# FFI RAPPORT

## MASKINLÆRINGSTEKNIKKER FOR KLASSIFISERING

NILSEN Asgeir Egil

**FFI/RAPPORT-2001/01501**

FFISYS/806/161

Godkjent
Kjeller 23 august 2001

Bent Erik Bakken
Forskningssjef

# MASKINLÆRINGSTEKNIKKER FOR KLASSIFISERING

NILSEN Asgeir Egil

FFI/RAPPORT-2001/01501

**FORSVARETS FORSKNINGSINSTITUTT**
**Norwegian Defence Research Establishment**
Postboks 25, 2027 Kjeller, Norge

**FORSVARETS FORSKNINGSINSTITUTT (FFI)**
**Norwegian Defence Research Establishment**

**P O BOX 25**
**NO-2027 KJELLER, NORWAY**

**UNCLASSIFIED**

**SECURITY CLASSIFICATION OF THIS PAGE (when data entered)**

## REPORT DOCUMENTATION PAGE

| 1) PUBL/REPORT NUMBER | 2) SECURITY CLASSIFICATION | 3) NUMBER OF PAGES |
|---|---|---|
| FFI/RAPPORT-2001/01501 | UNCLASSIFIED | 105 |
| **1a) PROJECT REFERENCE** | **2a) DECLASSIFICATION/DOWNGRADING SCHEDULE** | |
| FFISYS/806/161 | - | |

**4) TITLE**

MASKINLÆRINGSTEKNIKKER FOR KLASSIFISERING

Machine Learning Techniques for Classification

**5) NAMES OF AUTHOR(S) IN FULL (surname first)**

NILSEN Asgeir Egil

**6) DISTRIBUTION STATEMENT**

Approved for public release. Distribution unlimited. (Offentlig tilgjengelig)

**7) INDEXING TERMS**

| IN ENGLISH: | IN NORWEGIAN: |
|---|---|
| a) Machine learning | a) Maskinlæring |
| b) Artificial intelligence | b) Kunstig intelligens |
| c) Neural networks | c) Nevrale nett |
| d) | d) |
| e) | e) |

**THESAURUS REFERENCE:**

**8) ABSTRACT**

This diploma thesis treats classification tasks in machine learning. It focuses on three types of classification algorithms: support vector machines, neural nets and decision trees. The scope of its treatment of neural nets is limited to feed-forward nets (i.e. neural nets without internal feedback loops), trained by backpropagation. Several types of decision trees exist. This thesis focuses on an algorithm called C4.5. As a part of the thesis, a classification module was written in VisualWorks 3.0 Smalltalk. The module consists of three submodules, each implementing one of the classification techniques mentioned. The three modules were tested on non-trivial examples. Some of the experiments showed that the classifiers were able to learn the training data perfectly, and that their parameters could be set to achieve good generalisation as well.

| 9) DATE | AUTHORIZED BY | POSITION |
|---|---|---|
| | This page only | |
| 23 August 2001 | Bent Erik Bakken | Director of Research |

ISBN-82-464-0534-9

**UNCLASSIFIED**

**SECURITY CLASSIFICATION OF THIS PAGE (when data entered)**

## FORORD

Maskinlæring er en relativt ung vitenskap som de siste 15 årene har begynt å blomstre. Denne diplomoppgaven omhandler klassifisering som er et viktig område innenfor maskinlæring. Klassifisering i maskinlæring henspeiler på algoritmer som kan lære og som kan generalisere.

Jeg vil sende en takk min veileder Ole Martin Halck og faglærer Oddvar Hallingstad for hjelp og støtte underveis i denne diplomoppgaven og til FFI for at jeg fikk lov til å skrive denne oppgaven og for alle fasiliteter.

**SAMMENDRAG**

Denne diplomoppgaven er gitt av Forsvarets Forskningsinstitutt. Oppgaven omhandler klassifisering i maskinlæring.

Klassifisering i maskinlæring benyttes i f.eks mønstergjenkjenning og beslutningsstøtte. Klassifiseringsalgoritmene må kunne lære og de må kunne generalisere. Læringen kan foregå ved at klassifiseringsalgoritmene grupperer objekter som ligner på hverandre i klasser eller at de blir gitt eksempler på objekter som tilhører forskjellige klasser. Generaliseringen består i at klassifiseringsalgoritmene kan klassifisere objekter som de ikke har sett før.

Denne diplomoppgaven omhandler tre typer klassifiseringsalgoritmer: Støttevektormaskiner, nevrale nettverk og beslutningstrær.

Støttevektormaskiner er basert på å representere kunnskap som punkter i et egenskapsrom og separere forskjellige klasser ved hjelp av et optimalt plan som kalles hyperplan når det er mer enn tre egenskaper/dimensjoner.

Nevrale nettverk er bygd opp av kunstige nevroner som etterlikner noe av funksjonaliteten til biologiske nevroner. Signalene som overføres mellom nevronene blir vektet. Når nettverket lærer, justeres denne vektingen. Dette kan sammenliknes med justeringen av koblingen mellom biologiske nevroner. Oppgaven omhandler nettverk uten interne tilbakekoblinger ("Feed Forward"-nettverk) som læres opp ved hjelp av *tilbakepropagering* ("Backpropagation"-algoritmen).

Det fins flere typer beslutningstrær. I denne oppgaven fokuseres det på en algoritme som heter C4.5. Denne algoritmen benytter informasjonsteori for å konstruere beslutningstreet. Treet dannes ved å sammenligne nytten med hensyn til informasjonsverdi ved å splitte opp treningseksemplene på de mulige verdiene til en egenskap. Den egenskapen som gir best resultat velges som forgreningspunkt i treet og prosessen fortsetter rekursivt for de oppsplittede treningseksemplene.

Som en del av oppgaven ble det skrevet en klassifiseringsmodul i Smalltalk/ VisualWorks3.0. Modulen består av tre moduler som implementerer hver av de omtalte klassifiseringsalgoritmene. Modulen som implementerer nevrale nettverk var skrevet fra før av ansatte ved FFI, så i forbindelse med den ble det bare skrevet kode som knytter den til resten av modulen.

De tre modulene ble testet på ikke-trivielle eksempler. Resultatene viste at klassifiseringsmodulene hadde bra egenskaper med hensyn til å lære og til å klassifisere. Siden treningssettene var konsistente og uten støy, var det mulig å se om klassifiseringsalgoritmene var gode til å lære ved å lære dem opp og teste dem på samme

datasett. Dette ga ingen feil. Generaliseringsevnen viste seg å være bra. Algoritmene feilklassifiserte mellom 3% og 11% av testdatasettet. Det nevrale nettverket viste best resultater.

**INNHOLD**

# MASKINLÆRINGSTEKNIKKER FOR KLASSIFISERING

## 1 INNLEDNING

Denne diplomoppgaven omhandler klassifiseringsalgoritmer. Det fokuseres på kunstige nevrale nettverk, støttevektormaskiner (SVM) og beslutningstrær. Kun én algoritme for beslutningstrær vil bli omtalt: Denne algoritmen kalles C4.5 og er forfattet av J. Ross Quinlan.

En klassifiseringsalgoritme mottar et sett med treningsdata og genererer en klassifikator som er istand til å separere dette settet i klasser. Læres klassifikatoren opp til å separere treningsdata på angitte klasser, kalles dette overvåket læring. Dette skiller seg fra ikke-overvåket læring der klassifiseringsalgoritmen selv finner ut hvordan treningsdatasettet skal separeres. Nevrale nettverk kan læres opp på begge måter, mens SVM og beslutningstrær kun kan læres opp ved overvåket læring. Hensikten med opplæringen er å gjøre klassifikatoren i stand til å klassifisere objekter med ukjent klasse. Da er det nødvendig at klassifikatoren har evnen til å generalisere på grunnlag av informasjonen i treningsdatasettet.

De tre klassifikatorene som presenteres er prinsipielt forskjellige. Kunstige nevrale nettverk er bygd opp av elementer som etterlikner noe av funksjonaliteten til biologiske nevroner. Det kunstige nevrale nettverket prøver å fange opp noe av den menneskelige hjerne sin evne til å tenke. Det fins forskjellige typer nettverk og forskjellige algoritmer for opptrening. En kjent algoritme for opptrening av nettverk uten interne tilbakekoblinger (“Feed forward”-nettverk) er den *generaliserte delta-regelen for trening ved tilbakepropagering*. Før denne algoritmen ble oppdaget, fantes det ingen systematisk måte å trene opp kunstige nevrale nettverk med ubegrenset læreevne. Da denne algoritmen ble kjent på midten av 1980-tallet ga den maskinlæring som fagfelt en ny giv.

C4.5 er en etterkommer av algoritmen ID3. J. Ross Quinlan er også opphavsmannen til ID3. ID3 stammer fra 1960-tallet mens C4.5 stammer fra slutten av 1980-tallet. Siden den tid har C4.5 blitt modifisert og nye utgaver har sett dagens lys. Algoritmen C4.5 tar utgangspunkt i enkel informasjonsteori. Algoritmen konstruerer et beslutningstre der egenskapene til objektene danner forgreningspunkter og settet av mulige verdier til hver egenskap danner grenene ut fra tilhørende forgreningspunkt. Hver egenskap utnyttes kun én gang i hver gren.Ved konstruksjon av treet benyttes et grådighetsprinsipp ved valg av egenskap som skal danne forgreningspunkt. Kriteriet for valg av egenskap er forskjellen i informasjonsinnholdet i den delen av treningsdatasettet som finner veien til det aktuelle forgreningspunktet før og etter at dette datasettet splittes på denne egenskapens mulige verdier/forgreninger. Egenskapen som gir størst forskjell i informasjonsinnhold velges. Det fins ingen garanti for at dette gir den globale optimale klassifikatoren, men denne grådighetsalgoritmen har vist seg å fungere bra i praksis.

Opphavsmannen til støttevektormaskiner er Vladimir Vapnik. Det teoretiske grunnlaget for konseptet ble lagt på midten av 1960-tallet. Støttevektormaskiner tar utgangspunkt i at egenskapene til et objekt representerer et punkt i rommet som utspennes av egenskapvektorene. Støttevektormaskinen læres opp ved å finne det hyperplanet som separerer punktmengdene i

treningsdatabasen slik at klassifikatoren klassifiserer ukjente objekter så korrekt som mulig. På begynnelsen av 1990-tallet fikk konseptet ny giv da en metode ble oppdaget for å foreta lineære separasjoner i rom med mye høyere dimensjon enn rommet som egenskapsvektorene spenner ut, noe som gjorde det mulig å separere punktmengder av forskjellig klasse som ikke lot seg separere ved hjelp av hyperplan i egenskapsrommet. I motsetning til kunstige nevrale nettverk og beslutningstrær er ikke støttevektormaskin-algoritmen en grådighetsalgoritme. Men heller ikke denne algoritmen garanterer at den resulterende klassifikatoren blir optimal.

I de tre kapitlene som følger vil det bli gitt en nærmere beskrivelse av de tre klassifikatorene. Kapittel 5 gir en beskrivelse og resultater av en rekke forsøk som er blitt utført med de tre klassifikatorene med tildels egenproduserte implementasjoner. Kapittel 6 inneholder en diskusjon av resultatene og forskjellige aspekter ved de tre klassifikatorene. Kapittel 7 inneholder en kort oppsummering/konklusjon.

# 2 NEVRALE NETTVERK

## 2.1 Introduksjon

Kunstige nevrale nettverk er bygd opp av elementer som modellerer noe av funksjonaliteten til biologiske nevroner. Disse elementene er organisert på en måte som kan være relatert til anatomien i hjernen. Nevrale nettverk kan lære ved å modifisere sin oppførsel som respons på endringer i det omkringliggende miljøet. Når nevrale nettverk er trent, har de evnen til generalisere i den forstand at de til en viss grad ikke er sensitive for mindre variasjoner i inngangsdata. Nevrale nettverk vil derfor kunne klassifisere inngangsdata overlagret støy.

Ved å trene nevrale nettverk til å kjenne igjen et objekt ved å vise nettverket varianter av objektet eller objektet overlagret støy, vil nevrale nettverk være i stand til å abstrahere, slik at det vil kunne være i stand til å trekke ut de karakteristiske trekk ved elementene i treningsdatabasen, og reprodusere deres karakteristika. F.eks. ved å trene nevrale nettverk på å kjenne igjen bokstaven “A”, ved å benytte inngangsdata overlagret støy, kan nettverket reprodusere en idealisert “A” som det ikke har sett før.

Siden nevrale nettverk er et forsøk på å fange inn noe av den menneskelige intelligensen, er det egnet til oppgaver der nettverket må lære, generalisere og abstrahere. Dette er egenskaper som er viktige innen f.eks. mønstergjenkjenning.

Et nevralt nettverk vil alltid kunne klassifisere et objekt. Det fins imidlertid ingen garanti for at klassifiseringen er riktig. Dette er analogt for mennesker: En person kan ligne så mye på noen man kjenner, at man feilaktig antar at det er den aktuelle personen. Man vil egentlig bare kunne si at man kjenner personen med en viss sannsynlighet. Et nevralt nettverk vil også kunne vite noe om sannsynligheten for at klassifiseringen er korrekt. Noe av svakheten med et nevralt nettverk er at det i liten grad vil kunne forklare på en forståelig måte hvilke kriterier som ligger til grunn for en gitt klassifisering.Nevrale nettverk vil f.eks. ikke kunne generere enkle “if then else” setninger som vil kunne tolkes av et menneske. Analogt vil man kunne si at et menneske har noe av det samme problemet hvis klassifiseringsoppgaven blir kompleks nok. F.eks. vil det være vanskelig å lage et sett av “if then else” setninger som separerer en gjennomsnittlig person

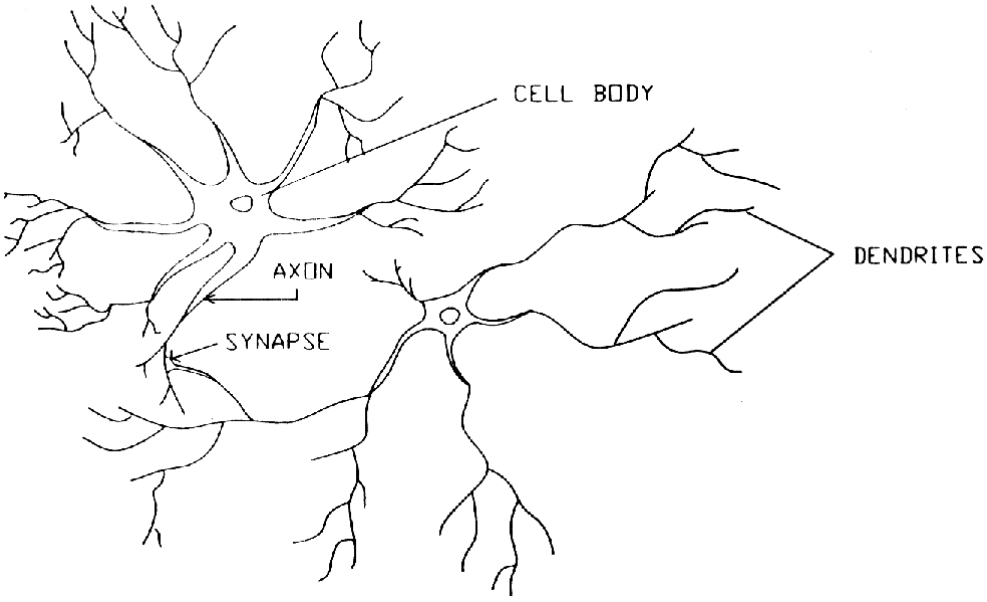
fra alle andre basert på ansiktets utseende. Stoffet som danner grunnlaget for teksten som følger fram til kapittel 2.3 er for det meste funnet i (6).

## 2.2 Fundamentale aspekter ved kunstige nevrale nettverk

### 2.2.1 Det biologiske nevron

Menneskets ufattelig kompliserte hjerne er et felt man bare har begrenset kunnskap om. Kunstige nevrale nettverk er et produkt av noe av den kunnskapen man har klart å tilegne seg innen dette feltet. For å få en forståelse av tankegangen bak kunstige nevrale nettverk er det derfor en fordel å ha innsikt i elementære aspekter innen biologiske nevrale nettverk.

Et biologisk nevralt nettverk er bygd opp av nevroner. Det er estimert at mennesket har et antall på rundt $10^{11}$ nevroner. Videre, at disse inngår i rundt $10^{15}$ forbindelser som kan være opptil rundt én meter lange. Hvert nevron har noen av de samme egenskapene som andre celler, men de har unike egenskaper for å motta, prosessere og sende elektrokjemiske signaler over nevrale kommunikasjonskanaler. Figur 2.1 (hentet fra (6) ) viser to biologiske nevroner. *Dendritter* forgrener seg fra cellekroppen til andre nevroner, hvor de mottar signaler i et knutepunkt som kalles *synapse*. Signalene kombineres i cellekroppen, og dersom det resulterende signalet overstiger en grense, responderer nevronet ved å avfyre elektriske impulser gjennom avfyringskanalen som kalles *axon*. I enden er axonkanalen forgrenet og danner forbindelseskanaler med et hundre- eller tusentalls dendritter. Noen signaler medvirker til avfyringen av impulser og noen motvirker. Det er ofte kun denne svært forenklede beskrivelsen av funksjonaliteten som modelleres i kunstige nevrale nettverk.




*Figur 2.1 Nevral cellestruktur*

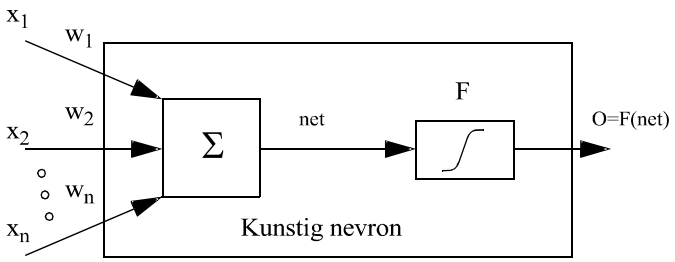
Virkeligheten er imidlertid kompleks. Det er f.eks. oppdaget hundrevis av forskjellige typer nevroner. Videre er egentlig ikke synapsen sammenlignbar med et elektrisk koblingspunkt. Overgangen mellom axonet og synapsen kalles *synaptisk kløft*. I dette området diffunderer stoffer som kalles *nevrotransmittere*. Disse skilles ut når axonet sender ut en elektrisk impuls.

Nevrotransmitterne er de egentlige signalene som prosesseres i cellekroppen. Det er til nå oppdaget mer enn tredve forskjellige nevrotransmittere.

Det forekommer også forbindelser mellom axoner, mellom axoner og cellekropp og mellom dendritter. Kunnskap om funksjonaliteten til disse koblingene er ennå lite kjent.

### 2.2.2 Det kunstige nevron

Det kunstige nevronet er en forenklet modell av det biologiske. *x*-signalene i figuren representerer signalene fra axonene. Disse multipliseres med sine vekter *w*, før summasjonen. Vektene representerer styrken til den synaptiske koblingen. Summasjonen representerer prosesseringen av signalene i cellekroppen.

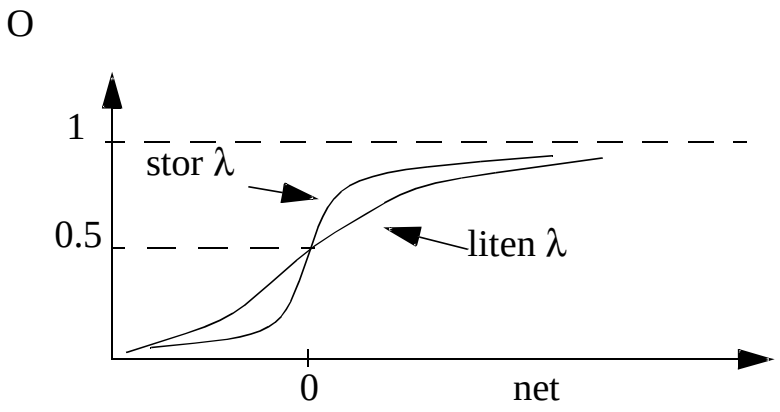



*Figur 2.2 Kunstig nevron*

Aktiveringsfunksjonen *F* har som regel "S"-form. Den mest vanlige, kalt logistikk-funksjonen, har den matematiske formen:

$$F(x) = \frac{1}{(1+e^{-lx})} \tag{2.1}$$

Denne funksjonen representerer en ulineær forsterkning. Det er ikke kjent for undertegnede om aktiveringsfunksjonen pr. i dag kan relateres til det biologiske nevronet, men det kan tenkes at overføringsfunksjonen mellom axon og dendritt har denne formen. Aktiveringsfunksjonen tjener flere formål: Den gjør det mulig for nevrale nettverk å approksimere enhver ulineær funksjon (uten den ulineære funksjonen, ville enhver flerlags nettverkskonfigurasjon kunne reduseres til ett lag). Funksjonen er deriverbar, noe som er viktig når læringsalgoritmer skal implementeres. I tillegg forsterker den små signaler mer enn store signaler (den deriverte er størst for x=0). Dette er viktig, ellers produserer ikke små signaler nok respons på utgangen til å skille dem fra overlagret støy på de store signalene.




*Figur 2.3 Logistikk-funksjonen*

En annen aktiveringsfunksjon som også er vanlig er:

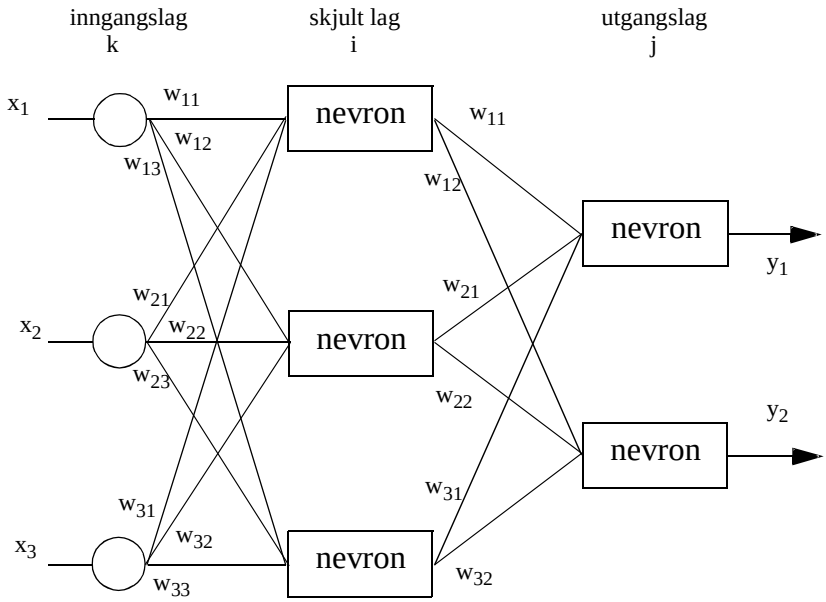
$$F(x) = \tanh(x) \tag{2.2}$$

Denne har også "S"-form, men er symmetrisk om origo.

Det kunstige nevronet skiller seg fra det biologiske på flere måter: Det er ingen tidsforsinkelser i det kunstige nevronet. Signalene i det biologiske nevronet forplanter seg bare med begrenset hastighet. I tillegg skal signalene prosesseres. Det biologiske nevronet sender diskrete signaler. Dette er modellert som kontinuerlige signaler i det kunstige nevronet. Her er det også et aspekt som vedrører axonets pulsfrekvens som kommer inn. Et viktig og hittil ubesvart spørsmål er hvor mye av det biologiske nevrons natur som er hensiktsmessig å modellere for å implementere et optimalt kunstig nevralt nettverk.

### 2.2.3 Nevral nettverksstruktur

Det fins flere typer kunstige nevrale nettverk. Det er mest vanlig å skille mellom "Feed Forward"-nettverk og "Recurrent"-nettverk. "Feed Forward"-nettverk har ingen tilbakekoblinger fra utgangen på et lag til egen inngang eller tidligere lag. Det er derimot kriteriet for et "Recurrent"-nettverk. Men "Feed Forward"-nettverk kan egentlig betraktes som er et spesialtilfelle av et "Recurrent"-nettverk. Figur 2.4 viser prinsipielt hvordan et "Feed Forward"-nettverk er bygd opp. Nettverket har to lag, inngangsvektoren har dimensjon lik tre og utgangsvektoren har dimensjon lik to. I det første laget er antall nevroner lik tre, og i det andre laget to. Sirklene representerer fordelingspunkter.




*Figur 2.4 “Feed Forward”-nettverk med ett skjult lag*

Et “Recurrent”-nettverk er åpenbart det som ligger nærmest biologien. Et biologisk nevralt nettverk kan ha et utall tilbakekoblingssløyfer. Pga. tilbakekoblingene og tidsforsinkelsene må det eksistere en dynamikk. Et nevrons utgangsverdi blir altså en funksjon av tidligere inngangsverdier og utgangsverdier. Derfor vil et slikt nettverk ha en form for korttidshukommelse. Dette kan ikke sies å være tilfellet for et “Feed Forward”-nettverk.

### 2.2.4 Læringsprosesser

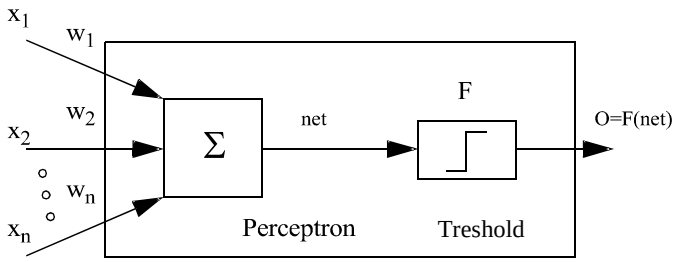
Formålet med læringen er å justere vektene slik at inngangsvektorene blir riktig klassifisert. Man skiller mellom to typer læringsprosesser: *Overvåket læring* og *ikke overvåket læring*. Prinsippet ved overvåket læring er at det nevrale nettverket blir matet med en mengde inngangsvektorer. Utgangsvektoren sammenliknes med den korrekte utgangsvektoren, og vektene justeres for redusere differansen mellom disse før sekvensen kjøres på nytt. Dette er en prosess som ikke trenger å konvergere eller den kan konvergere til et lokalt optimum.

I ikke overvåket læring mates nettverket med inngangsvektorer og vektene justeres med det formål at utgangene skal kunne separere de forskjellige klassene. Inngangsvektorer som likner på hverandre vil da gi tilnærmet samme utgangsvektor.Utgangsvektoren sammenliknes altså ikke med en idealisert utgangsvektor. Før læringsprosessen er omme er det vanskelig å vite hvordan utgangsvektoren responderer på de forskjellige klassene, men dette kan finnes i etterkant ved testing.

Ikke overvåket læring hevdes å være mer nærliggende biologisk læring. Årsaken er at idealiserte utgangsvektorer ikke alltid forekommer når biologiske individer lærer. F.eks kan det tenkes at barn lærer at runde ting kan rulle ved å observere at mer eller mindre runde ting ruller. Det er nok ikke vanlig at foreldrene prøver å formidle denne kunnskapen ved f.eks å gi ros dersom poden kan signalisere at et eple ruller bedre enn en kloss. Imidlertid vil jeg hevde det at mye læringen i en students hverdag er overvåket læring. “2+2=5“ skrives sjelden på tavla uten at læreren protesterer.

### 2.2.5 Representasjon i “Feed Forward”-nettverk

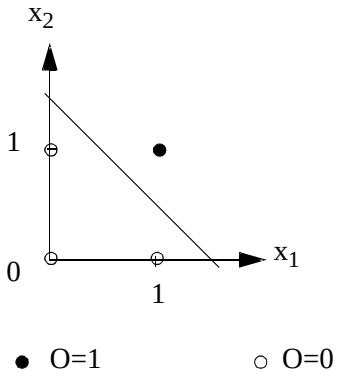
Et perseptron er et kunstig nevron, der aktiveringsfunksjonen *F* kun er en terskelfunksjon. På 1960-tallet ble ettlags nettverk med perceptroner viet mye oppmerksomhet. Flerlags nettverk fantes det på det tidspunkt ikke treningsalgoritmer for. Et innblikk i “Feed Forward”-nettverk bestående av perceptroner, gir også et innblikk i hva som kan representeres gjennom forskjellige konfigurasjoner av denne type nettverk.




*Figur 2.5 Perseptron*

Dersom inngangsvektoren f.eks er todimensjonal og binær, vil fire inngangskombinasjoner være mulig av ($x_1$,$x_2$): (0,0), (0,1), (1,0), (1,1). Utgangsvektoren kan være 0 eller 1. Den boolske funksjonen: $O = x_1$ & $x_2$ , vil ha sannhetstabellen gitt ved tabell 2.1 .

| $x_1$ | $x_2$ | $O$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |




*Tabell 2.1 Sannhetstabell*

*Figur 2.6 Grafisk framstilling av punktene i sannhetstabellen*

Inngangen på terskelelementet vil være gitt ved:

$$net = x_1 w_1 + x_2 w_2 \tag{2.3}$$

Når *net* er mindre enn terskel lik *T*, vil utgangen være null, og ellers lik én. Dvs.

$$\begin{aligned} T \geq (x_1 w_1 + x_2 w_2) &\Rightarrow O = 0 \\ T < (x_1 w_1 + x_2 w_2) &\Rightarrow O = 1 \end{aligned} \tag{2.4}$$

Grafisk vil linjen:

$$T = x_1 w_1 + x_2 w_2 \quad , \tag{2.5}$$

skille punktene som har *O* lik 0 fra de som har *O* lik 1. Settes for eksempel *T*=0,75 og $w_1$ = $w_2$ =0,5 vil dette gi linjen:

$$x_2 = -x_1 + 1{,}5 \quad , \tag{2.6}$$

som er tegnet inn på figuren. Et perseptron vil kunne implementere alle de grunnleggende "boolske" funksjonene uten XOR- og NXOR-funksjonen. For XOR- og NXOR-funksjonen vil det ikke være mulig å segregere to av punktene med samme utgangsverdi ved hjelp av kun én linje.

Dersom flere perceptroner benyttes i det første laget, vil hver representere en linje. For punkter på den ene siden av linja vil utgangen på perceptronet bli lik én og for punkter på den andre siden vil utgangen bli lik null. Ved å utføre boolske operasjoner (typisk: AND-operasjon) på utgangene i det første laget, vil en hvilken som helst region kunne separeres fra resten. De boolske operasjonene implementeres ved å legge på ett nytt lag i nettverket. Dersom det er flere separate regioner, vil disse kunne kombineres ved å utføre boolske operasjoner (typisk: OR-operasjon) på utgangene i det andre laget. De boolske operasjonene implementeres ved å legge på ett nytt lag i nettverket. Dette kan generaliseres til et n-dimensjonalt rom, der linjene i eksempelet byttes ut med hyperplan. Derfor kan i prinsippet alle segmenteringer representeres ved et trelags nettverk. Det er faktisk kun nødvendig med to lag. Flere forfattere har uavhengig ført bevis for at et nettverk med kun ett skjult lag og sigmoid-aktiveringsfunksjoner (som f.eks varianter av funksjonene gitt i (2.1) og (2.2)) kan approksimere en vilkårlig funksjon med vilkårlig nøyaktighet. Noen bevis vil ikke bli gjengitt her, men det henvises istedet til (7).

## 2.3 Delta-regelen og tilbakepropagering

*Delta-regel*-algoritmen er en algoritme for å trene vektene i et nettverk med kun ett lag nevroner. *Tilbakepropagering* er en generalisering av delta-regelen for flere lag nevroner.

Tilbakepropagering ble beskrevet av flere forfattere på omtrent samme tid. Imidlertid tillegges arbeidet til Rumelhart, Hinton og Williams (9) mye av æren for metoden. Metoden deres bærer navnet: *Den generaliserte delta-regelen for trening ved tilbakepropagering*. På midten av 1980-tallet begynte algoritmen å bli kjent i vitenskapelige miljøer. Ideen bak algoritmen er å benytte

differansen mellom ønsket respons og virkelig respons på utgangen, og forplante denne feilen gjennom nettverket. Feilen benyttes til å justere vektene. Dette gjøres for hele treningssettet og gjentas helt til feilen på utgangen er akseptabel eller antall iterasjoner har nådd en grense. Stoffet som danner grunnlaget for den følgende beskrivelsen av delta-regelen og tilbakepropagering er funnet i (1).

### 2.3.1 Delta-regelen

Algoritmen kan beskrives ved følgende steg:

Steg 1: Gi vektene små tilfeldige verdier.

Steg 2: Velg en inngangsvektor med tilhørende utgangsvektor.

Steg 3: Utfør en beregning av den virkelige utgangsvektoren og lagre alle aktuelle mellomregninger.

Steg 4: Beregn differansen mellom ønsket og beregnet utgangsvektor. Juster vektene.

Steg 5: Hvis denne differansen er mindre enn gitt av et bestemt kriterium for alle treningspar i treningssettet eller antall iterasjoner har nådd en bestemt grense, så avsluttes sekvensen. Ellers, returner til steg 2.

Algoritmen minimaliserer summen av kvadratene til differansen mellom ønsket respons kalt $d_i$, på utgang $i$, og virkelig respons, kalt $O_i$. Denne differansen er representert ved *Error* som er gitt ved ligning:

$$Error = 0{,}5\sum_i (d_i - O_i)^2 \tag{2.7}$$

Vektene justeres ved å foreta et skritt i optimal retning på tangentflaten til *Error*. Matematisk kan dette uttrykkes som følger:

$$\Delta w_k = -c\frac{\partial}{\partial w_k} Error \quad , \tag{2.8}$$

der $c$ er en positiv konstant som representerer læringsraten.

Videre følger en matematisk øvelse for å substituere kjente størrelser inn i den foregående ligningen.

Kjerneregelen anvendt på (2.7) gir:

$$\frac{\partial}{\partial w_k} Error = \frac{\partial Error}{\partial O_i}\frac{\partial O_i}{\partial w_k} \tag{2.9}$$

Den partielt deriverte av (2.7) er gitt ved:

$$\frac{\partial Error}{\partial O_i} = -(d_i - O_i) \tag{2.10}$$

$O_i$ er definert som utgangen av aktiveringselementet og er gitt ved:

$$O_i = F(net_i) = F(\sum x_i w_i) \quad , \tag{2.11}$$

der $F$ er aktiveringsfunksjonen.

Den partielt deriverte av $O_i$ mhp. vekt $w_k$ er gitt ved:

$$\frac{\partial O_i}{\partial w_k} = x_k F'(net_i) \tag{2.12}$$

(2.10) og (2.12) innsatt i (2.9) gir:

$$\frac{\partial}{\partial w_k} Error = -(d_i - O_i) x_k F'(net_i) \tag{2.13}$$

(2.13) innsatt i (2.8) gir:

$$\Delta w_k = -c \frac{\partial}{\partial w_k} Error = c(d_i - O_i) x_k F'(net_i) \tag{2.14}$$

Ligning 2.14 benyttes i algoritmens steg 4, der vektene justeres.

### 2.3.2 Tilbakepropagering

Tilbakepropageringsalgoritmen kan beskrives ved de samme fem steg som i Delta-regel-algoritmen. Men prosedyren for å justere vektene blir forskjellig pga. at tilbakepropageringsalgoritmen også skal kunne håndtere nettverk med skjulte lag.

For å justere vektene som tilhører utgangslaget benyttes Delta-regel-algoritmen beskrevet i det foregående. Dersom logistikk-funksjonen benyttes, vil den deriverte av aktiveringsfunksjonen være gitt ved:

$$F'(net_i) = F(net_i)(1 - F(net_i)) = O_{i(}(1 - O_i) \tag{2.15}$$

(2.15) innsatt i (2.14) leder til (2.16) som er formelen for å justere vektene i utgangslaget (representert ved indeks *j*).

$$\Delta w_{jk} = -c(d_j - O_j) O_j (1 - O_j) x_{jk} \tag{2.16}$$

Siden $d_j$ ikke er definert for de skjulte lagene, kan ikke denne formelen benyttes for å justere de interne vektene.

I det generelle tilfellet justeres vektene ved å foreta et skritt i optimal retning på tangentflaten til *Error*. Matematisk kan dette uttrykkes som følger:

$$\Delta w_{ki} = -c \frac{\partial}{\partial w_{ki}} Error \tag{2.17}$$

Indeks *i* og *k* representerer to etterfølgende lag. For å utlede formelen for å justere de interne vektene, gjøres videre den antakelsen at nettverket har ett skjult lag representert ved indeks *i*, utgangslaget er representert ved indeks *j* og inngangslaget er representert ved indeks *k* (se figur 2.4).

Videre følger en matematisk øvelse for å substituere kjente størrelser inn i den foregående ligningen.

Kjerneregelen gir:

$$\frac{\partial}{\partial w_{ki}} Error = \frac{\partial}{\partial net_i} Error \frac{\partial}{\partial w_{ki}} net_i \tag{2.18}$$

$w_{\mathrm{ki}}$, som er vekten som multiplisert med utgangen til nevron *k* i *k*-laget, danner én av inngangene til nevron *i* i *i*-laget.

Per definisjon er $net_{\mathrm{i}}$ gitt ved:

$$net_i = \sum_k x_k w_{ki} \tag{2.19}$$

Den partielt deriverte av (2.19) med hensyn på $w_{\mathrm{ki}}$ er gitt ved:

$$\frac{\partial}{\partial w_{ki}} net_i = x_k \tag{2.20}$$

Per definisjon er delta til nevron *i* i *i*-laget gitt ved:

$$-delta_i = \frac{\partial}{\partial net_i} Error = \frac{\partial}{\partial O_i} Error \frac{\partial O_i}{\partial net_i} \tag{2.21}$$

(2.15) gir:

$$\frac{\partial O_i}{\partial net_i} = O_i (1 - O_i) \tag{2.22}$$

Kjerneregelen anvendt gir:

(2.23)

$$\frac{\partial}{\partial O_i} Error = \sum_j \frac{\partial}{\partial net_j} Error \frac{\partial net_j}{\partial O_i}$$

Årsaken til summasjonen er at utgang fra nevron *i* i *i*-laget danner forbindelse med inngangene til et antall av nevronene i *j*-laget.

Dersom $O_i$ spiller rollen som $x_k$ i (2.19) leder dette til:

$$\frac{\partial net_j}{\partial O_i} = w_{ij} \tag{2.24}$$

Delta for utgangslaget *j* er gitt ved:

$$delta_j = -\frac{\partial}{\partial net_j} Error = (d_j - O_j)F'(net_j) = (d_j - O_j)O_j(1 - O_j) \tag{2.25}$$

Den siste ligningen initialiserer rekursjonen. Det er kun aktuelt med rekursjon dersom det er mer enn ett skjult lag.

Ligning 2.24 og 2.25 innsatt i 2.23 gir:

$$\frac{\partial}{\partial O_i} Error = \sum_j - delta_j w_{ij} \tag{2.26}$$

Ligning 2.26 og 2.22 innsatt i 2.21 gir:

$$- delta_i = O_i(1 - O_i)\sum_j - delta_j w_{ij} \tag{2.27}$$

Den siste ligningen er rekursiv. Ligningen forplanter feilen på utgangen gjennom nettverket ett nivå av gangen. Det er her verdt å legge merke til at forsterkningen til aktiveringsfunksjonen (se ligning 2.22) inngår som faktor i ligningen. Der vektene er små, reduseres forplantningen av feilen bakover i nettverket. Forsterkningen til aktiveringsfunksjonen er størst for små inngangsverdier. Dette medfører at små verdier på inngangene til nevronene også kan påvirke vektene i justeringsprosessen. Utfra (2.27) kan man se at vektene ikke kan initialiseres slik at alle vektene er lik null. Det er fordi ligningen da blir lik null og ingen feil forplantes videre. Initialiseringsverdiene kan også være kritiske for algoritmens konvergens, når de er ulik null.

Ligning (2.21) og (2.20) innsatt i (2.18) gir:

$$\frac{\partial}{\partial w_{ki}} Error = \frac{\partial}{\partial net_i} Error \frac{\partial}{\partial w_{ki}} net_i = -delta_i x_k \tag{2.28}$$

Ligning 2.27 innsatt 2.28 gir:

(2.29)

$$\frac{\partial}{\partial w_{ki}} Error = x_k O_i (1 - O_i) \sum_j - delta_j w_{ij}$$

Ligning 2.29 innsatt 2.17 gir:

$$\Delta w_{ki} = -c \frac{\partial}{\partial w_{ki}} Error = -c x_k O_i (1 - O_i) \sum_j - delta_j w_{ij} \tag{2.30}$$

Ligning 2.30 og 2.27 er de primære ligningene som benyttes i algoritmens steg 4.

Læringsrate *c* kan være en kritisk parameter. Dersom *c* er for stor, konvergerer ikke algoritmen, og dersom *c* er for liten konvergerer algoritmen sakte. Ofte kan det være hensiktsmessig at *c* er stor i begynnelsen slik at alle klassene oppdages raskt, og reduseres etterhvert. Siden *Error*-funksjonen kan ha lokale minimum, er konvergens til global løsning ikke garantert.

# 3 STØTTEVEKTORMASKINER

Støttevektormaskiner (SVM) er en type klassifiseringsalgoritme som i de senere årene har vært viet endel oppmerksomhet. Det teoretiske fundamentet for SVM er Vladimir Vapnik tilkjent mye av æren for. Videre gis det en enkel beskrivelse av konseptet før noe av bakgrunnsteorien presenteres. Stoffet som danner grunnlaget for teksten som følger i dette kapittelet er funnet i (3), (4), (5) og (8).

## 3.1 Lineære støttevektormaskiner

### 3.1.1 Hyperplan

*Det optimale hyperplanet:* Det antas et sett med treningsvektorer som tilhører to forskjellige klasser,

$$(y_1, \mathbf{x_1}), \cdots, (y_l, \mathbf{x}_l), \qquad \mathbf{x} \in R^n, \qquad y \in \{-1,1\}$$

og et hyperplan gitt ved:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{3.1}$$

Treningsvektorene representerer punkter i et n-dimensjonalt rom. Hver egenskap i $\boldsymbol{x}$ representerer en ny dimensjon. Vektoren $\boldsymbol{w}$ er normalvektoren til planet og parameteren *b* kan tolkes utfra at *b* dividert med den euklidiske normen til vektoren $\boldsymbol{w}$ ($b/\|\boldsymbol{w}\|$) er den korteste avstanden fra origo til planet.

Et sett med vektorer er sagt å være optimalt separert av et hyperplan hvis klassene er separert uten feil og distansen mellom nærmeste vektor og hyperplanet er maksimal. Videre antas det at et slikt hyperplan eksisterer. Det vil si at datasettet er lineært separabelt.

*Kanoniske hyperplan:* Distansen fra et punkt $\boldsymbol{x}$ til et hyperplan med parametre ($\boldsymbol{w}$,*b*) er gitt ved:

$$d(\mathbf{w}, b, \mathbf{x}) = \frac{|\mathbf{w} \cdot \mathbf{x} + b|}{\|\mathbf{w}\|} \tag{3.2}$$

Det introduseres et såkalt *kanonisk hyperplan* der parameterne ***w***, *b* til hyperplanet blir begrenset av ligningen:

$$\min_{x_i} |\mathbf{w} \cdot \mathbf{x} + b| = 1 \tag{3.3}$$

Ligning (3.3) sier at for egenskapsvektorer (x-vektorer) i treningsdatabasen, skal den minimale verdien av absoluttverdien til utrykket i (3.3), være lik tallet 1. Dette medfører:

$$\min d(\mathbf{w}, b, \mathbf{x}) = \frac{1}{\|\mathbf{w}\|} \tag{3.4}$$

Dette betyr at resultatet av begrensningen blir at distansen fra nærmeste punkt i datasettet til hyperplanet blir lik den inverse av normen til vekt-vektoren ***w***.

Avstandene $d_+$ og *d-* innføres. Disse er henholdsvis den korteste distansen mellom positiv klasse og hyperplanet og den korteste distansen mellom negativ klasse og hyperplanet. Marginen som skiller klassene er da definert som summen av avstandene $d_+$ og $d_-$. I henhold til ligningen over må *marginen* som skiller de to klassene være lik $2/\|\boldsymbol{w}\|$ ($d_+ = d_- = 1/\|\boldsymbol{w}\|$). Det optimale hyperplanet fås ved å finne hyperplanet med størst margin. Fra (3.4) går det fram at maksimering av marginen er ekvivalent med minimering av $\|\boldsymbol{w}\|$. Dersom det er to klasser kan problemet formuleres som følger:

$$\min \quad \Psi(\mathbf{w}) = \frac{1}{2} \| \mathbf{w} \|^2 \tag{3.5}$$

$$\mathbf{x_i} \cdot \mathbf{w} + b \geq 1, \quad for \quad y_i = 1 \tag{3.6}$$

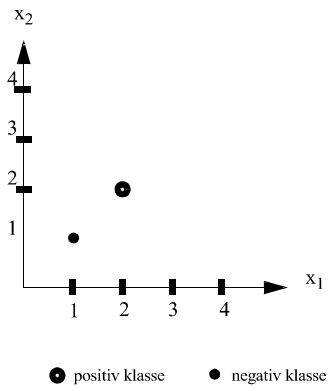
$$\mathbf{x_i} \cdot \mathbf{w} + b \leq -1, \quad for \quad y_i = -1 \tag{3.7}$$

Likningene:

$$\mathbf{x_i} \cdot \mathbf{w} + b = 1 \tag{3.8}$$

$$\mathbf{x_i} \cdot \mathbf{w} + b = -1 \tag{3.9}$$

danner to hyperplan som ligger parallellforskjøvet på hver sin side av hyperplanet i (3.1) med avstand *d*, gitt av (3.4). Området mellom disse hyperplanene kalles marginen, og her ligger ingen punkter dersom det er mulig å separere klassene. Alle punktene av den ene klassen ligger på en side av hyperplanene gitt av (3.8) og (3.9) mens punktene av den andre klassen ligger på den motsatte siden av hyperplanene gitt av (3.8) og (3.9) . For å illustrere noen enkle poenger gis det et eksempel. Figur 3.1 viser to egenskapsvektorer av forskjellig klasse representert ved to punkter i planet. Informasjonen på figur 3.1 er formalisert i tabell 3.1.




*Figur 3.1 Grafisk fremstilling av to egenskapsvektorer av forskjellig klasse*

| | $x_1$ | $x_2$ |
|---|---|---|
| $x_1$ | 1 | 2 |
| $x_2$ | 1 | 2 |
| $y$ | -1 | 1 |

*Tabell 3.1 Tabellen formaliserer informasjonen på figur 3.1.*

I henhold til (3.5), (3.6) og (3.7), kan dette skrives:

$$\min \quad \Psi(\mathbf{w}) = \frac{1}{2} \| \mathbf{w} \|^2$$
$$2w_1 + 2w_2 + b \geq 1 \quad for \quad y_2 = 1$$
$$w_1 + w_2 + b \leq -1 \quad for \quad y_1 = -1$$

Siden det er kun ett punkt i hver klasse vil punktene være de nærmeste punktene til hyperplanet i sin klasse (punktene blir da *støttevektorer*). Derfor kan problemet forenkles til kun å inkludere likhetsbetingelser og løsningen blir som følger:

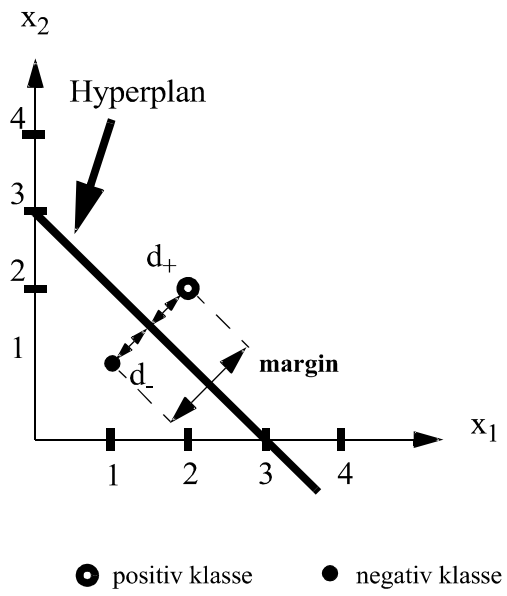
$$\min \quad \Psi(\mathbf{w}) = \frac{1}{2} \| \mathbf{w} \|^2$$
$$2w_1 + 2w_2 + b = 1 \quad for \quad y_2 = 1$$
$$w_1 + w_2 + b = -1 \quad for \quad y_1 = -1$$

$$\min \quad \Psi(\mathbf{w}) = 0{,}5\,(w_1^{\,2} + w_2^{\,2})$$
$$\Rightarrow 2w_1 + 2w_2 + b = 1$$
$$w_1 + w_2 = 2$$

$$w_1 = 1$$
$$\Rightarrow \quad w_2 = 1$$
$$b = -3$$

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$
$$\Rightarrow x_2 = -x_1 + 3 \qquad (3.10)$$

Ligning (3.10) er ligningen for hyperplanet som i to dimensjoner kun er en linje. Hyperplanet er tegnet inn på figur 3.2.



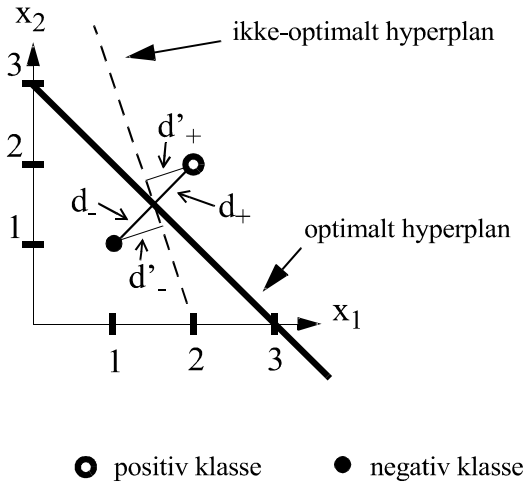

*Figur 3.2 Grafisk fremstilling av to egenskapsvektorer av forskjellig klasse separert av et hyperplan*

Hyperplanets normal vil være parallell med vektoren mellom de to punktene på figuren. Årsaken til det kan det være verdt å se nærmere på. På neste figur er det tegnet inn et ikke-optimalt hyperplan for å illustrere dette. Parametrene velges på grunnlag av begrensningene beskrevet tidligere gitt ved:

$$2w_1 + 2w_2 + b = 1$$
$$w_1 + w_2 = 2$$

Velger $w_1$ lik 1.5, da blir $w_2$ lik 0.5 og $b$ lik -3. Dette gir hyperplanet:

$$\begin{aligned} & \mathbf{w} \cdot \mathbf{x} + b = 0 \\ \Rightarrow \quad & 1{,}5x_1 + 0{,}5x_2 - 3 = 0 \\ \Rightarrow \quad & x_2 = -3x_1 + 6 \end{aligned} \tag{3.11}$$




*Figur 3.3* *Grafisk fremstilling av to egenskapsvektorer av forskjellig klasse separert av et optimalt hyperplan (hel-trukket) og et ikke-optimalt hyperplan (stiplet).*

Det går frem av figur 3.3 at marginen ($d'_+ + d'_-$) for det ikke-optimale hyperplanet, er mindre enn marginen ($d_+ + d_-$) for det optimale hyperplanet. Begrensningene gjør at $d'_+ = d'_- = 1/\|\boldsymbol{w}\|$ som beskrevet tidligere, slik at $\|\boldsymbol{w}\|$ må være større for det ikke-optimale hyperplanet enn for det optimale hyperplanet.

### 3.1.2 Det duale problemet

Optimeringsproblemet kan omformuleres slik:

$$\begin{aligned} & \min \quad \Psi(\mathbf{w}) = \frac{1}{2} \| \mathbf{w} \|^2 \\ & (\mathbf{x_i} \cdot \mathbf{w} + b) y_i \geq 1 \quad , \end{aligned} \tag{3.12}$$

og videre til et Lagrangeproblem med primal Lagrangefunksjon:

$$\min_{\mathbf{w},b} \quad L(\mathbf{w},b,a) = \min_{\mathbf{w},b}(1/2 \parallel \mathbf{w} \parallel^2 - \sum_{i=1}^{l} a_i\{[\mathbf{x}_i \cdot \mathbf{w} + b]y_i - 1\}) \quad , \qquad der \quad a_i \geq 0 \tag{3.13}$$

For å gjøre det enklere å løse dette optimeringsproblemet dvs. finne *b* og ***w***, er det vanlig å løse det duale problemet først. Det duale problemet formuleres slik:

$$\max_a \quad W(a) = \max_a \quad \{\min_{\mathbf{w},b} \quad L(\mathbf{w},b,a)\} \tag{3.14}$$

Det primale problemet betraktes først. Følgende betingelser må være oppfylt:

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \quad \sum_{i=1}^{l} a_i y_i = 0$$

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \Rightarrow \quad \sum_{i=1}^{l} a_i \mathbf{x_i} y_i = 0 \tag{3.15}$$

Innsatt i Lagrangefunksjonen, i to trinn, for det primale problemet gir dette:

$$\max_a \quad W(a,b) = \max_a \left[ \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} a_i a_j y_i y_j (\mathbf{x_i} \cdot \mathbf{x_j}) - \sum_{i=1}^{l} a_i \left\{ \left[ \mathbf{x_i} \cdot \sum_{j=1}^{l} a_j \mathbf{x_j} y_j + b \right] y_i - 1 \right\} \right]$$

$$= \max_a \left[ \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} a_i a_j y_i y_j (\mathbf{x_i} \cdot \mathbf{x_j}) - \sum_{i=1}^{l} \sum_{j=1}^{l} a_i a_j y_i y_j (\mathbf{x_i} \cdot \mathbf{x_j}) - b \sum_{i=1}^{l} a_i y_i + \sum_{i=1}^{l} a_i \right] \tag{3.16}$$

$$\max_a \quad W(a,b) = \max_a \left[ -\frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} a_i a_j y_i y_j (\mathbf{x_i} \cdot \mathbf{x_j}) + \sum_{i=1}^{l} a_i \right]$$

$$a = \arg\max_a \quad W(a,b) = \arg\max_a \left[ -\frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} a_i a_j y_i y_j (\mathbf{x_i} \cdot \mathbf{x_j}) + \sum_{i=1}^{l} a_i \right]$$

$$a_i \geq 0 \tag{3.17}$$

$$\sum_{i=1}^{l} a_i y_i = 0$$

Som det går fram av ligningene har optimaliseringsproblemet kvadratisk kriteriefunksjon og lineære bibetingelser.

Når det duale problemet er løst kan *b* og ***w*** beregnes som følger:

$$\mathbf{w} = \sum_{i=1}^{l} a_i \mathbf{x_i} y_i$$

$$b = -\frac{1}{2} \mathbf{w} \cdot [\mathbf{x_r} + \mathbf{x_s}] \quad , \tag{3.18}$$

der $\boldsymbol{x}_r$ og $\boldsymbol{x}_s$ er støttevektorer fra hver klasse som ligger på hver sitt plan gitt av (3.8) og (3.9). Det vil si at følgende krav er tilfredsstilt:

$$a_r, a_s > 0, \quad y_r = 1, \quad y_s = -1. \tag{3.19}$$

I denne sammenheng er det verdt å poengtere at en støttevektor er definert som en vektor $\boldsymbol{x}_i$ med tilhørende $\alpha_i$ større enn null.
For å se at beregningsmetodikken for *b* stemmer, er det som nevnt slik at to punkter som tilfredsstiller kravene over ligger på hver sin side av hyperplanet og ligger i hvert sitt grenseplan. Da kan formelen for *b* utledes som følger:

$$\begin{aligned} \mathbf{x_r} \cdot \mathbf{w} + b &= 1 \\ \mathbf{x_s} \cdot \mathbf{w} + b &= -1 \\ \Rightarrow \mathbf{w} \cdot (\mathbf{x_r} + \mathbf{x_s}) + 2b &= 0 \\ \Rightarrow b &= -0{,}5\mathbf{w} \cdot [\mathbf{x_r} + \mathbf{x_s}] \end{aligned} \tag{3.20}$$

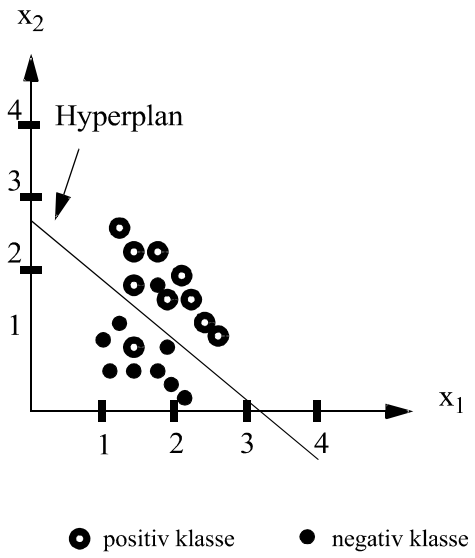
*b* og $\boldsymbol{w}$ er parametrene til klassifikatoren. En hard klassifikator er gitt ved:

$$f(\mathbf{x}) = \text{sign}(\mathbf{x} \cdot \mathbf{w} + b) \tag{3.21}$$

Dersom *f* er lik 1 for en x-vektor indikerer dette positiv klasse, og dersom *f* er lik -1 for en x-vektor indikerer dette negativ klasse.

### 3.1.3 Generalisering av tilfeller som antas lineært separable

Dersom datasettet er befengt med støy slik at et optimalt hyperplan ikke eksisterer må den foregående teorien modifiseres til å takle dette. I figur 3.4 går det fram hva som menes med støy. Noen tilfeldige punkter havner i klyngen sammen med punkter som tilhører den andre klassen og gjør det umulig å separere klassene med et optimalt hyperplan.




*Figur 3.4* *Grafisk fremstilling av egenskapsvektorer av forskjellig klasse separert av et hyperplan. To støy-punkter er inkludert i punkt-konfigurasjonen.*

For å tillate at punkter kan ligge på feil side av grenseplanene gitt av:

$$(\mathbf{x}_i \cdot \mathbf{w} + b)\, y_i = 1\,, \tag{3.22}$$

må kravet:

$$(\mathbf{x}_i \cdot \mathbf{w} + b) y_i \geq 1 \tag{3.23}$$

slakkes på. Dette gjøres ved å inkludere en slakkvariabel $\xi_i$ som kreves større eller lik null. Når slakkvariabelen er positiv betyr det at punktet ligger innenfor marginen eller på feil side av marginen som illustrert på figur 3.5. Slakkvariabelen er et mål for misklassifisering, og det er ønskelig at hyperplanet skal være slik at den totale graden av misklassifisering er minimal. Derfor inkluderes summen av alle slakkvariablene i optimalfunksjonen. Problemet blir da seende slik ut:

$$\begin{aligned} \min \quad & \Psi(\mathbf{w}) = \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i=1}^{l} x_i \\ & (\mathbf{x}_i \cdot \mathbf{w} + b) y_i \geq 1 - x_i \\ & x_i \geq 0, \end{aligned} \tag{3.24}$$

der *C* er en positiv konstant.

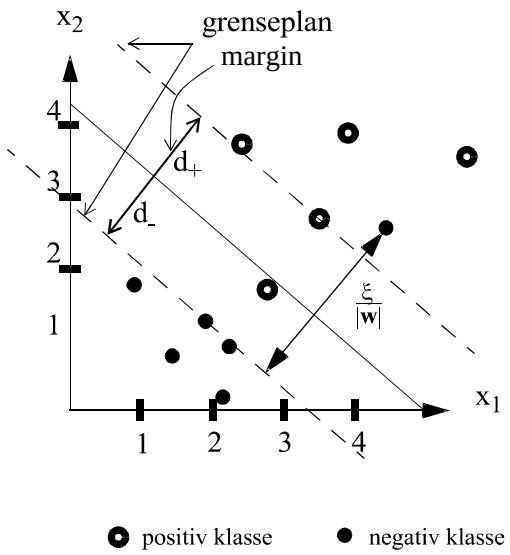
Dette kan omformes til et Lagrange-problem. Imidlertid medfører ikke innføringen av slakkvariabelen at formen til det duale Lagrange-problemet forandres i særlig grad. Det duale problemet blir seende slik ut:

$$a = \arg\max_a \quad W(a,b) = \arg\max_a \left[ -\frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} a_i a_j y_i y_j (\mathbf{x_i} \cdot \mathbf{x_j}) + \sum_{i=1}^{l} a_i \right]$$

$$C \geq a_i \geq 0 \tag{3.25}$$

$$\sum_{i=1}^{l} a_i y_i = 0$$

Den eneste endringen er at $\alpha_i$ får $C$ som øvre skranke.




*Figur 3.5* *Grafisk fremstilling av egenskapsvektorer av forskjellig klasse separert av et hyperplan. Ett støy-punkt er inkludert i punkt-konfigurasjonen.*

**Betraktninger knyttet til parameteren *C***

Når *C* øker, straffes det mer å anlegge hyperplanet slik at punkter havner innenfor marginen eller på feil side av marginen. Dette medfører at marginen minker og at de nevnte punktene får større innflytelse på hyperplanet. Derfor kan evnen til å generalisere reduseres når det er støydata til stede. Dersom *C* minker, minker vektleggingen av punkter som havner på feil side av grenseplanene. Dvs. at marginen øker. Dersom punkter som ikke er støydata av den grunn blir liggende innenfor marginen, kan dette medføre at generaliseringsevnen minker. Dersom punkter som ikke er støydata blir liggende på feil side av hyperplanet, er det hensiktsmessig å benytte en ulineær SVM.

## 3.2 Ulineære støttevektormaskiner

### 3.2.1 Introduksjon av ulineære SVM

Dersom klassifiseringsproblemet er av en slik art at det ikke er naturlig å utføre en lineær separasjon, kan en ulineær SVM benyttes. Den foregående teorien kan med visse modifikasjoner benyttes. Metoden går ut å utføre en lineær separasjon som beskrevet i det foregående i et rom med høyere dimensjon enn det inngangsvektorene eksisterer i.

De lineært ikke-separable inngangsvektorene transformeres via en transformasjon $\Phi(\boldsymbol{x})$ til et rom med høyere dimensjon der de er lineært separable. Optimeringsproblemet blir da seende ut som i (3.26).

$$a = \arg\max_a \quad W(a,b) = \arg\max_a \left[ -\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} a_i a_j y_i y_j (\Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j})) + \sum_{i=1}^{l} a_i \right]$$

$$C \geq a_i \geq 0 \tag{3.26}$$

$$\sum_{i=1}^{l} a_i y_i = 0$$

Dersom transformasjonen $\Phi$ benyttes på x-vektorene i ligningene for klassifikator-funksjonen *f(***x**), forekommer denne kun som indreprodukt slik som i (3.26). Dette indreproduktet kalles kjernen. Kjernen kan altså skrives:

$$K(\mathbf{x},\mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}). \tag{3.27}$$

Når indreproduktet erstattes av kjernen, er klassifikator-funksjonen gitt ved:

$$f(\mathbf{x}) = \text{sign}\left( \sum_{SV} a_i y_i K(\mathbf{x_i}, \mathbf{x}) + b \right)$$

$$\sum_{SV} a_i y_i K(\mathbf{x_i}, \mathbf{x}) = \mathbf{w} \cdot \mathbf{x} \tag{3.28}$$

$$b = -\frac{1}{2}\sum_{SV} a_i y_i \left[ K(\mathbf{x_i}, \mathbf{x_r}) + K(\mathbf{x_i}, \mathbf{x_s}) \right]$$

SV betyr at det er kun støttevektorene som inngår i summasjonen.

Som det går fram av ligningene er det ikke nødvendig å kjenne F(x); det er nok å kjenne kjernen. Dette er mye av poenget. F(x) kan i prinsippet transformere x til et rom med uendelig dimensjon uten at det kompliserer problemet.

Kjernen kan velges utifra visse kriterier som jeg kommer tilbake til etter et eksempel.

Anta at datavektorene er i $R^2$, og at det benyttes en kjerne gitt av:

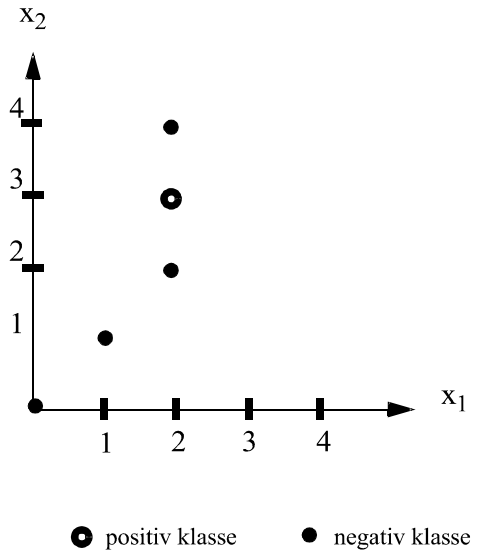
$$K(\mathbf{x_i}, \mathbf{x_j}) = \left(\mathbf{x_i} \cdot \mathbf{x_j}\right)^2. \tag{3.29}$$

Denne kjernen kan har uendelig mange transformasjoner $\Phi(\boldsymbol{x})$. Én av transformasjonene er gitt ved:

$$\Phi(\mathbf{x}) = \begin{bmatrix} x_1^{\,2} \\ \sqrt{2}x_1 x_2 \\ x_2^{\,2} \end{bmatrix} \tag{3.30}$$

som transformerer inngangsvektorer i $R^2$ over i $R^3$.

Figur 3.6 og tabell 3.2 angir datapunktene i eksemplet.




*Figur 3.6* *Grafisk fremstilling av egenskapsvektorer av forskjellig klasse som ikke kan separeres av et optimalt hyperplan i det todimesjonale rommet som spennes ut av inngangsvektorene.*

| | $\boldsymbol{x_1}$ | $\boldsymbol{x_2}$ | $\boldsymbol{x_3}$ | $\boldsymbol{x_4}$ | $\boldsymbol{x_5}$ |
|---|---|---|---|---|---|
| $x_1$ | 0 | 1 | 2 | 2 | 2 |
| $x_2$ | 0 | 1 | 2 | 3 | 4 |
| $\boldsymbol{y}$ | -1 | -1 | -1 | 1 | -1 |

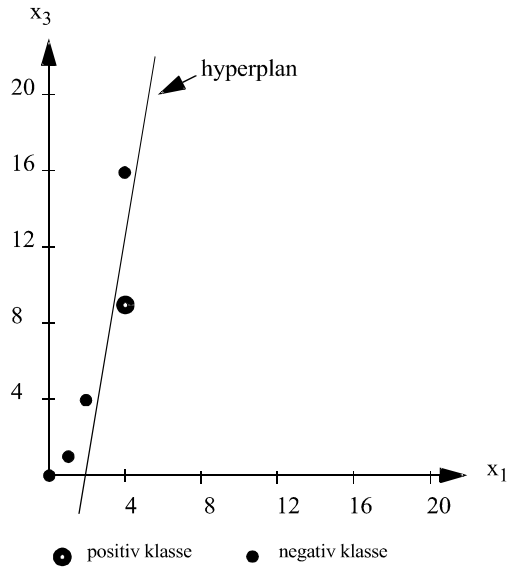
*Tabell 3.2* *Tabellen formaliserer informasjonen på figur 3.6*

Som det går fram av figuren over er det ikke mulig å separere de to klassene med ett enkelt hyperplan (linje i to dimensjoner). Benytter $\Phi(\boldsymbol{x})$ og mapper datavektorene over i tre dimensjoner.

| | $x_1$ i 2D | $x_2$ i 2D | $x_1$ i 3D | $x_2$ i 3D | $x_2$ i 3D |
|---|---|---|---|---|---|
| $\boldsymbol{x_1}$ | 0 | 0 | 0 | 0 | 0 |
| $\boldsymbol{x_2}$ | 1 | 1 | 1 | 1,4 | 1 |
| $\boldsymbol{x_3}$ | 2 | 2 | 2 | 5,7 | 4 |
| $\boldsymbol{x_4}$ | 2 | 3 | 4 | 8,5 | 9 |
| $\boldsymbol{x_5}$ | 2 | 4 | 4 | 11,3 | 16 |

*Tabell 3.3* *Tabellen viser hvordan de forskjellige vektorene xi blir seende ut når de transformeres fra to til tre dimensjoner.*

Figur 3.7 viser hvordan punktsamlingen transformert til det tredimensjonale rommet ser ut sett fra "$x_1x_3$"-perspektiv.




*Figur 3.7* *Grafisk fremstilling av egenskapsvektorer av forskjellig klasse som ikke kan separeres av et optimalt hyperplan i det todimesjonale rommet som spennes ut av inngangsvektorene, men som ved en transformasjon til et tredimensjonalt rom kan separeres med et hyperplan. Figuren viser et hyperplan som separerer klassene sett fra "$x_1$ $x_3$"-perspektiv.*

Som det går fram av figuren er det nå mulig å utføre en lineær separasjon av de to klassene.

De betraktninger som ble gitt angående parameteren *C* for lineære SVM, gjelder også for ulineære SVM.

### 3.2.2 Typiske kjernefunksjoner og Mercers krav

Rommet som inngangsvektorene transformeres til, kalles ofte for et *Hilbert-rom* og jeg vil benytte symbolet *H* til å referere til dette. Kravene til dette rommet er at det er et lineært rom der indreproduktet og den tilhørende normen er definert, slik at hvis en kjerne skal være anvendbar, må det eksistere et par $\{\Phi(\boldsymbol{x}),H\}$ som tilfredstiller kravene over. Mercers krav benyttes til å kontrollere dette. Mercers krav lyder som følger:

Det finnes en transformasjon $\Phi$ og en kjerne

$$K(\mathbf{x},\mathbf{y}) = \sum_{\mathbf{i}} \Phi(\mathbf{y})_{\mathbf{i}} \Phi(\mathbf{x})_{\mathbf{i}}$$

hvis og bare hvis, det for en g(**x**) slik at

$$\int g(\mathbf{x})^2 d\mathbf{x}$$

er endelig også tilfredsstiller

$$\int K(\mathbf{y},\mathbf{x}) g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0.$$

Mercers krav gir en garanti for at en kjerne er anvendbar men forteller ikke hvordan $F$ kan konstrueres eller hva $H$ er. Imidlertid fins det en rekke klasser av kjernefunksjoner som det er ført bevis for tilfredstiller Mercers krav.

Følgende klasser av kjernefunksjoner har vært vanlig å benytte i forskningsaktiviteten knyttet til SVM:

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y}+1)^p \tag{3.31}$$

$$K(\mathbf{x},\mathbf{y}) = \tanh(k(\mathbf{x}\cdot\mathbf{y})-d) \tag{3.32}$$

$$K(\mathbf{x},\mathbf{y}) = e^{(-\|\mathbf{x}-\mathbf{y}\|^2)/(2s^2)} \tag{3.33}$$

Ligning (3.31) gir klassifikator som er polynomisk av grad $p$. Dimensjonen til $H$ er avhengig av dimensjonen til rommet inngangsvektorene eksisterer i, kalt $dL$, og den polynomiske graden $p$. For den polynomiske kjernen:

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y})^p \tag{3.34}$$

er dimensjonen til $H$ gitt ved:

$$\dim(H) = \binom{d_L+p-1}{p} = \frac{(d_L+p-1)!}{p!(d_L-1)!} \tag{3.35}$$

Dette medfører at dim($H$) kan gjøres mye større enn $dL$. Det gjelder også for klassen av polynomiske kjerner gitt i (3.31). Ligning (3.32) gir en klassifikator som er ekvivalent med en type nevralt nettverk med “S”-formet aktiveringsfunksjon og ett skjult lag. Antallet støttevektorer vil bestemme antallet vekter som må justeres i de to lagene i det ekvivalente nettverket. Antallet vekter i det første laget vil være lik antall støttevektorer multiplisert med dL, og i det andre laget, lik antall støttevektorer. Kompleksiteten til nettverket vil derfor tilpasses treningsdatasettet. Kjernen tilfredstiller bare Mercers krav for bestemte kombinasjoner av parameterverdiene $d$ og $k$.

Ligning (3.33) gir kjernen til en klassifikator som er ekvivalent med en gaussisk radialbasis-funksjon (RBF) klassifikator (4). En RBF-klassifikator er gitt ved:

$$D(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{l}(w_i e^{(-\|\mathbf{x}-\mathbf{x}_i\|^2)/c_i} + b\right) \tag{3.36}$$

Dersom RBF-kjernen benyttes, vil en SVM-klassifikator se slik ut:

$$F(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{l}(a_i y_i)e^{(-\|\mathbf{x}-\mathbf{x}_i\|^2)/(2s^2)} + b\right) \tag{3.37}$$

Ved sammenligning av de to siste ligningene går det fram at sentrene $x_i$, tilsvarer støttevektorer, vektene $w_i$ tilsvarer $\alpha_i y_i$ og $b$ spiller samme rolle. $c_i$ og $\sigma$ spiller samme rolle. $\sigma$ er en kjerneparameter.

For å finne parametrene til $D$ kan flere metoder benyttes. Metodene gir ulike parametre og generelt forskjellig sentre. SVM gir en konsistent måte å finne parametrene på, og støttevektorene (tilsvarende sentrene) velges automatisk til å passe klassifiseringsproblemet.

### 3.3 Strukturell risikominimalisering

I det foregående er det gitt en enkel beskrivelse av konseptet ved SVM. I det følgende blir det gitt en skisse av teorien som ligger til grunn.

Gitt et sett med treningsdata:

$$(y_1, \mathbf{x_1}), \cdots, (y_l, \mathbf{x}_l), \qquad \mathbf{x} \in R^n, \qquad y \in \{-1,1\}$$

og et sett funksjoner:

$$f_a : (a \in \Lambda)\}, \quad f_a : R^N \rightarrow \{-1,1\}$$

Målet er å finne den funksjonen $f_{a*}(\mathbf{x})$ som er den beste transformasjonen fra **x** til $y$. En minimalisering av funksjonen:

$$R(a) = \int \left| f_a(\mathbf{x}) - y \right| dP(\mathbf{x}, y), \tag{3.38}$$

med hensyn på $f_\alpha$ gir $f_{\alpha*}$. $R(\alpha)$ som kalles den virkelige risikoen er ukjent fordi sannsynlighetsfordelingen $P(\boldsymbol{x},y)$ er ukjent. Imidlertid kan den empiriske risikoen beregnes som følger:

$$R_{emp}(a) = \frac{1}{l} \sum_{i=1}^{l} \left| f_a(\mathbf{x_i}) - y_i \right| \tag{3.39}$$

Det er rett fram å minimalisere denne og finne en optimal $f_\alpha$, men resultatet garanterer ikke at en liten $R_{\text{emp}}(\alpha)$ gir en liten $R(\alpha)$ når treningssettet er relativt lite. Det vil si at selv om treningsdatasettet klassifiseres uten feil, kan testsettet bli klassifisert med relativt mange feil.

Prinsippet ved *Strukturell risikominimalisering* gir et svar på dette problemet: For en $a$ inkludert i $L$ vil med en sannsynlighet på minst $1\text{-}h$, følgende skranke holde:

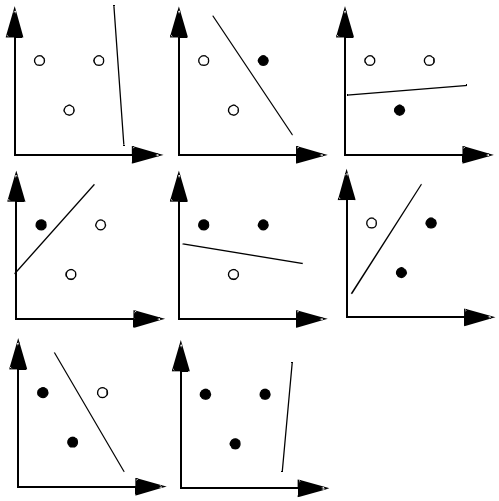
$$R(a) \leq R_{emp}(a) + \Phi\left(\frac{h}{l}, \frac{\log(h)}{l}\right), \tag{3.40}$$

der F kalles konfidens-intervall og er definert som:

$$\Phi\left(\frac{h}{l},\frac{\log(h)}{l}\right)=\sqrt{\frac{h\left(\log\left(\left(\frac{2l}{h}\right)+1\right)\right)-\log\left(\frac{h}{4}\right)}{l}} \tag{3.41}$$

*h* kalles VC-dimensjonen for settet av funksjoner. Dersom det er to klasser, vil VC-dimensjonen være lik det maksimale antallet punkter *k*, som kan separeres på $2^k$ måter med $f_a$. Figur 3.8 viser et tilfelle der det er tre punkter i datasettet og $f_a$ er et sett av linjer. Her er VC-dimensjonen lik tre fordi det er ikke mulig å inkludere ett punkt til og oppnå å separere de to klassene på $2^k$ måter.

Det maksimale antallet punkter som er lineært separabel på $2^k$ måter er gitt av dimensjonen til inngangsvektoren pluss én. Dette stemmer overens med figur 3.8 der dimensjonen på inngangsvektoren er to og tre punkter er lineært separabel på $2^3$ måter.



*Figur 3.8 Grafisk fremstilling av tre egenskapsvektorer som separeres på $2^k$ forskjellige måter.*

Beskrankningen på den virkelige risikoen $R(\alpha)$, kan kontrolleres ved å kontrollere $R_{emp}$ og *h*. $R_{emp}$ kontrolleres ved å velge den funksjonen i settet som minimaliserer *R*, altså $f_{\alpha*}$. *h* avhenger av det sett av funksjoner som kan implementeres. SVM algoritmen er basert på hyperplan.

I (3.4) ble distansen fra nærmeste punkt i datasettet til hyperplanet utledet:

$$\min d(\mathbf{w},b,\mathbf{x})=\frac{1}{\|\mathbf{w}\|}$$

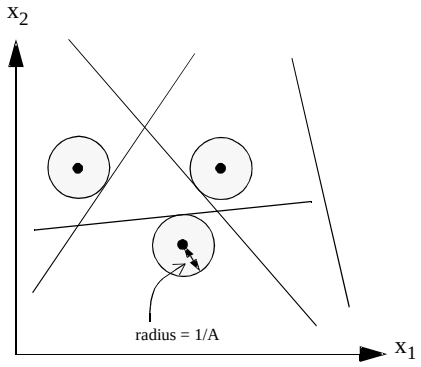
Dersom det antas at følgende kriterium holder:

$$\|\mathbf{w}\|\leq A \tag{3.42}$$

medfører dette:

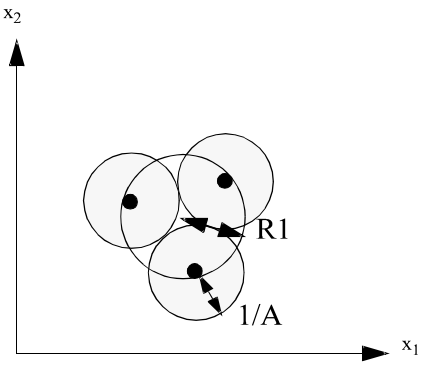
$$\min d(\mathbf{w}, b, \mathbf{x}) = \frac{1}{A} \tag{3.43}$$

Betydningen av dette er illustrert på neste figur 3.9.




*Figur 3.9* *Grafisk fremstilling av tre egenskapsvektorer i planet, som separeres av hyperplan. Figuren viser hvordan (3.43) begrenser hvor hyperplanene kan plasseres.*

Ingen av hyperplanene kan komme nærmere punktene enn 1/*A*. Hva har dette med VC-dimensjonen å gjøre? For å besvare spørsmålet innføres en "hyperkule" med radius R1 i N dimensjoner som omslutter alle punktene. I to dimensjoner med tre punkter kan dette se ut som på figur 3.10.




*Figur 3.10* *Grafisk fremstilling av tre egenskapsvektorer i planet. Figuren viser hvordan (3.43) og hyperkula med radius R1, begrenser hvor hyperplanene kan plasseres.*

For det første, som tidligere nevnt, kan ikke *h* være større enn dimensjonen til det aktuelle rommet pluss én ved lineær separasjon. For det andre er det åpenbart at det ikke er mulig å oppnå denne verdien for *h* på figur 3.10. Det er så trangt inne i "R1-hyperkula" at det ikke er mulig å separere "1/*A*-hyperkulene" med hyperplan. Den eneste måten å utføre $2^k$ forskjellige separasjoner, er å se bort fra ett av punktene. I dette tilfellet blir *h* lik to. Dersom *1/A* er noe mindre blir imidlertid *h* lik tre. Fra eksempelet går det altså frem at *h* er begrenset av antall "1/*A*-hyperkuler" som er mulig å separere på $2^k$ måter med sentrum inne i "R1-hyperkula". Det er vist at denne begrensningen er gitt ved:

$$h \leq R1^2 A^2. \tag{3.44}$$

Dette medfører at *h* er begrenset som følger:

$$h \leq \min(\mathrm{R1}^2 A^2, \dim(\mathbf{x}) + 1) \tag{3.45}$$

Siden ||***w***|| er mindre eller lik *A* er minimalisering av ||***w***|| ekvivalent med å minimalisere *h*. Utfra ligningen (3.41) for Φ, går det fram at for de aktuelle verdiene av *h* og *l* er Φ monotont minkende for minkende verdier av *h*. Derfor tilsvarer minimalisering av ||***w***||, å minimalisere Φ.

I (3.24) ble følgende kriteriefunksjonen og skranker presentert:

$$\begin{aligned} \min \quad & \Psi(\mathbf{w}) = \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i=1}^{l} x_i \\ & (\mathbf{x_i} \cdot \mathbf{w} + b) y_i \geq 1 - x_i \\ & x_i \geq 0. \end{aligned}$$

Leddet

$$C \sum_{i=1}^{l} x_i \,,$$

danner et øvre bånd på den empiriske risikoen. Parameteren *C* benyttes for å justere støttevektormaskinen til å takle støyen i treningsdata. Imidlertid så er det en sammenheng mellom Φ og den empiriske risikoen: Dersom *h* øker, kan den empiriske risikoen minke. Dette er lett å tenke seg dersom man tar utgangspunkt i et klassifiseringsproblem som ikke er lineært separabelt. Ved å transformere treningsdatasettet til et rom med høyere dimensjon er det sannsynlig at *h* øker og at datasettet, i større grad, blir lineært separabelt. Det vil si at den empiriske risikoen minker. Siden det er summen mellom Φ og $R_{\mathrm{emp}}$ som skal minimaliseres, betyr dette at det fins en avveining mellom liten *h* som gir liten Φ og stor *h* som gir liten $R_{\mathrm{emp}}$. Det er allerede nevnt at Φ er monotont minkende for minkende verdier av *h.* Utfra det foregående konkluderes det med at SVM implementerer prinsippet ved Strukturell risikominimalisering.

## 3.4 Eksempler som illustrerer forskjellige sider ved SVM

For å illustrere egenskaper ved SVM, er det hensiktsmessig å benytte kun to dimensjoner på attributt-vektoren. Kode skrevet i Matlab (se Appendiks D.3) benyttes for å generere resultatene som følger.

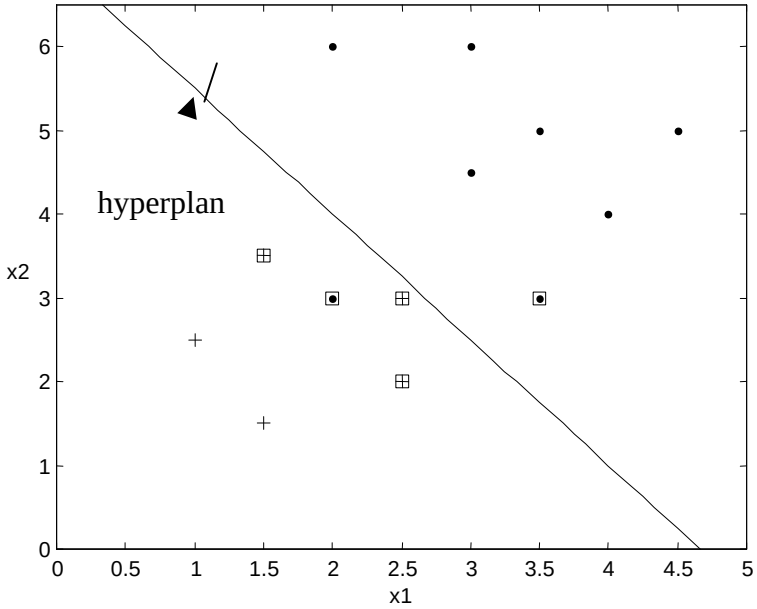
### 3.4.1 Variasjon av C for lineære SVM

I det neste eksempelet benyttes lineær separasjon i inngangsvektorrommet med forskjellige verdier for *C* som er vekten til summen av slakkvariablene i optimalkriteriet.

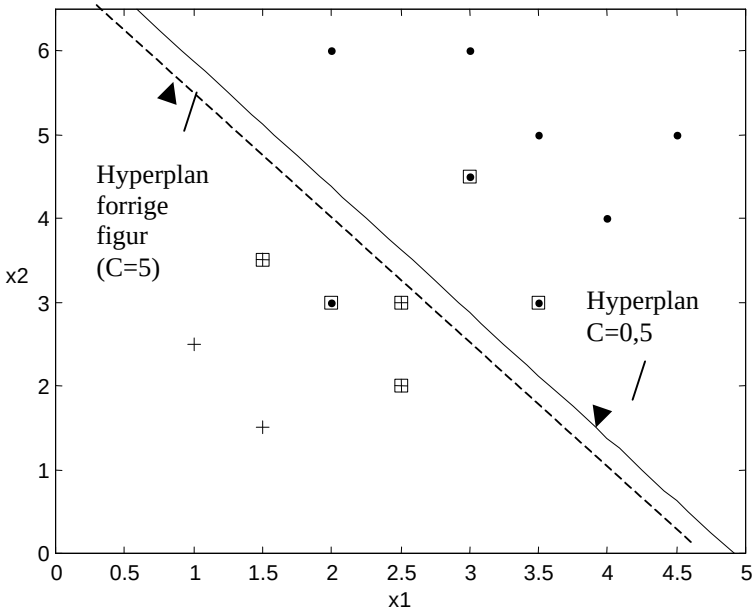
I de neste to grafene benyttes den samme punktkonfigurasjonen. Punktene i grafene er av to forskjellige klasser. Koordinatene for punktene og tilhørende verdier for α er gitt i tabell 3.4.

| klasse | 1 | 1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | 1,5 | 2,5 | 1,0 | 2,5 | 1,5 | 3,5 | 4,0 | 3,0 | 3,5 | 4,5 | 3,0 | 2,0 | 2,0 |
| $x_2$ | 1,5 | 2,0 | 2,5 | 3,0 | 3,5 | 3,0 | 4,0 | 4,5 | 5,0 | 5,0 | 6,0 | 6,0 | 3,0 |
| α,C=5 | 0 | 1,38 | 0 | 5 | 1,2 | 2,5 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |
| α,C=0,5 | 0 | 0,35 | 0 | 0,5 | 0,5 | 0,5 | 0 | 0,35 | 0 | 0 | 0 | 0 | 0,5 |

*Tabell 3.4* *Koordinater for egenskapsvektorer med tilhørende klassifisering og alfa-verdier for forskjellige verdier av parameteren C. Tabellen gjelder for lineær separasjon.*




*Figur 3.11* *Figuren viser lineær separasjon for C lik 5, av egenskapsvektorer med forskjellig klasse. Kryss markerer positiv klasse og prikk markerer negativ klasse. Firkant markerer at vektoren er en støttevektor.*




*Figur 3.12 Figuren viser lineær separasjon for C lik 0.5, av egenskapsvektorer med forskjellig klasse. Kryss markerer positiv klasse og prikk markerer negativ klasse. Firkant markerer at vektoren er en støttevektor.*

Det fremgår av figur 3.11 og 3.12 at klassifikatoren blir mindre følsom for støypunktet i (2.0,3.0) når *C* reduseres fra 5 til 0,5. Dette er intuitivt fordi elementer som ligger på "gal" side av marginen teller mindre når vekten *C*, reduseres.

### 3.4.2 Ulineær separasjon av lineært separabelt tilfelle

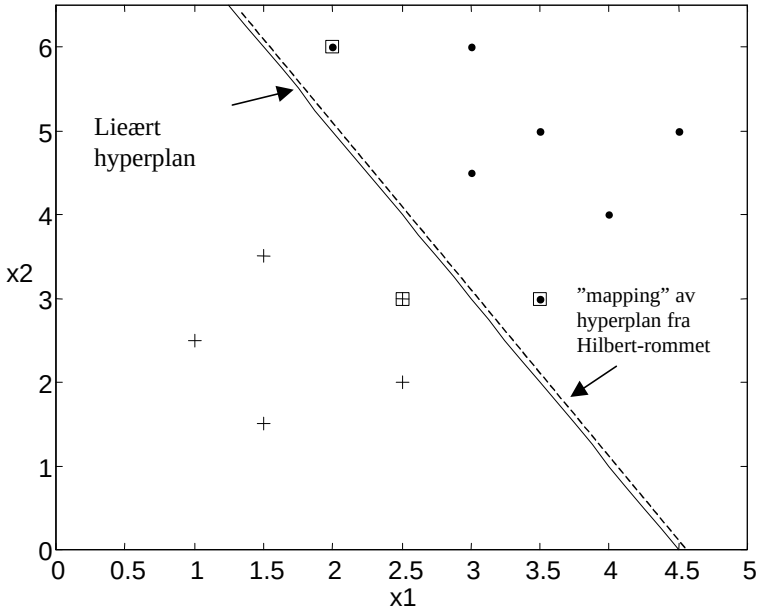
Når ulineære kjerner benyttes, vil ikke hyperplanet tilsvare et plan når det transformeres ned i vektorrommet til inngangsvektorene. Det kan likevel være interssant å se om lineær separasjon og ulineær separasjon alltid gir svært ulikt resultat for det lineært separable tilfellet. Det benyttes en ulineær polynomisk kjerne av 2. grad for den ulineære separasjonen. Kjernen er gitt ved:

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y}+1)^{grad}, \quad \text{grad} = 2. \tag{3.46}$$

Koordinatene for punktene og tilhørende verdier for $\alpha$ er gitt i tabell 3.5.

| klasse | 1 | 1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | 1,5 | 2,5 | 1,0 | 2,5 | 1,5 | 3,5 | 4,0 | 3,0 | 3,5 | 4,5 | 3,0 | 2,0 |
| $x_2$ | 1,5 | 2,0 | 2,5 | 3,0 | 3,5 | 3,0 | 4,0 | 4,5 | 5,0 | 5,0 | 6,0 | 6,0 |
| $\alpha$,C=5 lineær | 0 | 0 | 0 | 2,5 | 0 | 2,2 | 0 | 0 | 0 | 0 | 0 | 0,33 |
| $\alpha$,C=0,5 ulineær | 0 | 0 | 0 | 3,7E-2 | 0 | 3,5E-2 | 0 | 0 | 0 | 0 | 0 | 2,0E-3 |

*Tabell 3.5 Koordinater for egenskapsvektorer med tilhørende klassifisering og alfa-verdier for C lik 5. Tabellen gjelder for lineær separasjon og ulineær separasjon.*




*Figur 3.13* *Figuren viser lineær og ulineær separasjon for C lik 0.5, av egenskapsvektorer med forskjellig klasse. Kryss markerer positiv klasse og prikk markerer negativ klasse. Firkant markerer at vektoren er en støttevektor.*

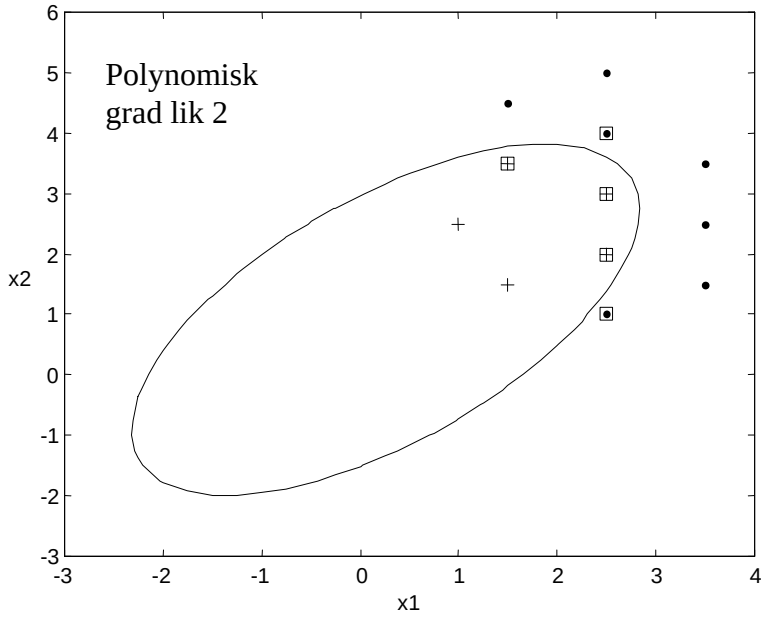
Det fremgår av figur 3.13 at ulineær og lineær separasjon for dette tilfellet, gir omtrent samme resultat i det avbildede området.

### 3.4.3 Ulineær separasjon av lineært ikke-separabelt tilfelle

Når ulineære polynomiske kjerner benyttes, kan det være interessant å se hva som skjer når graden økes. Det benyttes polynomiske kjerner av 2. og 6. grad og data er gitt i tabell 3.6.

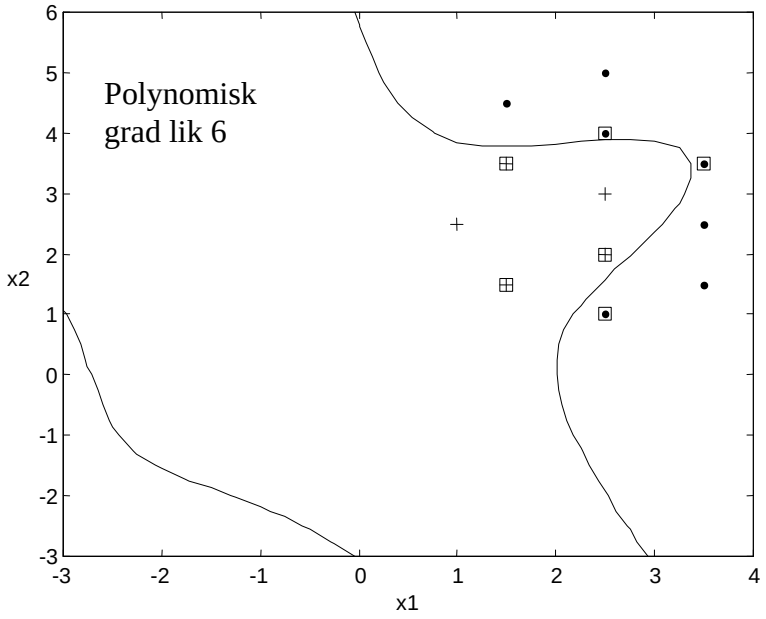
| klasse | 1 | 1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | 1,5 | 2,5 | 1,0 | 2,5 | 1,5 | 2,5 | 3,5 | 3,5 | 3,5 | 2,5 | 2,5 | 1,5 |
| $x_2$ | 1,5 | 2,0 | 2,5 | 3,0 | 3,5 | 1,0 | 1,5 | 2,5 | 3,5 | 4,0 | 5,0 | 4,5 |
| α,C=5 grad=2 | 0 | 0,54 | 0 | 1,6 | 0,32 | 1,2 | 0 | 0 | 0 | 1,3 | 0 | 0 |
| α,C=5 grad=6 | 1,0E5 | 2,0E-5 | 0 | 0 | 3,0E-6 | 3,1E-5 | 0 | 0 | 0,0 | 1,0E-2 | 0 | 0 |

*Tabell 3.6* *Koordinater for egenskapsvektorer med tilhørende klassifisering og alfa-verdier for C lik 5. Tabellen gjelder for ulineær separasjon med 2. og 6. grads polynomisk kjerne.*




*Figur 3.14* *Figuren viser ulineær separasjon for C lik 5, av egenskapsvektorer med forskjellig klasse. Kryss markerer positiv klasse og prikk markerer negativ klasse. Firkant markerer at vektoren er en støttevektor. Den polynomiske graden til kjernen er lik 2.*

Som der fremgår av figur 3.14 og 3.15, er det mulig å separere treningsdata som ikke er lineært separabel ved å benytte ulineær separasjon. Som det også fremgår, retter hyperplanet seg ut umiddelbart etter at det "forlater" punktmengden for figur 3.15. En linje separerer klassene best i dette området. Imidlertid ser det ut til at dette har sin pris, ettersom separasjonen i området der punktmengden befinner seg, ikke ser ut til å bli bedre når den polynomiske graden økes.




*Figur 3.15* *Figuren viser ulineær separasjon for C lik 5, av egenskapsvektorer med forskjellig klasse. Kryss markerer positiv klasse og prikk markerer negativ klasse. Firkant markerer at vektoren er en støttevektor. Den polynomiske graden til kjernen er lik 6.*

### 3.4.4 Ulineær separasjon med kjerne av for lav kapasitet

Det er ikke alltid det er mulig å utføre en separasjon fordi settet av mulige funksjoner har for liten kapasitet. Følgende polynomiske kjerne kan bare implementere grenseflater i et to-dimensjonalt inngangsvektorrom som har kjeglesnitt-form:

$$K(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j})^2 \tag{3.47}$$

En mulig transformasjon $\Phi$, er gitt ved ligning (3.30):

$$\Phi(\mathbf{x}) = \begin{bmatrix} x_1^{\,2} \\ \sqrt{2}x_1x_2 \\ x_2^{\,2} \end{bmatrix}. \tag{3.48}$$

Dersom denne settes inn for kjernen i klassifikator-funksjonen $f(\boldsymbol{x})$ (3.28), vil dette gi et polynom som har grad lik to. Derfor er det kun mulig å implementere kjeglesnitt med denne kjernen. Det settet av klassifikator-funksjoner som er mulig å implementere med denne kjernen, har ikke stor nok kapasitet til å separere datasettet gitt i tabell 3.7.

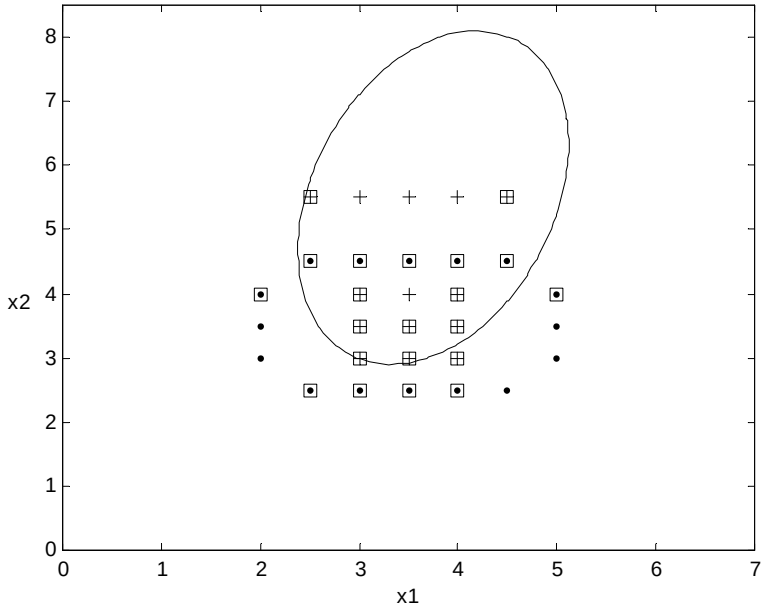
| klasse | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | 2,5 | 3,0 | 3,5 | 4,0 | 4,5 | 3,0 | 3,5 | 4,0 | 3,0 | 3,5 | 4,0 | 3,0 |
| $x_2$ | 5,5 | 5,5 | 5,5 | 5,5 | 5,5 | 4,0 | 4,0 | 4,0 | 3,5 | 3,5 | 3,5 | 3,0 |
| α,C=5 grad=6 | 0 | 0 | 0 | 2,0E-4 | 0 | 17E-4 | 0 | 2,0E-4 | 0 | 0 | 0 | 0 |
| α,C=5 grad=2 | 5 | 0 | 0 | 0 | 2,7 | 5 | 0,19 | 5 | 5 | 5 | 5 | 5 |

| klasse | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | 3,5 | 4,0 | 2,5 | 3,0 | 3,5 | 4,0 | 4,5 | 2,0 | 2,0 | 2,0 | 5,0 | 5,0 |
| $x_2$ | 3,0 | 3,0 | 4,5 | 4,5 | 4,5 | 4,5 | 4,5 | 4,0 | 3,5 | 3,0 | 4,0 | 3,5 |
| α,C=5 grad=6 | 0 | 2,0E-4 | 0 | 3,0E-4 | 12E-4 | 0 | 0 | 0 | 1,0E-4 | 0 | 1,0E-4 | 0 |
| α,C=5 grad=2 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 2,9 | 0 | 0 | 3,8 | 0 |

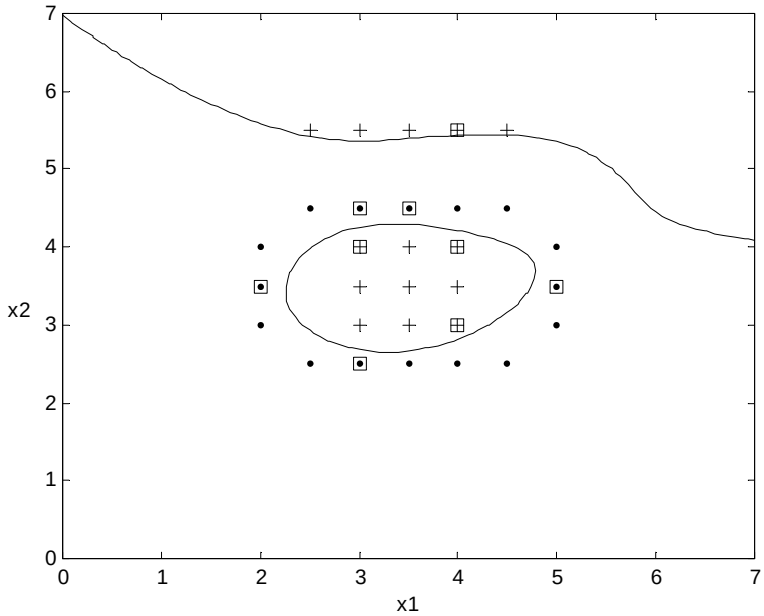
| klasse | -1 | -1 | -1 | -1 | -1 | -1 |
|---|---|---|---|---|---|---|
| $x_1$ | 5,0 | 2,5 | 3,0 | 3,5 | 4,0 | 4,5 |
| $x_2$ | 3,0 | 2,5 | 2,5 | 2,5 | 2,5 | 2,5 |
| α,C=5 grad=6 | 0 | 0 | 7,0E-4 | 0 | 0 | 0 |
| α,C=5 grad=2 | 0 | 4,5 | 5 | 5 | 1,7 | 0 |

*Tabell 3.7 Koordinater for egenskapsvektorer med tilhørende klassifisering og alfa-verdier for C lik 5. Tabellen gjelder for ulineær separasjon med 2. og 6. grads polynomisk kjerne.*

Det går fram av figur 3.16 og 3.17 at det ikke fins et kjeglesnitt som kan separere de to punktmengdene, men en kjerne av polynomisk 6. grad klarer det fint. Årsaken til at den separerende flaten synes å mangle symmetri, kan skyldes for dårlig numerisk representasjon.




*Figur 3.16* *Figuren viser ulineær separasjon for C lik 5, av egenskapsvektorer med forskjellig klasse. Kryss markerer positiv klasse og prikk markerer negativ klasse. Firkant markerer at vektoren er en støttevektor. Den polynomiske graden til kjernen er lik 2.*




*Figur 3.17* *Figuren viser ulineær separasjon for C lik 5, av egenskapsvektorer med forskjellig klasse. Kryss markerer positiv klasse og prikk markerer negativ klasse. Firkant markerer at vektoren er en støttevektor. Den polynomiske graden til kjernen er lik 6.*

# 4 C4.5-ALGORITMEN

## 4.1 Introduksjon

C4.5 er en maskinlæringsalgoritme som hører til under klassen beslutningstrær. Algoritmen er forfattet av J. Ross Quinlan og er en videreføring av ID3 av samme forfatter.

Beslutningstreet genereres ved hjelp av et sett treningsdata som består av inngangsvektorer med tilhørende klassifisering. Treet består av noder som representerer egenskapene som er nedfelt i treningsdatasettet. For å konfigurere treet benyttes teori fra informasjonsvitenskapen. Etter at treet er generert, forenkles det for å bli kvitt grener som ikke bidrar positivt. Til dette benyttes en heuristisk algoritme. Treet gir statistisk informasjon. Det er derfor mulig å vurdere hvor godt beslutningsgrunnlaget er. Det genererte treet kan benyttes direkte til klassifisering av inngangsvektorer av ukjent klasse. C4.5 genererer også produksjonsregler i form av "if-then-else"-setninger og foretar forenklinger av disse. Dette er hensiktsmessig i den grad det kan bidra til at et menneske får en dypere forståelse av hva som skjuler seg i datamaterialet. Stoffet som danner grunnlaget for teksten som følger i dette kapittelet er funnet i (2).

## 4.2 Strukturen til treningsdatasettet

Konstruksjon av et beslutningstre er basert på ett sett av inngangsdata. Dette settet består av en samling inngangsvektorer med tilhørende klassifisering. Inngangsvektorene er bygd opp av attributter og hver enkel attributt har sin egen mengde av lovlige verdier (kategorier). Klassifiseringsvariabelen til en vektor har også sinmengde av lovlige verdier. For å illustrere dette gis følgende "egenproduserte" eksempel som undertegnede har valgt å kalle *Dørvaktens dilemma*.

| gruppe-tilhørighet | alder | alkohol-påvirkning | antall gruppe-medlemmer fremmøtt | klær | dag | velkommen |
|---|---|---|---|---|---|---|
| B-gjengen | <16 | ikke synlig | <2 | Dongeri | Mandag-Torsdag | Ja |
| Pusur-klubben | 16<=x<18 | lett synlig | 2<=x<=4 | Bikini | Fredag-Lørdag | Nei |
| Kvinnegruppa OTTAR | >=18 | svært synlig | >4 | Dress | Søndag | Senere |
| Modellklubben Emanuelle | | | | Kjole | | |
| Andre | | | | Annet | | |

*Tabell 4.1 Tabellen viser datastrukturen til eksempelet*

Dørvakter er som kjent nødt til å avgjøre hvilke gjester som er velkommen og kan slippes inn og hvem som ikke er velkommen. I tabell 4.1 øverste rad på venstre side av dobbeltlinjen er attributtene listet opp, og i kolonnen under hver attributt er de tilhørende lovlige verdiene listet opp. Klassevariabelen er øverst i høyre hjørne i tabellen og i kolonnen under er de tilhørende lovlige verdiene listet opp.

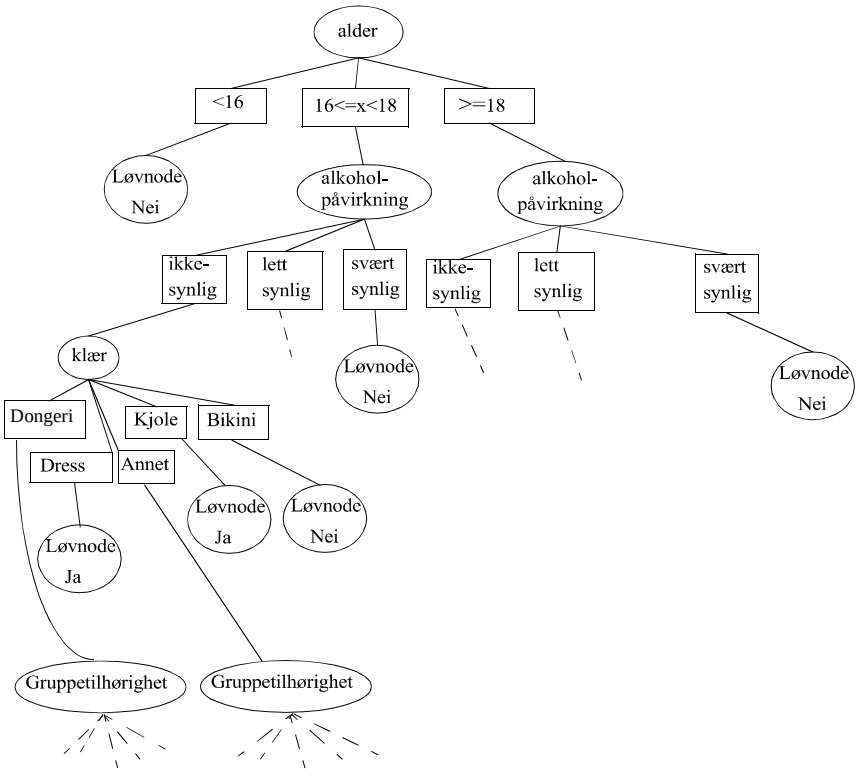
Tre mulige inngangsvektorer (egenskapsvektorer) med tilhørende klassifisering er gitt i tabell 4.2 for å illustrere betydningen av datastrukturen i tabell 4.1.

| gruppe-tilhørighet | alder | alkohol-påvirkning | antall gruppe-medlemmer fremmøtt | klær | dag | velkommen |
|---|---|---|---|---|---|---|
| B-gjengen | 16<=x< 18 | lett synlig | <2 | Dongeri | Fredag-Lørdag | Senere |
| Pusur-klubben | >=18 | svært synlig | >4 | Bikini | Søndag | Nei |
| Modellklubben Emanuelle | <16 | lett synlig | <2 | Annet | Fredag-Lørdag | Nei |

*Tabell 4.2 Tabellen viser tre mulige inngangsvektorer med tilhørende klassifisering.*

## 4.3 Konstruksjon av beslutningstrær

Et beslutningstre konstrueres i C4.5 ved å evaluere informasjonsverdien for hver attributt og splitte treet på den mest informative attributten over attributtens lovlige verdier. For å illustrere dette videreføres eksempelet Dørvaktens dilemma. Figur 4.1 viser en del av et mulig beslutningstre.




*Figur 4.1 Figuren viser en del av et mulig beslutningstre for eksempelet Dørvaktens dilemma.*

Gitt en inngangsvektor uten klassifisering:

| gruppe-tilhørighet | alder | alkohol-påvirkning | antall gruppe-medlemmer fremmøtt | klær | dag |
|---|---|---|---|---|---|
| Modellklubben Emanuelle | <16 | lett synlig | <2 | Annet | Fredag-Lørdag |

*Tabell 4.3 Tabellen viser en inngangsvektor uten klassifisering.*

Dersom inngangsvektoren skal klassifiseres, undersøkes først roten av treet. Roten er i dette tilfellet attributten *alder*. Siden alder er lik "<16" for "datasampelet" undersøkes noden under testen for "<16". Dette er en løvnode og klassen er lik "Nei". Konklusjon: Modellen slipper ikke inn.

### 4.3.1 Gain-kriteriet

For å evaluere informasjonsverdien til en attributt kan *gain-kriteriet* benyttes. Dette er basert på følgende lov innen informasjonsteorien: Informasjonsverdien til en melding avhenger av meldingens forekomstsannsynlighet og kan måles i bit gitt av den negative toer-logaritmen til forekomstsannsynligheten.

F.eks vil meldingene A,B,C og D med forekomstsannsynlighetene 1/8, 1/8, 2/8 og 4/8, ha informasjonsverdiene 3,3,2 og 1 bit. Dersom melding A har sannsynlighet lik 1, vil informasjonsverdien være lik null. Dette er håndgripelig, fordi dersom det er kjent at den samme meldingen forekommer hele tiden, har denne meldingen ingen informasjonsverdi. Overført til treningsdatasamlingen, vil attributtene og klassen til "samplene" representere meldinger. $S$ representerer treningsdatasamlingen og *info*($S$) representerer informasjonverdien dersom kun klassen tas i betraktning. Sannsynligheten til en klasse $C_j$ er gitt ved:

$\frac{\text{freq}(C_j, S)}{|S|}$, der freq($C_j$,$S$) er hyppigheten av $C_j$ i $S$ og $|S|$ er antall elementer i $S$.

Av det foregående følger det at informasjonsverdien er gitt ved:

$$\textit{informasjonsverdi} = -\log_2\left(\frac{\text{freq}(C_j, S)}{|S|}\right)\textit{bit}. \tag{4.1}$$

For å finne den gjennomsnittlige informasjonen for alle klassene, summeres produktene av klassenes informasjonsverdier og tilhørende forekomstsannsynligheter. *info*($S$) er da gitt ved:

$$\text{info}(S) = -\sum_{j=1}^{k}\left(\frac{\text{freq}(C_j, S)}{|S|}\right)\log_2\left(\frac{\text{freq}(C_j, S)}{|S|}\right) \tag{4.2}$$

Når $S$ splittes på en attributt, blir $S$ delt opp i nye datasamlinger $S_i$ der elementene har felles attributtverdi. Ved å splitte $S$ på en attributt kan informasjonsverdien reduseres eller forbli den

samme. Dersom den reduseres til null, betyr det at dataelementene som hører til under hver attributtverdi, har samme klasse. Dette er optimalt. Dersom dataelementene som hører til under hver attributtverdi, har klasseverdier som er fordelt på samme måte som i *S*, vil informasjonsverdien bli uendret. Ved å splitte *S* på denne attributten foregår det altså ingen utkrystallisering av de forskjellige klassene. Informasjonsverdien ved å splitte *S* på attributt *X* kalles $\text{info}_X(S)$. Informasjonsgevinsten ved splittingen kalles *gain* og er gitt ved:

$$\text{gain}(X) = \text{info}(S) - \text{info}_X(S) \tag{4.3}$$

$\text{info}_X$ er definert som den gjennomsnittlige informasjonsverdien til $S_i$ og er gitt ved:

$$\text{info}_X(S) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} \text{info}(S_i) \tag{4.4}$$

Den attributten som har størst *gain* velges. Treet splittes kun én gang på hver attributt i hver gren av treet.

Med utgangspunkt i Dørvaktens dilemma skal *gain* for attributten alder beregnes. Det fins 100 treningstilfeller med nødvendige data gitt av tabell 4.4.

| alder | $\|S_i\|$ | antall av klassen Ja | antall av klassen Nei | antall av klassen Senere |
|---|---|---|---|---|
| 16<=x<18 | 30 | 10 | 20 | 0 |
| >=18 | 60 | 50 | 5 | 5 |
| <16 | 10 | 0 | 10 | 0 |

*Tabell 4.4 Tabellen viser frekvensdata for attributten alder i eksempelet.*

Først beregnes *info*(*S*). Fra tabellen går det fram at det totale antallet tilfeller av klasse lik "Ja" er 60, tilfeller av klasse lik "Nei" er 35 og tilfeller av klasse lik "Senere" er 5. *info*(*S*) er da gitt ved:

$$\text{info(S)} = -\left(\frac{60}{100}\right)\log_2\left(\frac{60}{100}\right) - \left(\frac{35}{100}\right)\log_2\left(\frac{35}{100}\right) - \left(\frac{5}{100}\right)\log_2\left(\frac{5}{100}\right) = 1{,}19 bit$$

$S_1$ representerer treningstilfellene som har attributtverdien "16<=x<18", $S_2$ representerer treningstilfellene som har attributtverdien ">=18" og $S_3$ representerer treningstilfellene som har attributtverdien "<16". Dette gir:

$$\text{info}(S_1) = -\left(\frac{10}{30}\right)\log_2\left(\frac{10}{30}\right) - \left(\frac{20}{30}\right)\log_2\left(\frac{20}{30}\right) = 0{,}92 bit$$

$$\text{info}(S_2) = -\left(\frac{50}{60}\right)\log_2\left(\frac{50}{60}\right) - \left(\frac{5}{60}\right)\log_2\left(\frac{5}{60}\right) - \left(\frac{5}{60}\right)\log_2\left(\frac{5}{60}\right) = 0{,}82 bit$$

$$\text{info}(S_3) = -\left(\frac{10}{10}\right)\log_2\left(\frac{10}{10}\right) = 0bit$$

$$\text{info}_X(S) = \frac{30}{100}\times 0{,}92 + \frac{60}{100}\times 0{,}82 + \frac{10}{100}\times 0 = 0{,}77bit$$

$$\text{gain}(X) = 1{,}19 - 0{,}77 = 0{,}42$$

### 4.3.2 Kriteriet gain ratio

*Gain*-kriteriet har én svakhet: Kriteriet har sterk preferanse for attributter med mange attributtverdier. Dersom f.eks attributtverdiene til attributten *klær*, i eksempelet Dørvaktens dilemma, hadde inkludert alle de forskjellige påkledningene og klær var den aktuelle splittattributten, ville $|S_i|$ (antall personer med samme bekledning *i*) muligens blitt lik én og $info_X$ blitt lik null. Dette betyr at *gain* blir maksimal. Fornuftsmessig er det ikke lurt å inkludere alle de forskjellige påkledningene, men det kan likevel være tilfeller der det er naturlig at det er store forskjeller i antall attributtverdier under hver attributt.

Istedet for *gain*-kriteriet benyttes *gain ratio*-kriteriet i C4.5. *Gain ratio*-kriteriet normaliserer *gain* med informasjonsverdien i å splitte opp *S*. Denne informasjonsverdien kalles *split info*(*X*) og er gitt ved:

$$\text{split info}(X) = -\sum_{j=1}^{n}\left(\frac{|S_i|}{|S|}\right)\log_2\left(\frac{|S_i|}{|S|}\right) \qquad (4.5)$$

*Gain* ratio er da gitt ved:

$$\text{gain ratio}(X) = \text{gain}(X) / \text{split info}(X) \qquad (4.6)$$

I videreføringen av eksempelet Dørvaktens dilemma, tilsvarer dette:

$$\text{split info}(X) = -\left(\frac{60}{100}\right)\log_2\left(\frac{60}{100}\right) - \left(\frac{30}{100}\right)\log_2\left(\frac{30}{100}\right) - \left(\frac{10}{100}\right)\log_2\left(\frac{10}{100}\right) = 1{,}30bit$$

$$\text{gain ratio}(X) = 0{,}42 / 1{,}30 = 0{,}32$$

Kriteriet kan imidlertid være ustabilt og misvisende for små verdier av *split info* (*X*). Dette løses ved å kun velge blant de attributtene som har over gjennomsnittlig *gain*.

### 4.3.3 Ukjente attributtverdier

Dersom noe av informasjonen i inngangsvektoren er ukjent, er det to muligheter i opptreningsfasen: Enten å forkaste vektoren eller å forsøke å utnytte den informasjonen vektoren inneholder. Dersom treningsdatasettet er relativt lite i forhold til antall attributtverdier og antall klasseverdier eller det er mange attributtverdier som mangler totalt, er det mye å vinne på å utnytte informasjonen som ligger i glisne inngangsvektorer. Dessuten, hvis en mengde av

inngangsvektorene som skal klassifiseres mangler enkelte attributtverdier, er det viktig at algoritmen takler dette på en god måte.

Hvordan takler så C4.5 glisne inngangsvektorer?

Videre benyttes eksempelet "Dørvaktens dilemma" for å illustrere dette. Anta at i 10 av 100 tilfeller er ikke dørvakten i stand til å aldersbestemme gjesten i henhold til de gitte kategoriene, pga. mistanke om at vedkommende har falsk ID eller lyver om alder.

Aktuelle data for de komplette tilfellene er gitt i tabell 4.5.

| alder | $\|S_i\|$ | antall av klassen Ja | antall av klassen Nei | antall av klassen Senere |
|---|---|---|---|---|
| 16<=x<18 | 25 | 10 | 15 | 0 |
| >=18 | 60 | 50 | 5 | 5 |
| <16 | 5 | 0 | 5 | 0 |

*Tabell 4.5 Tabellen viser frekvensdata for attributten alder i eksempelet. Kun de 90 dataelementene der alder er kjent er oppgitt.*

På bakgrunn av de 90 tilfellene beregnes info(*S'*) og $\text{info}_X(S')$ på samme måte som i det foregående. *S'* representerer samlingen av de komplette tilfellene. Dette gir:

info(*S'*) = 1,4

$\text{info}_X(S') = 0{,}81$

*Gain* beregnes ved å summere to produkter. Det første produktet er prosentandelen for komplette tilfeller mht. attributten alder, multiplisert med *gain* for komplette tilfeller. Det andre produktet er prosentandelen for inkomplette tilfeller multiplisert med *gain* for inkomplette tilfeller. Siden *gain* for inkomplette tilfeller ikke har informasjonsverdi for denne attributten, må verdien være lik null, slik at *gain* er gitt ved:

$$\text{gain} = (\textit{prosentandel komplette tilfeller mhp. aktuell attributt}) \times (\text{info}(S') - \text{info}_X(S')) \qquad (4.7)$$

I eksempelet er *gain* gitt ved:

$$\text{gain} = 90/100 \times (1{,}14 - 0{,}81) = 0{,}29 bit$$

Beregningsmetodikken for *split info* (*X*) må imidlertid også modifiseres siden samlingen av inkomplette tilfeller representerer en ny kategori (representert ved siste ledd i neste ligning) som kommer i tillegg til attributtens kategorier . Ligning (4.5) gir:

$$\text{split info}(X) = -\left(\frac{60}{100}\right)\log_2\left(\frac{60}{100}\right) - \left(\frac{25}{100}\right)\log_2\left(\frac{25}{100}\right) -$$

$$\left(\frac{5}{100}\right)\log_2\left(\frac{5}{100}\right) - \left(\frac{10}{100}\right)\log_2\left(\frac{10}{100}\right) = 1{,}49 bit$$

*gain ratio* er i henhold til ligning (4.6) gitt ved:

$$\text{gain ratio}(X) = 0{,}29/1{,}49 = 0{,}19$$

Når *S* splittes på en attributt blir *S* delt opp i nye datasamlinger $S_i$ der elementene har felles attributtverdi. De tilfellene som mangler den aktuelle attributten overføres til alle de nye forgreningene og gis en vekt som representerer sannsynligheten for riktig kategori. I utgangspunktet har alle treningstilfeller vekt lik 1. Tilfellene med komplett informasjon har vekt lik 1 hele tiden. For hver gang treningsdatasamlingen splittes, vil vekten i treningstilfeller der den aktuelle attributten fins, bli uforandret. Når et treningstilfelle mangler splittattributten, vil den nye vekten beregnes ved å multiplisere den gamle vekten med prosentandelen av kjente treningstilfeller som tilhører den aktuelle kategorien. Dette illustreres i den følgende utvidelsen av eksempelet Dørvaktens dilemma:

Det antas at dørvakten ikke er istand til å aldersbestemme 10 av de 100 gjestene pga. mistanke om falsk ID og/eller løgn, og at aktuell kjent data er gitt av tabell 4.6.

| alder | $\|S_i\|$ |
|---|---|
| 16<=x<18 | 25 |
| >=18 | 60 |
| <16 | 5 |

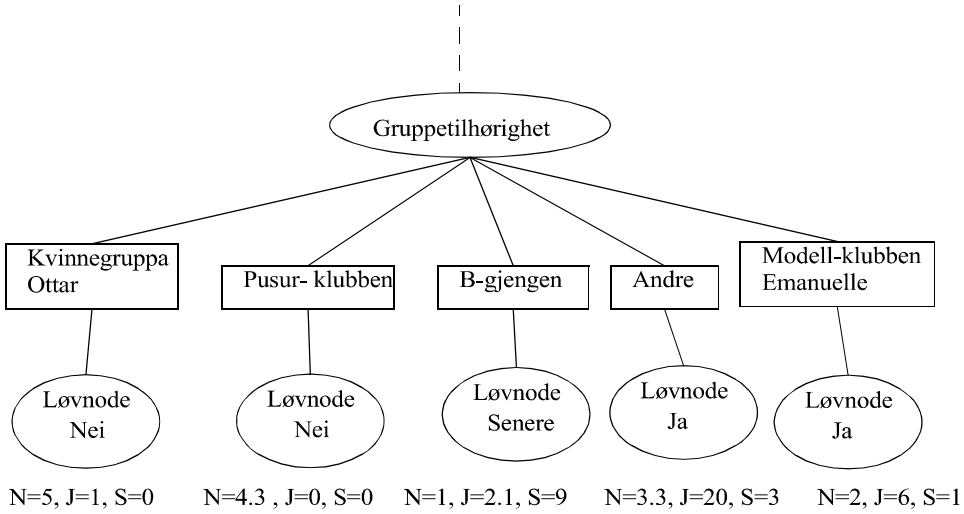
*Tabell 4.6 Tabellen viser utdrag av aktuelle data fra tabell 4.5.*

De 10 tilfellene der alder er ukjent vil bli inkludert i de tre datasamlingene som dannes når de treningstilfellene der attributten er kjent splittes. Med utgangspunkt i beslutningstreet i figur 4.1, er størrelsen på treningsdatasamlingen som havner på noden “alkohol påvirkning” og tilfredstiller testen “16<=x<18” lik 35 (25 dataelementer med komplett informasjon pluss 10 dataelementer med inkomplett informasjon).

Før denne splittingen var det 90 kjente tilfeller, og 25 av dem hører til kategorien “16<=x<18”. Vekten til de treningstilfellene der splitt-attributten er ukjent, er derfor gitt ved:

*ny_vekt = gammel_vekt x5/90 = 0.8,* der gammel_vekt i dette tilfellet var lik 1. Dersom alle vekter summeres, vil summen være konstant hele tiden. De samme modifiserte formlene som ble presentert i det foregående kan nå benyttes, men med den semantiske endringen at $|S_i|$ eller $|S|$ representerer summen av vektene, og ikke antall treningselementer i en samling.

Dersom et testtilfelle skal klassifiseres og enkelte attributtverdier mangler, gjennomføres klassifiseringen ved å følge de forskjellige mulige grenene i treet fra rot til løvnoder, og deretter kan sannsynligheten for tilhørighet i de forskjellige klassene beregnes.




*Figur 4.2* *Figuren viser en del av et mulig beslutningstre for eksempelet "Dørvaktens dilemma". "N=5" betyr at 5 dataelementer fra treningsdatabasen havnet på denne noden og var klassifisert som "Nei". "J" representerer klassen "Ja" og "S" klassen "Senere".*

Når et testtilfelle ender opp på noden Gruppetilhørighet på siste figur, og attributtverdien er ukjent, dvs. dørvakten klarer ikke bestemme vedkommendes "Gruppetilhørighet", kan likevel dørvakten (med litt kveldsundervisning) beregne sannsynligheten for om gjesten er velkommen nå, senere eller ikke i det hele tatt. Summerer først treningstilfellene for hver klasse-kategori:

|Nei| = 5+4,3+1+3,3+2 =15,6
|Ja| = 1+0+2,1+20+6 = 29,1
|Senere|= 9+3+1 = 13
Sum = |Nei|+|Ja|+|Senere| = 57,7
Sannsynlighet for kategori:
-Nei: 15,6/Sum = 0,27
-Ja: 29,1/Sum = 0,50
-Senere: 13/Sum = 0,22.

Siden sannsynligheten for "Ja" er størst, konkluderer dørvakten med at det er best å slippe vedkommede inn, selv om han ikke kjenner "Gruppetilhørighet" i dette tilfellet. I C4.5 lagres kun antall treningstilfeller og antall feilklassifiseringer på hver løvnode. Dersom det er kun to klasser er det mulig å beregne sannsynlighetene på samme måte som i det foregående, men dersom det er flere enn to klasser må antallet treningstilfeller av alle klasser som havner på en løvnode lagres som i eksempelet.

## 4.4 Forenkling av trær

### 4.4.1 Motivasjon for forenkling av trær

Genererte trær kan bli store uten at de nødvendigvis inneholder så mye informasjon. Et forenklet tre kan faktisk gi bedre resultater for testdata. Dette kan illustreres ved et eksempel: Dersom man tar utgangspunkt i et datasett klassifisert etter to kategorier: "Ja" og "Nei" med henholdsvis ($p$-1=0.25) 25% og ($p$=0.75) 75% fordelt på hver kategori. Antall attributter er f.eks. 20 og antall datasampler er rikelig. Anta at det ikke er noen sammenheng mellom attributtene og klassene. Denne antakelsen betyr at attributtene ikke har noen informasjonsverdi. Sannsynligheten for at et uklassifisert sampel bli klassifisert av beslutningstreet som "Ja" vil være 25% og som "Nei" 75%. (Vel, dette er ikke helt riktig. Det forutsetter at det er ikke er inkonsistent data, dvs. at "sampler" med like attributtverdier ikke er klassifisert forskjellig. Imidlertid benytter jeg denne naive forutsetningen for å illustrere poenget.) Så, hva er da sannsynligheten for feilklassifisering?

Det er to muligheter for feilklassifisering: Enten er "sampelet" av klassen "Ja" med 25% sannsynlighet og blir klassifisert som "Nei" med 75% sannsynlighet eller så er klassen "Nei" med 75% sannsynlighet og blir klassifisert som "Ja" med 25% sannsynlighet. Sannsynligheten for feil blir i begge tilfeller lik: $p(p\text{-}1)$. Og følgelig blir den totale sannsynligheten for feil lik: $2p(p\text{-}1)$. Med tallene i eksempelet blir dette 37.5% sannsynlighet for feil. Dersom treet forenkles slik at det bare inneholder en node og denne er av klasse "Nei" vil sannsynligheten for feil være 25%. Dvs. at det enkle treet med kun en node gir best resultat i dette tilfellet.

### 4.4.2 Metode for forenkling av trær

I C4.5 er det valgt å generere beslutningstreet først og å forenkle det etterpå. Etter at treet er generert vil det være kjent hvor stor mengde av treningsdata som har funnet veien til hver løvnode og hvor stor mengde som er feilklassifisert. Hvis det forventede antallet feilklassifiseringer for en forgrening i et tre hadde vært kjent, ville det naturlige være å erstatte forgreningen med en enkelt node hvis det forventede antallet av feilklassifiseringer ble redusert. Dvs. at dersom sannsynlighetsfordelingen under hver node hadde vært kjent hadde det vært uproblematisk å suksessivt forenkle treet fra løvnodene og opp mot roten av treet. Det ligger i sakens natur at slik informasjon er vanskelig tilgjengelig. Noe heuristisk løses dette ved å anta at antallet datasampler ($N$) og misklassifiseringer ($E$) under hver node inngår i en binomisk fordeling.

For å illustrere binomiske fordelinger benyttes ofte et eksempel med å trekke kuler fra en urne. Det er to typer kuler, f.eks. svart og hvit kule. Sannsynligheten for å trekke en svart kule er $p$ og følgelig er sannsynligheten for å trekke en hvit kule $q=1\text{-}p$. Det trekkes et antall $n$, kuler med tilbakelegging, og det er uten betydning hvilken rekkefølge kulene trekkes i. Sannsynligheten for å trekke $x$ svarte kuler er da gitt ved:

$$f(x) = \binom{n}{x} p^x q^{n-x} \tag{4.8}$$

Hvordan kan en binomisk fordeling settes i sammenheng med fordelingene omtalt over? Først og fremst er det i mangel på bedre viten at en slik fordeling benyttes. For å sette det i

sammenheng antas at av *N* treningstilfeller observeres *E* misklassifiseringer. Analogt, av *n* trukne kuler trekkes *b* svarte. Da kan det sies at når et testdatasett klassifiseres vil det analoge være å trekke *n* nye kuler. Men da må datamengden som havner på denne noden være like stor som under genereringen av treet. I dette tilfellet vil antakelsen om tilbakelegging gjelde som for kuleeksempelet. I likhet med i kuleeksempelet er det likegyldig i hvilken rekkefølge de enkelte misklassifiseringer ankommer noden. Nærmere en begrunnelse for analogien er det vanskelig å komme.

Som verdi for forventet antall feil er det egentlig *p* i kuleanalogien som er interessant. For få en viss robusthet i heuristikken beregnes et øvre konfidensbånd for *E* med konfidensnivå 25%. Konfidensbåndet er implisitt gitt ved:

$$0{,}25 = \sum_{0}^{x} \binom{n}{x} p^x q^{n-x} \tag{4.9}$$

I ligningen settes *N* inn for *n*, og *E* inn for *x*.

Konfidensbåndet er gitt av *p* i siste ligning og vil bli referert til som $U_{25\%}$.

Det øvre konfidensbånd for *E* beregnes for alle løvnodene. Videre sammenliknes summen av disse med det tilsvarende konfidensbåndet til modernoden. Konfidensbåndet til modernoden beregnes ved å velge den mest forekommende klassen under noden som nodens klassifisering. Da regnes de treningstilfellene som tilhører andre klasser, som gjennomløper noden, som feil. Da er *N* og *E* for noden kjent og det øvre konfidensbåndet til *E* kan beregnes på samme måten som for løvnodene. Dersom det øvre konfidensbåndet til *E* er mindre for modernoden enn summen av konfidensbåndene for løvnodene, indikerer dette at løvnodene bør forenkles bort.

Etter dette kriteriet forenkles treet inntil ingen flere forenklinger er aktuelle å utføre. Det er viktig å være klar over at det er bare aktuelt å erstatte barna til en attributtnode med en løvnode når alle disse er løvnoder.

## 4.5 Regler

Selv etter at et beslutningstre er forenklet, kan det være stort, uoversiktlig og inneholde irrelevante kriterier. Dersom noe av hensikten er å gi mennesker mere innsikt i en beslutningsprosess, er det bedre å generere et sett med forenklede regler. Regler genereres fra det uforenklede treet ved å starte fra roten og følge grenene til løvnodene. Hver slik sti er unik, og representerer en regel. F.eks er en bestemt sti i figur 4.1 gitt ved følgende regel:

**if**
    alder = ”16<=x<18”
    alkoholpåvirkning = “svært synlig”
**then** klasse = “Nei”

Det er en implisitt konjunksjon mellom de to testene i regelen.

Treet kan erstattes med et sett av slike regler. Et slikt sett med regler representerer imidlertid ikke noe mer forståelig for et menneske enn det opprinnelige treet gjør. På liknende måte som for treet, forenkles regelsettet. Ved å gjøre denne forenklingen på en hensiktsmessig måte, dannes et regelsett som klassifiserer like godt som det forenklede treet og som i tillegg kan gi forståelse.

Forenklingsprosessen gjøres i tre steg. Først fjernes ikke-signifikante tester fra reglene. Deretter grupperes reglene på de forskjellige klassene og regler som ikke forbedrer klassifiseringen fjernes. Til slutt ordnes reglene i en optimal rekkefølge og "default"-klassen (testtilfeller som ikke dekkes av reglene blir klassifisert som "default") velges.

### 4.5.1 Fjerning av ikke-signifikante tester fra reglene

For å vurdere om en enkelt test som f.eks alder = "16<=x<18" bør fjernes fra regelen, sammenliknes det øvre konfidensbåndet på forventet antall feil ($U_{25\%}$ omtalt i forrige delkapittel) for regelen med og uten testen. Dersom fjerning av en av testene gir en lavere verdi på konfidensbåndet for forventet antall feil enn tilfellet var før testen ble fjernet, indikerer dette at testen er kandidat til å fjernes. Bare én test fjernes, nemlig den som gir lavest verdi av konfidensbåndet for forventet antall feil. Deretter genereres nye verdier for konfidensbåndene og prosessen gjentaes inntil det ikke oppnås noen lavere verdi av konfidensbåndet på forventet antall feil ved å fjerne tester fra regelen.

Fra forrige delkapittel er det gitt en forklaring på hva konfidensbåndet $U_{25\%}$ er. Inngangsparametre til $U_{25\%}$ er $N$ og $E$. I dette tilfellet er $N$ antall treningstilfeller som tilfredstiller testene i regelen, og $E$ er antallet av disse som ikke tilhører klassen gitt av regelen. Generelt blir $N$ og $E$ forskjellig for alle varianter av regelen som dannes når tester fjernes.

### 4.5.2 Fjerning av regler

Når flere regler dannes for samme klasse, er det ofte slik at noen regler inneholder andre regler innenfor klassen. Da er det hensiktsmessig å fjerne unødvendige regler. Der er også mulig at enkelte regler bidrar så mye til feilklassifisering at de med fordel kan fjernes. En beskrivelse av metoden som benyttes for å fjerne uhensiktsmessige regler og den teoretiske bakgrunnen, vil ikke bli gitt her. Istedet henvises det til (2).

### 4.5.3 Rangering av regler

Når det endelige settet av regler foreligger, må disse rangeres. Et testtilfelle kan teste positivt på flere regler. Klassifiseringen kan derfor gi forskjellige resultater alt etter hvilke rekkefølge reglene er rangert i. I C4.5 rangeres reglene i blokker. Hver blokk inneholder reglene for én klasse. De blokkene som feilaktig tester positivt på flest treningstilfeller rangeres sist, slik at testtilfeller skal få mulighet til å teste positivt på blokker med bedre statistikk. "Default"-klassen som benyttes når testtilfellet ikke tester positivt på noen av blokkene, velges som den klassen som ikke dekkes av noen regel og som flest treningstilfeller hører inn under.

### 4.5.4 Windowing

*Windowing* er en teknikk som anvendes i C4.5. For store datasett kan denne teknikken gjøre algoritmen raskere, men i hovedsak benyttes windowing for å optimalisere treet. C4.5 er en typisk grådighets-algoritme, og det er ikke garantert at det beste treet genereres.Teknikken går ut

på å trekke et subsett av treningstilfellene slik at klassene blir ca. likt fordelt. Subsettet benyttes for å generere treet. Resten av treningsettet klassifiseres ved hjelp av det genererte treet, og halvparten av de feilklassifiserte tilfellene inkluderes i subsettet. Denne prosessen fortsetter inntil det prosentvise antallet av alle treningstilfellene som havner på en løvnode med annen klasse, har konvergert.

Denne teknikken forbedres ved å generere flere trær og utføre den beskrevne prosessen på alle, og deretter velge det beste treet. Dvs. det treet som har det laveste antall treningstilfeller som havner på en løvnode med annen klasse.

I denne sammenheng skal det sies, at dersom flere tilfeldige trær genereres, og klassifikatoren baseres på regler fra hvert av trærne, gir dette stort sett en bedre klassifikator.

## 4.6 Kontinuerlige variable

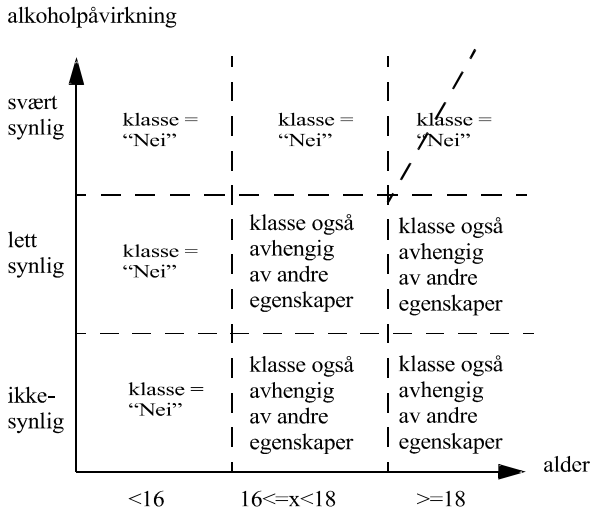
I denne oppgaven ble det skrevet programkode som implementerer noen viktige aspekter ved C4.5. Blant annet ble det skrevet programkode inspirert av C4.5 som takler kontinuerlige variable. Denne algoritmen genererer et tre for hver kontinuerlig attributt. Treet genereres etter samme metode som det som er beskrevet i det foregående med noen modifikasjoner. Først sorteres alle tilfellene i samlingen etter stigende verdi for attributtet, så foretas en splitting av samlingen på basis av "gain"-kriteriet. Denne rekursjonen fortsetter inntil *gain* blir lik null eller at grensen for maksimum antall splittnivåer nåes. En kontinuerlig attributt kan da f.eks bli splittet opp tre områder. Grensen for maksimum antall splittnivåer settes i initialiseringen til modulen. Det genererte treet blir binært, slik at de tilfellene der den aktuelle kontinuerlige attributten har en verdi som er mindre enn *splittverdien* blir overført til den ene grenen ut fra det aktuelle knutepunktet og de tilfellene der den aktuelle kontinuerlige attributten har en verdi som er større eller lik splittverdien blir overført til den andre grenen.

Treene benyttes til to formål: De benyttes til å generere tester som legges inn i datastrukturen for de kontinuerlige attributtene og de benyttes til å preprosessere kontinuerlige inngangsvariable. Med å generere tester menes f.eks å splitte tallområdet til en kontinuerlig attributt opp i områdene x<=10, 10<x<=100 og x>=100. Med preprosessering av kontinuerlige inngangsvariable menes f.eks å gi de kontinuerlige verdiene 10,1 og 15,2 for en kontinuerlig attributt i datasamlingen som skal klassifiseres, verdien 10<x<=100.

Grensen for maksimum antall splittnivåer er helt essensiell for evnen til å generalisere og må derfor settes med omhu. Dersom denne grensen settes for stor, vil det bli generert like mange tester for hvert attributt som det er tilfeller i data samlingen som benyttes i opplæringsfasen.

## 4.7 Svakheter ved C4.5

Den største svakheten med algoritmen er at den deler opp beslutningsrommet i såkalte hyperrektangler, og kun det. For å illustrere betydningen av dette videreføres eksempelet Dørvaktens dilemma. Det tas det utgangspunkt i to de første nivåene i beslutningstreet i figur 4.1. Dette danner grunnlaget for å lage en projeksjon av beslutningsrommet ned i planet som på figur 4.3.




*Figur 4.3 Figuren viser en projisert flate i beslutningsrommet for eksempelet Dørvaktens dilemma.*

Figur 4.3 illustrerer hva som menes med at algoritmen deler opp beslutningsrommet i hyperrektangler. Oppdelingen av beslutningsflaten i figur 4.3 i rektangler er fornuftig, men det kan meget godt tenkes at dørvakten ikke bare tenker i bokser og også foretar sin vurdering basert på den grove diagonale stiplede linjen i boksen oppe til høyre i figuren, slik at personer som havner over linjen i "Nei" boksen oppe i høyre hjørne ikke slipper inn, mens de som havner under slipper inn. For å realiserer dette må kategoriene ">=18" og "svært synlig" fragmenteres (f.eks "21>x>=18" og x>=21 og "ganske synlig" og "synlig på 150m avstand"). Dette kan føre til mye oppsplitting av treet og behov for mye treningsdata. I tillegg er det stor fare for at generaliseringsevnen ikke forbedres ved fragmentering.

# 5 EKSPERIMENTER

Som en del av denne oppgaven er det skrevet programkode i Smalltalk VisualWorks3.0. Programkoden implementerer prinsipielle sider av C4.5. Med støtte fra Matlab, implementerer programkoden også SVM samt et grensesnitt mot en modul, produsert ved FFI, som implementerer et nevralt nettverk. De deler av C4.5 som er implementert, er de deler som omhandler generering, forenkling og utskrift av beslutningstreet og klassifisering av testtilfeller. I tillegg er det skrevet programkode inspirert av C4.5 som takler kontinuerlige variable. Modulen som implementerer prinsipielle sider av C4.5, vil bli referert til som B4.5. SVM-modulen og B4.5 inngår i en klassifiseringsmodul. I denne modulen inngår også den nevnte modulen som implementerer et nevralt nettverk med ett skjult lag der tilbakepropagering benyttes for å justere vektene. I det følgende vil noen eksperimentelle tester av programmodulene bli presentert. Programmodulene er beskrevet i Appendiks A og koden er gitt i Appendiks D.

## 5.1 Car Evaluation Database

I det første forsøket benyttes data hentet fra UCI-databasen (10).

Størrelsen på datasettet er 1728 tilfeller som dekker attributtrommet fullstendig. Antall attributter er 6. Datastrukturen er gitt ved:

| buying | maint | doors | persons | lug_boot | safety | class |
|---|---|---|---|---|---|---|
| v-high | v-high | 2 | 2 | small | low | unacc |
| high | high | 3 | 4 | med | med | acc |
| med | med | 4 | more | big | high | good |
| low | low | more | | | | v-good |

*Tabell 5.1 Datastruktur for Car Evaluation Database*

Det fins ingen manglende attributter, og klassene er fordelt som i tabell 5.2.

| class | N | N[%] |
|---|---|---|
| unacc | 1210 | 70,02 |
| acc | 384 | 22,22 |
| good | 69 | 3,99 |
| v-good | 69 | 3,76 |

*Tabell 5.2 Fordeling av klassene i Car Evaluation Database*

Litt mer informasjon om datasettet er gitt i Appendiks C.2.

### 5.1.1 Beslutningstre generert fra Car Evaluation Database

Ved å benytte programmodulen for B4.5 og hele databasen ble beslutningstreet i Appendiks B generert. Forenklingsrutinen i B4.5 ga beslutningstreet avtegnet i figur 5.1–3.

Antall forventede feil er 119 som utgjør 6,9% av 1728 som er størrelsen på datasettet. Med antall forventede feil menes det antall feil som forekommer ved å benytte det forenklede treet til å klassifisere treningsdatasettet. I figur 5.1-3 gir parantesen bak hver klassifisering informasjon på formen (*N*/*E*), der *N* er antall klassifisert på denne noden og *E* er andelen som er feilklassifisert. Det er summen av alle feilklassifiserte som menes med forventet antall feil.

For å teste modulens evne til å generalisere, ble data settet delt i to like store disjunkte sett, treningssett og testsett, ved tilfeldig trekking. Ved å benytte programmodulen for B4.5 og treningsettet ble det forenklede beslutningstreet på figur 5.4 generert.

Forventet antall feil for det forenklede treet er 85 som er 9,8% av treningssettet, men det forekommer 132 feil for testdatasettet som utgjør 15% av dette. Når treet ikke forenkles forekommer det 95 feil for testdatasettet som utgjør 11% av dette.

```
safety = low: unacc (576/0)
safety = med
        persons = 2: unacc (192/0)
        persons = 4
                buying = vhigh
                        maint = vhigh: unacc (12/0)
                        maint = high: unacc (12/0)
                        maint = med
                                lug_boot = small: unacc (4/0)
                                lug_boot = med: unacc (4/2)
                                lug_boot = big: acc (4/0)
                        maint = low
                                lug_boot = small: unacc (4/0)
                                lug_boot = med: unacc (4/2)
                                lug_boot = big: acc (4/0)
                buying = high
                        lug_boot = small: unacc (16/0)
                        lug_boot = med: unacc (16/6)
                        lug_boot = big: acc (16/4)
                buying = med
                        maint = vhigh
                                lug_boot = small: unacc (4/0)
                                lug_boot = med: unacc (4/2)
                                lug_boot = big: acc (4/0)
                        maint = high
                                lug_boot = small: unacc (4/0)
                                lug_boot = med: unacc (4/2)
                                lug_boot = big: acc (4/0)
                        maint = med: acc (12/0)
                        maint = low
                                lug_boot = small: acc (4/0)
                                lug_boot = med: acc (4/2)
                                lug_boot = big: good (4/0)
                buying = low
                        maint = vhigh
                                lug_boot = small: unacc (4/0)
                                lug_boot = med: unacc (4/2)
                                lug_boot = big: acc (4/0)
                        maint = high: acc (12/0)
                        maint = med
                                lug_boot = small: acc (4/0)
                                lug_boot = med: acc (4/2)
                                lug_boot = big: good (4/0)
                        maint = low
                                lug_boot = small: acc (4/0)
                                lug_boot = med: acc (4/2)
                                lug_boot = big: good (4/0)
```

*Figur 5.1 Del 1 av 3 av beslutningstre generert av B4.5 med basis i hele datasettet i Car Evaluation Database*

```
            persons = more
                lug_boot = small: unacc (64/15)
                lug_boot = med: acc (64/29)
                lug_boot = big
                    buying = vhigh
                        maint = vhigh: unacc (4/0)
                        maint = high: unacc (4/0)
                        maint = med: acc (4/0)
                        maint = low: acc (4/0)
                    buying = high: acc (16/4)
                    buying = med: acc (16/4)
                    buying = low
                        maint = vhigh: acc (4/0)
                        maint = high: acc (4/0)
                        maint = med: good (4/0)
                        maint = low: good (4/0)
    safety = high
            persons = 2: unacc (192/0)
            persons = 4
                buying = vhigh
                    maint = vhigh: unacc (12/0)
                    maint = high: unacc (12/0)
                    maint = med: acc (12/0)
                    maint = low: acc (12/0)
                buying = high
                    maint = vhigh: unacc (12/0)
                    maint = high: acc (12/0)
                    maint = med: acc (12/0)
                    maint = low: acc (12/0)
                buying = med
                    maint = vhigh: acc (12/0)
                    maint = high: acc (12/0)
                    maint = med
                        lug_boot = small: acc (4/0)
                        lug_boot = med: acc (4/2)
                        lug_boot = big: vgood (4/0)
                    maint = low
                        lug_boot = small: good (4/0)
                        lug_boot = med: good (4/2)
                        lug_boot = big: vgood (4/0)
```

*Figur 5.2* *Del 2 av 3 av beslutningstre generert av B4.5 med basis i hele datasettet i Car Evaluation Database*

```
            buying = low
                maint = vhigh: acc (12/0)
                maint = high
                    lug_boot = small: acc (4/0)
                    lug_boot = med: acc (4/2)
                    lug_boot = big: vgood (4/0)
                maint = med
                    lug_boot = small: good (4/0)
                    lug_boot = med: good (4/2)
                    lug_boot = big: vgood (4/0)
                maint = low
                    lug_boot = small: good (4/0)
                    lug_boot = med: good (4/2)
                    lug_boot = big: vgood (4/0)
    persons = more
            buying = vhigh
                maint = vhigh: unacc (12/0)
                maint = high: unacc (12/0)
                maint = med: acc (12/1)
                maint = low: acc (12/1)
            buying = high
                maint = vhigh: unacc (12/0)
                maint = high: acc (12/1)
                maint = med: acc (12/1)
                maint = low: acc (12/1)
            buying = med
                maint = vhigh: acc (12/1)
                maint = high: acc (12/1)
                maint = med: vgood (12/5)
                maint = low: vgood (12/5)
            buying = low
                maint = vhigh: acc (12/1)
                maint = high: vgood (12/5)
                maint = med: vgood (12/5)
                maint = low: vgood (12/5)
```

*Figur 5.3* *Del 3 av 3 av beslutningstre generert av B4.5 med basis i hele datasettet i Car Evaluation Database.*

```
safety = low: unacc (285/0)
safety = med
        persons = 2: unacc (84/0)
        persons = 4
                lug_boot = small: unacc (37/12)
                lug_boot = med
                        doors = 2: unacc (13/5)
                        doors = 3: unacc (12/3)
                        doors = 4: acc (9/2)
                        doors = 5more: acc (10/3)
                lug_boot = big: acc (31/7)
        persons = more
                buying = vhigh: unacc (24/5)
                buying = high: unacc (23/8)
                buying = med: acc (28/11)
                buying = low
                        maint = vhigh: acc (8/2)
                        maint = high: acc (5/0)
                        maint = med: good (6/3)
                        maint = low: good (8/2)
safety = high
        persons = 2: unacc (93/0)
        persons = 4
                buying = vhigh
                        maint = vhigh: unacc (10/0)
                        maint = high: unacc (7/0)
                        maint = med: acc (7/0)
                        maint = low: acc (6/0)
                buying = high: acc (18/4)
                buying = med: acc (24/9)
                buying = low
                        maint = vhigh: acc (9/0)
                        maint = high: vgood (3/1)
                        maint = med: vgood (8/3)
                        maint = low: good (3/1)
        persons = more
                buying = vhigh
                        maint = vhigh: unacc (5/0)
                        maint = high: unacc (6/0)
                        maint = med: acc (10/0)
                        maint = low: acc (10/1)
                buying = high
                        maint = vhigh: unacc (7/0)
                        maint = high: acc (5/0)
                        maint = med: acc (5/1)
                        maint = low: acc (4/1)
                buying = med
                        lug_boot = small: unacc (7/3)
                        lug_boot = med: acc (6/3)
                        lug_boot = big
                                maint = vhigh: acc (2/0)
                                maint = high: acc (2/0)
                                maint = med: vgood (1/0)
                                maint = low: vgood (3/0)
                buying = low
```

*Figur 5.4 Forenklet beslutningstre generert av B4.5 med basis i halve datasettet i Car Evaluation Database.*

Ved å overføre halvparten av treningssettet til testsettet ble det nye treningssettet på 432 elementer og testsettet på 1296 elementer. Det forenklede treet ble seende ut som på figur 5.5.

```
safety = low: unacc (144/0)
safety = med
        persons = 2: unacc (44/0)
        persons = 4
                lug_boot = small
                        maint = vhigh: unacc (5/0)
                        maint = high: unacc (6/0)
                        maint = med: acc (7/1)
                        maint = low: unacc (2/1)
                lug_boot = med: acc (28/13)
                lug_boot = big: acc (18/4)
        persons = more
                buying = vhigh: unacc (11/2)
                buying = high: unacc (12/3)
                buying = med
                        maint = vhigh: acc (3/1)
                        maint = high: unacc (4/2)
                        maint = med: acc (2/0)
                        maint = low: good (5/0)
                buying = low: acc (15/6)
safety = high
        persons = 2: unacc (41/0)
        persons = 4
                maint = vhigh: acc (11/3)
                maint = high
                        buying = vhigh: unacc (6/0)
                        buying = high: acc (3/0)
                        buying = med: acc (1/0)
                        buying = low: acc (1/0)
                maint = med: acc (10/5)
                maint = low: acc (8/1)
        persons = more
                buying = vhigh
                        maint = vhigh: unacc (1/0)
                        maint = high: unacc (4/0)
                        maint = med: acc (4/0)
                        maint = low: acc (6/1)
                buying = high: acc (10/4)
                buying = med: vgood (8/4)
                buying = low
                        maint = vhigh: acc (5/0)
                        maint = high: vgood (1/0)
                        maint = med: good (2/1)
                        maint = low: good (4/2)
```

*Figur 5.5 Forenklet beslutningstre generert av B4.5 med basis i 25% av datasettet i Car Evaluation Database.*

Forventet antall feil for treet er 54 som utgjør 13% av 432 som er størrelsen på treningssettet. Det forekommer 244 feil for testdatasettet som utgjør 19% av dette. Når treet ikke forenkles forekommer det 186 feil for testdatasettet som utgjør 14% av dette.

Siden klassene er rangert er det mest fornuftig å vekte feilene som i tabell 5.3.

| class | unacc | acc | good | v-good |
|---|---|---|---|---|
| unacc | 0 | 0,33 | 0,67 | 1 |
| acc | 0,33 | 0 | 0,33 | 0,67 |
| good | 0,67 | 0,33 | 0 | 0,33 |
| v-good | 1 | 0,67 | 0,33 | 0 |

*Tabell 5.3 Vekting av feilene. Tabellen er symmetrisk. Dersom f.eks klassifiseringen gir "good" når den riktige klassen er "unacc" er den vektede feilen lik 0.67.*

Resultatet blir som følger: 102,67 feil som utgjør 7,9% av testsettet for det forenklede treet, og 78,33 feil som utgjør 6,0% av testsettet for det uforenklede treet. Det er kun i tilfellet med 432 elementer i treningsettet at jeg benytter veiing av feilene. Det ser altså ut til at B4.5 generaliserer bra. Årsaken til at det forenklede treet viser dårligere resultater enn det uforenklede, er at det er benyttet et for pessimistisk øvre bånd på feil ($U_{CF}$, der CF var satt til 25%. Ved å redusere CF blir det øvre båndet mindre pessimistisk). Det fins ikke inkonsistent data i datasettet og det forekommer ikke støy (Se appendiks B for uforenklet tre basert på hele datasettet).

### 5.1.2 SVM anvendt på Car Evaluation Database

SVM benytter numeriske attributtverdier. Dersom attributtverdiene ikke er numeriske, må de gis numeriske verdier. Ordnede ikke-numeriske verdier bør forbli ordnet, fordi det da sannsynligvis blir lettere å separere klassene. Attributtverdiene i eksempelet transformeres til numeriske verdier som vist i tabell 5.4.

| buying | maint | doors | persons | lug_boot | safety |
|---|---|---|---|---|---|
| v-high => 4 | v-high => 4 | 2 => 1 | 2 => 1 | small => 1 | low =>1 |
| high => 3 | high => 3 | 3 =>2 | 4 =>2 | med =>2 | med =>2 |
| med =>2 | med =>2 | 4 => 3 | more =>3 | big => 3 | high =>3 |
| low =>1 | low =>1 | 5-more => 4 | | | |

*Tabell 5.4 Transformering av attributtverdiene til numeriske verdier.*

Det ble benyttet en polynomisk kjerne av 4. grad.

Det må læres opp en SVM for hver klasse. Støttevektor maskinen læres opp til å returnerer en positiv verdi når den klassifiserer en vektor av ønsket klasse og å returnere negative verdier for alle andre klasser. Når en vektor klassifiseres, velges den klassen som tilhører den klassifikatoren som returnerer høyest verdi. I dette tilfellet må det læres opp fire klassifikatorer. Ved først å benytte et treningsett på 100 elementer og testsettet på 1628 elementeret, forekom det 324 feil som utgjorde 22% av testsettet. Dette tilsvarer 143 vektede feil som utgjør 8,8% av testsettet.

Det var videre intensjonen å benytte treningsettet på 432 elementer for å sammenlikne med beslutningstreet. Det oppsto da et problem fordi algoritmen som løser det kvadratiske optimaliseringsproblemet i Matlab (Matlab løser optimaliseringsproblemet og returnerer løsningen til SVM-modulen) ikke greier å løse et så stort problem. Den Hessiske matrisen får dimensjon 432x432. For å benytte mest mulig av informasjonen i treningssettet på 432, ble det faktum at kun støttevektorene er relevante benyttet. Det ble tatt utgangspunkt i et treningssett på

144 elementer av de 432. Støttevektorene til de fire klassifikatorene ble beregnet. Klassifisering av de 288 gjenværende elementene ble utført, og alle elementer som ikke var støttevektorer for noen av de fire klassifikatorene ble fjernet fra settet. Deretter ble treningsettet inkrementert med de elementene som ble feilklassifisert inntil feilklassifisering av de gjenværende elementene ikke forekom. Selv om klassifikatoren klassifiserer de 432 elementene uten feil betyr det ikke at klassifikatoren er optimal mhp. valg av støttevektorer. Denne prosedyren ga en klassifikator som burde være dårligere enn dersom hele treningsettet ble benyttet i én kjøring av optimaliseringen. Imidlertid ble resultatene som følger: Ved et (suboptimalt) utvalg av 185 elementer i treningsettet på 432, ble testdatasettet på 1296 elementer klassifisert med 154 feil tilsvarende 12%. Antall veide feil var 57; tilsvarende 4,4% av testsettet.

### 5.1.3 Nevralt nettverk anvendt på Car Evaluation Database

Det nevrale nettverket som ble benyttet hadde kun ett skjult lag, og ble testet med 4 og 8 nevroner i det skjulte laget. Utgangslaget ble konfigurert med 4 nevroner. Kodingen av klassene på utgangen var enkel: Nettverket ble trent til å returnere 1 på utgangen som representerte den aktuelle klassen og null på de andre utgangene som vist i tabell 5.5.

| klasse | utgang 1 | utgang 2 | utgang 3 | utgang 4 |
|---|---|---|---|---|
| unacc | 1 | 0 | 0 | 0 |
| acc | 0 | 1 | 0 | 0 |
| good | 0 | 0 | 1 | 0 |
| v-good | 0 | 0 | 0 | 1 |

*Tabell 5.5 Koding av klassene til ønskede verdier på utgangene.*

Læringsraten var konstant lik 0.1 for alle forsøkene med det nevrale nettverket. Det ble benyttet 10000 treningsykler med treningssettet på 432 elementer. Da opplæringen ble brutt etter 10000 sykler var ikke nettverket i stand til å klassifisere treningsettet 100% riktig. Dette gjaldt for både for kjøringen med 8 og 4 nevroner i det skjulte laget. Resultatene av klassifiseringen av testsettet på 1296 elementer ble som gitt i tabell 5.6.

| antall noder i skjult lag | antall feil | prosentvis antall feil | antall vektede feil | prosentvis antall vektede feil |
|---|---|---|---|---|
| 4 | 146 | 11% | 56 | 4,3% |
| 8 | 116 | 9,0% | 46 | 3,6% |

*Tabell 5.6 Feilstatistikk for kjøringene med 4 og 8 nevroner i det skjulte laget.*

## 5.2 Wine Database

I det andre eksempelet er data hentet fra samme database på internett som for det første eksempelet (10).

Størrelsen på datasettet er 178 tilfeller. Antall attributter er 13. Alle attributtene har kontinuerlige verdier. Attributtene er gitt ved:

Alcohol, Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocyanins, Color intensity, Hue, OD280/OD315 of diluted wines og Proline.

Klassene er “1”, “2” og “3”. Disse er fordelt som følger:

- Klasse 1: 59 elementer
- Klasse 2: 71 elementer
- Klasse 3: 48 elementer.

Litt mer informasjon om datasettet er gitt i Appendiks C.1.

### 5.2.1 Beslutningstre generert fra Wine Database

Rutinene i B4.5 som diskretiserer kontinuerlige variable tillater at tallområdet for hvert attributt blir diskretisert opptil et bestemt antall områder. F.eks “$x$>2,57”, “2,47<$x$<=2.57, “1,82<$x$<=2,47” og “$x$<=1,82. Dette representerer fire områder som er det maksimale antallet som er mulig med de parametrene som var gitt for genereringen av treet. Det maksimale antallet tallområder som det opprinnelige tallområdet til en attributt diskretiseres i, bestemmer hvor godt vilkårlige funksjoner kan representeres. Dersom antallet tallområder blir for stort, blir generaliseringsevnen dårlig. Dette kan imidlertid kompenseres med forenkling av treet. Det er ikke gitt at forenkling av treet er naturlig i dette tilfellet.

Ved å benytte programmodulen for B4.5 og et tilfeldig valgt treningssett på 89 elementer av de 178, ble besluttningstreet på figur 5.6 generert. Forenklingsrutinen i B4.5 ga treet i figur 5.7.

For det ikke-forenklede treet forekommer det 6 feil som utgjør 6,7% av testdatasettet. Når treet forenkles forekommer det 12 feil som utgjør 13% av testdatasettet.

```
ColorIntensity = x>7.5
        OD280/OD315OfDilutedWines = 1.82<x<=2.47: 3 (1/0)
        OD280/OD315OfDilutedWines = 2.47<x<=2.57: 3 (0/0)
        OD280/OD315OfDilutedWines = x<=1.82: 3 (11/0)
        OD280/OD315OfDilutedWines = x>2.57: 1 (2/0)
ColorIntensity = 3.4<x<=7.5
        OD280/OD315OfDilutedWines = 1.82<x<=2.47
                Hue = 0.78<x<=0.85: 3 (0/0)
                Hue = x>1.02: 2 (1/0)
                Hue = x<=0.78: 3 (3/0)
                Hue = 0.85<x<=1.02: 3 (1/0)
        OD280/OD315OfDilutedWines = 2.47<x<=2.57: 1 (1/0)
        OD280/OD315OfDilutedWines = x<=1.82: 3 (11/0)
        OD280/OD315OfDilutedWines = x>2.57
                Flavanoids = x>3.69: 2 (1/0)
                Flavanoids = 1.09<x<=1.58: 1 (0/0)
                Flavanoids = 1.58<x<=3.69
                        Proline = 750<x<=985: 1 (6/0)
                        Proline = 510<x<=750
                                AlcalinityOfAsh = x<=17: 1 (1/0)
                                AlcalinityOfAsh = 17<x<=22.5: 2 (1/0)
                                AlcalinityOfAsh = 22.5<x<=25: 1 (0/0)
                                AlcalinityOfAsh = x>25: 1 (0/0)
                        Proline = x>985: 1 (21/0)
                        Proline = x<=510: 2 (2/0)
                Flavanoids = x<=1.09: 1 (0/0)
ColorIntensity = x<=3.4: 2 (26/0)
```

*Figur 5.6 Beslutningstre generert av B4.5 med basis i 50% av datasettet i Wine Database.*

```
ColorIntensity = x>7.5: 3 (14/2)
        ColorIntensity = 3.4<x<=7.5
                OD280/OD315OfDilutedWines = 1.82<x<=2.47: 3 (5/1)
                OD280/OD315OfDilutedWines = 2.47<x<=2.57: 1 (1/0)
                OD280/OD315OfDilutedWines = x<=1.82: 3 (11/0)
                OD280/OD315OfDilutedWines = x>2.57: 1 (32/4)
        ColorIntensity = x<=3.4: 2 (26/0)
```

*Figur 5.7 Forenklet beslutningstre generert av B4.5 med basis i 50% av datasettet i Wine Database.*

### 5.2.2 SVM anvendt på Wine Database

Det samme treningsettet på 89 elementer og testsettet på 89 ble benyttet som i det foregående. Det ble benyttet at utvalg av kjernefunksjoner. Treningssettet ble klassifisert 100% riktig for alle kjernefunksjonene. Resultatene for klassifiseringen av testsettet er summert opp i tabell 5.7.

| | Polynomisk kjerne 2. grad | Polynomisk kjerne 4. grad | RBF kjerne $\sigma$=30 |
|---|---|---|---|
| Antall feil | 11 | 10 | 11 |
| Prosentvis feil | 12 % | 11 % | 12 % |

*Tabell 5.7 Feilklassifisering for forskjellige kjerner.*

Det var nødvendig å skalere x-vektorene av numeriske årsaker. Verdien på *C*-parameteren var lik 5 under alle forsøkene.

### 5.2.3 Nevralt nettverk anvendt på Wine Database

Det ble testet med 4 og 13 nevroner i det skjulte laget og for 10000 og 20000 treningssykler. Det er kun kjøringen med 13 nevroner i det skjulte laget og 20000 treningssykler som er ført feilstatistikk på fordi de andre tilfellene ga svært dårlige resultater. Utgangslaget ble konfigurert med 3 nevroner. Kodingen av klassene på utgangen var enkel: Nettverket ble trent til å returnere 1 på utgangen som representerte den aktuelle klassen og null på de andre utgangene. Tabell 5.8 viser denne kodingen.

| klasse | utgang 1 | utgang 2 | utgang 3 |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 |

*Tabell 5.8 Koding av klassene til ønskede verdier på utgangene.*

Treningssettet var de samme 89 elementene som i det foregående. Resultatene av klassifiseringen av testsettet på 89 elementer ble 6 feil som tilsvarer 6,7% av testsettet.

## 5.3 Sammenligning av resultatene

Tabell 5.9 presenterer de beste resultatene for de tre klassifikatorene for eksemplene “Wine Database” med testsett og treningssett på 89 elementer, og Car Evaluation Database med treningssett og testsett på henholdsvis 432 og 1296 elementer.

| | B4.5 | SVM | Nevralt nettverk |
|---|---|---|---|
| Car Evaluation Database | 6,0 % | 4,4 % | 3,6 % |
| Wine DataBase | 6,7 % | 11 % | 6,7 % |

*Tabell 5.9 Tabellen viser prosent feilklassifisering i forsøkene med best resultat.*

# 6 DISKUSJON

## 6.1 Sammenlikning av sterke og svake sider ved de tre klassifikatorene

### 6.1.1 Evne til å formidle kunnskap

Egenskapen til å generere regler og formidle kunnskap er en av C4.5 sine sterke sider. Regler kan tolkes av mennesker dersom de ikke er for komplekse. SVM genererer ikke regler og formidler bare kunnskap indirekte gjennom resultatene fra klassifisering av datasett. Nevrale nettverk genererer heller ikke regler men det er mulig å dra ut kunnskap av verdiene på vektene i nettverket.

### 6.1.2 Evne til å lære

Evnen klassifikatorene har til å lære er det i prinsippet ingen begrensning på. SVM, beslutningstrær og nevrale nettverk kan i prinsippet tilnærme alle funksjoner.

Beslutningstre-algoritmen er garantert å klassifisere treningseksemplene omtrent så bra som mulig. Treet kan splittes opp helt til kun inkonsistente data eller mangelfulle data forårsaker feilklassifisering. Imidlertid har C4.5 en konsistent måte å dra nytte av mangelfull data i motsetning til de to andre klassifikatorene.

Ett problem som gjelder for nevrale nettverk, er at de ved opptrening kan gi klassifikatorer som verken klassifiserer treningsettet bra eller generaliserer bra på usette tilfeller. Tilbakepropageringsalgoritmen søker lokale optimum, og det er ikke sikkert at den finner noe optimum innen rimelig tid eller at det lokale optimumet er det samme som det globale. Det er f.eks mulig at antall noder er for lavt i de skjult lagene, eller at enkelte typer feilklassifisering gir lokale optimum.

SVM kan lære hvilke funksjoner som helst bare dimensjonen på Hilbert-rommet gjøres stor nok.

Betraktningene over støttes av resultatene i kapittel 5.

### 6.1.3 Evne til å generalisere

Evnen til å generalisere betyr evnen til å klassifisere usette tilfeller.

Beslutningstrær har en svakhet med at beslutningsrommet må deles opp i n-dimensjonale rektangler eller såkalte hyperrektangler. Dersom det virkelige beslutningsrommet ikke består av hyperrektangler, vil C4.5 i første fase av læringsfasen dele opp beslutningsrommet i en uforholdsmessig stor mengde hyperrektangler. I den andre fasen når treet forenkles, vil en rekke av disse hyperrektanglene bli fjernet. Resultatet blir at generaliseringsevnen reduseres i disse områdene. I områdene som ikke dekkes av tilstrekkelig treningsdata kan generaliseringsevnen bli veldig dårlig. I utgangspunktet blir en løvnode som det ikke havner noen treningstilfeller på i et tre gitt den mest forekommende klassen i modernoden. Utfra samme resonnement som over, vil denne generaliseringen miste mye av sin kraft dersom beslutningsrommet i dette området ikke består av hyperrektangler.

SVM implementerer et prinsipp som heter Strukturell risikominimalisering. Det som vil bestemme generaliseringsevnen til en SVM er verdien på parameteren *C* som representerer vektingen av feilklassifiseringen av treningsdata og dimensjonen på Hilbert-rommet. Dersom dimensjonen på Hilbert-rommet og *C* velges for høy, vil klassifikatoren kunne klassifisere støydata i treningssettet perfekt. Dette vil kunne ødelegge generaliseringsevnen totalt. Derfor bør dimensjonen på Hilbert-rommet gjøres for høy, og *C* parameteren bør tunes på bakgrunn av forsøk med klassifisering av testsett.

Noe av det samme problemet eksisterer for nevrale nettverk: Dersom det benyttes for mange noder i de skjulte lagene, vil nettverket få kapasitet til å klassifisere støydata slik at generaliseringsevnen ødelegges. I tillegg vil kompleksiteten på funksjonen utenfor det området som treningspunktene befinner seg i, kunne bli høy og gi dårlig generaliseringsevne i disse områdene også.

### 6.1.4 Evne til å takle mangelfull informasjon

C4.5 er den eneste av de tre klassifiseringsalgoritmene som takler mangelfulle inngangsvektorer på en konsistent måte. Dette er en verdifull egenskap siden det ofte er slik at ikke all informasjon er kjent.

## 6.2 Betraktninger knyttet til forsøkene med Wine Database og Car Evaluation Database

Som det går frem av tabell 5.9, så gir det nevrale nettverket best resultater totalt. B4.5 viser bra resultater både for tilfellet med diskrete attributter og for tilfellet med bare kontinuerlige attributter. Det er jo fint, for den delen av algoritmen som takler kontinuerlige attributter, har undertegnede laget med inspirasjon fra C4.5. SVM viser gode resultater for forsøket med “Car Evaluation Database”, men siden optimaliseringsalgoritmen i Matlab ikke var istand til takle hele treningssettet i det ene forsøket, er det mulig at resultatet kunne vært bedre. SVM gir ca. dobbelt så mange feilklassifiseringer som de to andre klassifikatorene som det går fram av tabell 5.9. Dette er påfallende og undertegnede finner ingen god forklaring.

## 6.3 Behov for regnekraft

Den minst regnekrevende algoritmen er B4.5 når det er få kontinuerlige attributter. Dersom det er mange kontinuerlige attributter er denne algoritmen ganske regnekrevende. Grunnen til dette er at tallområdet for hver attributt skal deles opp i områder. Dette gjøres ved å sortere treningsdatasamlingen i stigende rekkefølge på den aktuelle kontinuerlige attributten. Deretter evalueres splitt på alle verdiene for attributtet i den sorterte treningsdatasamlingen for *gain*-kriteriet. Så splittes samlingen på den verdien som gir størst *gain* og rekursjonen fortsetter ned til et bestemt nivå eller at alle klassene i en datasamling er like. Dette er ganske analogt med hvordan algoritmen fungerer globalt. Det genereres ett tre med tester for hver attributt. Dette er regnekrevende. Kjøringen som genererte det aktuelle resultatet i tabell 5.9 tok 15 minutter.

Kjøretiden til det nevrale nettverket er avhengig av størrelsen på nettverket, læringsraten og hvor lett det er å trene med de gitte treningsdata. Kjøringen tok rundt 30 minutter for “Car Evaluation Database” og rundt 50 minutter for “Wine Database”. Det hadde sikkert gått raskere dersom læringsraten (denne var 0.1, og alle inngangsvariable var skalert til ca. å ligge i området

null til én) hadde vært høyere i begynnelsen av kjøringen, for det tok ganske lang tid for nettverket å oppdage alle klassene.

Kjøretiden for SVM var moderat dvs. ca 3 minutter. Størrelsen på x-vektoren i optimaliseringsproblemet var på det meste rundt 190 elementer og optimaliseringen ble kjørt fire ganger siden det var fire klasser. Dersom problemet skal løses med dobbelt så mange elementer i x-vektoren, kan man regne med at kjøretiden iallfall åttedobles pga. at matriseinversjon er den tyngste operasjonen og er kubisk varierende med dimensjonen mhp. antall regneoperasjoner.

## 6.4 Erfaringer fra implementasjonen av klassifiseringsmodulen

Programmeringsspråket Smalltalk som ble benyttet for å implementere klassifiseringsmodulen hadde jeg ingen erfaringer med før jeg begynte å arbeide med denne oppgaven. Smalltalk er et rendyrket objektorientert språk, og siden jeg ikke hadde så mye programmeringserfaring med objektorienterte språk tok det litt tid å komme inn i den objektorienterte tankegangen. Imidlertid viste det seg at det er raskt å generere og "debugge" kode med Smalltalk VisualWorks3.0, når man først begynner å forstå syntaksen og hvordan utviklingsverktøyet fungerer. Årsaken til at det er raskt å generere kode er fordi linking og kompilering skjer parallelt med at koden skrives.

Det eneste problemet jeg hadde, var å implementere en algoritme for løsing av kvadratiske optimaliseringsproblemer. Det var i utgangspunktet heller dårlig stelt med matematiske programpakker til Smalltalk, men noe hjemmelaget fikk jeg tak i. Til slutt greide jeg å programmere en versjon av *Projisert Gradient* som virket, men det viste seg at for problemer av den størrelse som er aktuell å løse for SVM var dette en alt for ueffektiv algoritme. Det neste steget var å prøve å inkludere programkode skrevet i C eller C++. Dette er ikke rett fram, og jeg hadde for mangelfull dokumentasjon for hvordan dette gjøres til å våge å bruke veldig mye tid på det. Løsningen jeg etterhvert valgte var å benytte Matlab til å løse optimaliseringsproblemet. Jeg benyttet tekstfiler for å kommunisere med Matlab. Det som jeg da oppdaget, var at algoritmen i Matlab som løser kvadratiske optimaliseringsproblemer, ikke takler problemer som er større enn at x-vektoren har mer enn ca.180 elementer. Siden jeg alt hadde brukt mye tid på dette, ble denne løsningen den endelige. Men det er selvfølgelig mulig å erstatte denne løsningen i etterkant. Dette problemet kostet meg én måneds arbeid.

Dersom alle de tre modulene skulle lages fra grunnen av med metodene som er tilgjengelig i Smalltalk VisualWorks3.0 antar jeg at det ville være SVM modulen som ville være vanskeligst å implementere. Dette tror jeg på bakgrunn av de implementasjoner av løsning av kvadratiske optimaliseringsproblem som jeg har kommet over i C-kode. C4.5 er ikke like vanskelig å implementere men er til gjengjeld mer omfattende. C-koden til C4.5 er gitt i (1) og er på ca 100 sider i A-5 format. Modulen som implementerer det nevrale nettverket med ett skjult lag har jeg ikke satt meg godt nok inn i til å uttale meg om vanskelighetsgraden med å implementere denne. Men det ser ut som det er behov for langt mindre kode enn for de to andre modulene implementert fra grunnen av og i sin fulle form.

## 6.5 Videre arbeid

Det kunne vært interessant å teste C4.5 algoritmen i sin helhet. De delene av C4.5 som omfatter regler er ikke implementert i B4.5. Disse forenklingsrutinene skal ifølge forfatteren av C4.5 fungere bedre enn den som er implementert i B4.5.

Det som også kunne ha vært interessant, er å forurense datasettene i "Wine Database" og "Car Evaluation Data base" relativt mye og kjøre noen forsøk på dem, for å se hvordan de tre klassifikatorene taklet dette. Det er ikke sikkert at det nevrale nettverket ville kunne konvergert til en god løsning. Det er også mulig at *C*-parameteren for SVM måtte justeres for å takle dette.

Det hadde vært hensiktsmessig å anskaffe programkode som løser store kvadratiske optimaliseringsproblemer og gjøre klassifiseringsmodulen uavhengig av Matlab som ikke fungerer tilfredstillende for dette formålet.

# 7 KONKLUSJON

I denne oppgaven har tre typer klassifiseringsalgoritmer blitt beskrevet. De tre algoritmene er støttevektormaskin-algoritmen, nevrale nettverk og beslutningstre-algoritmen C4.5.

Det er blitt skrevet programkode i Smalltalk som implementerer en klassifiseringsmodul. De modulene som inngår i klassifiseringsmoduelen er en modul som implementerer en beslutningstre-algoritme inspirert av C4.5, en modul som implementerer støttevektormaskiner og en modul som implementerer nevrale nettverk. Koden i den siste modulen er skrevet av ansatte ved FFI.

Klassifiseringsmodulene er testet på to datasett med forskjellige egenskaper som er beregnet for å teste klassifiseringsalgoritmer. Alle klassifikatorene viste bra egenskaper til å lære og til å generalisere.

Det er litt vanskelig å uttale seg om hvilken algoritme som i prinsippet er best da alle algoritmene har sider som er sterkere enn de andre og alle har sider som er svakere enn de andre. Nevrale nettverk med tilbakepropagering har den svakheten at algoritmen gir lokalt optimale løsninger som kan være forskjellig fra det globale optimumet. Samtidig gir algoritmen svært gode resultater når læringsraten og størrelsen på nettverket velges riktig. Støttevektor-algoritmen har den svakheten at det et behov for å løse et ikke-trivielt optimaliseringsproblem. Men ideen bak algoritmen er veldig fornuftig med tanke på at den implementerer prinsippet ved strukturell risikominimalisering. C4.5 på sin side har den svakheten at den deler beslutningsrommet opp i hyperrektangler. Algoritmen er imidlertid den eneste som kan generere regler og direkte formidle kunnskap og som takler mangelfulle inngangsvektorer på en konsistent måte.

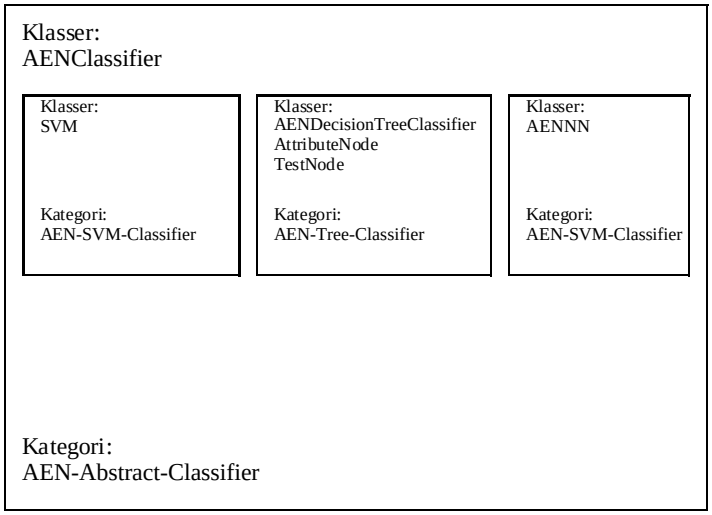
# APPENDIKS

# A

## A.1 Generell informasjon om klassifiseringsmodulen

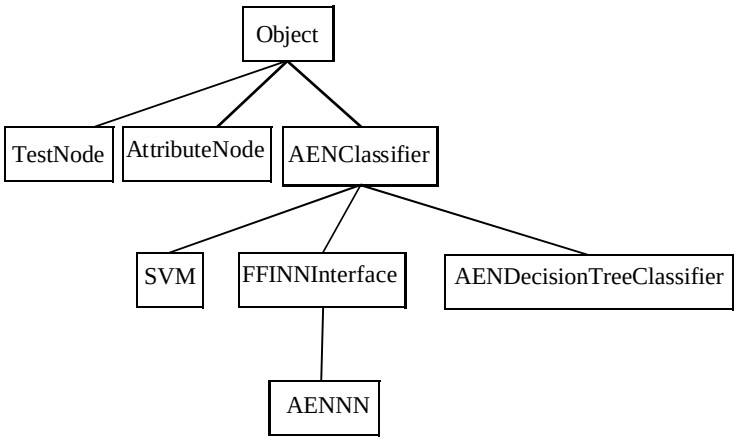
Det er gjort et forsøk på å dokumentere programkoden som implementerer klassifiseringsmodulen vha. kommentarer i koden. For uinnvidde i programmeringsspråket Smalltalk er det vanskelig å få noe ut av dette. Videre vil det bli gitt en (svært kortfattet) forklaring på hvordan modulen fungerer, uten at det må oppfattes som et forsøk på en utførlig dokumentasjon. Det er imidlertid gitt endel generell informasjon i teoridelen (kapittel 2,3 og 4). Programkoden er for orden skyld presentert i Appendiks D. Det krever imidlertid svært mye kunnskap om Smalltalk og algoritmene for å få noe ut av den. Det er litt av poenget at strukturen i menyene i utviklingsverktøyet skal hjelpe brukeren til å forstå og strukturere programkoden. Derfor er det egentlig best å laste programmet inn i utviklingsverktøyet for å skjønne hvordan det er strukturert og hvordan det fungerer.

I Smalltalk er klassene gruppert på kategorier for å gjøre koden mer oversiktlig. Figur A.1 viser hvordan dette er gjort for klassifiseringsmodulen.




*Figur A.1 Figuren viser hvordan klassene som undertegnede kan forfattet, er fordelt på kategorier.*

Figur A.2 viser klassestrukturen eller arverekkefølgen. Klassen Object er den klassen som ligger høyest i hierarkiet i Smalltalk. Alle klasser arver fra denne klassen.




*Figur A.2 Figuren viser hvordan klassene i figur A.1 inngår i klassehierarkiet.*

Klassen FFINNInterface er en abstrakt klasse for det nevrale nettverket som er implementert av ansatte ved FFI. AENClassifier er også en abstrakt klasse. Det er instanser av klassene TestNode og AttributeNode som benyttes til å bygge opp treet.

Instanser av klassene SVM, AENNN og AENDecisionTreeClassifier lyder kommandoene *learn* og *classify*. Datafilen for å lære opp klassifikatoren består av linjer med ett opplæringstilfelle på hver linje. Alle elementer i linjen er tabulatordelt med attributtene først og klassen til slutt. Det samme gjelder filen med testdata foruten at klassen ikke er inkludert. Datastrukturen må settes i initialiseringen til klassene. Dette gjelder hvor mange linjer test- og opplæringsfilen består av og hvilken type attributter som inngår. Diskrete attributter må gjøres til numeriske verdier før klassifisering med det nevrale nettverket og med SVM. Beslutningstreet behandler kontinuerlige variabler på en spesiell måte. Derfor er det viktig at modulen vet hvilke attributter som er kontinuerlige og hvilke som er diskrete. Dette gjøres ved at attributtene som er kontinuerlig gis typen *Cont* mens de andre kalles noe annet.

## A.2 Informasjon om SVM-modulen

Det som er spesielt med denne modulen er at den benytter støtte fra Matlab for å løse et kvadratisk optimaliseringsproblem. Kommunikasjon med Matlab foregår gjennom en fil som består av én bit. Når dette bittet er satt kan Matlab-filen *qpsvm.m* lese inn parametre fra tekst filene *H.dat* og *Aeq.dat* og utføre beregningen. Resultatet returneres til *alpha.txt* og kontrollbittet resettes. I SVM-modulen går prosessen i løkke mens den venter på at kontrollbittet skal "resettes" slik at *alpha.txt* kan leses og beregningen fortsettes. Neste gang prosessen ankommer stedet før den må vente på beregningen av alpha settes kontrollbittet slik at sykelen sluttes. C-parameteren må settes både i Matlab og SVM-modulen. Brukeren av SVM-modulen bør vite endel om hvordan parameterne til optimaliseringsrutinen i Matlab bør settes. Resultatet av kjøringen i Matlab bør også overvåkes. Det er klare begrensninger på størrelsen på problemene som denne rutinen (quadprog.m) kan løse. Typen kjerne som benyttes settes i initialiseringen.

## A.3 Informasjon om AENNN-modulen

Denne modulen har tre metoder som må tilpasses det gitte problemet. Det er beskrevet i den abstrakte klassen FFINNInterface hvordan funksjonene *postprocess*, *postprocessInverted:* og *preprocess:* skal implementeres. Ellers så må antall læringssykluser (antall ganger det kjøres gjennom opplæringstilfellene) settes i initialiseringen.

## A.4 Informasjon om AENDecisionTreeClassifier-modulen

Denne modulen implementerer noen viktige aspekter ved C4.5. De deler av C4.5 som er implementert, er de deler som omhandler generering, forenkling og utskrift av beslutningstreet og klassifisering av testtilfeller. I tillegg er det skrevet programkode inspirert av C4.5 som takler kontinuerlige variable.

# B

## B.1 Beslutningstre for hele datasettet i Car Evaluation Database

```
safety = low: unacc (576/0)
safety = med
        persons = 2: unacc (192/0)
        persons = 4
                buying = vhigh
                        maint = vhigh: unacc (12/0)
                        maint = high: unacc (12/0)
                        maint = med
                                lug_boot = small: unacc (4/0)
                                lug_boot = med
                                        doors = 2: unacc (1/0)
                                        doors = 3: unacc (1/0)
                                        doors = 4: acc (1/0)
                                        doors = 5more: acc (1/0)
                                lug_boot = big: acc (4/0)
                        maint = low
                                lug_boot = small: unacc (4/0)
                                lug_boot = med
                                        doors = 2: unacc (1/0)
                                        doors = 3: unacc (1/0)
                                        doors = 4: acc (1/0)
                                        doors = 5more: acc (1/0)
                                lug_boot = big: acc (4/0)
                buying = high
                        lug_boot = small: unacc (16/0)
                        lug_boot = med
                                doors = 2: unacc (4/0)
                                doors = 3: unacc (4/0)
                                doors = 4
                                        maint = vhigh: unacc (1/0)
                                        maint = high: acc (1/0)
                                        maint = med: acc (1/0)
                                        maint = low: acc (1/0)
                                doors = 5more
                                        maint = vhigh: unacc (1/0)
                                        maint = high: acc (1/0)
                                        maint = med: acc (1/0)
                                        maint = low: acc (1/0)
                        lug_boot = big
                                maint = vhigh: unacc (4/0)
                                maint = high: acc (4/0)
                                maint = med: acc (4/0)
                                maint = low: acc (4/0)
                buying = med
                        maint = vhigh
                                lug_boot = small: unacc (4/0)
                                lug_boot = med
                                        doors = 2: unacc (1/0)
                                        doors = 3: unacc (1/0)
                                        doors = 4: acc (1/0)
                                        doors = 5more: acc (1/0)
                                lug_boot = big: acc (4/0)
```

*Figur B.1 Beslutningstre del 1 av 8.*

```
                maint = high
                    lug_boot = small: unacc (4/0)
                    lug_boot = med
                        doors = 2: unacc (1/0)
                        doors = 3: unacc (1/0)
                        doors = 4: acc (1/0)
                        doors = 5more: acc (1/0)
                    lug_boot = big: acc (4/0)
                maint = med: acc (12/0)
                maint = low
                    lug_boot = small: acc (4/0)
                    lug_boot = med
                        doors = 2: acc (1/0)
                        doors = 3: acc (1/0)
                        doors = 4: good (1/0)
                        doors = 5more: good (1/0)
                    lug_boot = big: good (4/0)
            buying = low
                maint = vhigh
                    lug_boot = small: unacc (4/0)
                    lug_boot = med
                        doors = 2: unacc (1/0)
                        doors = 3: unacc (1/0)
                        doors = 4: acc (1/0)
                        doors = 5more: acc (1/0)
                    lug_boot = big: acc (4/0)
                maint = high: acc (12/0)
                maint = med
                    lug_boot = small: acc (4/0)
                    lug_boot = med
                        doors = 2: acc (1/0)
                        doors = 3: acc (1/0)
                        doors = 4: good (1/0)
                        doors = 5more: good (1/0)
                    lug_boot = big: good (4/0)
                maint = low
                    lug_boot = small: acc (4/0)
                    lug_boot = med
                        doors = 2: acc (1/0)
                        doors = 3: acc (1/0)
                        doors = 4: good (1/0)
                        doors = 5more: good (1/0)
                    lug_boot = big: good (4/0)
    persons = more
            lug_boot = small
                buying = vhigh: unacc (16/0)
                buying = high: unacc (16/0)
                buying = med
                    maint = vhigh: unacc (4/0)
                    maint = high: unacc (4/0)
                    maint = med
                        doors = 2: unacc (1/0)
                        doors = 3: acc (1/0)
                        doors = 4: acc (1/0)
                        doors = 5more: acc (1/0)
```

*Figur B.2 Beslutningstre del 2 av 8.*

```
                                maint = low
                                        doors = 2: unacc (1/0)
                                        doors = 3: acc (1/0)
                                        doors = 4: acc (1/0)
                                        doors = 5more: acc (1/0)
                        buying = low
                                maint = vhigh: unacc (4/0)
                                maint = high
                                        doors = 2: unacc (1/0)
                                        doors = 3: acc (1/0)
                                        doors = 4: acc (1/0)
                                        doors = 5more: acc (1/0)
                                maint = med
                                        doors = 2: unacc (1/0)
                                        doors = 3: acc (1/0)
                                        doors = 4: acc (1/0)
                                        doors = 5more: acc (1/0)
                                maint = low
                                        doors = 2: unacc (1/0)
                                        doors = 3: acc (1/0)
                                        doors = 4: acc (1/0)
                                        doors = 5more: acc (1/0)
        lug_boot = med
                        buying = vhigh
                                maint = vhigh: unacc (4/0)
                                maint = high: unacc (4/0)
                                maint = med
                                        doors = 2: unacc (1/0)
                                        doors = 3: acc (1/0)
                                        doors = 4: acc (1/0)
                                        doors = 5more: acc (1/0)
                                maint = low
                                        doors = 2: unacc (1/0)
                                        doors = 3: acc (1/0)
                                        doors = 4: acc (1/0)
                                        doors = 5more: acc (1/0)
                        buying = high
                                maint = vhigh: unacc (4/0)
                                maint = high
                                        doors = 2: unacc (1/0)
                                        doors = 3: acc (1/0)
                                        doors = 4: acc (1/0)
                                        doors = 5more: acc (1/0)
                                maint = med
                                        doors = 2: unacc (1/0)
                                        doors = 3: acc (1/0)
                                        doors = 4: acc (1/0)
                                        doors = 5more: acc (1/0)
                                maint = low
                                        doors = 2: unacc (1/0)
                                        doors = 3: acc (1/0)
                                        doors = 4: acc (1/0)
                                        doors = 5more: acc (1/0)
```

*Figur B.3 Beslutningstre del 3 av 8.*

```
                buying = med
                        maint = vhigh
                                doors = 2: unacc (1/0)
                                doors = 3: acc (1/0)
                                doors = 4: acc (1/0)
                                doors = 5more: acc (1/0)
                        maint = high
                                doors = 2: unacc (1/0)
                                doors = 3: acc (1/0)
                                doors = 4: acc (1/0)
                                doors = 5more: acc (1/0)
                        maint = med: acc (4/0)
                        maint = low
                                doors = 2: acc (1/0)
                                doors = 3: good (1/0)
                                doors = 4: good (1/0)
                                doors = 5more: good (1/0)
                buying = low
                        maint = vhigh
                                doors = 2: unacc (1/0)
                                doors = 3: acc (1/0)
                                doors = 4: acc (1/0)
                                doors = 5more: acc (1/0)
                        maint = high: acc (4/0)
                        maint = med
                                doors = 2: acc (1/0)
                                doors = 3: good (1/0)
                                doors = 4: good (1/0)
                                doors = 5more: good (1/0)
                        maint = low
                                doors = 2: acc (1/0)
                                doors = 3: good (1/0)
                                doors = 4: good (1/0)
                                doors = 5more: good (1/0)
lug_boot = big
        buying = vhigh
                maint = vhigh: unacc (4/0)
                maint = high: unacc (4/0)
                maint = med: acc (4/0)
                maint = low: acc (4/0)
        buying = high
                maint = vhigh: unacc (4/0)
                maint = high: acc (4/0)
                maint = med: acc (4/0)
                maint = low: acc (4/0)
        buying = med
                maint = vhigh: acc (4/0)
                maint = high: acc (4/0)
                maint = med: acc (4/0)
                maint = low: good (4/0)
        buying = low
                maint = vhigh: acc (4/0)
                maint = high: acc (4/0)
                maint = med: good (4/0)
                maint = low: good (4/0)
```

*Figur B.4 Beslutningstre del 4 av 8.*

```
safety = high
        persons = 2: unacc (192/0)
        persons = 4
                buying = vhigh
                        maint = vhigh: unacc (12/0)
                        maint = high: unacc (12/0)
                        maint = med: acc (12/0)
                        maint = low: acc (12/0)
                buying = high
                        maint = vhigh: unacc (12/0)
                        maint = high: acc (12/0)
                        maint = med: acc (12/0)
                        maint = low: acc (12/0)
                buying = med
                        maint = vhigh: acc (12/0)
                        maint = high: acc (12/0)
                        maint = med
                                lug_boot = small: acc (4/0)
                                lug_boot = med
                                        doors = 2: acc (1/0)
                                        doors = 3: acc (1/0)
                                        doors = 4: vgood (1/0)
                                        doors = 5more: vgood (1/0)
                                lug_boot = big: vgood (4/0)
                        maint = low
                                lug_boot = small: good (4/0)
                                lug_boot = med
                                        doors = 2: good (1/0)
                                        doors = 3: good (1/0)
                                        doors = 4: vgood (1/0)
                                        doors = 5more: vgood (1/0)
                                lug_boot = big: vgood (4/0)
                buying = low
                        maint = vhigh: acc (12/0)
                        maint = high
                                lug_boot = small: acc (4/0)
                                lug_boot = med
                                        doors = 2: acc (1/0)
                                        doors = 3: acc (1/0)
                                        doors = 4: vgood (1/0)
                                        doors = 5more: vgood (1/0)
                                lug_boot = big: vgood (4/0)
                        maint = med
                                lug_boot = small: good (4/0)
                                lug_boot = med
                                        doors = 2: good (1/0)
                                        doors = 3: good (1/0)
                                        doors = 4: vgood (1/0)
                                        doors = 5more: vgood (1/0)
                                lug_boot = big: vgood (4/0)
```

*Figur B.5 Beslutningstre del 5 av 8.*

```
                maint = low
                        lug_boot = small: good (4/0)
                        lug_boot = med
                                doors = 2: good (1/0)
                                doors = 3: good (1/0)
                                doors = 4: vgood (1/0)
                                doors = 5more: vgood (1/0)
                        lug_boot = big: vgood (4/0)
persons = more
        buying = vhigh
                maint = vhigh: unacc (12/0)
                maint = high: unacc (12/0)
                maint = med
                        doors = 2
                                lug_boot = small: unacc (1/0)
                                lug_boot = med: acc (1/0)
                                lug_boot = big: acc (1/0)
                        doors = 3: acc (3/0)
                        doors = 4: acc (3/0)
                        doors = 5more: acc (3/0)
                maint = low
                        doors = 2
                                lug_boot = small: unacc (1/0)
                                lug_boot = med: acc (1/0)
                                lug_boot = big: acc (1/0)
                        doors = 3: acc (3/0)
                        doors = 4: acc (3/0)
                        doors = 5more: acc (3/0)
        buying = high
                maint = vhigh: unacc (12/0)
                maint = high
                        doors = 2
                                lug_boot = small: unacc (1/0)
                                lug_boot = med: acc (1/0)
                                lug_boot = big: acc (1/0)
                        doors = 3: acc (3/0)
                        doors = 4: acc (3/0)
                        doors = 5more: acc (3/0)
                maint = med
                        doors = 2
                                lug_boot = small: unacc (1/0)
                                lug_boot = med: acc (1/0)
                                lug_boot = big: acc (1/0)
                        doors = 3: acc (3/0)
                        doors = 4: acc (3/0)
                        doors = 5more: acc (3/0)
                maint = low
                        doors = 2
                                lug_boot = small: unacc (1/0)
                                lug_boot = med: acc (1/0)
                                lug_boot = big: acc (1/0)
                        doors = 3: acc (3/0)
                        doors = 4: acc (3/0)
                        doors = 5more: acc (3/0)
```

*Figur B.6 Beslutningstre del 6 av 8.*

```
buying = med
        maint = vhigh
                doors = 2
                        lug_boot = small: unacc (1/0)
                        lug_boot = med: acc (1/0)
                        lug_boot = big: acc (1/0)
                doors = 3: acc (3/0)
                doors = 4: acc (3/0)
                doors = 5more: acc (3/0)
        maint = high
                doors = 2
                        lug_boot = small: unacc (1/0)
                        lug_boot = med: acc (1/0)
                        lug_boot = big: acc (1/0)
                doors = 3: acc (3/0)
                doors = 4: acc (3/0)
                doors = 5more: acc (3/0)
        maint = med
                lug_boot = small
                        doors = 2: unacc (1/0)
                        doors = 3: acc (1/0)
                        doors = 4: acc (1/0)
                        doors = 5more: acc (1/0)
                lug_boot = med
                        doors = 2: acc (1/0)
                        doors = 3: vgood (1/0)
                        doors = 4: vgood (1/0)
                        doors = 5more: vgood (1/0)
                lug_boot = big: vgood (4/0)
        maint = low
                lug_boot = small
                        doors = 2: unacc (1/0)
                        doors = 3: good (1/0)
                        doors = 4: good (1/0)
                        doors = 5more: good (1/0)
                lug_boot = med
                        doors = 2: good (1/0)
                        doors = 3: vgood (1/0)
                        doors = 4: vgood (1/0)
                        doors = 5more: vgood (1/0)
                lug_boot = big: vgood (4/0)
buying = low
        maint = vhigh
                doors = 2
                        lug_boot = small: unacc (1/0)
                        lug_boot = med: acc (1/0)
                        lug_boot = big: acc (1/0)
                doors = 3: acc (3/0)
                doors = 4: acc (3/0)
                doors = 5more: acc (3/0)
```

*Figur B.7 Beslutningstre del 7 av 8.*

```
maint = high
        lug_boot = small
                doors = 2: unacc (1/0)
                doors = 3: acc (1/0)
                doors = 4: acc (1/0)
                doors = 5more: acc (1/0)
        lug_boot = med
                doors = 2: acc (1/0)
                doors = 3: vgood (1/0)
                doors = 4: vgood (1/0)
                doors = 5more: vgood (1/0)
        lug_boot = big: vgood (4/0)
maint = med
        lug_boot = small
                doors = 2: unacc (1/0)
                doors = 3: good (1/0)
                doors = 4: good (1/0)
                doors = 5more: good (1/0)
        lug_boot = med
                doors = 2: good (1/0)
                doors = 3: vgood (1/0)
                doors = 4: vgood (1/0)
                doors = 5more: vgood (1/0)
        lug_boot = big: vgood (4/0)
maint = low
        lug_boot = small
                doors = 2: unacc (1/0)
                doors = 3: good (1/0)
                doors = 4: good (1/0)
                doors = 5more: good (1/0)
        lug_boot = med
                doors = 2: good (1/0)
                doors = 3: vgood (1/0)
                doors = 4: vgood (1/0)
                doors = 5more: vgood (1/0)
        lug_boot = big: vgood (4/0)
```

*Figur B.8 Beslutningstre del 8 av 8.*

# C

## C.1 Relevant informasjon om Wine Database

1. Title of Database: Wine recognition data
Updated Sept 21, 1998 by C.Blake : Added attribute information

2. Sources:
(a) Forina, M. et al, PARVUS - An Extendible Package for Data
Exploration, Classification and Correlation. Institute of Pharmaceutical
and Food Analysis and Technologies, Via Brigata Salerno,
16147 Genoa, Italy.

(b) Stefan Aeberhard, email: stefan@coral.cs.jcu.edu.au
(c) July 1991
3. Past Usage:

(1)
S. Aeberhard, D. Coomans and O. de Vel,
Comparison of Classifiers in High Dimensional Settings,
Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
Mathematics and Statistics, James Cook University of North Queensland.
(Also submitted to Technometrics).

The data was used with many others for comparing various
classifiers. The classes are separable, though only RDA
has achieved 100% correct classification.
(RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
(All results using the leave-one-out technique)

In a classification context, this is a well posed problem
with "well behaved" class structures. A good data set
for first testing of a new classifier, but not very
challenging.

(2)
S. Aeberhard, D. Coomans and O. de Vel,
"THE CLASSIFICATION PERFORMANCE OF RDA"
Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
Mathematics and Statistics, James Cook University of North Queensland.
(Also submitted to Journal of Chemometrics).

Here, the data was used to illustrate the superior performance of
the use of a new appreciation function with RDA.

4. Relevant Information:

-- These data are the results of a chemical analysis of
wines grown in the same region in Italy but derived from three
different cultivars.
The analysis determined the quantities of 13 constituents
found in each of the three types of wines.

-- I think that the initial data set had around 30 variables, but
for some reason I only have the 13 dimensional version.
I had a list of what the 30 or so variables were, but a.)
I lost it, and b.), I would not know which 13 variables
are included in the set.

-- The attributes are (dontated by Riccardo Leardi,
riclea@anchem.unige.it )
1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10)Color intensity
11)Hue
12)OD280/OD315 of diluted wines
13)Proline

5. Number of Instances

class 1 59
class 2 71
class 3 48

6. Number of Attributes

13

7. For Each Attribute:

All attributes are continuous

No statistics available, but suggest to standardise
variables for certain uses (e.g. for us with classifiers
which are NOT scale invariant)

NOTE: 1st attribute is class identifier (1-3)

8. Missing Attribute Values:

None

9. Class Distribution: number of instances per class

class 1 59
class 2 71
class 3 48

Kilde: (10)

## C.2 Relevant informasjon om Car Evaluation Database

1. Title: Car Evaluation Database

2. Sources:
(a) Creator: Marko Bohanec
(b) Donors: Marko Bohanec (marko.bohanec@ijs.si)
Blaz Zupan (blaz.zupan@ijs.si)
(c) Date: June, 1997

3. Past Usage:

The hierarchical decision model, from which this dataset is derived, was first presented in

M. Bohanec and V. Rajkovic: Knowledge acquisition and explanation for multi-attribute decision making. In 8th Intl Workshop on Expert Systems and their Applications, Avignon, France. pages 59-78, 1988.

Within machine-learning, this dataset was used for the evaluation of HINT (Hierarchy INduction Tool), which was proved to be able to completely reconstruct the original hierarchical model. This, together with a comparison with C4.5, is presented in

B. Zupan, M. Bohanec, I. Bratko, J. Demsar: Machine learning by function decomposition. ICML-97, Nashville, TN. 1997 (to appear)

4. Relevant Information Paragraph:

Car Evaluation Database was derived from a simple hierarchical decision model originally developed for the demonstration of DEX (M. Bohanec, V. Rajkovic: Expert system for decision making. Sistemica 1(1), pp. 145-157, 1990.). The model evaluates cars according to the following concept structure:

| | |
|---|---|
| CAR | car acceptability |
| . PRICE | overall price |
| . . buying | buying price |
| . . maint | price of the maintenance |
| . TECH | technical characteristics |
| . . COMFORT | comfort |
| . . . doors | number of doors |
| . . . persons | capacity in terms of persons to carry |
| . . . lug_boot | the size of luggage boot |
| . . safety | estimated safety of the car |

Input attributes are printed in lowercase. Besides the target concept (CAR), the model includes three intermediate concepts: PRICE, TECH, COMFORT. Every concept is in the original model related to its lower level descendants by a set of examples (for these examples sets see http://www-ai.ijs.si/BlazZupan/car.html).

The Car Evaluation Database contains examples with the structural information removed, i.e., directly relates CAR to the six input attributes: buying, maint, doors, persons, lug_boot, safety.

Because of known underlying concept structure, this database may be particularly useful for testing constructive induction and structure discovery methods.

5. Number of Instances: 1728
(instances completely cover the attribute space)

6. Number of Attributes: 6

7. Attribute Values:

| | |
|---|---|
| buying | v-high, high, med, low |
| maint | v-high, high, med, low |
| doors | 2, 3, 4, 5-more |
| persons | 2, 4, more |
| lug_boot | small, med, big |
| safety | low, med, high |

8. Missing Attribute Values: none

9. Class Distribution (number of instances per class)

| class | N | N[%] |
|---|---|---|
| unacc | 1210 | (70.023 %) |
| acc | 384 | (22.222 %) |
| good | 69 | ( 3.993 %) |
| v-good | 65 | ( 3.762 %) |

Kilde: (10).

# D

## D.1 Programkode i Smalltalk

```
Object subclass: #AENClassifier
          instanceVariableNames: 'elementTypesCollection
namesOfElements mainElementValueCollection
numberOfLinesTrainingData numberOfLinesTestData
numberOfElementsEachLine trainingDataFile testDataFile
mainCollection classificationOutputFile '
          classVariableNames: ''
          poolDictionaries: ''
          category: 'AEN-Abstract-Classifier'!


!AENClassifier methodsFor: 'other'!

indexOfElement:anElement in: aCollection

"Finner plasseringen til et element i en samling."


| t item numberOfElements |


t:=1.
item:=aCollection at:1.
numberOfElements:=aCollection size.

 [(item = anElement) or: [ t=numberOfElements]]
whileFalse: [ t:=t+1.item:=aCollection at:t. ].

^t!


indexOfMaxElementIn: aCollection

"Finner plasseringen til elementet med stoerst verdi i en
samling."

| t max maxIndex |

t:=1.
max:=aCollection at:1.
maxIndex:=1.
((aCollection size)-1) timesRepeat:
[t:=t+1.(max<(aCollection at:t))  ifTrue:[maxIndex:=t.
max:=aCollection at:t.] ].
^maxIndex!



readDataFromFileNumberOfElementsEachLine:
aNumberOfElementsEachLine numberOfLines: aNumberOfLines
datafile:aDatafile


          "Datafilen  bestaar av  linjer. Hver linje angir
ett trenings- eller testsampel.Elementene skilles med tab.
Antall datasampler  (numberOfLines), antall
attributter+klasse (numberOfElementsEachLine) og datafil
(dataFile) mÅsettes  i initialize"

          | tempMainCollection stream startCopy stoppCopy
aLine allLines readingBlock separator1 separator2 dataFile
|

          dataFile:= aDatafile asString asFilename.


          tempMainCollection:=OrderedCollection new.
          stream := dataFile readStream.
          separator1 := Character tab.
          separator2 := Character cr.

          startCopy:=1.
          stoppCopy:=aNumberOfElementsEachLine.







aLine:=OrderedCollection new.
          allLines := OrderedCollection new.

          readingBlock := [



[stream atEnd] whileFalse: [


          (aNumberOfElementsEachLine-1) timesRepeat:[
                    allLines add: (stream upTo:
separator1)].
                    allLines add: (stream upTo:
separator2).

          ]].
          readingBlock valueNowOrOnUnwindDo: [stream
close].

          aNumberOfLines timesRepeat:
          [aLine:=allLines copyFrom:startCopy to:stoppCopy
.
          startCopy:=startCopy+aNumberOfElementsEachLine.

stoppCopy:=stoppCopy+aNumberOfElementsEachLine.
tempMainCollection add:aLine].

^tempMainCollection!

removingElementFromCollection: aCollection elementIndex:
anElementIndex



| copy t |

copy:=OrderedCollection new.

t:=0.
aCollection size timesRepeat:[
t:=t+1.
(t=anElementIndex) ifFalse:[copy add:(aCollection at:t)]].



^copy!

returnNumericCollection:aCollection

|  k1 k2 indexInValueCollection |

          k1:=1.
          (((aCollection at:1) size)) timesRepeat:[
                    ((elementTypesCollection at:
k1)~='Cont') ifTrue:[
                              k2:=1.
                              aCollection size
timesRepeat:[

          indexInValueCollection:=self
indexOfElement:((aCollection at:k2) at:k1) in:
          (mainElementValueCollection at:k1).

                                        (aCollection at:k2)
at:k1 put:(indexInValueCollection).
                                        k2:=k2+1.
                              ].

                    ].



                    k1:=k1+1.
          ].


^aCollection!

returnXCollection:aCollection

| k1 tempXCollection xLength |

          xLength:=(aCollection at:1) size.
          tempXCollection:=OrderedCollection new.
          k1:=1.
          [k1 <= (aCollection size)] whileTrue: [

                    tempXCollection add:((aCollection
at:k1) copyFrom:1 to: (xLength-1) ).

          k1:=k1+1
].

^tempXCollection!


returnYCollection:aCollection

| k1 tempYCollection |
          tempYCollection:=OrderedCollection new.
          k1:=1.
          [k1 <= (aCollection size)] whileTrue: [

                    tempYCollection add:(aCollection at:k1)
last.

          k1:=k1+1
```

```
].

^tempYCollection! !


!AENClassifier methodsFor: 'subclassreponsibility'!

classify

^self subclassResponsibility!

learn

^self subclassResponsibility! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

AENClassifier class
          instanceVariableNames: ''!

!AENClassifier class methodsFor: 'new'!

new

^super new initialize! !



'From VisualWorksÂ(r), Release 3.0 of February 5, 1998 on
January 26, 2001 at 10:46:43 am'!

Object subclass: #TestNode
          instanceVariableNames: 'name nextNode '
          classVariableNames: 'ClassVarName1 ClassVarName2
'
          poolDictionaries: ''
          category: 'AEN-Tree-Classifier'!
TestNode comment:
'TestNode beskriver noder som benyttes til å angi tester på
attributtets verdi. Benytter en node for hver test som
legges i en samling inne i attributtnoden.


Instance Variables:

name      <String>  Attributtverdi
nextNode  <ClassOfVariable>   Referanse til en
Attributtnode.

Class Variables:

ClassVarName1       <ClassOfVariable>   description of
variable''''s function
ClassVarName2       <ClassOfVariable>   description of
variable''''s function'!


!TestNode methodsFor: 'initialize-release'!

initialize! !

!TestNode methodsFor: 'accessing'!

name

^name!

name:aName

name:=aName.!

nextNode

^nextNode!

nextNode: aNode

nextNode:=aNode.! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

TestNode class
          instanceVariableNames: ''!


!TestNode class methodsFor: 'ny'!

new

^super new initialize! !
1


'From VisualWorksÂ(r), Release 3.0 of February 5, 1998 on
January 26, 2001 at 10:46:38 am'!

Object subclass: #AttributeNode
          instanceVariableNames: 'nextNodeCollection name
number numberWrongClass classFrequencyCollection '
          classVariableNames: ''
          poolDictionaries: ''
          category: 'AEN-Tree-Classifier'!
AttributeNode comment:
'Dette er en klasse som beskriver en node som brukes til Å
bygge et besluttningstre. Noden representerer en attributt.

Instance Variables:

nextNodeCollection  <>        Her legges det inn en samling
av TestNoder eller en streng som sier at
                                                    noden er
en løvnode.

name      <String>  Attributtets navn eller klassen hvis
løvnode.
number    <Number>  Antall sampler som passerte denne noden
da treet ble skapt.
numberWrongClass Antall feil klassifiserte sampler. Er uten
betydning dersom noden ikke er en
                                lØvnode.

Class Variables:

ClassVarName1       <ClassOfVariable>   description of
variable''''s function
ClassVarName2       <ClassOfVariable>   description of
variable''''s function'!


!AttributeNode methodsFor: 'initialize-release'!

initialize
number:=0.
numberWrongClass:=0.! !

!AttributeNode methodsFor: 'accessing'!

classFrequencyCollection


^classFrequencyCollection!

classFrequencyCollection:aCollection


classFrequencyCollection:=aCollection.!

name


^name!

name:aName


name:=aName.!

nextNodeCollection

^nextNodeCollection!

nextNodeCollection: aCollectionTestNodes

nextNodeCollection:=aCollectionTestNodes.!

number

^number!

number:aNumber

number:=aNumber.!

numberWrongClass

^numberWrongClass!

numberWrongClass:aNumber

numberWrongClass:=aNumber.! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

AttributeNode class
          instanceVariableNames: ''!


!AttributeNode class methodsFor: 'ny'!

new

^super new initialize! !
1


'From VisualWorksÂ(r), Release 3.0 of February 5, 1998 on
January 26, 2001 at 10:45:31 am'!

AENClassifier subclass: #AENDecisionTreeClassifier
          instanceVariableNames: 'learningDataCollection
rootOfTree maxLevelContSplit rootNodeCollection '
          classVariableNames: ''
          poolDictionaries: ''
          category: 'AEN-Tree-Classifier'!
AENDecisionTreeClassifier comment:
'Objekter fra denne klassen kan benyttes til å generere og
skrive ut beslutningstrær. Samt å klassifisere data på
grunnlag av et besluttningstre.

Instance Variables:

anAENMainCollection <ClassOfVariable>   description of
variable''s function
mainElementValueCollection    <ClassOfVariable>
          description of variable''s function
elementTypesCollection        <ClassOfVariable>
          description of variable''s function
rootOfTree          <ClassOfVariable>   description of
variable''s function
namesOfElements     <ClassOfVariable>   description of
variable''s function'!


!AENDecisionTreeClassifier methodsFor: 'initialisering'!
```

```
initialize

"
Documentation

The datastructure is given by elementTypesCollection,
namesOfElements and mainElementValueCollection. The
folowing two examples show how to set these variables.

Example 1
elementTypesCollection:=#('Cont' 'Cont' 'Cont' 'Cont'
'Cont' 'Cont' 'Cont' 'Cont' 'Cont' 'Cont' 'Cont' 'Cont'
'Cont' 'Byorder').
namesOfElements:=#('Alcohol' 'MalicAcid' 'Ash'
'AlcalinityOfAsh' 'Magnesium' 'TotalPhenols' 'Flavanoids'
'NonflavanoidPhenols' 'Proanthocyanins' 'ColorIntensity'
'Hue' 'OD280/OD315OfDilutedWines' 'Proline'  'class').
mainElementValueCollection:=#( #() #() #() #() #()  #() #()
#() #()  #() #() #() #() #('1' '2' '3') ) .

Example 2

elementTypesCollection:=#('ByOrder' 'ByOrder' 'ByOrder'
'ByOrder' 'ByOrder' 'ByOrder' 'ByOrder').
namesOfElements:=#('buying' 'maint' 'doors' 'persons'
'lug_boot' 'safety' 'class').
mainElementValueCollection:=#( #('low' 'med' 'high'
'vhigh') #('low' 'med' 'high' 'vhigh') #( '2' '3' '4'
'5more'  ) #( '2' '4' 'more' )  #('small' 'med' 'big') #(
'low' 'med' 'high' ) #('unacc' 'acc' 'good' 'vgood') )



The maximum number of separate intervalls the continuous
attributes can be partitioned in is given by:
(maxLevelContSplit-1)^2.
Example 3

This means          that the continuous attributes can be
discretized in 4 regions:
maxLevelContSplit:=3.

Information about the size of the training and test files
must be given.

Example 4
numberOfLinesTrainingData :=89.
numberOfLinesTestData:=89.
numberOfElementsEachLine := namesOfElements size.
trainingDataFile:= 'd:\vw30\treningssettWine.txt'.
treeClassificationOutputFile:='d:\vw30\klassefiseringsres.t
xt'.
testDataFile:='d:\vw30\testsettWine.txt'.



Initialization of the tree

rootOfTree:=TestNode new.
rootOfTree name:'root'.

"! !

!AENDecisionTreeClassifier methodsFor: 'evaluation of
Information'!

classInfoOf:aFrequenzyOfClassesCollection
sizeOfSubCollection: aSize

"Beregner informasjonsverdien av en samling av sampler
fordelt pÅ et attributts verdier (splitinfo) eller en
samling av sampler  fordelt på klasser."

|info t|

(aSize=0) ifTrue:[info:=0. "Her unngÅs divisjon med null"
]
ifFalse:[
t:=0.
info:=0.
aFrequenzyOfClassesCollection size timesRepeat: [t:=t+1.
info:=info-(((aFrequenzyOfClassesCollection at: t)/aSize) *
(self myLogFunction:(   (aFrequenzyOfClassesCollection at:
t)/aSize) )) . ].
].

^info!

distributionOfClassesOnAttribute: anAENMainSubCollection
elementNumber: anElementNumber
elementValueCollection:elementValueCollection

"Genererer en samling med frekvensen til elementene i
datasubsamlingen
fordelt på  attributt- og klasse-verdiene."

|numberOfElements n t element frequenzyOfElementsCollection
elementCollection numberOfClasses classCollection class s
frequency |

elementCollection:=elementValueCollection at:
anElementNumber.
classCollection:=elementValueCollection last.

"Initierer frequenzyOfElementsCollection."

numberOfElements:=elementCollection  size.
frequenzyOfElementsCollection:=OrderedCollection new.
numberOfClasses:=(elementValueCollection last) size.
numberOfElements timesRepeat:[frequenzyOfElementsCollection
add:OrderedCollection new].
t:=1.
numberOfElements timesRepeat:
[numberOfClasses timesRepeat:
[(frequenzyOfElementsCollection at:t) add:0].t:=t+1.].


"Teller forekomster og inkrementerer de riktige elementene
i frequenzyOfElementsCollection."
n:=0.
anAENMainSubCollection size timesRepeat:[n:=n+1.
          element:= (anAENMainSubCollection at:n) at:
anElementNumber. class:=(anAENMainSubCollection at:n) last.
          t:=self indexOfElement:element
in:elementCollection.
          s:=self indexOfElement:class in: classCollection.
          frequency:=((frequenzyOfElementsCollection at: t)
at:s) +1.(frequenzyOfElementsCollection at: t) at:s
put:frequency   ].

^frequenzyOfElementsCollection!

frequencyOfElementCollection: anAENMainSubCollection
elementNumber: anElementNumber
elementValueCollection:elementValueCollection

"Genererer en samling med frekvensene til verdiene til
elementene i datasubsamlingen for
en attributt eller klasse "

|numberOfElements n t element frequenzyOfElementsCollection
elementCollection  frequency |

elementCollection:=elementValueCollection at:
anElementNumber.

numberOfElements:=elementCollection  size.
frequenzyOfElementsCollection:=OrderedCollection new.


"initialiserer frekvenssamlingen"
numberOfElements timesRepeat:
 [frequenzyOfElementsCollection  add:0. ].



n:=0.
anAENMainSubCollection size timesRepeat:[ n:=n+1.element:=
(anAENMainSubCollection at:n) at: anElementNumber.

"Finner ut posisjonen til  den detekterte verdien til
klassen eller attributtet i elementValueCollection for det
aktuelle attributtet eller klassen"
t:=self indexOfElement:element in:elementCollection.

"Oppdaterer frekvensen"

frequency:=(frequenzyOfElementsCollection at: t) + 1.
frequenzyOfElementsCollection at: t put:frequency   ].


^frequenzyOfElementsCollection!

gainCollection: anAENMainSubCollection
elementValueCollection:elementValueCollection

"Lager en samling med gainverdiene for splitting av
datasamlingen på de forskjellige attributtene."

|  InfoOfAnAENMainSubCollection gainCollection
infoCollection t tempFrequencyCollectionOfCollection
distributionOfClassesOnAttributeCollection numberOfElements
|

tempFrequencyCollectionOfCollection:=OrderedCollection new.
distributionOfClassesOnAttributeCollection:=OrderedCollecti
on new.
numberOfElements:=(anAENMainSubCollection at:1) size.

"Lager en samling av samlinger med frekvensinformasjon over
antall sampler fordelt pÅ de forskjellige attributtverdiene
pÅ hver attributt."
t:=0.
numberOfElements timesRepeat:[t:=t+1.
          tempFrequencyCollectionOfCollection add:(self
frequencyOfElementCollection: anAENMainSubCollection
elementNumber: t
          elementValueCollection:elementValueCollection).].


"Lager en samling av samlinger av samlinger med
frekvensinformasjon over antall sampler fordelt på de
forskjellige klassene på attributtverdiene på hver
attributt. Enkelt hva?!!"
t:=0.
(numberOfElements-1) timesRepeat:[t:=t+1.
          distributionOfClassesOnAttributeCollection
add:(self distributionOfClassesOnAttribute:
anAENMainSubCollection elementNumber: t
          elementValueCollection:elementValueCollection)].


InfoOfAnAENMainSubCollection :=self
classInfoOf:(tempFrequencyCollectionOfCollection last)
sizeOfSubCollection: anAENMainSubCollection size.

infoCollection:=self infoCollectionOfCollection:
anAENMainSubCollection frequencyCollection:
tempFrequencyCollectionOfCollection distribution:
```

```
distributionOfClassesOnAttributeCollection.

gainCollection:=OrderedCollection new.


t:=0.
infoCollection size timesRepeat: [t:=t+1.gainCollection
add:(InfoOfAnAENMainSubCollection -(infoCollection at:t))
].

^gainCollection!

gainRatioCollection: anAENMainSubCollection
elementValueCollection:elementValueCollection

"Lager en samling med gainratioverdiene for splitting av
datasamlingen på de forskjellige attributtene."

|  InfoOfAnAENMainSubCollection gainCollection
infoCollection t tempFrequencyCollectionOfCollection
distributionOfClassesOnAttributeCollection numberOfElements
splitCollection gainRatioCollection |

tempFrequencyCollectionOfCollection:=OrderedCollection new.
distributionOfClassesOnAttributeCollection:=OrderedCollecti
on new.
numberOfElements:=(anAENMainSubCollection at:1) size.

"Lager en samling av samlinger med frekvensinformasjon over
antall sampler fordelt pÅ de forskjellige attributtverdiene
pÅ hver attributt."
t:=0.
numberOfElements timesRepeat:[t:=t+1.
tempFrequencyCollectionOfCollection add:(self
frequencyOfElementCollection: anAENMainSubCollection
elementNumber: t
elementValueCollection:elementValueCollection).].

"Lager en samling av samlinger av samlinger med
frekvensinformasjon over antall sampler fordelt på de
forskjellige klassene pÅ attributtverdiene pÅ hver
attributt."
t:=0.
(numberOfElements-1) timesRepeat:[t:=t+1.
          distributionOfClassesOnAttributeCollection
add:(self distributionOfClassesOnAttribute:
anAENMainSubCollection elementNumber: t
          elementValueCollection:elementValueCollection)].


InfoOfAnAENMainSubCollection :=self
classInfoOf:(tempFrequencyCollectionOfCollection last)
sizeOfSubCollection: anAENMainSubCollection size.

infoCollection:=self infoCollectionOfCollection:
anAENMainSubCollection frequencyCollection:
tempFrequencyCollectionOfCollection distribution:
distributionOfClassesOnAttributeCollection.

gainCollection:=OrderedCollection new.
splitCollection:=OrderedCollection new.
gainRatioCollection:=OrderedCollection new.

t:=0.
infoCollection size timesRepeat: [t:=t+1.gainCollection
add:(InfoOfAnAENMainSubCollection -(infoCollection at:t)).

splitCollection add: (self
classInfoOf:(tempFrequencyCollectionOfCollection at:t)
sizeOfSubCollection:anAENMainSubCollection size ).

((splitCollection at:t)=0) ifTrue:[
          "UnngÅr divisjon med null. "
          gainRatioCollection add:0]
ifFalse:
          [gainRatioCollection add:((gainCollection
at:t)/(splitCollection at:t)) asFloat] ].

^gainRatioCollection!

infoCollectionOfCollection: anAENMainSubCollection
frequencyCollection: aFrequencyCollection distribution:
aDistribution

"Beregner informasjonen ved splitting av
anAENMainSubCollection over alle attributter.
aFrequencyCollection og aDistribution inneholder data  for
all attributter."

| numberOfAttributes infoCollection t |

numberOfAttributes:=((anAENMainSubCollection at:1) size)-1.

infoCollection:= OrderedCollection new.

t:=0.
numberOfAttributes timesRepeat: [t:=t+1. infoCollection
add: [self infoOfCollection: anAENMainSubCollection
frequencyCollection: (aFrequencyCollection at:t)
distribution: (aDistribution at:t) ] value].

^infoCollection!

infoOfCollection: anAENMainSubCollection
frequencyCollection: aFrequencyCollection distribution:
aDistribution

"Beregner informasjonen ved splitting av
anAENMainSubCollection over en attributt.
aFrequencyCollection og aDistribution inneholder data kun
for en attributt."

|t info sizeOfCollection|


sizeOfCollection:=anAENMainSubCollection size.

t:=0.
info:=0.
aFrequencyCollection size timesRepeat: [t:=t+1.
info:=info+(((aFrequencyCollection at:
t)/sizeOfCollection)*(self classInfoOf: (aDistribution
at:t) sizeOfSubCollection: (aFrequencyCollection at: t)))
].

^info! !

!AENDecisionTreeClassifier methodsFor: 'continAttributes'!

classifyContAttr: aName    treeNode: aNode valueOfparent:
aParentName

"Med utgangspunkt i det genererte binÆre treet, som ikke er
det samme som benyttes i den overordnede klassifiseringen,
klassifiseres de kontinuerlige input-dataene. Man kan si at
de diskretiseres basert pÅ en gitt oppøsning etter
informasjons innhold"
| classValue |

(((aNode nextNode name) asNumber)>=(aName asNumber))
ifTrue:[

          (((aNode nextNode nextNodeCollection at:1)
nextNode nextNodeCollection)='LeafNode' ) ifTrue:[
                    ('<' = aNode name) ifTrue:[

          ^classValue:=(aParentName,'<','x','<=',(aNode
nextNode name)).

                    ]
                    ifFalse:[
                              ^classValue:=('x','<=',(aNode
nextNode name)).

                    ].
          ]
          ifFalse:[
          ^classValue:=self classifyContAttr: aName
treeNode: (aNode nextNode nextNodeCollection      at:1)
valueOfparent:      aParentName
          ].
]
ifFalse:[
(((aNode nextNode nextNodeCollection at:2) nextNode
nextNodeCollection) ='LeafNode' ) ifTrue:[
                    ('<'=aNode name) ifTrue:[
                    ^classValue:=('x','>',(aNode nextNode
name)).

                    ]
                    ifFalse:[
                    ^classValue:=((aNode nextNode
                    name),'<','x','<=',aParentName).

                    ].
          ]
          ifFalse:[
          ^classValue:=self classifyContAttr: aName
          treeNode: (aNode nextNode nextNodeCollection
          at:2) valueOfparent:          aParentName
          ].
].


^classValue!

generateTreeForAContAttr: anAENMainSubCollection
elementValueCollection:anElementValueCollection
attributeIndex:anAttrIndex treeLevel:aLevel
maxLevel:aMaxLevel treeNode: aNode

"genererer et binÆrt tree basert pÅ informasjonsverdien av
Å splitte attributtet, og der oppøsningen spesifisert ved
aMaxLevel som angir maksdybden i treet"


|  sizeOfMainCollection t collectionA collectionB
classIndex classFreqencyCollectionA
classFreqencyCollectionB splittFrequencyCollection
distributionCollection info
classFreqencyAENMainSubCollection infoAENMainSubCollection
gain gainCollection maxIndex level maxClassIndex |

level:=aLevel+1.


sizeOfMainCollection:=anAENMainSubCollection size.
classIndex:=(anAENMainSubCollection at:1) size.
OrderedCollection new.
classFreqencyAENMainSubCollection:=self
frequencyOfElementCollection: anAENMainSubCollection
elementNumber: classIndex
elementValueCollection:anElementValueCollection.

infoAENMainSubCollection:=self
classInfoOf:classFreqencyAENMainSubCollection
sizeOfSubCollection: anAENMainSubCollection size.
gainCollection:=OrderedCollection new. "benytter kun gain
(og ikke gainratio) ettersom splittingen skjer innenfor
```

```
attributtet, og ikke mellom attributtene"

t:=0.
sizeOfMainCollection timesRepeat:[t:=t+1.
          "Videre genereres gainverdiene"
          splittFrequencyCollection:=OrderedCollection new.
          splittFrequencyCollection add:t;
add:(sizeOfMainCollection-t).

          collectionA:=anAENMainSubCollection copyFrom:1
to:t.
          classFreqencyCollectionA:=self
frequencyOfElementCollection: collectionA elementNumber:
classIndex
          elementValueCollection:anElementValueCollection.


          collectionB:=anAENMainSubCollection
copyFrom:(t+1) to:sizeOfMainCollection.
          classFreqencyCollectionB:=self
frequencyOfElementCollection: collectionB elementNumber:
classIndex
          elementValueCollection:anElementValueCollection.

          distributionCollection:=OrderedCollection new.
          distributionCollection
add:classFreqencyCollectionA ;
add:classFreqencyCollectionB.

          info:=self infoOfCollection:
anAENMainSubCollection frequencyCollection:
splittFrequencyCollection distribution:
distributionCollection.
          gain:=(infoAENMainSubCollection -info).

          gainCollection add:(gain asFloat).

].


maxIndex:=self indexOfMaxElementIn: gainCollection.

(((gainCollection at:maxIndex)=0)|(level=aMaxLevel))
ifTrue:[
          "Stopper oppsplittingen nÅr enten den
spesifiserte dybden er nÅdd eller gain er null"

          aNode nextNode:AttributeNode new. "Lager en ny
atributtnode"
          aNode nextNode nextNodeCollection:'LeafNode'.
          aNode nextNode number:sizeOfMainCollection.
          maxClassIndex:=self
indexOfMaxElementIn:classFreqencyAENMainSubCollection.
          aNode nextNode
numberWrongClass:(sizeOfMainCollection -
(classFreqencyAENMainSubCollection at:maxClassIndex)).
          aNode nextNode name:(anElementValueCollection
last at:maxClassIndex).
          aNode nextNode
classFrequencyCollection:classFreqencyAENMainSubCollection.

]
ifFalse:
[

          aNode nextNode:AttributeNode new. "Lager en ny
atributtnode"
          aNode nextNode
nextNodeCollection:OrderedCollection new.
          aNode nextNode nextNodeCollection add:TestNode
new;add:TestNode new.
          aNode nextNode number:sizeOfMainCollection.
          aNode nextNode name:((anAENMainSubCollection
at:maxIndex) at:anAttrIndex).
          aNode nextNode
classFrequencyCollection:classFreqencyAENMainSubCollection.
          (aNode nextNode nextNodeCollection at:1)
name:'>'.
          (aNode nextNode nextNodeCollection at:2)
name:'<'.

"rekursive kall"

self generateTreeForAContAttr: (anAENMainSubCollection
copyFrom:1 to:maxIndex)
elementValueCollection:anElementValueCollection
attributeIndex:anAttrIndex treeLevel:level
maxLevel:aMaxLevel treeNode: (aNode nextNode
nextNodeCollection at:1).

self generateTreeForAContAttr: (anAENMainSubCollection
copyFrom:(maxIndex+1) to:sizeOfMainCollection)
elementValueCollection:anElementValueCollection
attributeIndex:anAttrIndex treeLevel:level
maxLevel:aMaxLevel treeNode: (aNode nextNode
nextNodeCollection at:2).
]!

preprosessContAttrInCollection:anAENMainSubCollection
byNumberOfLevels:aMaxLevel
elementValueCollection:anElementValueCollection
elementTypesCollection: anElementTypesCollection

"basert pÅ det genererte binÆre treet oppdateres
inputdatasamlingen og elementverdisamlingen. Dette er
hovedfunksjonen i denne kategorien"

| t rootNode aSortedAENCollection tempCollection |
rootNodeCollection:=Array new:(anElementTypesCollection
size -1).

t:=0.
(anElementTypesCollection size -1) timesRepeat:[t:=t+1.

          ((anElementTypesCollection at:t) = 'Cont' )
ifTrue:[

          rootNode:=TestNode new.


          rootNodeCollection at:t put:rootNode.
          rootNode name: 'root'.

          aSortedAENCollection:=(self
sortingOfAENMainCollection:anAENMainSubCollection
byContAttributeNr:t) asOrderedCollection.

          "genererer det binÆre treet"
          self generateTreeForAContAttr:
aSortedAENCollection
elementValueCollection:anElementValueCollection
attributeIndex:t treeLevel:0  maxLevel:aMaxLevel treeNode:
rootNode.

          "BÅde ny datainputsamling og elementverdisamling
returneres i tempCollection"

          tempCollection:=self
separateContAttrAndGenAttrValues:  anAENMainSubCollection
treeNode: rootNode  attributeIndex:t.


          anElementValueCollection at:t put:(tempCollection
asOrderedCollection).


          ].
].!

preprosessContAttrInTestCollection:anAENMainSubCollection



| t |

t:=0.
(elementTypesCollection size -1) timesRepeat:[t:=t+1.

          ((elementTypesCollection at:t) = 'Cont' )
ifTrue:[


          self separateContAttrAndGenAttrValues:
anAENMainSubCollection    treeNode:     (rootNodeCollection
at:t)     attributeIndex:t.

          ].
].!

separateContAttrAndGenAttrValues: anAENMainSubCollection
treeNode: aNode attributeIndex:anAttrIndex

"med utgangspunkt i et generert binÆrt tree diskretiseres
de kontinuerlige verdiene i datasamlingen for en attributt.
BÅde elementverdisamlingen og inputsamlingen oppdateres og
returneres. For kontinuerlige attributter er
elementverdisamlingen i utgangspunktet tom"

| t classValue anElementInAENMainSubCollection
elementValueSet |
          t:=0.


elementValueSet:=Set new.

anAENMainSubCollection size timesRepeat:[t:=t+1.
          anElementInAENMainSubCollection:=anAENMainSubColl
ection at:t.

          (aNode nextNode nextNodeCollection ='LeafNode' )
ifTrue:[
                    classValue:=('x','=',(aNode nextNode
name)).
                    elementValueSet add:classValue.
                    anElementInAENMainSubCollection
at:anAttrIndex               put:classValue.
          ] ifFalse:[
                    (((aNode nextNode name)
asNumber)>=((anElementInAENMainSubCollection
at:anAttrIndex)asNumber)) ifTrue:[

                              (((aNode nextNode
nextNodeCollection at:1) nextNode
nextNodeCollection)='LeafNode' )        ifTrue:[


          classValue:=('x','<=',(aNode nextNode name)).
                                        elementValueSet
add:classValue.

          anElementInAENMainSubCollection at:anAttrIndex
                    put:classValue.
                              ]

                              ifFalse:[
                                        classValue:=self
classifyContAttr: (anElementInAENMainSubCollection
at:anAttrIndex)       treeNode: (aNode nextNode
                              nextNodeCollection  at:1)
valueOfparent:      (aNode nextNode name) .
                                        elementValueSet
```

```
add:classValue.

          anElementInAENMainSubCollection at:anAttrIndex
put:classValue.
                              ].
                    ]
                    ifFalse:[
                              (((aNode nextNode
nextNodeCollection at:2) nextNode
nextNodeCollection)='LeafNode' ) ifTrue:[


          classValue:=('x','>',(aNode nextNode name)).
                                        elementValueSet
add:classValue.

          anElementInAENMainSubCollection at:anAttrIndex
                    put:classValue.

                              ]
                              ifFalse:[
                                        classValue:=self
classifyContAttr: (anElementInAENMainSubCollection
at:anAttrIndex)      treeNode: (aNode nextNode
                              nextNodeCollection  at:2)
valueOfparent:      (aNode nextNode name) .
                                        elementValueSet
add:classValue.

          anElementInAENMainSubCollection at:anAttrIndex
put:classValue.
                              ].

                    ].

          ].

]."loop"



^elementValueSet.! !

!AENDecisionTreeClassifier methodsFor: 'accessing'!

elementTypesCollection


^elementTypesCollection!

learningDataCollection

^learningDataCollection!

learningDataCollection:aCollection

learningDataCollection:=aCollection!

mainElementValueCollection


^mainElementValueCollection!

mainElementValueCollection: aMainElementValueCollection


mainElementValueCollection:=aMainElementValueCollection.!

namesOfElements

^namesOfElements!

rootOfTree

^rootOfTree!

rootOfTree:aNode

rootOfTree:=aNode.! !

!AENDecisionTreeClassifier methodsFor:
'manipulatingCollections'!

removingElementFromCollection: aCollection elementIndex:
anElementIndex


| copy t |

copy:=OrderedCollection new.

t:=0.
aCollection size timesRepeat:[
t:=t+1.
(t=anElementIndex) ifFalse:[copy add:(aCollection at:t)]].



^copy!

returnSortedVersionOfCollection:aCollection
byContAttributeNr:anIndex

| temp |
          temp:= aCollection asSortedCollection:[:t :s|((t
at:anIndex)asNumber)<((s at:anIndex)asNumber)].

^temp!

sortingOfAENMainCollection:theAENMainCollection
byContAttributeNr:anIndex

| temp |
          temp:=theAENMainCollection asSortedCollection:[:t
:s|((t at:anIndex)asNumber)<((s at:anIndex)asNumber)].

^temp!

splittingOfCollection: anAENMainSubCollection dividedOn:
attributeNo elementValueCollection:elementValueCollection


"Denne metoden lager en ordnet samling med subsamlinger av
'anAENMainSubCollection' splittet over en angitt
attributt."

|numberOfElements n t element ElementCollection
tempCollection|

tempCollection:= OrderedCollection new.

"Lager en samling av riktig antall subsamlinger."
ElementCollection:=elementValueCollection at: attributeNo.
numberOfElements:=ElementCollection  size.
numberOfElements timesRepeat:[tempCollection
add:OrderedCollection new].

n:=0.
anAENMainSubCollection size timesRepeat:[n:=n+1.
element:= (anAENMainSubCollection at:n) at: attributeNo.
t:=self indexOfElement:element in: ElementCollection.
(tempCollection at: t) add:(anAENMainSubCollection at:n) ].
^tempCollection! !

!AENDecisionTreeClassifier methodsFor: 'private'!

calculatePredictedErrorByNumberOfSamples:aN
withANumberOfErrors:aE

"Utfører en iterasjon for Å finne forventet antall feil.
Det tas utgangspunkt i en binomisk fordeling"

| temp p x tol temp1 |

((aE=0) & (aN=0)) ifFalse:[
temp:=1000.
p:=0.

tol:=0.01.
[temp>=0.25] whileTrue:[temp:=0. x:=0.p:=p+tol .(aE+1)
timesRepeat:[temp1:=self Cn:aN k:x .
temp:=temp+(((temp1)*(p**x))*((1-p)**(aN-x))).x:=x+1] ].
].

((aE=0) & (aN=0)) ifTrue:[p:=0].


^p*aN!

Cn:aNumberN k:aNumberK

| temp |

          ((aNumberN=aNumberK)|(aNumberK=0)) ifTrue:[^1].

          ((aNumberN-aNumberK)>aNumberK) ifTrue:[
                    temp:=(self myFakToNum:aNumberN
fromNum:(aNumberN-aNumberK+1))/(self myFakToNum:aNumberK
fromNum:1)
                    ] ifFalse:[
                    temp:=(self myFakToNum:aNumberN
                    fromNum:(aNumberK+1))/(self
myFakToNum:(aNumberN-aNumberK) fromNum:1)
          ].




^temp!

myFakToNum:aNumA fromNum:aNumB

| temp t |
          (aNumA=aNumB ) ifTrue: [^temp:=aNumB] ifFalse:[

temp:=aNumB.
t:=aNumB.

(aNumA-aNumB) timesRepeat:[temp:=temp*(t+1).t:=t+1.].


].


^temp!

myLogFunction:aNumber

"Logfunksjonen liker ikke null eller mindre som input.
Logfunksjonen multipliseres imidlertid med null i den
senere beregninga dersom input er null, slik at nÅr input
er null gÅr det greit at null returneres av denne
funksjonen."

|t|
(aNumber<=0) ifTrue:[t:=0.] .
```

```
(aNumber<=0) ifFalse:[t:=aNumber log:2].

^t! !

!AENDecisionTreeClassifier methodsFor: 'other'!

classifySimpleDataCollection:anAENTestDataCollection node:
aNode


"Metoden klassifiserer ett datasett. Parameteren 'aNode' er
rotnoden til treet nÅr metoden kalles, og siden i de
rekursive kallene representerer den  en testnode."

| attribute indexAttribute attributeValue indexValue
classification |

"Finner vei i forgreningspunktet."
attribute:=aNode nextNode name.
indexAttribute:=self indexOfElement: attribute in:
namesOfElements.
attributeValue:=anAENTestDataCollection at: indexAttribute.
indexValue:=self indexOfElement:attributeValue  in:
(mainElementValueCollection at:indexAttribute).

((aNode nextNode nextNodeCollection at:indexValue) nextNode
nextNodeCollection='LeafNode') ifTrue: [
          "Øvnode"
          classification:=((aNode nextNode
nextNodeCollection at:indexValue) nextNode name)] "Skriver
ut klasse"
ifFalse:[
          "Rekursivt kall"
          classification:=self
classifySimpleDataCollection:anAENTestDataCollection node:
(aNode nextNode nextNodeCollection at:indexValue)

].

^classification! !

!AENDecisionTreeClassifier methodsFor: 'mainmetods'!

classify

self classifyDataCollectionRootNode: rootOfTree.!

classifyDataCollectionRootNode: aNode

"Metoden klassifiserer  alle datasettene. Og forutsetter at
'anAENTestDataCollection' er en ordnet samling med datasett
der datasettene ogsÅ er ordnete samlinger. Se
initialisering. Parameteren 'aNode' er rotnoden til treet
nÅr metoden kalles, og siden i de rekursive kallene
representerer den  en testnode."

| t classeficationCollection testDataCollection filename
stream k2 |

testDataCollection:=self
readDataFromFileNumberOfElementsEachLine:
(numberOfElementsEachLine-1) numberOfLines:
numberOfLinesTestData datafile:testDataFile .
classeficationCollection:=OrderedCollection new.

self preprosessContAttrInTestCollection:testDataCollection
.


t:=0.
testDataCollection size timesRepeat:[t:=t+1.
classeficationCollection add:( self
classifySimpleDataCollection:(testDataCollection at:t)
node: aNode
    ) ].


filename := classificationOutputFile asFilename.

"Creating the file."
stream := filename writeStream.


[
          k2:=1.
          [k2 <= classeficationCollection size] whileTrue:
[
                    stream
nextPutAll:(classeficationCollection at:k2).
                    stream nextPutAll:'
'.          "return"
                    k2:=k2+1.
          ].
]
valueNowOrOnUnwindDo: [stream close].!

drawTree:aNode deep: aDeep


"Denne funksjonen skriver ut det genererte beslutningstreet
ved at man kaller funksjonen med beslutningstreets rootnode
som innparameter for 'aNode'. Parameteren 'aDeep' angir
innrykket pÅ utskriften. Innrykket inkrementeres for hvert
dybdenivÅ i treet. I det initielle kallet kan 'aDeep'
settes til  f.eks. '2'. "

| numberOfTests anOtherDeep t |

numberOfTests:=aNode nextNode nextNodeCollection size.
"Antall forgreninger pÅ dette nivÅet i treet."


anOtherDeep:=aDeep +1.
t:=0.

numberOfTests timesRepeat:[t:=t+1. Transcript
crtab:anOtherDeep.
Transcript show:(aNode nextNode name); "skriver navn pÅ
attributt"
show:' = ' ;
show:(((aNode nextNode nextNodeCollection) at:t) name).
"skriver  navn pÅ testverdi i aktuell forgrening."
(((aNode nextNode nextNodeCollection) at:t) nextNode
nextNodeCollection  = 'LeafNode' ) ifTrue:[Transcript
show:': ' ;
          show:(((aNode nextNode nextNodeCollection) at:t)
nextNode name) ; "skriver navn pÅ klasse"
          show:' (' ;
          show: (((aNode nextNode nextNodeCollection) at:t)
nextNode number) printString; "skriver antall sampler som
ble klassifisert pÅ denne noden."
          show:'/';
          show: (((aNode nextNode nextNodeCollection) at:t)
nextNode numberWrongClass) printString; "skriver antall
sampler som ble missklassifisert        pÅ denne noden."
          show:')' . "teller:=teller+1. (teller=28)
ifTrue:[self halt. Transcript clear. teller:=0 ]" ]
ifFalse: [ self drawTree:((aNode nextNode
nextNodeCollection) at:t) deep: anOtherDeep]. "rekursivt
kall"

].!

generateTree: anAENMainSubCollection
elementValueCollection:anElementValueCollection treeNode:
aNode namesOfElements:aNameCollection

"Metoden lager treet basert pÅ datasamlingen og
informasjonen om datastrukturen."

| gainRatioCollection splittIndex splittCollection
numberOfSubCollections  t anOtherAENMainSubCollection s
newAENMainSubCollection temp anOtherElementValueCollection
anOtherNameCollection tempNodeCollection tempTestNode
classIndex frequencyCollection maxIndex numberOfClasses
myFrequencyCollection |

"Danner en samling av 'gainratios' for splitt pÅ alle
attributter."
numberOfClasses:=   anElementValueCollection last size.
gainRatioCollection:=self gainRatioCollection:
anAENMainSubCollection
elementValueCollection:anElementValueCollection.


temp:=0.
gainRatioCollection do:[:each| temp:=temp+each].

"Dersom alle 'gainratios' er null,  oppnÅr man ingenting
med Å splitte blokken videre opp fordi alle
attributtverdiene er like. Imidlertid trenger ikke klassen
Å være den samme for alle samplene pga. inkonsistent
informasjon."

(temp=0) ifTrue:[


          "Genererer en samling med klassefrekvensene"

          classIndex:=(anAENMainSubCollection at:1) size.
          frequencyCollection:=self
frequencyOfElementCollection: anAENMainSubCollection
elementNumber: classIndex
          elementValueCollection:anElementValueCollection.

          "Velger klassen som er mest forekommende"
          maxIndex:=self indexOfMaxElementIn:
frequencyCollection.


          aNode nextNode:AttributeNode new.
          aNode nextNode
classFrequencyCollection:frequencyCollection.
          aNode nextNode nextNodeCollection:'LeafNode'.
                    "Setter inn en løvnode"
          aNode nextNode name:((anElementValueCollection
last) at:maxIndex) .          "Gir den klassen med stØrst
forekomstrate."
          aNode nextNode number:(anAENMainSubCollection
size).    "Angir antall sampler som havnet i løvnoden."
          aNode nextNode
numberWrongClass:(anAENMainSubCollection size-
(frequencyCollection at:maxIndex)).
^self].

"Splitter blokken med samplene pÅ den attributten som har
stØrst gainratio "
splittIndex:=self indexOfMaxElementIn:gainRatioCollection.
splittCollection:=self splittingOfCollection:
anAENMainSubCollection dividedOn: splittIndex
elementValueCollection:anElementValueCollection.

numberOfSubCollections:=(anElementValueCollection
at:splittIndex) size.
aNode nextNode:AttributeNode new. "Lager en ny
atributtnode"
aNode nextNode name:(aNameCollection
at:splittIndex)."Setter navn pÅ attributt."
aNode nextNode number:(anAENMainSubCollection size)."Setter
```

```
antall sampler."
classIndex:=(anAENMainSubCollection at:1) size.
frequencyCollection:=self frequencyOfElementCollection:
anAENMainSubCollection elementNumber: classIndex
elementValueCollection:anElementValueCollection.
maxIndex:=self indexOfMaxElementIn: frequencyCollection.

"Setter inn frekvensfordelingen til samplene som strømmer
gjennom noden i noden."
aNode nextNode
classFrequencyCollection:frequencyCollection.

"Lager en ordnet samling med testnoder inne i
attributtnoden."
tempNodeCollection:=OrderedCollection new.
t:=0.
numberOfSubCollections timesRepeat:[t:=t+1.
tempTestNode:=TestNode new. tempTestNode
name:((anElementValueCollection at:splittIndex) at:t).
tempNodeCollection add: tempTestNode  ].
aNode nextNode nextNodeCollection: tempNodeCollection.

"Fjerner splittattributtet fra samlingen med
attributtverdier."
anOtherElementValueCollection:=self
removingElementFromCollection: anElementValueCollection
elementIndex: splittIndex.

"Fjerner splittattributtet fra samlingen med
attributtnavn."
anOtherNameCollection:=self removingElementFromCollection:
aNameCollection elementIndex: splittIndex.

t:=0.
numberOfSubCollections timesRepeat:[t:=t+1.
anOtherAENMainSubCollection:=  (splittCollection at:t).
(anOtherAENMainSubCollection isEmpty)
ifTrue:[

          "Dersom subsamlingen er tom lages en løvnode med
default klasse. Antall sampler er initiert til null."

          (aNode nextNode nextNodeCollection at:t)
nextNode:AttributeNode new.
          (aNode nextNode nextNodeCollection at:t) nextNode
name:( (anElementValueCollection last) at:maxIndex)  .
"Setter klassen til mest      forekommende klasse til moder
attributtet."

          (aNode nextNode nextNodeCollection at:t) nextNode
nextNodeCollection:'LeafNode'.
          myFrequencyCollection:= OrderedCollection new.
          numberOfClasses
timesRepeat:[myFrequencyCollection add:0].
          (aNode nextNode nextNodeCollection at:t) nextNode
classFrequencyCollection:myFrequencyCollection.
          ]
 ifFalse:[
          "Lager en ny samling med splittattributtet
fjernet fra subsamlingen."
          newAENMainSubCollection:=OrderedCollection new.
s:=0.
          (anOtherAENMainSubCollection size)
timesRepeat:[s:=s+1. newAENMainSubCollection add:(
          self removingElementFromCollection:
(anOtherAENMainSubCollection at:s) elementIndex:
splittIndex)].
          "Rekursivt kall."
          self generateTree: newAENMainSubCollection
elementValueCollection:anOtherElementValueCollection
treeNode: (((aNode nextNode   nextNodeCollection) at:t) )
namesOfElements:anOtherNameCollection ]].!

learn


          mainCollection:=self
readDataFromFileNumberOfElementsEachLine:
numberOfElementsEachLine numberOfLines:
numberOfLinesTrainingData datafile:trainingDataFile.


self preprosessContAttrInCollection:mainCollection
byNumberOfLevels:maxLevelContSplit
elementValueCollection:mainElementValueCollection
elementTypesCollection: elementTypesCollection.

self generateTree: mainCollection
elementValueCollection:mainElementValueCollection treeNode:
rootOfTree namesOfElements:namesOfElements.

self pruneTree:rootOfTree
elementValueCollection:mainElementValueCollection.
self drawTree:(self rootOfTree) deep: 1.!

pruneTree:aNode
elementValueCollection:anElementValueCollection

"funksjonen friserer bort de delene av det genererte treet
som later til Å bære ikke signifikant og villedende
informasjon"

| numberOfTests  t accumulatedNodeClassDistribution
numberOfSamplesUnderNode predictedErrorCollection num err
predictedErrorOfAllBranches maxIndex numberOfLeafNodes
pruned |

numberOfTests:=aNode nextNode nextNodeCollection size.
"Antall forgreninger pÅ dette nivÅet i treet."
anElementValueCollection last size.
numberOfLeafNodes:=0.


t:=0.
numberOfSamplesUnderNode:=0.
predictedErrorCollection := OrderedCollection new.

[numberOfTests>t] whileTrue:[t:=t+1.


(((aNode nextNode nextNodeCollection) at:t) nextNode
nextNodeCollection  = 'LeafNode' ) ifTrue:[
          numberOfLeafNodes:=numberOfLeafNodes+1.
          num:=((aNode nextNode nextNodeCollection) at:t)
nextNode number.
          err:=((aNode nextNode nextNodeCollection) at:t)
nextNode numberWrongClass.
          numberOfSamplesUnderNode:=numberOfSamplesUnderNod
e +num.
          predictedErrorCollection add:(self
calculatePredictedErrorByNumberOfSamples:num
withANumberOfErrors:err).


 ]
ifFalse: [ pruned:=self pruneTree:((aNode nextNode
nextNodeCollection) at:t)
elementValueCollection:anElementValueCollection.
          (pruned) ifTrue:[
          t:=t-1].  "Dette mÅ til ellers sÅ glemmes denne
grenen nÅr den har blitt frisert, nÅr den egentlig skal
være med i den videre friseringen"
 ]. "rekursivt kall"

].

(numberOfTests=numberOfLeafNodes) ifTrue:[
          accumulatedNodeClassDistribution:=aNode nextNode
classFrequencyCollection.
          predictedErrorOfAllBranches:=0.
          predictedErrorCollection do:[:each|
          predictedErrorOfAllBranches:=predictedErrorOfAllB
ranches+each].
          maxIndex:=self
indexOfMaxElementIn:accumulatedNodeClassDistribution.
"Finner mest        forekommende klasse"

          (predictedErrorOfAllBranches
>=(numberOfSamplesUnderNode-
(accumulatedNodeClassDistribution       at:maxIndex)))
ifTrue:[
          "Det forventes mer feil ved Å beholde
forgreningen. Derfor skjæres den bort. En ny lØvnode settes
inn       i stedet. Data oppdateres."

                    aNode nextNode
classFrequencyCollection:accumulatedNodeClassDistribution.
                    aNode nextNode
nextNodeCollection:'LeafNode'. "Setter  lØvnode"
                    aNode nextNode
name:((anElementValueCollection last) at:maxIndex) . "Gir
den klassen med               stØrst forekomstrate."
                    aNode nextNode
number:numberOfSamplesUnderNode."Angir antall sampler som
havnet i            lØvnoden."
                    aNode nextNode
          numberWrongClass:(numberOfSamplesUnderNode-
(accumulatedNodeClassDistribution at:
          maxIndex)).
          ^true
          ].
].
^false! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

AENDecisionTreeClassifier class
          instanceVariableNames: ''!


!AENDecisionTreeClassifier class methodsFor: 'ny'!

new


^super new initialize! !
1


AENClassifier subclass: #SVM
          instanceVariableNames: 'kernel indexCollection
bias xCollection yCollection cPar alphaCollection
yCollectionOfCollections alphaFile ctrlFile setCtrlFile
HessianFile AeqFile '
          classVariableNames: ''
          poolDictionaries: ''
          category: 'AEN-SVM-Classifier'!


!SVM methodsFor: 'mainmetods'!

classify


| testDataCollection tempCollection mainIndexCol
biasCollection classCollection tempIndexCol k1 k2 maxIndex
tempMax temp filename stream |

testDataCollection:=self
readDataFromFileNumberOfElementsEachLine:
(numberOfElementsEachLine-1) numberOfLines:
```

```
numberOfLinesTestData datafile:testDataFile .

tempCollection:=self
returnNumericCollection:testDataCollection.

biasCollection:=OrderedCollection new.
mainIndexCol:=OrderedCollection new.
classCollection:=OrderedCollection new.

k1:=1.
alphaCollection size timesRepeat:[
          tempIndexCol:=self
findIndexOfSupportVectorsAlphaCol:(alphaCollection at:k1)
          YCol:(yCollectionOfCollections at:k1).
          mainIndexCol add:tempIndexCol.
          k1:=k1+1.
].

k1:=1.
alphaCollection size timesRepeat:[
          biasCollection add:(self returnBiasBInputX:
xCollection inputY:(yCollectionOfCollections      at:k1)
          alphaCollection: (alphaCollection at:k1)
constraintC:cPar  indexCollection:      (mainIndexCol
at:k1) ).


          k1:=k1+1.
].


k2:=1.
tempCollection size timesRepeat:[
          k1:=1.
          maxIndex:=1.
          tempMax:=-1e-30.
          alphaCollection size timesRepeat:[
                    temp:=(self classifyInputX:
(tempCollection at:k2) yCol: (yCollectionOfCollections
at:k1)              alphaCol:(alphaCollection at:k1)
indexCol:(mainIndexCol at:k1) bias: (biasCollection
                    at:k1)).
                    (temp >tempMax) ifTrue:[maxIndex:=k1.
tempMax:=temp].
                    k1:=k1+1.
          ].

          classCollection add:((mainElementValueCollection
last) at:maxIndex).
          k2:=k2+1.
].

filename := classificationOutputFile asFilename.

"Creating the file."
stream := filename writeStream.


[
          k2:=1.
          [k2 <= classCollection size] whileTrue: [

                    stream nextPutAll:(classCollection
at:k2).
                    stream nextPutAll:'
'.          "return"
                    k2:=k2+1.
          ].
]
valueNowOrOnUnwindDo: [stream close].!

classifyInputX: aX yCol: aYCollection alphaCol:anAlphaCol
indexCol:anIndexCol bias:aBias

          | k1 numberOfSV temp index theKernel |

"aX is a simple orderedCollection"

numberOfSV:= (anIndexCol at:5) size.
temp:=0.

k1:=1.

          [k1 <= numberOfSV] whileTrue: [
          index:=(anIndexCol at:5) at:k1.
          theKernel:=kernel value:(aX) value: (xCollection
at:index).

          temp:=temp+((anAlphaCol at:index )*(aYCollection
at:index)*theKernel).
          k1:=k1+1.
].

temp:=temp+aBias.


^temp!

findIndexOfSupportVectorsAlphaCol:anAlphaCollection
YCol:aYCollection



"funksjonen finner indeksene til stÃ¸ttevektorene.Disse
lagres som samlinger i  collect5 som  1C-,  -1C-,  1C, and
-1C. Der  1C- betyr klasse
1 og 0<alfa<C etc..    "


          | k1   collect1 collect2 collect3 collect4
collect5 epsilon collect6 |




          collect1:=OrderedCollection new.
          collect2:=OrderedCollection new.
          collect3:=OrderedCollection new.
          collect4:=OrderedCollection new.
          collect5:=OrderedCollection new.
          collect6:=OrderedCollection new.
          epsilon:=1e-14.

          k1:=1.
          [k1 <= (aYCollection size)] whileTrue: [
                    (((anAlphaCollection at:k1)
>epsilon)&((anAlphaCollection at:k1)<(cPar-epsilon)) )
                    ifTrue:[
                    ((aYCollection at:k1)=1)
ifTrue:[collect1 add:k1] ifFalse:[collect2 add:k1].

                    ].
                    ((anAlphaCollection at:k1) >=(cPar-
epsilon)) ifTrue:[
                    ((aYCollection at:k1)=1)
ifTrue:[collect3 add:k1] ifFalse:[collect4 add:k1].

                    ].


          k1:=k1+1.
].
collect5:=OrderedCollection new.
collect5:=collect1,collect2,collect3,collect4.
collect6 add:collect1 ; add: collect2 ; add: collect3 ;
add: collect4 ; add: collect5.


 ^collect6!

learn



mainCollection:=self
readDataFromFileNumberOfElementsEachLine:
numberOfElementsEachLine numberOfLines:
numberOfLinesTrainingData datafile:trainingDataFile.

alphaCollection:=self
returnAlphaCollection:(mainCollection).! !

!SVM methodsFor: 'initialize'!

initialize

          | kernelType  |

          "Read the comments in the file qpsvm.m"


          kernelType:='RBF'. "Bestemmer type kjerne"

          (kernelType='Linear') ifTrue:[
                    "kernel = x1 dot x2"
                    kernel:=[:x1 : x2 |
                    |dotProduct|
                    dotProduct:=0 . 1 to:x1 size do:[:i
                    |         dotProduct:=dotProduct +
(((x1 at:i) asNumber)* ((x2 at:i) asNumber))]. dotProduct].

          ].
          (kernelType='Polynomial') ifTrue:[
                    "kernel = (x1 dot x2 + 1)^polydeg"
                    kernel:=[:x1 : x2 |
                    |dotProduct polydeg|
                    dotProduct:=0. polydeg:=2. 1 to:x1 size
do:[:i
                    |         dotProduct:=dotProduct +
(((x1 at:i) asNumber)*((x2 at:i) asNumber))].
((dotProduct+1)**polydeg)].

          ].

(kernelType='RBF') ifTrue:[
                    "kernel = (x1 dot x2 + 1)^polydeg"
                    kernel:=[:x1 : x2 |
                    |temp1 o|

                    temp1:=0. o:=30. 1 to:x1 size do:[:i
                    |         temp1:=temp1 + (((((x1 at:i)
asNumber)-((x2 at:i) asNumber)))**2)]. ((temp1 *(-
1)/(2*(o**2))) exp) ].

          ].

          "Here it is possible to include even more
kernelfunctions"

          "The parameter C"
          cPar:=5.

"
The datastructure is given by elementTypesCollection,
namesOfElements and mainElementValueCollection. The
folowing two examples show how to set these variables.

Example 1
elementTypesCollection:=#('Cont' 'Cont' 'Cont' 'Cont'
```

```
'Cont' 'Cont' 'Cont' 'Cont' 'Cont' 'Cont' 'Cont' 'Cont'
'Cont' 'Byorder').
namesOfElements:=#('Alcohol' 'MalicAcid' 'Ash'
'AlcalinityOfAsh' 'Magnesium' 'TotalPhenols' 'Flavanoids'
'NonflavanoidPhenols' 'Proanthocyanins' 'ColorIntensity'
'Hue' 'OD280/OD315OfDilutedWines' 'Proline'  'class').
mainElementValueCollection:=#( #() #() #() #() #()  #() #()
#() #()  #() #() #() #() #('1' '2' '3') ) .

Example 2

elementTypesCollection:=#('ByOrder' 'ByOrder' 'ByOrder'
'ByOrder' 'ByOrder' 'ByOrder' 'ByOrder').
namesOfElements:=#('buying' 'maint' 'doors' 'persons'
'lug_boot' 'safety' 'class').
mainElementValueCollection:=#( #('low' 'med' 'high'
'vhigh') #('low' 'med' 'high' 'vhigh') #( '2' '3' '4'
'5more'  ) #( '2' '4' 'more' )  #('small' 'med' 'big') #(
'low' 'med' 'high' ) #('unacc' 'acc' 'good' 'vgood') )



Information about the size of the training and test files
must be given.

Example 4
numberOfLinesTrainingData :=89.
numberOfLinesTestData:=89.
numberOfElementsEachLine := namesOfElements size.
trainingDataFile:= 'd:\vw30\treningssettWine.txt'.
treeClassificationOutputFile:='d:\vw30\klassefiseringsres.t
xt'.
testDataFile:='d:\vw30\testsettWine.txt'.


          elementTypesCollection:=#('Cont' 'Cont' 'Cont'
'Cont' 'Cont' 'Cont' 'Cont' 'Cont' 'Cont' 'Cont' 'Cont'
'Cont' 'Cont' 'Byorder').
namesOfElements:=#('Alcohol' 'MalicAcid' 'Ash'
'AlcalinityOfAsh' 'Magnesium' 'TotalPhenols' 'Flavanoids'
'NonflavanoidPhenols' 'Proanthocyanins' 'ColorIntensity'
'Hue' 'OD280/OD315OfDilutedWines' 'Proline'  'klasse').
mainElementValueCollection:=#( #() #() #() #() #()  #() #()
#() #()  #() #() #() #() #('1' '2' '3') ) .

numberOfLinesTrainingData :=89.
numberOfLinesTestData:=89.
numberOfElementsEachLine := namesOfElements size.
trainingDataFile:= 'd:\vw30\treningssettWine1.txt'.
treeClassificationOutputFile:='d:\vw30\klassefiseringsres.t
xt'.
testDataFile:='d:\vw30\testsettWine2.txt'.


alphaFile:= 'd:\aen\alpha.txt' asFilename.
ctrlFile:= 'd:\aen\ctrlbit.txt' asFilename.
setCtrlFile := 'd:\aen\ctrlbit.txt' asFilename.
HessianFile := 'd:\aen\H.dat' asFilename.
AeqFile := 'd:\aen\Aeq.dat' asFilename.
"! !

!SVM methodsFor: 'private'!

altFindTwoDiffrentClassSupportVectorsIndexCollection:anInde
xCollection alphaCollection: anAlphaCollection


          "This function find the indexes of two
supportvectors of diffrent class lying on the margin"


          | k1 collect index1 minTemp1 minTemp2 index2 temp
posCollection negCollection |

          collect:=OrderedCollection new.
          minTemp1:=cPar.
          minTemp2:=cPar.
          index1:=-1.
          index2:=-1.
          posCollection:=anIndexCollection at:1.
          negCollection:=anIndexCollection at:2.

          k1:=1.
          [k1 <= (posCollection size)] whileTrue: [
                    temp:=((anAlphaCollection
at:(posCollection at:k1) )-(cPar/2)) abs.
                    (temp<minTemp1) ifTrue:[minTemp1:=temp.
index1:=(posCollection at:k1)].
          k1:=k1+1.
          ].

          k1:=1.
          [k1 <= (negCollection size)] whileTrue: [
                    temp:=((anAlphaCollection
at:(negCollection at:k1) )-(cPar/2)) abs.
                    (temp<minTemp2) ifTrue:[minTemp2:=temp.
index2:=(negCollection at:k1)].
          k1:=k1+1.
          ].



((index1=-1) or: [index2=-1]) ifTrue:[Transcript
show:'function failed'.^self].

collect   add:index1;add: index2.
 ^collect.!

returnCollectionOfNumericYCollections:aCollection


"This function returns an ordered collection with the y-
vectors representing the classes. An element representing a
learningsample of the class belonging to the y.vector is
given a 1, and an element representing a learningsample of
another class not belonging to the y.vector is given a -1"

| k1 k2 tempCollectionOfYCollections tempYCollection temp |

          tempYCollection:=self
returnYCollection:aCollection.
          tempCollectionOfYCollections:=OrderedCollection
new.
          k1:=1.
          (mainElementValueCollection last size)
timesRepeat:[
                    temp:=OrderedCollection new.
                    k2:=1.
                    tempYCollection size timesRepeat:[
                              ((tempYCollection
at:k2)=((mainElementValueCollection last) at:k1)) ifTrue:[
                              temp add:1
                              ] ifFalse:[
                              temp add:-1
                              ].

                              k2:=k2+1.
                    ].


                    tempCollectionOfYCollections add:temp.


                    k1:=k1+1.
          ].


^tempCollectionOfYCollections!

returnHMatrix

"This function returns the Hessian matrix"


| k1 k2 matrixDim collectMatrix theKernel temp |



matrixDim:=yCollection size.

collectMatrix:=Array new:matrixDim.
k1:=1.
          [k1 <= matrixDim] whileTrue: [
                    collectMatrix at: k1 put: (Array
new:matrixDim).
          k1:=k1+1
].



          k1:=1.
          [k1 <= matrixDim] whileTrue: [
                    k2:=k1.
                    [k2 <= matrixDim] whileTrue: [

                              theKernel:=kernel
value:(xCollection at:k1) value: (xCollection at:k2) .

                              temp:=(theKernel*(yCollection
at:k2)*(yCollection at:k1)).
                              (collectMatrix at:k1) at:k2
put:temp.
                              (k1~=k2)
ifTrue:[(collectMatrix at:k2) at:k1 put:temp.].

                              k2:=k2+1
                    ].


                    k1:=k1+1
          ].



          ^collectMatrix! !

!SVM methodsFor: 'other'!

readAndReturnAlpha


          |  stream separator line readingBlock |


          stream := alphaFile readStream.
          separator := $,.    "comma"

          line:=OrderedCollection new.


          readingBlock := [
          [stream atEnd] whileFalse: [
          line add: (stream upTo: separator) asNumber]].

          readingBlock valueNowOrOnUnwindDo: [stream
close].
```

```
^line!

readAndReturnCtrlBit


          |  stream separator line readingBlock |

          "kontrollbittet består bare av et tegn. 0 eller
1.
          Se kommentarer i qpsvm.m-filen."



          stream := ctrlFile readStream.
          separator := $,.    "comma"

          line:=OrderedCollection new.


          readingBlock := [
          [stream atEnd] whileFalse: [
          line add: (stream upTo: separator) ]].
          readingBlock valueNowOrOnUnwindDo: [stream
close].


^line at:1!

returnAlpha


| wait |

"The parameters of the QP-problem is written to file and
the controlbit is set"

          self writeQpParToFile.
          self setCtrlBit.


          wait:=1.
          [wait=1] whileTrue:[
                    (self readAndReturnCtrlBit='0')
ifTrue:[ wait:=0 ].
          ].

          ^self readAndReturnAlpha!

returnAlphaCollection:aCollection

"Solves the QP-problem for each class and returns a
collection with solution-vectors or alpha-vectors"

|  k1 tempAlphaCollection tempXCollection |


          tempXCollection:=self
returnXCollection:aCollection.
          xCollection:=self
returnNumericCollection:tempXCollection.

          yCollectionOfCollections:=self
returnCollectionOfNumericYCollections:aCollection.

          tempAlphaCollection:=OrderedCollection new.
          k1:=1.
          yCollectionOfCollections size timesRepeat:[
                    yCollection:=yCollectionOfCollections
at:k1.
                    tempAlphaCollection add:(self
returnAlpha).
                    k1:=k1+1.
          ].

^tempAlphaCollection!

returnBiasBInputX: aXCollection inputY:aYCollection
alphaCollection: anAlphaCollection constraintC:aC
indexCollection:anIndexCollection

          | k1 kernelS temp kernelR b supportVectorIndexCol
index tempCollect indexS indexR |

"This function returns the bias b"
"IndexS and IndexR gives the indexes of two vectors of
diffrent class on the margin"



supportVectorIndexCol:=anIndexCollection at:5.


tempCollect:=self
altFindTwoDiffrentClassSupportVectorsIndexCollection:anInde
xCollection alphaCollection: anAlphaCollection.
indexS:= tempCollect at:1.
indexR:= tempCollect at:2.
temp:=0.
k1:=1.
          [k1 <= (supportVectorIndexCol size)] whileTrue: [
          index:=supportVectorIndexCol at:k1.
          kernelS:=kernel value:(aXCollection at:index)
value:(aXCollection at:indexS).
          kernelR:=kernel value:(aXCollection at:index)
value:(aXCollection at:indexR).

          temp:=temp+((anAlphaCollection
at:index)*(aYCollection at:index)*(kernelS + kernelR)).
          k1:=k1+1.
].

b:=temp*(-0.5).

^b!

setCtrlBit

|  stream |


"Creating the file."
stream := setCtrlFile writeStream.


[stream nextPutAll:'1' .

]
valueNowOrOnUnwindDo: [stream close].!

writeQpParToFile

| stream H matrixDim k1 k2 temp |


"Creating the file."
stream := HessianFile writeStream.

H:=self returnHMatrix.

[


          matrixDim:=yCollection size.

          k1:=1.
          [k1 <= matrixDim] whileTrue: [
                    k2:=1.
                    [k2 <= matrixDim] whileTrue: [
                              temp:=((H at:k1) at:k2)
printString.
                              stream nextPutAll:temp.
                              stream nextPutAll:'  '.
                              (matrixDim=k2) ifTrue:[stream
nextPutAll:'
'].
                              k2:=k2+1.
                    ].
          k1:=k1+1.
          ].
] valueNowOrOnUnwindDo: [stream close].



"Creating the file."
stream := AeqFile writeStream.

[k1:=1.
          [k1 <= (matrixDim)] whileTrue: [

                    temp:=(yCollection at:k1) printString.
                    stream nextPutAll:temp.
                    stream nextPutAll:'  '.

                    k1:=k1+1.
          ].
]
valueNowOrOnUnwindDo: [stream close].! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

SVM class
          instanceVariableNames: ''!


!SVM class methodsFor: 'new'!

new

^super new initialize! !

1


FFINNInterface subclass: #AENNN
          instanceVariableNames: 'numberOfLearningCycles '
          classVariableNames: ''
          poolDictionaries: ''
          category: 'AEN-NN-Classifier'!


!AENNN methodsFor: 'init'!

initialize

"
Documentation

Configuration of the nettwork:

self createNNInput:13 hidden:13 output:3 fileName:'nn-
file.dat'.

self readFromFile ifFalse:[ Dialog warn: 'NN file ',nn
fileName,' not found or incorrect -- \randomizing weights'
withCRs.
self randomizeFrom:-1 to: 1].

Number of passes trough the training examples:

numberOfLearningCycles:=10000.
```

```
The datastructure is given by elementTypesCollection,
namesOfElements and mainElementValueCollection. The
folowing two examples show how to set these variables.

Example 1
elementTypesCollection:=#('Cont' 'Cont' 'Cont' 'Cont'
'Cont' 'Cont' 'Cont' 'Cont' 'Cont' 'Cont' 'Cont' 'Cont'
'Cont' 'Byorder').
namesOfElements:=#('Alcohol' 'MalicAcid' 'Ash'
'AlcalinityOfAsh' 'Magnesium' 'TotalPhenols' 'Flavanoids'
'NonflavanoidPhenols' 'Proanthocyanins' 'ColorIntensity'
'Hue' 'OD280/OD315OfDilutedWines' 'Proline'  'class').
mainElementValueCollection:=#( #() #() #() #() #()  #() #()
#() #()  #() #() #() #() #('1' '2' '3') ) .



Example 2

elementTypesCollection:=#('ByOrder' 'ByOrder' 'ByOrder'
'ByOrder' 'ByOrder' 'ByOrder' 'ByOrder').
namesOfElements:=#('buying' 'maint' 'doors' 'persons'
'lug_boot' 'safety' 'class').
mainElementValueCollection:=#( #('low' 'med' 'high'
'vhigh') #('low' 'med' 'high' 'vhigh') #( '2' '3' '4'
'5more'  ) #( '2' '4' 'more' )  #('small' 'med' 'big') #(
'low' 'med' 'high' ) #('unacc' 'acc' 'good' 'vgood') )



Information about the size of the training and test files
must be given.

Example 4
numberOfLinesTrainingData :=89.
numberOfLinesTestData:=89.
numberOfElementsEachLine := namesOfElements size.
trainingDataFile:= 'd:\vw30\treningssettWine.txt'.
classificationOutputFile:='d:\vw30\klassefiseringsres.txt'.
testDataFile:='d:\vw30\testsettWine.txt'.

"! !


!AENNN methodsFor: 'other'!

postprocess

"
This method will receive as anArray the nn's output vector
(of length outputLength).
It should deliver whatever output is appropriate.
Assume the output is 1-dimensional, and the desired output
is 10 times the nn's output.
Then it would be:

          ^10 * (output at:1)

"



postprocessInverted: anObject


"
This method is only used for training.
It takes a desired output-object as input, and transformes
it to a corresponding nn-output.
It should be the inverse of postProcess.
In the example given for that method, the code would be:

          output at:1 put:(anObject * 0.1)

"



preprocess: anObject

"
This method should register the input values for the nn,
using the method reg:.
Assume anObject has numeric attributes a1 and a2 to be
given as input, where a2 should be scaled down by a factor
5.
Then the code would be:

          self reg: anObject a1.
          self reg: (anObject a2) *0.2.
"



self reg: (anObject at:13) asNumber *0.001.! !

!AENNN methodsFor: 'mainmetods'!

classify

| testDataCollection tempCollection classCollection k1 k2
maxIndex filename stream |

          classCollection:=OrderedCollection new.

          testDataCollection:=self
readDataFromFileNumberOfElementsEachLine:
          (numberOfElementsEachLine-1) numberOfLines:
          numberOfLinesTestData

          datafile:testDataFile .

          tempCollection:=self
returnNumericCollection:testDataCollection .

          k1:=1.
          tempCollection size timesRepeat:[
                    maxIndex:=self evaluate:
(tempCollection at:k1).
                    classCollection
add:((mainElementValueCollection last) at:maxIndex).
                    k1:=k1+1.
          ].

          filename :=classificationOutputFile asFilename.

          "Creating the file."
          stream := filename writeStream.


          [
                    k2:=1.
                    [k2 <= classCollection size] whileTrue:
[
                              stream
nextPutAll:(classCollection at:k2).
                              stream nextPutAll:'
'.                     "return"
                              k2:=k2+1.
                    ].
          ]
          valueNowOrOnUnwindDo: [stream close].!




learn



| tempCol tempXCol tempYCol k1 k2  |

mainCollection:=self
readDataFromFileNumberOfElementsEachLine:
numberOfElementsEachLine numberOfLines:
numberOfLinesTrainingData datafile:trainingDataFile.

tempCol:=self returnNumericCollection:mainCollection.

tempXCol:=self returnXCollection:tempCol.
tempYCol:=self returnYCollection:tempCol.

k2:=1.
          [k2 <= numberOfLearningCycles] whileTrue: [

          k2 \\ 10 = 0 ifTrue: [Transcript show: k2
printString; tab].

          k1:=1.
                    [k1 <= (tempYCol size)] whileTrue: [
                              self train: (tempXCol at:k1)
feedback: (tempYCol at:k1).


                    k1:=k1+1.
          ].
k2:=k2+1.
].

"Write the values of the weights to file"
self writeToFile.! !

1
```

## D.2 Programkode i matlab som støtter SVM-modulen

```
%qpsvm

%filen har ansvaret for beregning av alfa og må startes opp før image-filen

%før qpsvm kjøres, settes innholdet i ctrlbit.txt til 0
%ctrlbit.txt må settes til 1 for at beregning skal utføres
%ctrlbit resettes for hver beregning
%stopp må settes manuelt til 1 for å stoppe beregningssyklusen
%fem filer benyttes qpsvm.m, stopp.txt, ctrlbit.txt, alpha.txt, qpinput.m
%qpinput, inneholder matrisene som quadprog benytter og
%resultatvektoren skrives til alpha.txt

cPar=5;
stopp=0;
while stopp==0,
   pause(0.1);
   [ctrlbit]=textread('ctrlbit.txt','%u')

   if ctrlbit==1
      load H.dat;
      load Aeq.dat;
      dim=length(Aeq);
      LB=zeros(dim,1);
      UB=ones(dim,1)*cPar;
      f=ones(dim,1)*(-1);
      beq=0;

      OPTIONS = OPTIMSET('MaxIter',5000,'Preconditioner','on','DiffMinChange',1e-12,'Display','iter')
      [alpha]=quadprog(H,f,[],[],Aeq,beq,LB,UB);
      fid=fopen('alpha.txt','w');
                  fprintf(fid,'%16.14f,', alpha); %legger inn et komma mellom alfa-verdiene
                  fclose(fid);

      ctrlbit=0;
      fid=fopen('ctrlbit.txt','r+');
                  fprintf(fid,'%u', ctrlbit);
                  fclose(fid);

   end

   [stopp]=textread('stoppbit.txt','%u')

end
```

## D.3 Programkode i matlab som genererer resultatene i kapittel 5.4

```
 %SVM

X=[1.5 2.5 1.0 2.5 1.5 2.5 3.5 3.5 3.5 2.5 2.5 1.5 ;
   1.5 2.0 2.5 3.0 3.5 1.0 1.5 2.5 3.5 4.0 5.0 4.5 ];

y=[1  ;1;  1;  1;  1; -1; -1; -1; -1; -1; -1; -1];

c=5;
index=0;
[alpha]=AENGenAlpha(X,y,c)

N=length(alpha);
k=1;
for I=1:N,
   if alpha(I)>0.0000000000000001
      index(k)=I;k=k+1;
   end
end

N=length(index);

mintemp1=c;
mintemp2=c;

for I=1:N,
   temp=abs((c/2)-alpha(index(I)));
   if y(index(I))==1
      if temp<mintemp1
         mintemp1=temp;
         indexS=index(I);
      end
   else
      if temp<mintemp2
         mintemp2=temp;
         indexR=index(I);
      end
   end
end

[b]=
AENcalculateScalarBForNonLinSep(X,y,alpha,indexR,indexS);

AENcalNonLinPoints;


------------------------------------------------------

function [X]= AENkernel1(x,y)


  %ulineær sparasjon
  X=((x'*y)+1)^6;

------------------------------------------------------

function [F]= AENfun1(xcl,x,y,alpha,b,index)


N=length(index);
temp=0;
for I=1:N,

   temp= temp+
alpha(index(I))*y(index(I))*AENkernel1(xcl,x(:,index(I)));
end

F=temp+b;


------------------------------------------------------


function [X]=
AENcalculateScalarBForNonLinSep(x,y,alpha,indexR,indexS)

sizeX=size(x);
numXvectors=sizeX(2);

X=0;
for I=1:numXvectors,

   Ks= AENkernel1(x(:,I),x(:,indexS));
   Kr= AENkernel1(x(:,I),x(:,indexR));


   X= X+  y(I)*alpha(I)*(Ks+Kr);

end

X=X*(-0.5);


------------------------------------------------------

function [G] = AENMatrixG(y,x)


sizeX=size(x);
numXvectors=sizeX(2);

G=0;
for I=1:numXvectors,
   for J=1:numXvectors,

      G(I,J)=y(I)*y(J)*AENkernel1(x(:,I),x(:,J));

   end

end

------------------------------------------------------------
---------------------------------------
function [alpha]=AENGenAlpha(X,y,c)


sizeX=size(X);
numXvectors=sizeX(2);
A=0;b=0;Aeq=0;beq=0;g=0;G=0;


[G] = AENMatrixG(y,X);

%genererer g-vektoren
for I=1:numXvectors,
   g(I)=-1;
end


%genererer A-matrisen
A=0;
for I=1:numXvectors,
   for J=1:numXvectors,
           if I==J
       A(I,J)=1;
    else
       A(I,J)=0;
    end
         end
end

for I=1:numXvectors,
   for J=1:numXvectors,
           if I==J
       A(I+numXvectors,J)=-1;
    else
       A(I+numXvectors,J)=0;
    end
         end
end



%genererer b-vektoren

for I=1:numXvectors,
   b(I)=c;
end
for I=1:numXvectors,
         b(I+numXvectors)=0;
end



 Aeq=y';
 beq=0;

 [alpha,FVAL,EXITFLAG] = quadprog(G,g,A,b,Aeq,beq)

------------------------------------------------------------
-----------------------------
%AENcalNonLinPoints.m

xcl=0;
D=0;
M=0;
x1=-3:0.25:4;
N1=length(x1);
x2=-3:0.25:6;
N2=length(x2);
for I=1:N1,
   for J=1:N2,
      xcl(1,1)=x1(I);xcl(2,1)=x2(J);
      [D]=AENfun1(xcl,X,y,alpha,b,index);
M(J,I)=D;

   end

end
contour(x1,x2,M,[0 0],'k');
hold on
x1pos=[1.5 2.5 1.0 2.5 1.5 ]';
x2pos=[1.5 2.0 2.5 3.0 3.5 ]';
plot(x1pos,x2pos,'k+')

x1neg=[2.5 3.5 3.5 3.5 2.5 2.5 1.5 ]';
x2neg=[1.0 1.5 2.5 3.5 4.0 5.0 4.5 ]';
plot(x1neg,x2neg,'k.')

x1sv=[1.5 2.5 1.5 2.5 2.5 3.5]';
x2sv=[1.5 2.0 3.5 1.0 4.0 3.5]';
plot(x1sv,x2sv,'ks')

xlabel('x1')
ylabel('x2')
hold off
```

**Litteratur**

(1) Luger, F.G., og Stubblefield, W.A. 1999. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving, third edition.* Addison Wesley Longman.

(2) Quinlan, J.R. 1993. *C4.5: Programs for machine learning.* Morgan Kaufman Publishers.

(3) Schölkopf, B., Burges, C., og Vapnik, V. 1995. Extracting Support Data for a Given Task, i *Proceedings, First International Conference on Knowledge Discovery & Data Mining*. AAAI Press, Menlo Park, CA, 252-257.

(4) Schölkopf, B., Vapnik, V., Sung, K., Burges, C., Girosi, F., Niyogi, P., og Poggio,T. 1996. *Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers*. A.I Memo No.1599, C.B.C.L. Paper No. 142.

(5) Christopher J.C. Burges. 1998. *A Tutorial on Support Vector Machines for Pattern Recognition.* Data Mining and Knowledge Discovery 2, 121-167.

(6) Wasserman, P.D. 1989. *Neural Computing: Theory and Practice*. Van Nostrand Reinhold.

(7) Cybenko, G.1989. Approximation by superpositions of a sigmoidal function, *Mathematics of Control Signals and Systems*, 2,303-314.

(8) Cortes, C., og Vapnik, V. 1995. Support-Vector Networks. *Machine Learning*, 20, 273-297.

(9) Rumelhart, D.E., Hinton, G.E., og Williams, J.R. 1986. Learning Internal Representation by error propagation. *Parallell Distributed Processing: Explorations in the Microstructures of Cognition, vol. 1: Foundations*, Rumelhart, D.E, et al. eds., MIT Press, Cambridge, Mass., 318-362.

(10) Blake, C.L. og Mertz, C.J.1998: UCI repository of machine learning databases, University of California, Department of Information and Computer Science. URL: http:// www.ics.uci.edu/~mlearn/MLRepository.html.

# FORDELINGSLISTE

**FFISYS** **Dato:** 23 august 2001

| RAPPORTTYPE (KRYSS AV) | RAPPORT NR. | REFERANSE | RAPPORTENS DATO |
|---|---|---|---|
| X RAPP ☐ NOTAT ☐ RR | 2001/01501 | FFISYS/806/161 | 23 august 2001 |
| RAPPORTENS BESKYTTELSESGRAD | | ANTALL EKS UTSTEDT | ANTALL SIDER |
| UGRADERT | | 39 | 105 |
| RAPPORTENS TITTEL | | FORFATTER(E) | |
| MASKINLÆRINGSTEKNIKKER FOR KLASSIFISERING | | NILSEN Asgeir Egil | |
| FORDELING GODKJENT AV FORSKNINGSSJEF: | | FORDELING GODKJENT AV AVDELINGSSJEF: | |

## EKSTERN FORDELING

| ANTALL | EKS NR | TIL |
|---|---|---|
| 1 | | Prof A Aamodt, NTNU |
| 1 | | Prof O Hallingstad, UniK |
| 1 | | Prof H R Jervell, UiO |
| 1 | | Prof II R A Fjellheim, UniK |
| 1 | | KK J K Nyhus, FSTS |
| 1 | | Lufttjenesteinspektoratet |
| 1 | | Asgeir E Nilsen<br>Plahteskogen 12<br>1363 Høvik<br><br>www.ffi.no |

## INTERN FORDELING

| ANTALL | EKS NR | TIL |
|---|---|---|
| 14 | | FFI-Bibl |
| 1 | | Adm direktør/stabssjef |
| 1 | | FFIE |
| 5 | | FFISYS |
| 1 | | FFIBM |
| 1 | | FFIN |
| 1 | | R H Solstrand, FFISYS |
| 1 | | B E Bakken, FFISYS |
| 1 | | J E Torp, FFISYS |
| 1 | | F A Dahl, FFISYS |
| 1 | | J M Gilljam, FFISYS |
| 1 | | O Graasvoll, FFISYS |
| 1 | | O M Halck, FFISYS |
| 1 | | T Kråkenes, FFISYS |
| 1 | | I Dyrdal, FFIBM |
| | | FFI-veven |

FFI-K1 Retningslinjer for fordeling og forsendelse er gitt i Oraklet, Bind I, Bestemmelser om publikasjoner for Forsvarets forskningsinstitutt, pkt 2 og 5. Benytt ny side om nødvendig.