CHALMERS

GÖTEBORGS UNIVERSITET

Maskininlärning för klassificeringsalgoritmer

En jämförande studie av linjär regression, Winnow, Perceptron och artificiella neurala nätverk

Kandidatarbete inom civilingenjörsutbildningen vid Chalmers

Rebecca Gedda
Martin Gullbrandson
Aron Ivarsson
Wenjin Yuan

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2019

Maskininlärning för klassificeringsalgoritmer

En jämförande studie av linjär regression, Winnow, Perceptron och artificiella neurala nätverk

Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk matematik vid Chalmers
Rebecca Gedda Martin Gullbrandson

Kandidatarbete i matematik inom civilingenjörsprogrammet Maskinteknik vid Chalmers
Aron Ivarsson

Kandidatarbete i matematik inom civilingenjörsprogrammet Bioteknik vid Chalmers
Wenjin Yuan

Handledare: Larisa Beilina
Examinator: Ulla Dinger & Maria Roginskaya

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2019

Populärvetenskaplig presentation

Maskininlärning har under de sista åren väckt mycket stort intresse inom teknikvärlden men det har även blivit något av ett vardagsämne i hemmen. Men vad är då maskininlärning och vad finns det egentligen för begränsningar med det? Maskininlärning innebär framförallt att en dator får se en mängd exempel och att dessa sedan hjälper datorn att känna igen nya saker, eller fatta nya beslut tack vare sin tidigare erfarenhet.

Ett av de vanligaste sättet för inlärning inom den klassiska maskininlärningen är det som kallas övervakat lärande (eng. *supervised learning*). Det fungerar på många sätt som när ett barn lär sig att känna igen nya saker, så som exempelvis en lampa. Barnet pekar på en lampa och noterar dess form, storlek, värme, ljus och mycket mer och föräldern säger ”lampa”. Senare kanske barnet pekar på en stol och frågar ”lampa”? Föräldern svarar då och säger ” Nej, stol”. Allt eftersom detta upprepas med olika saker lär sig barnet att känna igen lampor av alla möjliga former och storlekar och behöver inte längre att någon förklarar vad det är. Barnet har lärt sig något. Det har lärt sig att använda sin tidigare erfarenhet till att göra bedömningar av helt nya föremål. På liknande sätt fungerar övervakat lärande för maskininlärning. Ett program får se ett exempel och ger ett förslag på vad det är. Är förslaget fel så korrigeras bedömningen något och programmet fortsätter med ett nytt exempel. Detta upprepas för en mängd olika exempel, och efter en viss tid har programmet lärt sig att känna igen de karakteristiska dragen för de saker som den sett, och kan på så sätt känna igen helt nya objekt av samma kategori. Hur själva lärandet går till och principerna bakom hur datorn lär sig att känna igen framtida exempel kan variera kraftigt och det finns många olika algoritmer som flitigt används under olika omständigheter och beroende på vad som ska läras.

Några av algoritmerna som används inom den klassiska maskininlärningen är linjär regression, Winnow, Perceptron och artificiella neurala nätverk vilka också är de algoritmer som behandlas i studien. Dessa algoritmer fungerar alla på olika sätt men syftar till att uppnå samma sak, att få en så bra bedömning av framtida exempel som möjligt baserat på tidigare exempel. Dessa fyra algoritmer förklaras och beskrivs i viss detalj i rapporten och deras tillförlitlighet undersöks och jämförs. En sak som visas är att hur exemplen ser ut kan kraftigt påverka resultatet och hur bra datorn lär sig att känna igen framtida exempel. Man kan även se att vissa tillämpningar innebär att specifika algoritmer tycks mer lämpade. Utöver det visas även vissa kända metoder för förbättring av grundmetoderna som också utvärderas och jämförs. Det framkommer att alla algoritmerna har stor potential, och för tillämpningar med komplexa data och då stor noggrannhet krävs finns många små korrigeringar, förändringar och kombinationer som kan ge ytterligare förbättrat resultat. Korrigeringarna är dock beroende på varje tillämpning och det är svårt att ge för mycket generella direktiv, algoritmerna får istället anpassas till varje situation.

I studien utvärderas algoritmerna på två fall då datan är genererad och relativt enkel. För denna data ses att alla algoritmerna kan i de flesta fall uppnå en felfri klassificering. I det tredje fallet är datan hämtad från en databas för maskininlärning där datan representerar patienter på en dermatologisk klinik och där klassifikation motsvarar korrekt diagnos. I det fallet kan en klassificering av korrekt diagnos ske vid $97,2$ procent av fallen och under vissa antagande ända upp till $98,6$ procent.

Dessa resultat visar vilken kraftig och tydlig potential som finns för maskininlärning inom många fält. Rapporten är därför uppmuntrande för vidare utveckling och tillämpning av algoritmerna samtidigt som den visar att det finns många existerande metoder som kan höja resultaten ytterligare även utan att byta till nya ännu mer komplexa algoritmer.

Sammanfattning

Rapporten undersöker fyra av de mest grundläggande klassificeringsalgoritmerna för maskininlärning: linjär regression, Perceptron-algoritmen, Winnow-algoritmen och en flerlagersperceptron (eng. *multilayer Perceptron*) som är en typ av artificiellt neuralt nätverk. Den matematiska bakgrunden för samtliga algoritmer sammanfattas och ger algoritmerna en tydligare matematisk koppling. Vidare implementeras samtliga algoritmer från grunden i `MATLAB`, där koden går att finna i appendix.

Huvudsakligen används andel felklassificeringar som mått på algoritmernas resultat, utöver detta används precision (eng. *precision*) och täckning (eng. *recall*) som komplementerande mått. Algortimernas förmåga att klassificera jämförs genom klassifikation av två simulerade datamängder och därefter dermatologisk data hämtad från en offentlig databas. De simulerade datamängderna skapas i `MATLAB` och valda så att de introducerar stegvis vissa svårigheter för algoritmerna. Medianen av andelen felklassificerade exempel uppgår för båda simulerade datamängder till 0 % för samtliga algoritmer. Utöver detta lyckas de fyra algoritmerna uppnå ett medianfel på 2.78 % på den dermatologiska datan även om viss spridning och vissa skillnader mellan algoritmerna kan noteras.

Slutligen implementeras två resultathöjande algoritmer och dessa resultat utvärderas i kontrast till resultaten utan dessa.

Abstract

This report examines four basic machine learning algorithms for classification problems: linear regression,the Precptron-algorithm, the Winnow-algorithm and a type of artificial neural network called multi-layer Perceptron. The mathematical foundation of these algorithms are developed throughout the report and provides the reader with a better understanding of the algorithms. Furthermore the algorithms are implemented in `MATLAB` and the code is availible in appendix for reference.

The proportion of missclassified data is used as the main indicator of an algorithms performance, however the precison and recall of the algorithms is also taken into account. Three different data sets are tested: two data sets are simulated in `MATLAB` and a third dermatologic data set is gathered from a public database. These data sets are used to evaluate the performance of the algorithms. The simulated sets are chosen such that they introduce certain difficulties for the classification of the data.

Finally, two performance boosting algorithms are implemented and tested for each classification algorithm. The result of using the boosting algorithms are compared to the results of not using them.

The median of the proportion of missclassified examples is for both simulated data sets 0 % for all algorithms. Moreover the four algorithms achieves a median error of 2.78 % for the dermatologic data set even if some spread and differences between the algorithms are noted.

Innehåll

Förord

Denna rapport är ett kandidatarbete vid Chalmers Tekniska Högskola, skriven av Rebecca Gedda, Martin Gullbrandson, Aron Ivarsson och Wenjin Yuan. De indviduella prestationerna redogörs i en loggbok[1] och sammanfattas här.

Rebecca är huvudförfattare av inledningen, linjär regression, prestandamått, klassificeringsresultat och inverkan av avgränsningar, samt är medförfattare av artificiella neurala nätverk. Utöver dessa texter har Rebecca ansvarat för filhantering (SVN), rapportens struktur och hantering av referenser.

Martin är huvudförfattare av beskrivning av använd data, Winnow, Perceptron och diskussionen, samt är medförfattare av metaalgoritmer, klassificeringsresultat och antal attributs påverkan på inlärningen. Martin har också haft ansvarat för genomförandet av simulerad data, En-mot-Alla, Winnow, Perceptron och visualisering av algoritmerna och resultat. Tillsammans med Aron har Martin testat och gjort urval av algoritmer utöver de fyra som ingår i projektbeskrivningen.

Aron är huvudförfattare av populärvetenskapliga presentationen och artificiella neurala nätverk samt är medförfattare av inledning och diskussion. Dessutom har Aron ansvarat för genomförandet av algoritmerna: linjär regression, Winnow, ANN, *k*-NN, Lrw, Test Train, Test Train Num samt Vandermonde. Han också utvärderat ytterligare algoritmer och tillsammans med Martin gjort urval inför rapporten. Aron har också lagt extra mycket tid på att studera den litterära bakgrunden till algoritmerna och gjort urval för innehållet till rapporten.

Wenjin är huvudförfattare av metaalgoritmer och antal attributs påverkan på inlärningen samt medförfattare av populärvetenskapliga presentatonen, beskrivning av använd data och klassificeringsresultat.

Korrekturläsning och textrevidering har gjorts tillsammans i grupp. Avslutningsvis vill vi tacka vår handledare, Larisa Beilina, för stöd genom projektets gång. Vidare vill vi även tacka fackspråk för språklig återkoppling.

[1] Med loggbok menas här gruppens dagbok och tidslogg.

1 Inledning

Klassifikationer görs varje dag baserat på olika aspekter och kan hjälpa oss att ta beslut. Ett enkelt exempel kan vara hur en frukt klassificeras som bra eller dålig, varav en sedan väljer om den vill ha frukten eller inte. I ett samhälle med mer och mer data blir klassificering allt viktigare då det hjälper oss att ta beslut inom många olika tillämpningssområden. Med en stor mängd data kan en dator läras att fatta belsut baserat på tidigare information. Lärandet kan göras på två sätt: oövervakat eller övervakat lärande (eng. *unsupervised* respektive *supervised learning*). Undersökningen baseras på övervakat lärande, där datorn tränas med hjälp av datasamlingar som är förklassificerade. Under senare delar av 1990-talet och framåt görs framsteg inom maskininlärning, där bildigenkänning är ett framträdande exempel. 2012 lyckas en dator att känna igen katter utifrån bildmaterial på internet med hjälp av bildigenkänningstekniker, som är ett exempel på hur klassificering med hjälp av maskininlärning kan användas. Bildklassificering är ett av flera områden för maskininlärning med många spännande applikationer, inom allt från att hitta vänner i bilder till att detektera hinder för självkörande bilar eller diagnostisera sjukdomar och mycket mer.

Maskininlärning är en nyare vetenskap som succesivt har vuxit fram ur matematiken och implementeras senare med hjälp av datorer. Under början av 1900-talet raffineras och formuleras statistiska metoder som utgör konceptuella pelare inom maskininlärning [1] och databehandling. Men hjälp av minstakvadratmetoden etableras *linjär regression* som snabbt klassificerar datakluster algebraiskt. Mellan 1950 och 1970 tar den så kallade AI-vintern plats och utveckling avstannar. Även om intresset för maskininlärning svalnade under AI-vintern introducerade Frank Rosenblatt 1958 *Perceptron-algortimen* [2]. Algoritmen bearbetar, till skillnad från linjär regression, en datapunkt i taget och anpassar klassificeringsalgoritmen itterativt. Nick Littlestone tog 1989 fram *Winnow-algoritmen* [3] som fick sitt namn från en sållningsmetod som används på sädesfält. För att skilja agnar från veten kastas säden upp i luften och vinden för bort axen medan kornen faller ner. Under 1990-talet tar maskininlärningen fart på nytt då datorvetenskapen etableras och datorer kan användas för att implementera algoritmer, vilket förenklar de tidigare krävande beräkningarna. Under 2000-talet har algortimer som försöker immitera tankeprocessen hos människor vuxit fram. *Artificiella neurala nätverk* bygger på en beslutstagningsprocess som liknar den som sker i neuroner hos människor.

Utöver de grundalgoritmer som används inom maskininlärning så har flera metoder utvecklats för att förbättra algoritmernas resultat. Många av dessa metoder kan implementeras på en mängd olika algoritmer men resultaten av förbättringen kan variera kraftigt mellan fallen. Tre av algoritmerna, linjär regression, Winnow samt Perceptron, är av sin natur binära, vilket innebär att de enbart kan klassificera två klasser. En-Mot-Alla-metoden (eng. *One vs. All*) generaliserar binära klassificeringsalgoritmer till att klassificera flera klasser. Några exempel på andra typer av förbättringar är k-NN (k-Nearest-Neighbour) som kan användas till förbehandling av datasamlingar, Bagging (härstammar från *bootstrap aggregating*) som bygger på flera klassificeringsförsök [4], och adaptiv inlärningshastighet [5].

För att jämföra prestandan hos algoritmerna linjär regression, Winnow, Perceptron samt artificiella neurala nätverk finns ett behov av att tydligt kunna utvärdera prestandan för respektive algoritm. I många tillämpningar räcker det inte med att endast mäta andelen fel då det inte är givet att alla fel har samma betydelse och då datan kan ha mycket olika utformning. En applikation som letar efter sällsynta händelser kan få väldigt bra andel fel (eng. *error rate*) om den alltid klassificerar att händelsen inte uppträtt, trots att en sådan applikation skulle vara helt oanvändbar. Därför används täckning (eng. *recall*) och precision (eng. *precision*) i stor grad som komplementerande mått på prestandan till andelen fel. Utöver dessa prestandamått finns en mängd olika saker att ta i beaktning som algoritmkomplexitet, känslighet för datans utformning etc.

Tidigare studier på de individuella algoritmerna har genomförts [6, 7], men i det snabbt växande fältet av maskininlärning är det av stor vikt att veta hur bra resultat som kan väntas, och att känna till grunderna till de algoritmer som används, både för att kunna välja rätt algoritm och för kunna göra att en betydande analys av resultatet. Vidare är det viktigt att känna till några av de förbättringsmöjligheter som existerar för algoritmerna och hur dessa kan påverka resultatet av en given algoritm.

1.1 Syfte

Syftet med studien är att förbättra användningen av algoritmer för maskininlärning genom att skapa förståelse kring hur fyra av de klassiska klassificeringsalgoritmerna, linjär regression, Winnow, Perceptron och artificiella neurala nätverk, klarar av att klassificera datasamlingar. Studien försöker således utvärdera tillförlitligheten av algoritmerna under olika förhållanden och skapa en jämförelse mellan algoritmerna baserat

på valda prestandamått. Förhoppningen med studien är en ökad förståelse för vid vilka omständigheter varje algoritm är att föredra, och vilka resultat som kan väntas.

1.2 Problem och uppgifter

I undersökningen implementeras, jämförs och analyseras följande algoritmer: linjär regression, Winnow, Perceptron och ANN (artificiella neurala nätverk). Vid analys undersöks tillförlitlighet, applicerbarhet och komplexitet med hjälp av vissa valda prestandamått. De huvudsakliga prestandamåtten som används är andel fel, täckning och precision.

Då linjär regression, Winnow och Perceptron alla är binära klassificerare, implementeras även en övergripande metod som kallas En-mot-Alla-metoden. Denna metod implementeras för att klassificera fler än två klasser med de binära klassificerarna. En-mot-Alla behöver dock inte användas vid utvärdering av ANN då algoritmen kan klassificera ett godtyckligt antal klasser i sitt grundutförande.

Utöver En-mot-Alla så implementeras två resultathöjande metoder och resultaten från dessa utvärderas. De två metoderna är Bagging och förbehandling av data med k-NN. För att ge en ökad förståelse kring algoritmerna presenteras en del av matematiken och grunderna bakom varje algoritm.

Ofta är data samlad från verkligheten relativt komplex och det kan vara svårt att få en bra överblick över och att visualisera hur algoritmerna fungerar. På grund av detta simuleras egen data i ökande komplexitet för att ge en bättre förståelse och överblick över logiken bakom algoritmerna. Efter detta utvärderas algoritmerna även på mer komplex data hämtad från verkligheten.

1.3 Avgränsningar

Vissa avgränsningar görs för studien. Algoritmerna utvärderas på tre olika dataset, två genererade och ett dataset som beskriver vissa attribut hos patienter vid en dermatologiklinik, där klasserna representerar olika medicinska diagnoser. De två genererade dataseten är separerbara medans det är okänt huruvida dermatologiska datan är separerbar eller inte. Då ANN utvärderas, görs detta endast för en specifik typ av ANN kallad flerlagersperceptron (eng. *Multi-Layer Perceptron*) med ett lager dolda neuroner. Utöver detta utvärderas linjär regression, Winnow och Perceptron endast med polynom av grad ett och två. Resultaten på prestandamåtten är medianen av 100 försök. Vidare är inlärningshastigheten konstant i all undersökningar och all implementation sker i `MATLAB`.

1.4 Etiska aspekter

Studien använder insamlad data som berör männsikor. Denna data är anonymiserad och hämtad från en öppen plattform för maskininlärning. Datan väcker därav ingen etisk problematik. Vidare används datan i undersöknigen endast som statistiska datapunkter för prestandautvärdering av algoritmerna. Detta anses därför inte vara något etiskt problem. Vid implementation av algoritmerna på andra dataset uppmanas användaren till att själv verifiera testresultaten på det specifika datasetet. Undersökningens resultat är inte nödvändigtvis överförbara till andra dataset utan kontroll av resultaten.

2 Metod

Genom undersökning av de fyra algoritmerna individuellt kan en jämförelse av för- och nackdelar med algoritmerna göras. Först undersöks linjär regression som med en tydlig matematisk koppling ger en förståelse för hur klassificering kan ske. Därefter undersöks Winnow som är en grundläggande form av maskininlärning [8]. Sedan studeras en nära besläktad algoritm vid namn Perceptron. Jämföresle mellan de olika klasserna görs för att se med vilken säkerhet algorimterna klassificerarar datapunkter. För klassificering behöver punkterna vara separerbara, exempelvis linjärt separerbara. Med linjärt separerbar menas att det finns reella tal $w_1, \ldots, w_n$ så att för två dataset A och B med n attribut $\sum_{i=1}^{n} w_i x_i > \Theta$ för alla punkter som tillhör A och $\sum_{i=1}^{n} w_i x_i < \Theta$ för punkter i B där $\Theta \in \mathbb{R}$ är ett tröskelvärde. Kvadratiskt separerbara dataset kan separeras på samma sätt som de linjära med skillnaden att x_i då även innehåller kvadratiska termer så väl som blandtermer vilket ger en större flexibilitet. Vidare i rapporten syftar begreppet separerbartill linjärt eller kvadratiskt separerbar. Slutligen undersöks ANN som en vidgning av Perceptron. Samtliga algoritmer testas på klasser med minst två attribut och tillförlitligheten testas med hjälp av ett testset som är skillt från

träningsetet. Genomgående i undersökningen används också boken *An Introduction to Machine Learnig* [9] som intuitiv bakgrund till studien.

2.1 Beskrivning av använd data

Arbete med maskininlärning kräver mycket data för att både träna algoritmer men även för utvärdering. Målet är att träna algoritmerna på en given träningsmängd till att urskilja två eller flera klasser för att senare kunna ge en förutsägelse om testdata. Då reell data kan vara både arbetskrävande att införskaffa samt analysera kan simulerad data med fördel användas. Dessutom är simulerad data fullständigt kontrollerad av programmeraren och kan konstrueras på så sätt som önskas och anpassas efter specifika klassificeringsproblem. Trots vissa fördelar med simulerad data är syftet med maskininlärning att uppfylla verkliga krav. Därför har samtliga algoritmer även testats på verklig data rörande dermatologi [10].

2.1.1 Simulerad data

Under studien har data simulerats via att parametrisera ett antal funktioner för att representera klasser. Ett uniformt brus har sedan adderats till parametriseringen för att få klasser som är mindre förutbestämda. Parametriseringen sker då enligt

$$y_\sigma(t) = y(t)(1+\delta\alpha),\ t \in [0,T], \tag{1}$$

där $T \in \mathbb{R} > 0$ är slutet på parametriseringen. Bruset består av $\alpha \sim U(-1,1)$ samt brusnivån $\delta \in [0,1]$, där U är den likformiga sannoliksfördelningen. Ekvation (1) kan användas för att simulera data som är separerbara eller icke-separerbara; med mindre eller mera brus och på så sätt uppfylla olika klassificeringskrav. I det här arbetet behandlas algoritmerna i upp till andragradens polynom, därmed har linjärt samt kvadratiskt separerbara dataset simulerats.

2.1.2 Dermatologisk data

Den verkliga data som analyseras berör hudsjukdommar och hämtas från en öppen dataplattform [10]. Målet är att genom den insamlade datan avgöra vilken av de fem olika hudsjukdommar som patienten har eller om patienten är frisk. Datan har 34 olika attribut per patient, med olika aspekter så som ålder eller släktanlag. Notera att vissa patienter saknar vissa uppgifter, dessa patienter tas bort innan maskininlärningen startar.

2.2 Algoritmer

Undersökningen utvärderar fyra algoritmer: linjär regression, Winnow, Perceptron och artificiella neurala nätverk. För varje algoritm redogörs bakgrunden och teorin. Linjär regression är en optimeringsalgoritm för skapa en linje som minimerar avståndet mellan datapunkter tillhörande olika klasser. Detta kan avnändas för att klassificera datapunkter. Winnow och Perceptron däremot är två stycken feldrivna iterativa algoritmer vars inlärning sker via multiplikation samt addition. Båda algoritmerna fungerar på sådant sätt att först antas en hypotes om datapunkters klasser. Om hypotesen inte stämmer korrigeras vikterna något. Denna process upprepas för alla datapunkter tills alla är rättklassficerade då algoritmen stannar. Idé för artificiella neurala nätverk kommer från det biologiska. Nätverket består av tre delar: insignal, dolda neuroner och utsignal, vilket efterliknar det biologiska neuronnätet. I det aktuella fallet behandlas ett ANN som är en flerlagersperceptron.

2.2.1 Linjär regression

För att utföra en linjär regression för klassificeringar används matematiska verktyg som minsta kvadratmetoden och nivåytor. Med en given samling av t datapunkter med $n \in \mathbb{N}$ attribut kan punkterna placeras ut i rummet $V \subseteq \mathbb{R}^n$. Ett exempel med enbart ett attribut kan vara ålder på en person, det vill säga att en samling av åldersdata kan placeras på en rak linje. Vidare är målet att klassificera om personen är barn eller vuxen.

I många fall kan två klasser separeras med hjälp av ett polynom av grad d. Klasserna separeras då genom att klasserna befinner sig på var sin sida av en nivåyta för polynomet. Ekvation

$$p(\mathbf{a}) = w_0 + w_1a_1 + w_2a_1^2 + w_3a_1a_2 + ... + w_{(m-2)}a_na_{n-1} + w_{(m-1)}a_n + w_ma_n^2 \tag{2}$$

är ett exempel på ett sådant polynom där $d = 2$. Här är $a_j,\ j \in \{1,..n\}$ de attribut eller dimensioner i det aktuella fallet och $w_i,\ i \in \{0,..m\}$ är de vikter som ska bestämmas. Om (2) istället transformeras via variabelsubstitution till grad 1 kan polynomet skrivas

$$p(\mathbf{x}) = w_0x_0 + w_1x_1 + w_2x_2 + ... + w_{(m-1)}x_{(m-1)} + w_mx_m, \tag{3}$$

där $\mathbf{x}$ är en omskrivning av attributen till linjär form. Notera att det nu finns lika många variabler som vikter att bestämma. Detta är nödvändigt vid användning av till exempel matrismultiplikation. Målet är nu att bestämma vikterna $w_i,\ i = \{0,...,m\}$ vilket kan göras med hjälp av minstakvadratmetoden då linjär regression är baserad på samma logik. Systemet som ska lösas blir därför $\mathbf{Xw} = \bar{\mathbf{c}}$.

Antag att matrisen $\mathbf{X}$ har dimension $t \times m$ och då innehåller t stycken datapunkter alla beskrivna av de m attributen, alltså på formen

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \dots & x_{1,m} \\ 1 & x_{2,1} & x_{2,2} & \dots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{t,1} & x_{t,2} & \dots & x_{t,m} \end{bmatrix}. \tag{4}$$

Klasstillhörigheterna för datan i $\mathbf{X}$ definieras som $\bar{\mathbf{c}}$. Notera att matrisen kan innehålla valfri bas som vill användas, i rapporten kommer däremot enbart polynom av grad ett eller två att användas.

Residualen $\mathbf{r} = \mathbf{c} - \mathbf{Xw} = \mathbf{Xw} - \mathbf{c}$ är i 2-normen ett mått på hur långt ifrån en given datapunkt är ifrån vårt polynom. En uppsättning vikter erhålls, $\mathbf{w}$, där minimeringsproblemet

$$\min_{\mathbf{w}} ||\mathbf{r}||_2^2 = \min_{\mathbf{w}} \sum_{i=1}^{n} r_i^2 = \min_{\mathbf{w}} \sum_{i=1}^{n} (\mathbf{Xw}_i - \mathbf{c})^2.$$

vill lösas, se *Numerical Linear Algebra: Theory and Applications* för härledning [11]. Om $m < t$ är systemet överbestämt då det finns fler rader än kolumner, för denna specifika applikation innebär det att det finns fler datapunkter än vikter att justera. För att lösa ekvationssystemet med avseende på $\mathbf{w}$ används normalekvationerna

$$\mathbf{Xw} = \bar{\mathbf{c}} \quad \rightarrow \quad \mathbf{X}^T\mathbf{Xw} = \mathbf{X}^T\bar{\mathbf{c}} \quad \rightarrow \quad \mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\bar{\mathbf{c}}, \tag{5}$$

där $\mathbf{w}_{\min}$ innehåller de koefficienter som minimerar avstånder mellan datapunkterna och ett polynom enligt ekvation (3). Om konditionstalet på matris $\mathbf{X}$ är lågt kan andra metoder behövas för att lösa systemet, exempelvis QR- eller SVD-uppdelning av matris $\mathbf{X}$, fördjupning kan ses i Bishops bok [12] (s. 143-145).

Den uppsättning vikter, $\mathbf{w}$ som erhålls från (5) är en yta som är anpassad för att minimera avvikelsen till datan i 2-normen. Genom att fastställa ett tröskelvärde, Θ, kan en nivåkurva eller nivåyta erhållas vilket i teorin separerar klasserna. Givet en datapunkt $\mathbf{x}$ kan en hypotetisk klasstillhörighet,

$$h(\mathbf{x}) = \begin{cases} 1 \text{ om } p(\mathbf{x}) > \Theta \\ 0 \text{ om } p(\mathbf{x}) \leq \Theta \end{cases}, \tag{6}$$

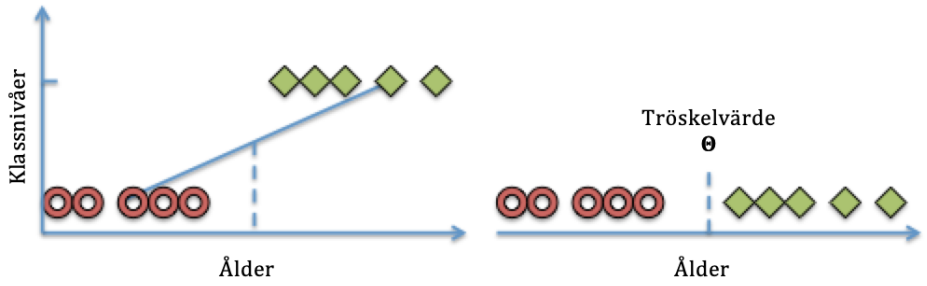
nu beräknas. Genom att förändra Θ kan olika hypoteser beräknas givet samma $\mathbf{w}$. Under studien används enbart ett konstant värde av $\Theta = 0.5$ att användas.

För att illustrera hur algoritmen fungerar beskrivs ett enkelt exempel. En mängd data innehåller information om varje individs ålder samt huruvida personen är vuxen eller ett barn. Låt nu de som är vuxna få klasstillhörigheten 1 och barnen få klassen 0. Genom att lösa ekvation (5) beräknas vikterna $\mathbf{w}$ som beskriver en linje. Figur 1 illusterar hur en sådan linje kan se ut där x-axeln visar attributet och y-axeln är klasstillhörigheten. Vidare ses i figuren hur tröskelvärdet Θ används för att i detta fallet finna en punkt som separerar klasserna, denna punkt är i det aktuella exemplet en ålder för när en individ anses som vuxen.

2.2.2 Winnow

Som namnet indikerar kan Winnow fördelaktig användas när datapunkterna har många attribut, då den snabbt kan sålla ut vilka attribut som är viktiga och förkasta de som inte används till klassificering. Algoritmen försöker iterativt finna en uppsättning vikter som effektivt separerar två klasser. För att förändra vikterna använder algoritmen en multiplikativ uppdatering av vikterna [6]. Likt linjär regression används ett polynom enligt ekvation

$$p(\mathbf{x}) = w_0x_0 + w_1x_1 + w_2x_2 + ... + w_{(m-1)}x_{(m-1)} + w_mx_m, \tag{7}$$



Figur 1: Illustration av hur linjär regression används för att optimera en linje mellan alla datapunkter på två olika klassnivåer (vänster). Brytpunkten $\Theta \in \mathbb{R}$ ansätts längs axeln för klassnivå, och i det här arbetet antar Θ ett konstant värde. Den angivna brytpunkten projiceras ner på den ursprunliga dimensionen och en klassificering erhålls (höger) .

för att separera klasserna, och målet är att bestämma vikterna i vektorn $\mathbf{w}$ av storlek $1 \times m$. En datapunkt $\mathbf{x}$ kan nu klassificeras genom att beräkna vår hypotes

$$h(\mathbf{x}) = \begin{cases} 1 \text{ om } p(\mathbf{x}) > \Theta \\ 0 \text{ om } p(\mathbf{x}) \leq \Theta \end{cases}, \tag{8}$$

där $\Theta \in \mathbb{R}$ är ett tröskelvärde som kan justeras för att uppnå bästa känslighet av kurvan. Låt $c(\mathbf{x}) \in \{0,1\}$ beteckna den riktiga klassen för punkten $\mathbf{x}$, då definieras en lyckad klassifikation som de fall då $c(\mathbf{x}) = h(\mathbf{x})$. Låt $\mathbf{w_0} = w_{i,\,0}$ där $i \in \{0,1,...,m\}$ vara en uppsättningen av slumpvist valda startvikter, som sedan uppdateras iterativt med hjälp av

$$w_{i,\,j+1} = w_i \cdot \alpha^{(c(\mathbf{x})-h(\mathbf{x}))\cdot x_i},\ i \in \{0,1,...,m\} \tag{9}$$

för att träna vikterna. Låt $\alpha \in \{\mathbb{R} > 1\}$ vara en typ av förändringsfaktor och $[c(\mathbf{x}) - h(\mathbf{x})] = e(\mathbf{x}) \in \{-1,0,1\}$ avgör riktigningen för korrigeringen. Om $e(\mathbf{x}) = 0$ stämmer hypotesen $h(\mathbf{x})$ i ekvation (8) och ekvation (9) medför då ingen förändring av vikterna. Stämmer hypotesen inte kommer däremot vikterna justeras för att förbättra polynomet. Om polynomet kan separera klasserna korrekt konvergerar algoritmen mot en lösning. Notera att ekvation (9) multiplicerar varje enkilt attribut i exponenten. Implementering kan ses i appendix B.4. De attribut som är 0 kommer alltså inte att förändras. Samtidigt, i och med den multiplikativa strukturen, förändras vikterna fort vilket ger den sållande egenskapen som algoritmen är döpt efter.

2.2.3 Perceptron

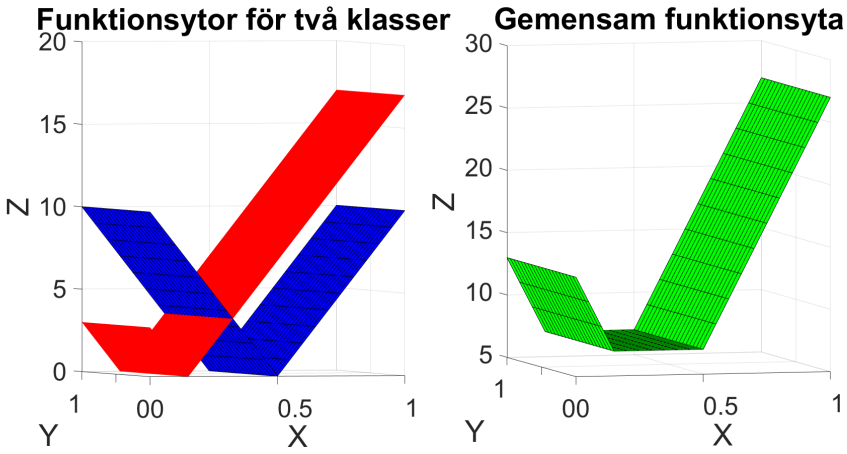
Likt Winnow är Perceptron en binär klassificeringsalgoritm som iterativt kan åstadkomma ett polynom som skiljer två klasser åt. På samma sätt som i Winnow struktureras ett klassificeringspolynom och implementering kan ses i appendix B.5. Genom att sedan transformera polynomet till ett första gradspolynom

$$p(\mathbf{x}) = w_0x_0 + w_1x_1 + w_2x_2 + ... + w_{(m-1)}x_{(m-1)} + w_mx_m,$$

fås en uppsättning vikter w_i att bestämmas utifrån indata x_i där $i \in \{0,...,m\}$. Notera att polynomet har samma form som ekvation (3). Fortsatt beräknas en hypotes h om klasstillhörighet, notera likheten med ekvation (8) för Winnow, låt $h(\mathbf{x})$ definieras som

$$h(\mathbf{x}) = \begin{cases} 1 \text{ om } p(\mathbf{x}) > 0 \\ 0 \text{ om } p(\mathbf{x}) \leq 0 \end{cases}$$

och representerar vår hypotes. Låt nu $c(\mathbf{x})$ vara den faktiskta klassen för datapunkten $\mathbf{x}$, alltså $c(\mathbf{x}) \in \{0,1\}$. Fel och riktning för korrigering beräknas på samma sätt som för Winnow, alltså $[c(\mathbf{x}) - h(\mathbf{x})] = e(\mathbf{x}) \in \{-1,0,1\}$ där $e(\mathbf{x}) = 0$ betyder att hypotesen stämmer.



Figur 2: Visualisering av funktionsytor för de 2 klasserna separerade. Notera hur gradienten är samma för de olika ytorna i kanterna (vänster). Visualisering av den gemensamma funktionsytan för minimeringsproblemet (höger). I detta fallet är det tydligt att ett minimum existerar, vidare ses den ytan där minimum finns sammanfaller med den yta som separerar klasserna.

Vikterna förbättras nu iterativt så att polynom (2.2.3) separerar de två klasserna. Låt de initiala vikterna $\mathbf{w_0} = w_{i,0}\ i \in \{0,...,m\}$ vara slumpvist valda vikter, till exempel kan en normalfördelning användas. Uppdatera vikterna iterativt genom att applicera

$$w_{i,j+1} = w_{i,j} + \eta e(\mathbf{x}) x_i\ i \in \{0,1,...,m\}. \tag{10}$$

Notera att när $e(\mathbf{x}) \neq 0$ uppdateras vikten, η är då inlärningshastigheten. Ekvation (10) finner ett separerande polynom förutsatt att de två klasserna är separerbara vid den grad som studeras [8].

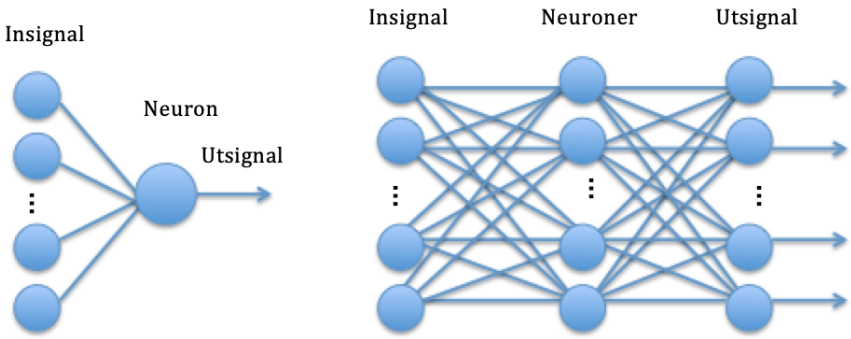
Det finns stora likheter mellan (10) och optimeringsmetoden gradientsökning,

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \alpha_o \nabla f(\mathbf{a}_n),$$

som också är iterativ [13]. Steglängden $\alpha_o \in \{\mathbb{R} > 0\}$, till skillnad från steglängden i Winnow där $\alpha > 1$, eftersom uppdateringen är additativ. Gradienten i den klassiska optimeringsmetoden motsvaras i (10) av $e(\mathbf{x})x_i$. Den specifika termen $e(\mathbf{x})$ kan brytas ner till riktning som grundas på om och hur vår hypotes skiljde sig mot det korrekta svaret, x_i ser då till att varje dimension justerar polynomet i rätt riktning. Inlärningshastigheten $\eta \in \{\mathbb{R} > 0\}$ motsvarar steglängden, α_o, och dikterar hur långt längs gradienten varje steg går.

Det blir här tydligt att klassifikation av två klasser görs om till ett optimeringsproblem med hjälp av Perceptron. Funktionsytan för optimeringsproblemet har då en diskret gradient motsvarande av $e(\mathbf{x})$, vilket innebär att dess funktion går att beräkna. Låt studera ett enkelt exempel där två klasser representeras av varsin rak linje i xy-planet, fallet är trivialt linjärt separerbar sålänge linjerna är parallella men inte sammanfaller. Utifrån $e(\mathbf{x})$, det vill säga gradienten, kan figur 2 konstrueras. Klasserna representeras här av den linje för varje yta där gradienten är 0. Ytorna adderas med varandra och genrerar ytan till höger i figur 2, notera att det finns en tydlig area med minimum för funktionsytan. När Perceptron tränas studeras varje punkt enskilt. För varje enskild punkt behöver lösningen inte befinna sig i den yta som i figur 2 är minimum. Om Perceptron tvingas iterera tills att $e(\mathbf{x}) = 0$ för alla punkter bergränsas ytan till minimum i figur 2. Ett minimum för optimiseringsproblemet eller ett polynom som separerar klassernna i fallet av ett klassifikationsproblem har då funnits.

Perceptron är alltså en binär klassificeringsalgoritm som kan åstadkomma en funktion som skiljer två klasser åt. Oftast hanteras data som är mer komplexad än enbart två klasser och ytterligare hjälpalgoritmer kan då användas. Perceptroner kan också kombineras i ett flerlagersnätverk. Ett sådant system kallas för ett artificiellt neuralt nätverk och återkommer senare i rapporten. Skillnaden på uppsättning mellan Perceptron och den av ett artificiellt neuralt nätverk kan ses i figur 3.



Figur 3: Illustration av Perceptron (vänster) och artificiellt neuralt nätverk (höger). Notera hur ANN består av en sammansättning av Perceptroner, och bildar en så kallad flerlagersperceptron.

2.2.4 Artificiella neurala nätverk

ANN (eng. *Artificial Neural Networks*) är ett samlingsnamn på algoritmer för maskininlärning som försöker efterlikna ett biologiskt neuralt nätverk. Algoritmer för ANN gjordes populära genom Rumelhart och McClellands publikation [14]. Det finns en mängd olika sorters artificiella neurala nätverk som alla bygger på något olika principer men ett av de vanligaste nätverken är en så kallad flerlagersperceptron. I den här studien utvärderas endast en flerlagersperceptron. För en mer omfattande beskrivning av olika typer av ANN samt en mer detaljerad beskrivning av flerlagersperceptronen så hänvisas till *Artificial Neural Networks A Practical Course* [15].

I ett enkelt artificiellt neuralt nätverk finns tre delar; insignal, dolda neuroner och utsignal enligt figur 3. Insignalerna är alla attribut som tillhör ett givet träningsexempel $x_{ij}^{(1)}$, där $i \in \{1,...,t\}$ och $j \in \{1,...,n\}$. Index i representerar vilket träningsexempel som används och j attribut. Exponenten indikerar att det första lagret studeras, i det här fallet insignalerna i figur 3. Träningsexemplet representeras av attributvektorn $\mathbf{x}_i^{(1)} = [x_{i1}^{(1)}, x_{i2}^{(1)}, x_{i3}^{(1)}, ..., x_{in}^{(1)}]$ samt en indikator för vilken klass exemplet tillhör. Klassen för ett träningsexempel betecknas $c(\mathbf{x}_i^{(1)}) = c_s$, där $s \in \{1,...,K\}$ är mängden av olika klasser. Detta gör att matrisen $\mathbf{X}^{(1)}$ med alla attribut för samtliga träningsexempel i träningssetet får utseende

$$\mathbf{X}^{(1)} = \begin{bmatrix} x_{11}^{(1)} & x_{12}^{(1)} & x_{13}^{(1)} & \dots & x_{1n}^{(1)} \\ x_{21}^{(1)} & x_{22}^{(1)} & x_{23}^{(1)} & \dots & x_{2n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \\ x_{t1}^{(1)} & x_{t2}^{(1)} & x_{t3}^{(1)} & \dots & x_{tn}^{(1)} \end{bmatrix}, \tag{11}$$

där t är antalet träningsexempel och n är antalet attribut som beskriver varje träningsexempel. Vanligtvis normeras attributvektorn då värdena på de olika attributen kan vara mycket olika beroende på vad attributen representerar. Normering görs enligt följande beräkning

$$z_{ij} = \frac{x_{ij}^{(1)} - \min(\mathbf{x}_i^{(1)})}{\max(\mathbf{x}_i^{(1)}) - \min(\mathbf{x}_i^{(1)})},$$

och gör att varje attribut väger lika mycket. För enkelhets skull kommer träningsexemplen i fortsättning antas redan vara normerade. Detta antagande görs för att minska antalet beteckningar så i fortsättningen antas följande: $x_{ij}^{(1)} = z_{ij}$.

Insignalerna $\mathbf{x}_i^{(1)}$ multipliceras med korresponderande vikter $\Omega^{(1)}$ som är en matris av storlek $p \times n$ och beskriver sambandet mellan första och andra lagret. Detta summeras därefter in till en dold neuron d_k, $k \in \{1,...,p\}$ där p betecknar antalet dolda neuroner. Hur många dolda neuroner som bör användas beror på komplexiteten hos klassificeringsproblemet där fler dolda neuroner innebär möjlighet till en mer flexibel lösning. En nackdel med ökat antal dolda neuroner är att beräkningstiden ökar, då fler vikter behöver tränas, och fler beräkningar behöver utföras. Vidare finns även risk för överanpassning (eng. *over fitting*)

vilket innebär att nätverket lyckas klassificera träningsexemplen väl utan att nödvändigtvis klara framtida testexempel lika bra på grund av en allt för komplex beslutsyta. På grund av svårigheten att fastställa ett optimalt antal dolda neuroner har algoritmer tagits fram för att göra just detta. Ash skrev redan 1989 en artikel [16] om hur optimalt antal noder kan bestämas.

Matrisen $\Omega^{(1)}$ med de individuella vikterna $\omega_{kj}^{(1)}$ till det dolda lagret kan skrivas

$$\Omega^{(1)} = \begin{bmatrix} \omega_{11}^{(1)} & \omega_{12}^{(1)} & \omega_{13}^{(1)} & \dots & \omega_{1n}^{(1)} \\ \omega_{21}^{(1)} & \omega_{22}^{(1)} & \omega_{23}^{(1)} & \dots & \omega_{2n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \\ \omega_{p1}^{(1)} & \omega_{p2}^{(1)} & \omega_{p3}^{(1)} & \dots & \omega_{pn}^{(1)} \end{bmatrix},$$

där index k indikerar vilken dold neuron som vikterna hör samman med och index j vilket attribut som det skall multipliceras med. Notera att antalet dolda neuroner kan, och brukar, skilja sig från antalet attribut.

Första steget i det neurala nätverket är multiplikationen mellan attributen och vikterna vilket görs enligt

$$\Sigma_k = \omega_k^{(1)} \left(x_i^{(1)}\right)^T,$$

där $(\mathbf{x}_i^{(1)})^T$ är transponatet av attributvektorn som beskriver träningsexempel x_i. I ekvation (2.2.4) är $\omega_k^{(1)}$ den rad av vikter som motsvarar varje dold neuron d_k. Detta gör att Σ_k är ett skalärt värde som används i överföringsfunktionen för neuron d_k och Σ är en vektor av storlek $1 \times p$ med element Σ_k. I den dolda neuronen används en överföringsfunktion vilken tar summan och ger i vanliga fall ett värde mellan 0 och 1. I studien används den historiskt vanliga överföringsfunktionen den så kallade Sigmoidfunktionen,

$$f(\Sigma_k) = \frac{1}{1+e^{-\Sigma_k}}. \tag{12}$$

En nackdel med denna överföringsfunktionen är att derivatan går mot noll då värdet av funktionen närmar sig antingen 0 eller 1. Detta påverkar senare hur snabbt nätverket närmar sig en lösning. För att motverka detta föreslår Kubat [9] (s. 91-106) att man kan använda endast den del av Sigmoidfunktionen som har en förhållandevis konstant derivata. Ett vanligt alternativ är den stegvis linjära överföringsfunktionen ReLU utöver denna finns vidare en mängd olika överföringsfunktioner och en mer detaljerade jämförelse och utvärdering av dessa kan läsas i en rapport av Yogitha och Mathivanan [17].

Efter att summan gått igenom överföringsfunktionen fås återigen ett värde på intervallet $[0,1]$ precis som för insignalerna. Detta gör att värdena i de dolda neuronerna kan ses som nästa lagers insignaler eller attribut som här betecknas $x_k^{(2)}$ för ett givet träningsexempel. För att minska antalet beteckningar behandlas bara ett träningsexempel i taget vilket innebär att $\mathbf{x}^{(2)}$ är en vektor av storlek $1 \times p$ där varje element $x_k^{(2)}$ kan ses som insignal eller attribut till andra lagret. Värdet på $x_k^{(2)}$ bestäms med hjälp av Sigmoidfunktionen från ekvation (12) enligt

$$x_k^{(2)} = f(\Sigma_k) = \frac{1}{1+e^{-\Sigma_k}} = \frac{1}{1+e^{-(\omega_k^{(1)}\mathbf{x}_i^{(1)T})}}, \tag{13}$$

där k indikerar vilken neuron som behandlas. Hela processen återupprepas sedan med vikterna för nästa lager vilka betecknas $\omega_{sk}^{(2)}$. Här är $\mathbf{x}^{(2)}$ en vektor med lika många element som antalet dolda neuroner, p, och $\Omega^{(2)}$ en matris av storlek $K \times p$ där K är antalet möjliga klasser. Utsignalsvektorn $\mathbf{y}$ består av element y_s där $s \in \{1,...,K\}$ och kan bestämmas genom

$$y_s = f(\omega_s^{(2)}\mathbf{x}^{(2)}), \tag{14}$$

där f är överföringsfunktionen. Detta kallas för framåtpropagering och på detta vis fås en utsignal från det artificiella neurala nätverket.

Utsignalen $\mathbf{y}$ är som tidigare nämnt en vektor med lika många element som möjliga klasser och då varje element i vektorn är värdet från överföringsfunktionen så kommer varje element $y_s \in [0,1]$. Detta värde på varje element representerar hur sannolikt det är att ett givet träningsexempel tillhör utneuronens motsvarande klass, där 1 innebär helt säkert. Man kan således se att för de första träningsexemplen så kommer alla element i $\mathbf{y}$ ha värden relativt nära 0.5 och allteftersom att nätverket tränar så kommer förhoppningsvis

ett värde att närmar sig 1 och resterande går mot 0. I vanliga fall uppnås dock inte detta exakt utan i ett testexempel med tre möjliga klasser kan en utsignal få formen $\mathbf{y} = [0.1, 0.9, 0.06]$. Detta visar att nätverket är förhållandevis säkert på att testexmplet tillhör klass c_2. I vissa fall då hög tillförlitlighet av klassifieringen är nödvändig kan ett krav sättas på skillnaden mellan de två största y_i för att klassificera och annars tillåta nätverket att avstå från klassificering.

För att algoritmen ska fungera så behöver nätverket tränas och optimera vikterna $\Omega^{(1)}$ och $\Omega^{(2)}$ så att utsignalen $\mathbf{y}$ blir så nära det korrekta värdet som möjligt. Detta innebär att det behöver introduceras ett sätt att beräkna felet för att sedan minimera detta. En funkational introduceras som

$$F(\omega_{kj}^{(1)}, \omega_{sk}^{(2)}) = \frac{1}{2}||t_s - y_s||^2 = \frac{1}{2}\sum_{s=1}^{K}(t_s - y_s)^2, \tag{15}$$

där $\mathbf{t} = t(x_i)$ är målvektorn vilken beror på det exempel som används. Då x_i kan tillhöra K olika klasser är $\mathbf{t}$ en vektor med K stycken binära element där varje element följer

$$t_s(x_i^{(1)}) = \begin{cases} 1, \text{ om } x_i^{(1)} \text{ tillhör klass } c_s \\ 0, \text{ om } x_i^{(1)} \text{ inte tillhör klass } c_s \end{cases}$$

Målet är att hitta en stationär punkt med avseende på ω så att $F'(\omega) = 0$ där $\omega = [\omega_{kj}^{(1)}, \omega_{sk}^{(2)}]$. Här är $F'(\omega)$ Fréchet derivatan, för vidare beskrivning se *Approximate global convergence and adaptivity for coefficient inverse problems* [18] (s. 40). I funktionalen (15) är y_s utsignalen som fås via framåtpropagering av exemplet (14). Om överföringsfunktionen är Sigmoidfunktionen i ekvation (12) fås

$$\begin{aligned} F'_{\omega_{sk}^{(2)}}(\omega_{sk}^{(2)}) &= (t_s - y_s)y_s(\omega_{sk}^{(2)})' = \\ &= (t_s - y_s)x_k^{(2)} \cdot f(\sum_k \omega_{sk}^{(2)} x_k^{(2)})(1 - f(\sum_k \omega_{sk}^{(2)} x_k^{(2)})) = \\ &= (t_s - y_s)x_k^{(2)} y_s(1 - y_s). \end{aligned}$$

$F'_{\omega_{sk}^{(2)}}/x_k^{(2)}$ brukar kallas för neuronernas ansvar och det brukar betecknas $\delta_k^{(2)}$ och defineras som $\delta_k^{(2)} = (t_s - y_s)y_s(1 - y_s)$. På liknande sätt kan även ansvaret $\delta_j^{(1)}$ för första lagret tas fram med hjälp av

$$\begin{aligned} F'_{\omega_{ij}^{(1)}}(\omega_{ij}^{(1)}) &= (t_s - y_s)y_s(\omega_{kj}^{(1)})' = \\ &= \big[h_j(1 - h_j) \cdot \big[\sum_s y_s(1 - y_s)(t_s - y_s)\omega_{sk}^{(2)}\big] \cdot x_{ij}^{(1)}\big], \end{aligned}$$

där $h_j = f(\sum_j \omega_{kj}^{(1)} x_{ij}^{(1)})$. Detta ger således att $\delta_j^{(1)} = h_j(1 - h_j) \cdot \sum_k \delta_k^{(2)} \omega_{sk}^{(2)}$. Med hjälp av $\delta_j^{(1)}$, $\delta_i^{(2)}$ går det nu att updatera vikterna $\omega_{kj}^{(1)}$, $\omega_{sk}^{(2)}$ med vanlig gradient uppdateringsformeln

$$\begin{aligned} \omega_{kj}^{(1)} &\leftarrow \omega_{kj}^{(1)} + \eta \delta_j^{(1)} x_{ij}^{(1)} \\ \omega_{sk}^{(2)} &\leftarrow \omega_{sk}^{(2)} + \eta \delta_k^{(2)} x_k^{(2)}, \end{aligned}$$

där η är steglängden för gradientuppdateringen av vikterna och $\eta \in [0, 1]$. För enklare modeller används ett konstant η som balanseras till en storlek som liten nog att komma nära det globala minimum som söks men stort nog för att inte fastna i lokala minimum. Samtidgt måste η vara stort nog att hålla sig inom en acceptabel beräkningstid. För mer komplexa ANN används ofta adaptiva steglängder som gradvis minskar desto närmare ett globalt minium som lösningen kommer.

En svårighet med ANN är spårbarhet i besluten. I många fall kan det vara viktigt att kunna förklara besluten och se varför ett visst exempel blev klassificerat på ett specificikt sätt. Detta är dock mycket svårt att göra med ett ANN på grund av dess komplexa strukter och istället ses nätverket som en svart låda där man stoppar in något och får ut något annat.

För att kunna använda ANN, även när spårbarhet i klassificering behövs, har ett antal metoder utvecklats för att öka spårbarheten. En av dessa kallas, Local Interpretable Model-agnostic Explanations (LIME)

vilken föreslogs av Ribeiro [19]. Denna modell försöker att lokalt approximera nätverket med en spårbar modell. Det globala systemet kan vara för komplext för att väl approximeras med en spårbar modell, men med LIME delar man upp den globala modellen och försöker approximera den lokalt istället vilket ofta ger betydligt bättre approximationer. Implementering av ett ANN kan ses i appendix B.6.

2.3 Metaalgoritmer - vidgande samt resultathöjande algoritmer

För att generallisera eller förbättra prestandan vid implementering kan övergripande algoritmer användas. Flertalet av de använda algoritmerna är binära, det krävs därför en metod för att göra dessa användbara vid tillfällen då det finns fler än två klasser. Till detta används En-mot-Alla-algoritmen. Resultaten kan också eventuellt förbättras med hjälp av metaalgoritmer, så som k-NN [9](s. 43-62) och Bagging [9] (s. 173-175). För att ta bort avvikande värden används k-NN medan Bagging går ut på att köra algortimerna ett antal gånger och ta det resultat som majoriteten av klassificerare erhåller.

2.3.1 En-mot-Alla-metoden

En-mot-Alla-metoden (eng. *One vs. All*) är en metod som används för klassificera flera klasser med en binär klassificeringsalgoritm. Detta gäller alltså linjär regression, Winnow och Perceptron. För att dessa algoritmer ska kunna skilja på fler klasser än två behövs En-mot-Alla-algoritmen implementeras. En-mot-Alla-algoritmen bygger på att det tränas K stycken klassificerare, givet att det finns K klasser, som används parallelt efter träningen. För att träna dessa klassificerare behöver träningssettet modifieras. För den i:e klassen, $i \in \{1,...,K\}$, skapas ett träningsset T_i som innehåller samma data som originalet med skillnaden att klassbeskrivningen i T_i endast innehåller information om huruvida datapunkten tillhör klass i eller inte. Med T_i utförs sedan en binär klassificering om tillhörighet till klass i. Det skapas på så vis en form av specialisering hos de olika versionerna av algoritmen, där förhoppningen är att varje version klarar av att klassificera en specifik klass. Alla enskilt tränade klassificerare kombineras sedan parallelt för att klassificera orginaldatan.

2.3.2 k-Nearest-Neighbour

En distansbaserad metod, k-NN (förkortning av eng. *k-Nearsest-Neighbour*), bygger på att klassificera en datapunkt med hjälp av de närmsta grannarna. Metoden introducerades 1951 av Fix och Hodges [20] och förfinades sedan av Cover och Hart [21, 22]. Algoritmen kan också användas till att förbehandla data, det är på detta vis som algoritmen används under studien. I algoritmen k-NN står k för vilket antal av de närmsta grannarna som studeras och i den följande undersökningen utvärderas resultaten med $k = 3$. Algoritmen förtydligar klassernas gränser genom att ta bort utstickande datapunkter som ligger ovanligt nära datapunkter som tillhör en annan klass. Detta görs genom att identifiera de k närmsta grannarna för en given datapunkt, till detta används den euklidiska normen som definieras enligt

$$d_E(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}.$$

Om den studerade punktens klass skiljer sig ifrån majoriteten av de k närliggande tas den studerade punkten bort. Detta resulterar i att brus och felmätta punkter som annars är placerade i till exempel gränsen mellan klasser tas bort. Kvar är ett dataset som tydligare illustrerar de sökta klasserna.

Ett annat fenomen som kan hanteras med att processa datan med k-NN är så kallade Tomek länkar. En Tomek länk är de fall då det bildas en kedja av punkter från en specifik klass som i någon mån avviker ifrån klassens huvudsakliga område. Kedjor som dessa skapar problem för klassificeringsalgoritmerna på så vis att de får en stor effekt trotts att de är ett fåtal datapunkter.

Algoritmen används alltså för att eliminera information från datasetet som kan ses som överflödig och är en oproportionelig börda för algoritmen eller information som är rent av felaktig.

2.3.3 Bagging

Bagging (ursprungligen från *bootstap aggregating*) är en metod för att minska modellvariansen hos en algoritm och introducerades 1994 av Breimann [4]. Det är känt inom statistiken att ökat antal observationer minskar variansen av medelvärdet hos populationen. Detta kan implementeras i maskininlärningen

och minska felet hos algoritmerna genom att samla in många olika träningsset från populationen, träna algoritmen från olika delmängder och köra dem på testseten för att erhålla olika förutsägelser.

Bagging använder en träningsmängd T för att skapa n stycken slumpmässiga lika stora delmängder T_i så att $T_i \subset T$. Datan som ingår i T_i tas slumpmässigt ur T med ersättning, alltså kan samma punkt finnas i T_i flera gånger. För varje delmängd skapas en klassificerare av den valda algoritmen, dessa tränas individuellt. För att klassificera en datapunkt kombineras då alla dessa klassificerare och viktas för att avgöra vilken klass som är mest trolig. För Bagging används en uniform viktning det vill säga alla algoritmerna räknas som lika trovärdiga. I och med att det tränas ett flertal klassificerare på olika delmängder minskar risken för att till exempel överanpassning inträffar.

2.4 Antal attribut och vikters påverkan för inlärning

Alla algoritmer använder sig av vikter som uppdateras under inlärningen. Därför bör hänsyn till dessa vikter att tas. Algoritmerna använder sig av olika många vikter beroende på antal klasser samt attribut som datan innehåller. Antal vikter påverkas även av graden på polynomet som används i det aktuella fallet. Detta gäller alla algoritmer utom ANN som använder sig av dolda neuroner vilka också påverkar antalet vikter. Antal vikter kan enkelt beräknas och påverkar hur komplexa algoritmerna är.

Linjär regression, Perceptron och Winnow använder sig av lika många vikter vars antal påverkas av gradtalet. Antag att varje träningsexempel $\mathbf{x} = (x_1, ..., x_n)$, i enlighet med ekvation (2), innehåller n attribut samt $K \in \mathbb{N}$ antal klasser. I de fall med polynom av första graden har samtliga algoritmer en linjär viktuppdatering vilket motsvarar antal attribut. Antalet vikter vid ett polynom av grad ett definieras som

$$N_a^{(1)} = n + 1,$$

då blir det totala antalet vikter $K \cdot N_a^{(1)}$.

Vid ett andragradspolynom ökar inte antal vikter linjärt med ökade antal attribut, utan $N_a^{(2)}$ ökar istället som en aritmetisksumma. För flera klasser med flera attribut hos ett andragradspolynom gäller likt det linjära fallet att $N = K \cdot N_a^{(2)}$. Med skillnaden att $N_a^{(2)}$ definieras som

$$N_a^{(2)} = \sum_{S=1}^{n+1} S = 0.5n^2 + 1.5n + 1.$$

Det ses att antalet vikter ökar snabbt för ett andragradspolynom vid en ökning av antalet attribut. Detta leder till att algoritmen kan ta lång tid vid tillämpning på dataset med många attribut.

I ANN används, istället för ett polynom, dolda neuroner för att klassificera datan. Som nämnt ovan multipliceras all indata med korresponderande vikter och summeras in till en dold neuron. Här betecknas totala antalet ingående vikter med V_{in} och antal dolda neuroner i nätverket med p. Därmed blir antal inmatade vikter till de dolda neuronerna $V_{in} = n \cdot p$. Vidare upprepas nu processen och antalet vikter ut från de dolda neuronerna fås av $V_{ut} = p \cdot K$. Det totala antalet vikter som beräknas för varje träningsexempel för ANN är alltså

$$V_{tot} = V_{in} + V_{ut} = p \cdot (n + K). \tag{16}$$

Ett ökat antal dolda neuroner ger nätverket i viss mån större anpassningsförmåga. Men att inte använda ett större antal dolda neuroner än nödvändigt är av stor vikt vid konstruktionen av ett ANN för att minimera antalet beräkningar.

Polynomets grad, i fallen av linjär regression, Winnow eller Perceptron, har också effekter utöver hur komplex algoritmen är. Klassificeringsalgoritmens VC-dimension (Vapnik-Chervonenkis-dimension) [23] beror på polynomets grad och beskriver den maximala mängden punkter ett polynom av ett visst gradtal kan separera i den angivna dimensionen [9] (s. 145-149). Vidare kan ett mått för storleken på träningsmängden som behövs för att en klassifikator av angiven grad med säkerhet ska klassificera datan [24] erhållas. Mängden data som krävs ökar med polynomets grad och är en bra anledning till att vara varsam med att använda polynom av högre grad.

2.5 Prestandamått

Efter genomgång av algoritmer vars mål är att klassificera data, finns intresse för hur väl respektive algoritm lyckas med detta. Detta avsnitt behandlar ett antal olika sätt att mäta resultatet på. Antalet klasser som

undersöks betecknas K, låt därför $i \in \{1,...,K\}$. I följande stycken betecknar $N_{T,i} = |\{c(\mathbf{x}) = h(\mathbf{x}) \cap c(\mathbf{x})) = i\}|$ antalet datapunkter som tillhör klass i och klassificerades korrekt. På liknande sätt avser $N_{F,i} = |\{c(\mathbf{x}) \neq h(\mathbf{x}) \cap c(\mathbf{x}) = i\}|$ antalet felklassificereade punkter för klass i. Felhalten (eng. *error rate*) betecknas som E från den första bokstaven i den engelska benäningen. Det uppenbara sättet att mäta hur bra algoritmen fungerar är genom att mäta hur stor andel av datan som klassifieras fel. Detta kan uttryckas som en kvot mellan antal felklassificeringar och totala antalet klassificeringar enligt

$$\mathrm{E} = \frac{\sum_{i=1}^{K} N_{F,i}}{\sum_{i=1}^{K} (N_{T,i} + N_{F,i})}, \tag{17}$$

där är $N_{F,i}$ antalet data som är felaktigt klassificeras som klass i och $N_{T,i}$ är då antalet data som är korrekt klassificerade. Då enbart andelen felklassificeringar berörs av måttet lider metoden av svårigheter att ge en nyanserad bild. För att få en bättre bild av hur klasserna klassificeras individellt undersöks metoderna precision och täckning [9] (s. 211-214). Precision (eng. *precision*) är ett mått på hur stor andel av varje klass som klassificerats korrekt, vilket belyser precisionen för algortimens klassificering. Likt (17) jämförs en kvot

$$\mathrm{P}(i) = \frac{N_{T,i}}{N_{T,i} + N_{F,i}},$$

där korrekta klassificeringar för en viss klass i jämförs mot totala antalet datapunkter som klassificerats till klassen. Notera att precisionsmåttet är klassspecifikt till skillnad från andelen felklassificeringar som är generell. Utöver att se hur väl en viss klass kan definieras, som precision anger, ligger intresse i att hur stor andel av de klassificerade datapunkterna i en viss klass i faktiskt gjorde det, vilket metoden täckning (eng. *recall*) utreder.

Till skillnad från precision jämför metoden täckning, som betecknas med R, andelen av det som klassificerades till en viss klass som faktiskt tillhörde klassen enligt

$$\mathrm{R}(i) = \frac{N_{T,i}}{N_{T,i} + N_{F,j}},$$

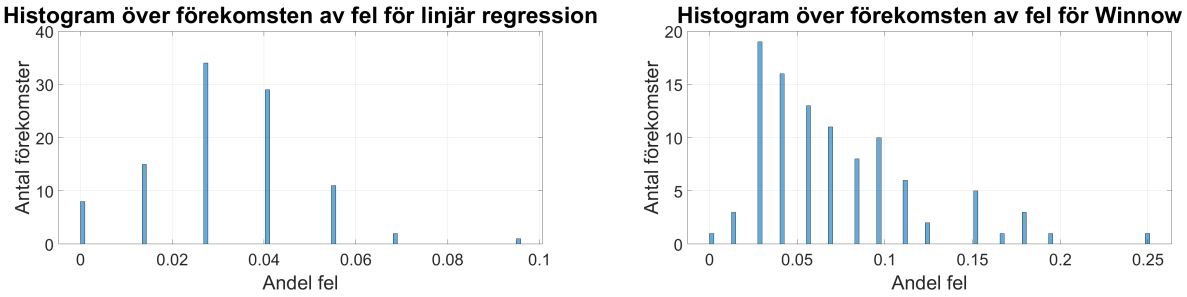
där $j \in \{1,...,K\} \neq i$ och $h(\mathbf{x}) = j$ samt $c(\mathbf{x}) = i$. Måttet jämför antalet datapunkter som klassificerades korrekt till en viss klass med totala antalet pukter som tillhör klassen. Det bör noteras att både täckning och precision är kopplat till antalet felklassificeringar och om inga punkter klassisferas fel kommer inte heller precisionen eller täckningen spegla några fel. Beroende på användningsområde bör olika vikt läggas vid täckning eller precision.

3 Klassificeringsresultat

Presentationen av resultatet utgår ifrån de olika fallen av datasamlingar. Inledningsvis undersöks hur algoritmerna klassificerar två klasser med två attribut och vidgar sedan resonemanget till fyra klasser med fem atrribut. Därefter undersöks hur algoritmerna klassificerar den dermatologiska datan som presenterades i avsnitt 2.1.2. Varje algoritm har tränats 100 oberoende gånger för att få en bättre förståelse för hur den presterar. Som kan ses i figur 4 är felen för algoritmerna inte kontinuerligt fördelade utan placerade i ett antal diskreta kategorier. Då fördelningen är diskret och uppenbart skev är inte medelvärde ett effektivt sätt att presentera datan. Istället kommer resultatet att presenteras som medianen av 100 oberoende genomgångar av algoritmen. Täckning och precision är specifika för varje enskild klass och presenteras som en vektor där varje tal korresponderar med en specifik klass.

3.1 Två klasser med två attribut

I tabell 1 presenteras det sammanlagda resultatet för algortimernas prestanda på det simulerade datasetet med två klasser och två attribut (förkortas A2K2) som illustreras i appendix A.1 och koden återfinns i B.1. Då värderna i tabellen är medianer av 100 oberoende genomgångar kommer det finnas skillnader i prestanda mellan varje sådan omgång. Intervallet i den första kolumnen sammanfattar spridningen på resultatet samtidigt som medianen ger ett konkret värde på vad som kan anses vara det förväntade felet för varje given algoritm.

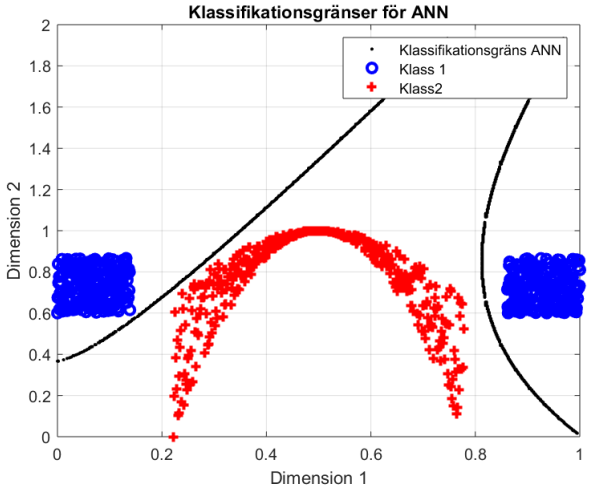


Figur 4: Histogramet presenterar felet efter varje körning av linjär regression med grad 1 på den dermatologiska datan (vänster). Histogramet presenterar felet efter varje körning av Winnow med grad 2 på den dermatologiska datan (höger). Notera likheten mellan histogramen i form av att de är diskreta samt skeva.

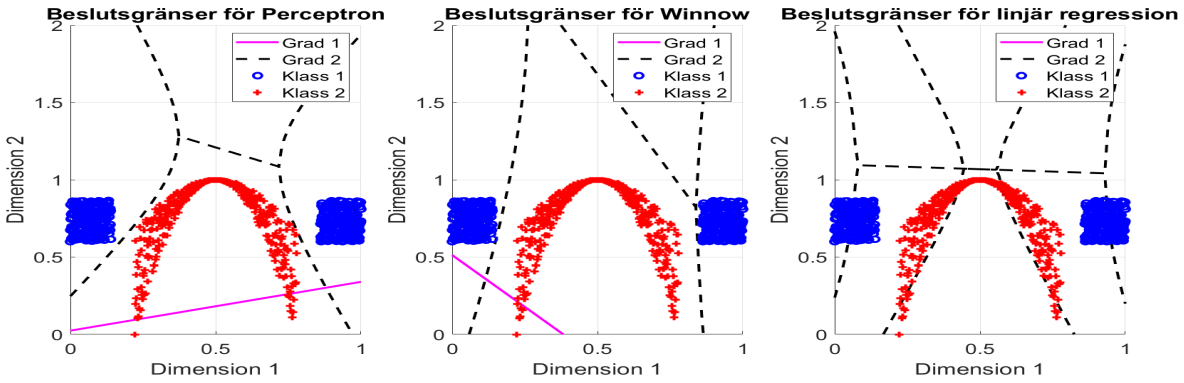
Som ses i tabell 1 är det stor skillnad i förmågan att klassificera mellan ett linjärt polynom jämfört med ett kvadratiskt, oberoende av algoritm. För de algoritmer med linjära polynom kan en stor variation i täckning och precision noteras för de olika algoritmerna. De tre algoritmer som använder sig av ett polynom för klassificering förbättrar alla sin prestanda avsevärt vid användning av ett kvadratiskt polynom jämfört med ett linjärt. En illustration av detta kan ses i figur 6 där kvadratiska polynomen är tydligt bättre anpassade. ANN skiljer sig i metod samt klassificering, som ses i figur 5 jämfört med figur 6. Klassificeringsgränsen som fås av ANN skiljer sig betydligt från resterande algoritmers gränser. Alla algoritmerna uppnår dock ett liknande resultat med avseende på andelen fel.

A2K2	**Grad 1**		
	Fel (%) [min max]	**Täckning (%)**	**Precision (%)**
Linear regression	31.87 [26 52]	[97.45; 45.45]	[62.47; 94.29]
Winnow	49.38 [28.75 67.50]	[50.62; 55.82]	[0; 49.95]
Perceptron	49.38 [31.25 55.63]	[0; 100]	[0; 51]
	Grad 2		
	Fel (%) [min max]	**Täckning (%)**	**Precision (%)**
Linear regression	0 [0 0]	[100; 100]	[100;100]
Winnow	0 [0 0.63]	[100; 100]	[100; 100]
Perceptron	0 [0 0.63]	[100; 100]	[100; 100]
	Neural Networks		
	Fel (%) [min max]	**Täckning (%)**	**Precision (%)**
Neural Networks	0 [0 3.13]	[100; 100]	[100; 100]

Tabell 1: Sammanställning av prestandan för respektive algoritm i fallet med två klasser med två attribut. Felet presenteras i procent för medianen av 100 oberoende körningar, samtidigt visas det minsta respektiva största felet för att presentera spridningen. Vidare är täckning och precision olika för varje enskild klass och presenteras som en vektor där varje tal korresponderar med medianen för varje specifik klass. Observera hur prestandan beror på gradtalet.



Figur 5: Illustration av datan för A2K2 där linjen visar på gränsen mellan klasserna enligt ett ANN efter träning. Märkvärt är att gränsen skiljer sig en del ifrån gränserna för de andra algoritmerna som kan ses i figur 6.



Figur 6: Illustration av datan för A2K2 där linjerna är de gränser som algoritmerna använder för att skilja mellan klasserna. Notera likheter för grad 2 för det olika algoritmerna samt att linjära polynom inte tycks klassificera datan bra, vilket också ses i tabell 1. Linjerna för linjär regression visar på två linjer, en för varje uppsättning vikter i träning. Den faktiskta gränsen är en kombination av dessa, utförande kan ses i detalj i appendix B.3. Notera också att det linjära polynomet för linjär regression saknas i bilden men alltså generar resultat på förhållandevis bra nivå.

3.2 Fyra klasser med fem attribut

Undersökning av fallet med fyra klasser och fem attribut (förkortas A5K4) testas på samma premisser som fallet med två klasser och två attribut. Implementeringen samt illustration kan ses i appendix B.2 respektive A.2 och resultaten ses i tabell 2. De flesta algoritmer tycks klara av att klassificera klasserna utan eller med små fel, undantaget är linjär regression som oberoende av graden på polynomet inte uppnår ett medianfel på 0 %. Värt att notera är att linjär regression uppvisar motsatt mönster från i tabell 1 då felet nu istället ökar för ett kvadratiskt polynom jämfört med ett linjärt. ANN presterar även för fyra klasser ett resultat i nivå med den bästa av de andra algoritmerna. Precision och täckning är för alla algoritmer, förutom linjär regression med ett kvadratiskt polynom, felfritt.

A5K4	**Fel (%)** [min max]	**Grad 1** **Täckning (%)**	**Precision (%)**
Linear regression	0.31 [0 2.19]	[100; 100; 100; 100]	[100; 100; 100; 100]
Winnow	0 [0 32]	[100; 100; 100; 100]	[100; 100; 100; 100]
Perceptron	0 [0 0.94]	[100; 100; 100; 100]	[100; 100; 100; 100]
	Fel (%) [min max]	**Grad 2** **Täckning (%)**	**Precision (%)**
Linear regression	32.81 [27 41]	[0; 71; 100; 96]	[25; 100; 43; 100]
Winnow	0 [0 0]	[100; 100; 100; 100]	[100; 100; 100; 100]
Perceptron	0 [0 1.25]	[100; 100; 100; 100]	[100; 100; 100; 100]
	Fel (%) [min max]	**Neural Networks** **Täckning (%)**	**Precision (%)**
Neural Networks	0 [0 0.94]	[100; 100; 100; 100]	[100; 100; 100; 100]

Tabell 2: Sammanställning av jämförelsemetoderna för respektive algoritm i fallet med fyra klasser och fem attribut. Felet presenteras i procent för medianen av 100 oberoende körningar, samtidigt visas det minsta respektiva största felet för att presentera spridningen. Vidare är täckning och precision olika för varje enskild klass och presenteras som en vektor där varje tal korresponderar med medianen för varje specifik klass.

3.3 Dermatologisk data

I fallet med dermatologisk data innehåller datan 34 olika attribut och sex olika klasser. Tabell 3 presenterar resultatet för algoritmerna givet att de datapunkter för vilket något attribut har saknat värde tagits bort. Det är uppenbart att den dermatologiska datan är svårare att klassificera då samtliga algortimer presterar sämre än i de föregående fallen. Det är däremot också värt att notera hur de linjära och kvadratiska polynomen presterar på liknande nivåer för Perceptron och Winnow. ANN presterar på samma nivå som Perceptron och Winnow. Linjär regression tycks med ett linjärt polynom klassificera datan väl men med det kvadratiska polynomet uppstår stora fel.

Dermatologisk	**Fel (%)** [min max]	**Grad 1** **Täckning (%)**	**Precision (%)**
Linear regression	2.78 [0 6.9]	[100; 90; 100; 91; 100; 100]	[100; 92; 100; 89; 100; 100]
Winnow	5.56 [0, 19.44]	[31 100 100 100 100 100]	[100 100 100 100 100 100]
Perceptron	5.56 [1.39 13.89]	[100; 85; 100; 88; 100; 100]	[83; 92; 100; 100; 100; 100]
	Fel (%) [min max]	**Grad 2** **Täckning (%)**	**Precision (%)**
Linear regression	81.94 [58 96]	[0; 18; 0; 47; 0; 0]	[0; 50; 0; 30; 5; 0]
Winnow	2.78 [0 13.89]	[33; 100; 100; 100; 100; 100]	[100; 100; 100; 100; 100; 100]
Perceptron	2.78 [0 8.33]	[100; 92; 100; 90; 100; 100]	[100; 90; 100; 90; 100; 100]
	Fel (%) [min max]	**Neural Networks** **Täckning (%)**	**Precision (%)**
Neural Networks	2.78 [0 11.11]	[100; 92; 100; 91; 100; 100]	[100; 92; 100; 90; 100; 100]

Tabell 3: Sammanställning av jämförelsemetoderna för respektive algoritm i fallet med dermatologiska datan. Felet presenteras i procent för medianen av 100 oberoende körningar, samtidigt visas det minsta respektiva största felet för att presentera spridningen. Vidare är täckning och precision olika för varje enskild klass och presenteras som en vektor där varje tal korresponderar med medianen för varje specifik klass.

3.4 Metaalgoritmer för dermatologisk data

För att eventuellt förbättra resultatet används två olika metoder för att få varje algoritm att prestera bättre, resultatet från dessa presenteras i tabell 4. Beroende på vilken av de två resultathöjande metaalgoritmerna som används presterar algoritmerna olika, däremot presterar varje algoritm samma medianfel per metametod. Även täckningen och precisionen för algoritmerna uppvisar stora likheter. Preceptron avviker något med Bagging och har en precision för klass ett som är sämre än de andra algortimerna, samtidigt är precisionen för klass två bättre än resterande algoritmer. Det förekommer också vissa skillander mellan de olika algoritmerna för det maximala felet som uppmäts med en möjlig tendes att k-NN i regel producerar lägre fel.

4 Diskussion

Nedan diskuteras vad som kan sägas om algoritmerns prestanda relativt varandra i respektive undersökt fall. Vidare förs en diskussion kring de avgränsningar som gjorts under rapportens gång samt algoritmernas komplexitet och vilka konsekvenser dessa potentiellt har för resultatet.

4.1 Algoritmprestanda

Respektive algoritm lyckades klassificera de olika fallen som beskrivs i resultatdelen med varierande framgång, stor skillnad upplevs generellt baserat på polynomets grad för de olika algoritmerna. Illustrationerna av de simulerade dataseten återfinns i appendix A och kan vara till nytta för att mer intuitivt förstå algoritmernas prestanda för de olika fallen. De olika fallen samt algoritmernas olika komplexitet diskuteras individuellt och jämförs sedan.

Metametoder	**Bagging**		
	Fel (%) [min max]	**Täckning (%)**	**Precision (%)**
Linear regression	2,78 [0 9,72]	[100; 91,7; 100; 93,3; 100; 100]	[100; 92,9; 100; 90; 100; 100]
Winnow	2,78 [0 6,94]	[100; 92; 100; 89; 100; 100]	[100; 91; 100; 90; 100; 100]
Perceptron	2,78 [0 9,72]	[100; 92; 100; 90; 100; 100]	[88; 100; 100; 91; 100; 100]
Neural Networks	2,78 [0 6,94]	[100; 92; 100; 100; 100; 100]	[100; 91; 100; 90; 100; 100]
	KNN		
	Fel (%) [min max]	**Täckning (%)**	**Precision (%)**
Linear regression	1,45 [0 7,25]	[100; 100; 100; 100; 100; 100]	[100; 100; 100; 100; 100; 100]
Winnow	1,45 [0 4,35]	[100; 100; 100; 100; 100; 100]	[100; 100; 100; 100; 100; 100]
Perceptron	1,45 [0 5,80]	[100; 100; 100; 100; 100; 100]	[100; 100; 100; 100; 100; 100]
Neural Networks	1,45 [0 4,35]	[100; 100; 100; 100; 100; 100]	[100; 100; 100; 100; 100; 100]

Tabell 4: Sammanställning av metametoderna för dermatologiska datan med klassificeringspolynom av grad två för Winnow och Perceptron samt grad ett för linjär regression. För Bagging används sju klassificerare och för KNN används $k = 3$. Felet presenteras i procent för medianen av 100 oberoende körningar, samtidigt visas det minsta respektiva största felet för att presentera spridningen. Vidare är täckning och precision olika för varje enskild klass och presenteras som en vektor där varje tal korresponderar med medianen för varje specifik klass.

4.1.1 Två klasser med två attribut

Givet reultaten i tabell 1 inses att datasetet är separerbart med hjälp av ett kvadratiskt polynom men inte med ett linjärt polynom. Detta återspeglas tydligt av värderna där linjära polynom får höga medianfel jämfört med de kvadratiska polynomen genomgående i tabell 1. Linjär regression, Perceptron och Winnow tycks alltså kunna klassificera punkterna utifrån ett andragradspolynom, medianen av felet är 0 % och för Perceptron samt Winnow är det största uppmätta felet fortfarnde litet. Intressant är att notera hur linjär regression får lägre fel, om än ett högt fel, då ett linjärt polynom används samtidigt som Perceptron och Winnow får högre medianfel när samma gradtal på polynomet används. Att Perceptron och Winnow inte förmår att skapa någon slags klassificering kan troligen ackrediteras till det att datan i detta fallet inte är linjärt separerbar och den iterativa algoritmen då inte konvergerar mot en korrekt lösning. Vidare ska det noteras att resultaten kan variera vid varje körning. Linjär regression är inte iterativ och tycks därför i detta fallet att undvika vissa komplikationer som Perceptron och Winnow behöver bearbeta. ANN har ett lågt medianfel men en större spridning på felet än resterande algoritmer med polynom av grad två. Det största uppmätta felet för ANN är $3,13$ % fel, vilket är större än största felet för Perceptron respektive Winnow. Då ANN är den mest komplexa algoritmen är det något förvånande att den i viss mån får sämre resultat än de enklare algoritmerna Perceptron, Winnow och linjär regression. En tänkbar anledning till detta skulle vara det faktum att ett konstant antal dolda neuroner används. Konsekvensen skulle eventuellt vara att nätverket är för flexibelt för träningsdatan och därför gör en mindre överanpassning. Medianfelet för ANN är dock 0 % och ett största fel på $3,13$ % är relativt lågt.

Då datan är separerbar bör även Perceptron och Winnow alltid uppnå ett fel på 0 %, att så inte är fallet skulle kunna bero på att ett dåligt urval har gjorts för träningsdatan respektive datan som jämförs med. Detta skulle kunna betyda att specifika träningssessioner inte representerar klasserna tillräckligt väl och därför presterar inte algoritmerna ett fel på 0 %. Därmed tycks den algoritm som presterar bäst för det genererade datasetet med två attribut och två klasser vara linjär regression.

I figur 6 kan det ses att klassificeringslinjerna för Winnow, Perceptron och linjär regression har alla en liknande form även om vissa skillnader kan noteras. Linjär regression tycks, utifrån bilden i figur 6, inte kunna klassificera data med någon av linjerna för grad två. Istället är algoritmen beroende av ett effektivt sätt att välja vilken av linjerna som ska användas vid vilket tillfälle. Winnow och Perceptron genererar båda beslutslinjer för andragradspolynom som ensamma kan klassificera alla punkter. Detta innebär att dessa lösningar är mindre känsliga för datans utformning och vilken metod som används för val av linje för klassificeringen. Det är värt att notera att klassificeringsgränsen för ANN som presenteras i figur 5 har en avvikande form jämfört med övriga algoritmer. Detta är väntat då ANN inte använder sig av ett polynom för sin klassificering på samma sätt om övriga algoritmer. I fallet A2K2 lyckas alla algoritmer att klassificera alla punkter korrekt men notera även att beslutslinjen för Winnow, Perceptron, linjär regression är bunden till en form som bestäms av polynomet medans ANN kan producera en större variation av beslutslinjer beroende på valet av antal dolda neuroner.

4.1.2 Fyra klasser med fem attribut

Resultaten för respektive algoritm i fallet med fyra klasser och fem attribut ses i tabell 2, där gradtalet tycks påverka framgången mindre än i A2K2. Observera att datan tycks vara linjärt separerbar, till skillnad från fallet med två klasser och två attribut. Notera även att alla algortimer förutom linjär regression, oberoende av polynomets grad, presterade bra med ett medianfel på 0 %. Perceptron och Winnow separerar klasserna med polynom av både grad ett och två. Om datan är linjärt separerbar är detta förväntat då ett kvadratiskt polynom erbjuder mer flexibilitet men en grad extra inte bör producera en överanpassning. Linjär regression blir märkvärt sämre med ett medianfel på $32,81$ % för ett polynom av grad två, jämfört med fallet med två klasser i tabell 1. Det finns flera potentiella förklarningar för detta: För det första används ett konstant tröskelvärde för algoritmen vilket kan vara problematiskt om datan är strukturerad på ett bland annat för kompakt sätt. Den andra potentiella förklaringen är att VC-graden innebär att för få data finns för att klassificera ett kvadratiskt polynom tillförlitligt. Detta skulle kunna innebära att klassificeraren misslyckas med uppgiften att klassificera datan. En tredje potentiell förklaring är att linjär regression inte på samma sätt som Perceptron eller Winnow är flexibel nog att få ett andragradspolynom att likna ett linjärt polynom. ANN presterar i detta fallet bra med ett medianfel på 0 % och ett största felvärde på $0,94$ %, vilket är betydligt bättre än linjär regression men på samma nivå som Perceptron och Winnow.

4.1.3 Dermatologisk data

Slutligen återfinns resultatet för den dermatologiska datan i tabell 3 där endast linjär regression med ett andragradspolynom utmärker sig som dåligt. Det är relativt tydligt att detta datasetet är mer komplext än de tidigare två då algoritmerna genomgående har ett medianfel på magnitud av $2-5$ %. Då den dermatologiska datan inte är konstruerad är det okänt huruvida den är separerbar eller inte. Det är möjligt att datan är separerbar då flera algoritmer uppnår 0 % fel vid upprepade tillfällen, vilket enbart borde ske om datan är separerbar. Felen för de olika algoritmerna är i regel också spridda med stor variation mellan den bästa och sämsta gången. En anledning till detta kan vara de relativt små dataseten som innebär att träningsmängden inte alltid på ett korrekt sätt representerar de faktiska klasserna. Träningsmängderna kan alltså representera orginaldatan olika och resultera i olika bra klassifikatorer.

ANN, linjär regression av grad ett samt Winnow och Perceptron av grad två presterar liknande men vissa skillnader kan noteras. Utifrån resultat har alla en potential att uppnå 0 % fel men Winnow har större spridning på felen. Algoritmerna tränas varje gång på olika träningsmängder vilket betyder att det avvikande värdet skulle kunna vara en tillfällighet. Då det inte tycks finnas någon anledning till att Winnow ska prestera betydligt värre än den liknande algoritmen Perceptron. Det bör också noteras att precisionen och täckningen skiljer de olika algoritmerna åt. Linjär regression, Perceptron och ANN får liknande värden på både precision samt täckning, om än att ANN tycks prestera något bättre. I dessa algoritmena har klass 2 och 4 suboptimala värden, vilket indikerar att dessa två klasser förväxlas av algoritmerna och är den stora bidragande faktorn till felet. För Winnow ses istället ett lågt värde för täckning av klass ett men en median av 100 % för precisionen av alla klasser. Detta tyder på att klass ett frekvent blir klassificerad som någon av de andra klasserna, i och med att medianprecisionen för alla klasserna är 100 % bör punkter från klass 1 relativt jämnt fördelas mellan de övriga klasserna. Detta skulle kunna vara en tillfällighet men det faktum att det sker frekvent under de 100 testerna som genomförts indikerar att Winnow för den dermatologiska datan använder sig av ett något annorlunda mönster för att detektera klasserna jämfört med de andra algoritmerna.

Att Winnow även med ett linjärt polynom uppvisar samma beteende kan anses som ytterligare belägg för att det är just algoritmen Winnow som producerar ett speciellt resultat. Då detta är en större avvikelse är vidare undersökning av intresse för att bekräfta resultatet.

Resultatet pekar alltså på att beroende av vilken klass som anses viktigast är antingen ANN eller Winnow med ett kvadratiskt polynom den mest fördelaktiga algoritmen för att klassificera den dermatologiska datan. Potentiellt skulle förändringar av Winnow så som justering av α eller Θ förbättra prestandan, samtidigt skulle mängden dolda neuroner eller implementation av överförningsfunktionen ReLU kunna öka prestandan eller effektiviteten för ANN. Alternativt skulle en kombination av Winnow och ANN kunna användas parallellt för att representera olika perspektiv av klassifikationen och därmed genom viktning ge ett bättre resultet.

4.1.4 Metaalgoritmers inverkan

I tabell 4 studeras resultatet av två övergripander metoder ämnade att förbättra klassifikationen av den dermatologiska datan. Resultatet av att använda någon form av metaalgoritm tycks vara effektivt baserat på medianfelet. För Bagging presterar nu alla algoritmer med ett medianfel på $2,78$ % och uppvisar inte en enda träningssession med fel över 10 %. För ANN förbättras däremot täckningen och precisionen, det är fortfarande klass 2 och 4 som är problematiska men till mindre grad med Bagging. Både förändringen och skillnaden är dock lite och variationen mellan algoritmerna saknar någon större statistisk signifikans.

Det är ansevärt att Winnow nu uppvisar samma mönster för täckning och precision som de andra algoritmerna, det vill säga klass 2 och 4 förväxlas. En potentiell kombination av Winnow med annan algoritm kan antas vara mindre effektiv när Bagging används då de inte kompleterar varandra på samma sätt.

En annan metod som undersöks är den av att använda k-NN, där $k = 3$. Alla algoritmer tycks uppvisa en förbättring med k-NN jämfört med tidigare resultat då samtliga nu presterar ett medianfel på $1,45$ %. Spridningen av resultatet för de olika träningssessionerna är även de kraftigt förbättrade med de små variationer som finns mellan algoritmerna troligtvis slumpmässiga. Att så stora fördelar med k-NN ses är positivt då det är en enkel algoritm att implementera. Användare bör däremot vara medvetna om att k-NN inte alltid är lämplig specifikt för små mängder då den minskar mängden ytterligare.

4.1.5 Algoritmkomplexitet

I och med att algoritmerna behandlar data och producerar en klassifikation på olika sätt påverkas också algoritmens komplexitet. Linjär regression är i detta avseendet den absolut snabbaste algoritmen då ingen inlärningsprocess används. Istället utförs invertering eller dekomposition av en matris samt matrismultiplikation vilket i regel är enkla operationer för dedikerad mjukvara. De övriga algoritmerna använder sig istället av varje enskild datapunkt för att iterativt producera ett polynom eller nätverk som kan klassificera datan. Den iterativa processen behöver inte alltid ge bättre resultat, som kan ses i tabeller 1, 2 och 3, däremot är det alltid en arbetskrävande process. Utöver detta så använder sig Perceptron, Winnow och ANN av uppsättningar av vikter som alla måste uppdateras. Vikterna kan definieras i matrisform men ett ökat gradtal eller, i fallet av ett ANN, dolda neuroner ökar mängden vikter avsevärt som beskrivs i avsnitt 2.4. Som ses i figur 3 är vikternas struktur mellan ett ANN och Perceptron markant annorlunda. Beroende på mängden attribut och dolda neuroner kommer de olika algoritmerna att ha olika många vikter. Dessutom kräver varje iteration av ANN en framåt propagation samt en bakåt propagation vilket effektivt fördubblar antalet operationer som utförs. Den data som har undersökts har i regel haft relativt få attribut, aldrig över 34. Givet en mängd av enbart 10 dolda neuroner överstiger vikterna i ANN de i Winnow eller Perceptron för alla studerade datamängder.

Bagging som förbättrar prestandan på algoritmen förvärrar alla potentiella problem med att träna en algoritm. Då implementation av Bagging använder flera olika klassificerare för algoritmerna har detta påtagande inverkan på träningstiden. ANN är den algoritm för vilket Bagging ter sig värst anpassad då ANN redan i sin ursprungliga form av avsevärt mer resurskrävande än de andra tre algoritmerna.

Den andra metaalgoritmen, k-NN, är tillskillnad från Bagging inte beroende på den huvudsakliga algortimens komplexitet. Istället är k-NN beroende på hur stort datasetet är, då dataseten som behandlats i rapporten är relativt små har k-NN haft en närmast försumbar betydelse på längden av träningen för algoritmerna. Trots detta har bra prestanda uppmäts för samtliga algoritmer med k-NN.

4.1.6 Jämförelse av prestanda för olika data

Genomgående i kapitel 3 ger ANN bra resultat samtidigt som de andra algoritmerna ger ett mer varierat resultat. ANN är på så vis en säkrare och mer flexibel algoritm att använda när strukturen på datan är okänd, men det blir här tydligt att detta är i utbyte mot en mer komplicerad och krävande inlärningsprocess. Avvägningen av vad som är åtrovärt beror på den givna applikationen och tillgängligheten att träna algoritmen. Kunskap om datan som ska klassificeras gör det också möjligt att använda mindre komplexa algoritmer men frågan som bör ställas om huruvida införskaffandet av datans struktur inte är svårare än att använda ett ANN.

Avvägningen mellan att införskaffa information om datan och att använda enklare algoritmer är något som skulle kunna studeras vidare. Mer specifikt finns intresse i att bättre förstå hur manuell kunskap om data för att välja en bra algoritm kan och bör jämföras med förmågan att med en dator förbehandla data.

4.2 Inverkan av avgränsningar

I avsnitt 1.3 beskrivs de val av avgränsningar som görs för arbetet. Beslut kring hur dataset simuleras samt begräsningar på algoritmerna påverkar självklart algoritmernas prestanda. Andra faktorer som även påverkar prestandan är hur resultaten mäts. I detta arbete används median istället för medelvärde, eller konstanta värden som i vidare studier kan varieras. Genomgående i arbetet testas algoritmerna i `MATLAB` vilket också kan ses som en avgränsning men som inte påverkar resultaten nämnvärt.

Vidare är det möjligt att andra sätt att mäta prestandan av algoritmerna hade producerat annorlunda resultat. Att på ett annat sätt bedöma algoritmerna baserat på mängden data som behövs, största spridning av fel eller tiden för träning skulle med säkerhet påverka resultatet. Detta är möjligheter för vidare studier, samtidigt kan den totala mängden fel, precision och täckning anses vara mer grundläggande.

4.2.1 Val av datasimuleringar

Under rapporten har två specifika dataset simulerats och använts, ett med två klasser och två attribut samt ett med fyra klasser och fem attribut. Validiteten i att just dessa, tillsammans med den dermatologiska datan, har använts för att testa algoritmerna går att diskutera. Syftet med de gjorda valen har varit att stegvis introducera en blandning av komplikationer för algoritmen. I det första datasetet testat ett grundläggande fall med separerbara klasser för att sedan i datasetet med fyra klasser och fem attribut introducera problematik så som icke-binär klassificering och fler attribut. Slutligen testas algoritmerna på den dermatologiska datan vars struktur är okänd. Det kan då påpekas att fler steg eller dataset där till exempel störningar eller överflödiga attribut introduceras var för sig skulle kunna bidra till ett bättre och tydligare resultat. Att lägga större fokus på varje sådan förändring och mindre på teorin för algoitmen är potentiellt intressant för vidare arbete.

4.2.2 Begränsningar på algoritmer

Implementeringen av algoritmerna linjär regression, Winnow samt Perceptron begränsas av vilken grad på klassificeringspolynomen som används. Genomgående i arbetet undersöks enbart polynom av grad ett eller två. Detta tycks inte vara ett problem eftersom algoritmerna presterar väl. Ett högre gradtal kräver även en större mängd data för effektiv klassificering. Alternativ som kan betraktas i vidare undersökning är Bellsplines eller periodiska funktioner så som sinus och cosinus.

4.2.3 Val av median

För att jämföra algoritmernas prestanda behövs ett mått på hur många felklassificeirngar som sker. Utöver de prestandamått som beskrivs i avsnitt 2.5 används medianen för att jämföra resultaten. Valet att använda median istället för medelvärde baseras på att fördelningen av felet för 100 oberoende körningar är skev. Medelvärde påverkas av avstickande värden, vilket inte medianen gör. För att ta hänsyn till avstickande värden betraktas förekomstintervallen. Ett alternativ till förekomstintervall är avvikelse från en fördelning. Då andelen fel inte följer normalfördelningen är standaravvikelse inte rättvisande. I resultattabellerna ses både median och intervallet för andelen fel, det vill säga största och minsta andelen felklassificeringar för 100 körningar. Genom denna kombination av median och förekomstintervall ges en helhetsbild av felet.

4.2.4 Konstant inlärningshastighet

Genomgående i undersökningen hålls inlärningshastigheterna för de olika algoritmerna konstanta. En adaptiv inlärningshastighet skulle kunna ha som fördel att låta algoritmerna konvergera till bättre lösningar. Det finns allså en tydlig utvidgningspotential genom att implementera detta. Vidare används ett konstant antal träningsomgångar, epoker, detta upplevs inte som ett problem då samtliga algoritmer har konvergerat innan det maximala antalet epoker har genomförts. Det finns däremot potential för detta att förändras vid en implementation av en adaptiv inlärningshastighet och bör då tas i beaktning.

5 Slutsats

Då studien har många inskränkningar och begränsningar kan den tydligaste slutsatsen vara att vidare studier inom områden så som vad för effekt de olika avgränsningarna har på algoritmernas prestanda. Utöver detta tycks det vara rimligt att kunna fastslå att ett ANN är den klassifikationsalgoritm som är mest lämpad att användas givet ett okänt problem då ANN kontinuerligt har presterat bland den bästa prestandan i undersökningen. Givet kunskap om problemet har linjär regression potentiellt ett lika bra resultat som ANN, samt Perceptron och Winnow, fast utan en inlärningsperiod. Även linjär regression har möjlighet att via vidare studier uppnå bättre resultat genom att utvärdera andra basvektorer eller optimera tröskelvärdet för dess klassifikation. Slutligen bör nämnas att förbehandling av data, till exempel via k-NN, ger ett markant förbättrat resultat. Jämfört med Bagging som producerar sämre resultat än k-NN men med betydligt mer beräkning blir vikten av att processa datan innan inlärning tydlig. Vidare studier skulle med fördel kunna inriktas på just processen av att behandla data på ett gynnsamt sätt.

Referenser

[1] R. J. Solomonoff. *A Formal Theory of Inductive Inference. Part II*. Tekn. rapport. 1964.

[2] F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." I: *Psychological review* 65.6 (1958), s. 386.

[3] N. Littlestone. *Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm*. Tekn. rapport. 1988, s. 285–318.

[4] L. Breiman. "Bagging predictors". I: *Machine Learning* 24.2 (okt. 1994), s. 123–140.

[5] T. Takase, S. Oyama och M. Kurihara. "Effective neural network training with adaptive learning rate based on training loss". I: *Neural Networks* 101 (maj 2018), s. 68–78.

[6] M. Balcan. *Machine Learning Theory The Winnow Algorithm*. Tekn. rapport.

[7] J. M. Keller och D. J. Hunt. *Incorporating Fuzzy Membership Functions into the Perceptron Algorithm*. Tekn. rapport.

[8] T. Julius och R. C. Gonzalez. *Pattern Recognition Principles*. 1974.

[9] M. Kubat. *An Introduction to Machine Learning*. Second. Springer, 2017.

[10] *Dermatology Data Set*. 1 jan. 1998. URL: `https://archive.ics.uci.edu/ml/datasets/Dermatology`.

[11] L. Beilina, E. Karchevskii och M. Karchevskii. *Numerical linear algebra: Theory and applications*. 2017.

[12] C. M. Bishop. "Pattern Recognition and Machine Learning". I: *Technometrics* 49.3 (aug. 2007).

[13] N. Andréasson, A. Evgrafov, M. Patriksson m. fl. *An Introduction to Continuous Optimization*. Third. Studentlitteratur, 2016, s. 289–296.

[14] D. E. Rumelhart och J. L. McClellands. "Parallel distributed processing". I: (1981).

[15] I. Nunes da Silva m. fl. *Artificial Neural Networks: A Practical Course*. Springer, Cham, 2017.

[16] T. Ash. "Dynamic Node Creation in Backpropagation Networks". I: *Connection Science* (1989), s. 365–375.

[17] R. Yogitha och G. Mathivanan. "Performance Analysis of Transfer Functions in an Artificial Neural Network". I: *International Conference on Communication and Signal Processing*. IEEE, 2018, s. 0393–0397.

[18] La. Beilina och M. V. Klibanov. *Approximate global convergence and adaptivity for coefficient inverse problems*. Vol. 9781441978059. Springer US, aug. 2012.

[19] T. Ribeiro Marco, S. Singh och C. Guestrin. "'Why Should I Trust You?': Explaining the Predictions of Any Classifier". unpublished. 2016.

[20] E. Fix och J. L. Hodges. *Discriminatory analysis, non-paramatric discrimination*. type. USAF School of Aviation Medicine, Randolph Field, Texas, 1951.

[21] T. M. Cover och P. E. Hart. "Nearest neighbor pattern classification". I: *Transactions on Information Theory* (1967), s. 21–27.

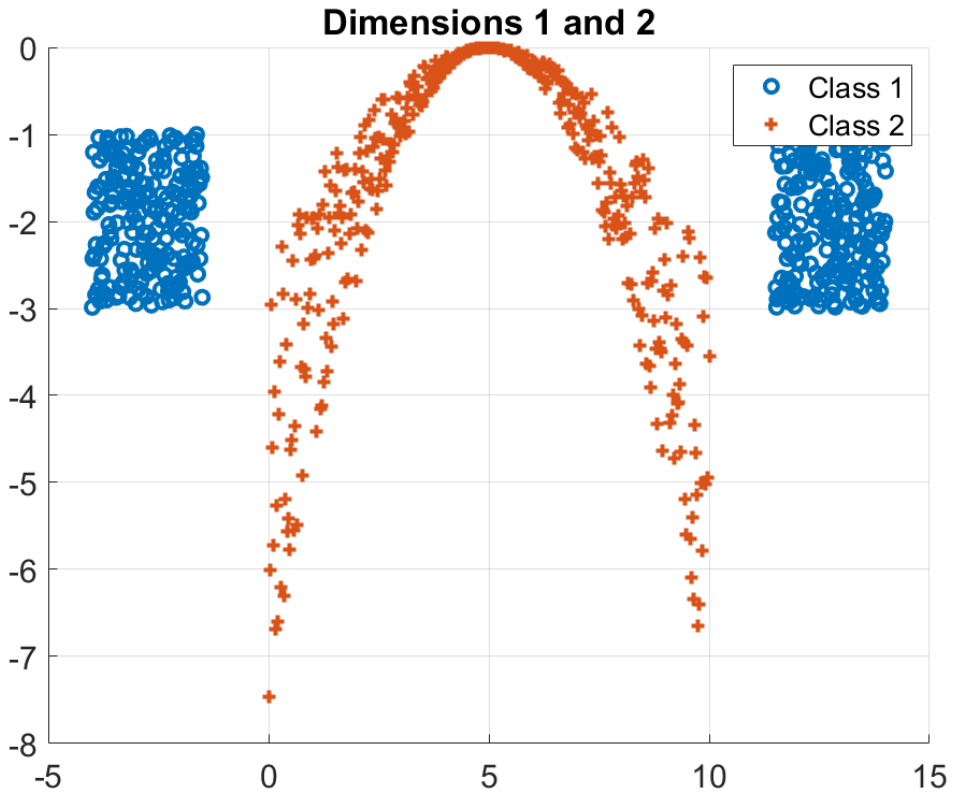
[22] T. M. Cover. "Estimation by the nearest neighbor rule". I: *Transactions on Information Theory* (1968), s. 50–55.

[23] V. N. Vapnik och A. Y. Chervonenkis. *Theory of Probability and its Applications*. 1971, s. 264–280.

[24] A. Blumer m. fl. "Learnability and the Vapnik-Chervonenkis dimension". I: *Journal of the ACM (JACM)* 36.4 (1989), s. 929–965.

A Appendix - Visualisering av data

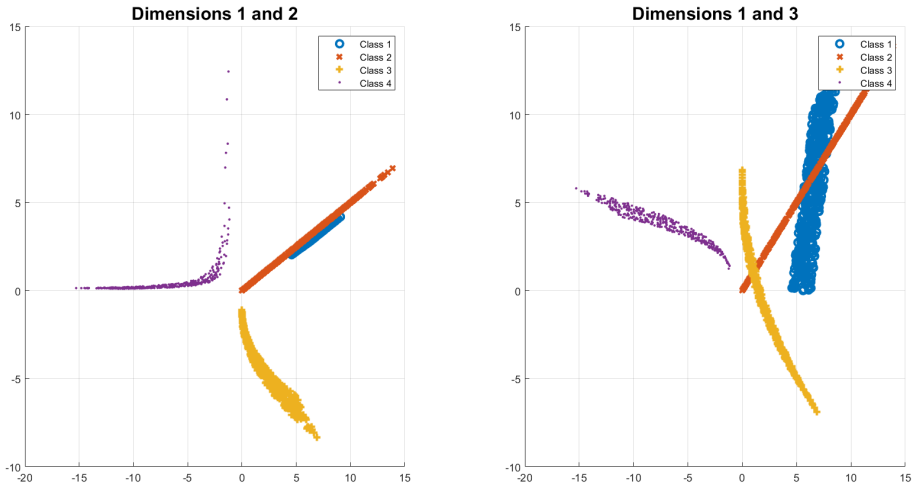
A.1 Visualisering av kvadratiskt separerbar data för två klasser med två attribut



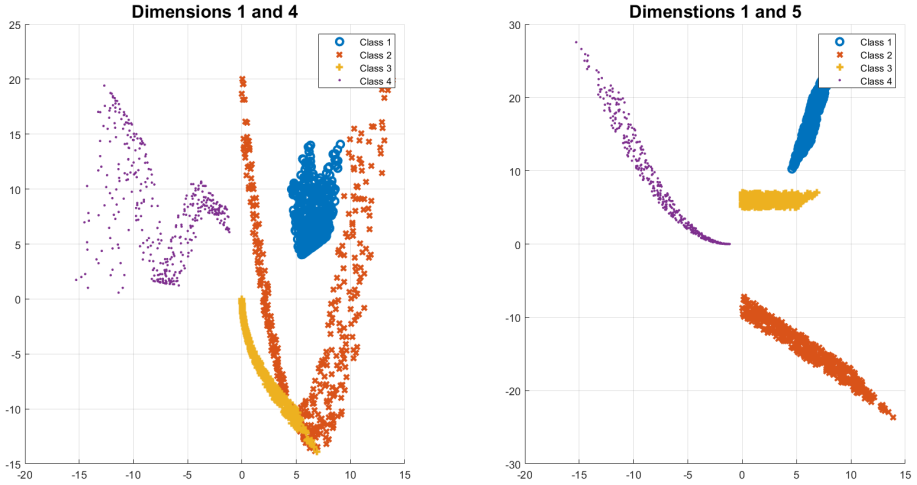
Figur 7: Illustration av ett kvadratiskt separerbart dataset med två attribut och två klasser. De olika attributen presenteras på axlarna och att datan är separerbar ses enkelt. De exakta funktionerna för klasserna samt kod för simuleringen finns i appendix B.1.

A.2 Visualisering av linjärt separerbar data för fyra klasser med fem attribut

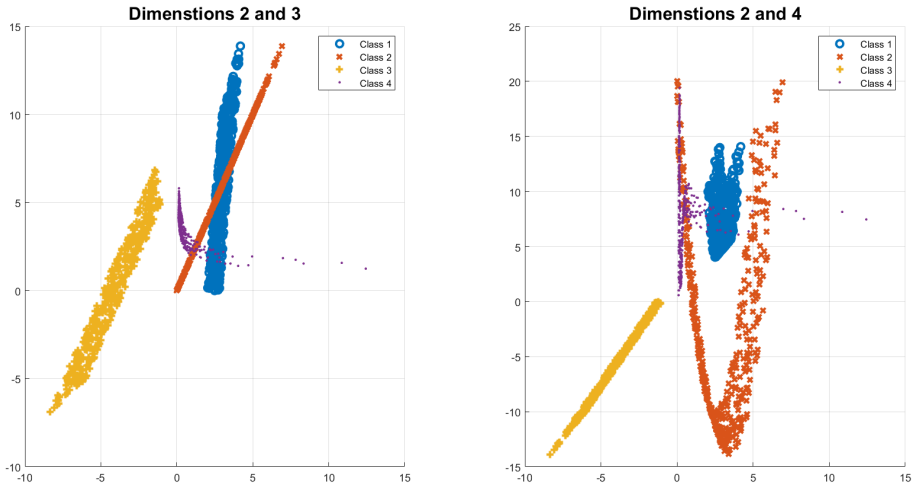
Bilderna som visualiserar klasserna för den simulerade datan med fyra klasser och fem attribut återfinns i figurer 8, 9, 10, 11 och 12. Visualisationen har delats upp i flera figurer på grund av svårigheten att presentera fler än två dimensioner samtidigt. Vidare återfinns koden för samtliga figuer i appendix B.2.



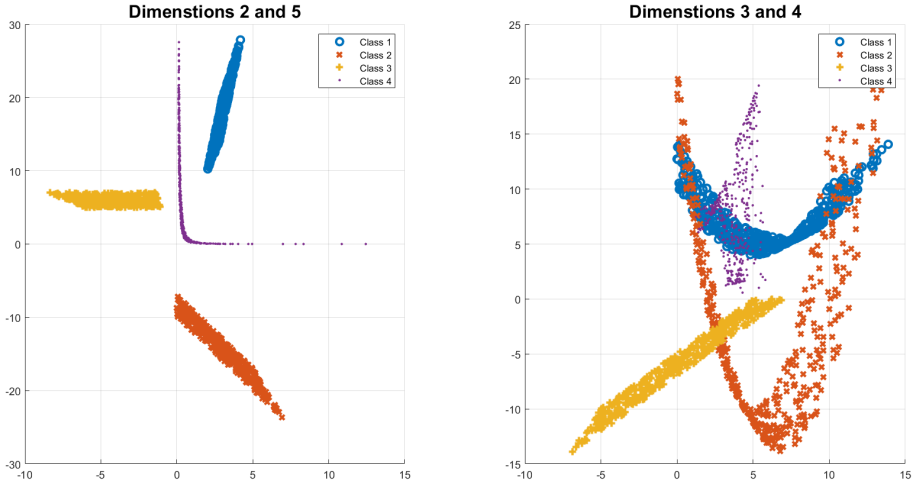
Figur 8: Illustration av alla klasser med avseende på attributen 1 och 2 (vänster) och attributen 1 och 3 (höger). Notera hur olika klasser framstår som separerbara beroende på perspektiv.



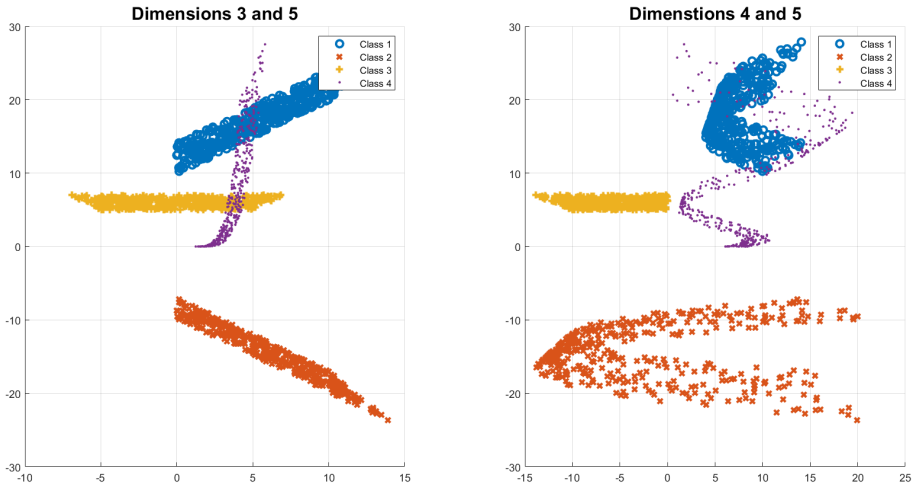
Figur 9: Illustration av alla klasser med avseende på attributen 1 och 4 (vänster) och attributen 1 och 5 (höger). Notera hur olika klasser framstår som separerbara beroende på perspektiv.



Figur 10: Illustration av alla klasser med avseende på attributen 2 och 3 (vänster) och attributen 2 och 4 (höger). Notera hur olika klasser framstår som separerbara beroende på perspektiv.



Figur 11: Illustration av alla klasser med avseende på attributen 2 och 5 (vänster) och attributen 3 och 4 (höger). Notera hur olika klasser framstår som separerbara beroende på perspektiv.



Figur 12: Illustration av alla klasser med avseende på attributen 3 och 5 (vänster) och attributen 4 och 5 (höger). Notera hur olika klasser framstår som separerbara beroende på perspektiv.

B Appendix - MATLAB-kod

B.1 Kod för datasimulering av två klasser med två attribut

```
%Creates a data set containing 2 attributes and 2 classes that is
    defined
%by given functions. Further it plots the classes and produces a text-
    file
%containing the data.
clc
clf
hold on

%Set-up of the variable of the parametrization.
T = 10;
nbrOfPoints = 400;
t = linspace(0,T,nbrOfPoints);

%Determines the noice and the level of it.
delta = 1;
alpha = 2*rand(nbrOfPoints,1);

%The choosen functions that is to be parametrized.
y1 =@(t) -1;
y2 =@(t) -((t-5).^2)/10
x1 =@(t) -4 + t/2 +heaviside(t-T/2)*13
x2 =@(t) t

%Each class consist of the function combined with the noise aswell as
    the
%last column indicating what class is represented.
Class1 = [x1(t)', (y1(t)'.*(1+alpha*delta)), zeros(length(t),1)];
Class2 = [x2(t)', (y2(t)'.*(1+alpha*delta)), ones(length(t),1)];

A = [Class1;
     Class2];

plot(Class1(:,1),Class1(:,2),'o')
plot(Class2(:,1),Class2(:,2),'o')

%Writes a txt file containing the data
csvwrite('Rapport_Square_A2_K2_Set3.txt',A)
```

B.2 Kod för datasimulering av fyra klasser med fem attribut

```
%Creates a data set containing 5 attributes and 4 classes that is
    defined
%by given functions. Further it plots the classes for each pair of
    dimensions
%and produces a text-file containing the data.

clc
clf
hold on

%Set-up of the variable of the parametrization.
```

```
T = 10;
nbrOfPoints = 400;
t = linspace(0,T,nbrOfPoints);

%Determines the noice and the level of it.
delta = 0.2;
alpha = 2*rand(nbrOfPoints,1);

%The choosen functions that is to be parametrized for each dimenstion and
%class
y1 =@(t) 2+t/10;
y2 =@(t) 0.5*t;
y3 = @(t) -0.5*t-1;
y4 =@(t) 1./(t+0.1);
x1 =@(t) 4.5 + t/5;
x2 =@(t) t;
x3 = @(t) (t.^2)./20;
x4 =@(t) -t-1;
z1 =@(t) t;
z2 =@(t) t;
z3 =@(t) -t+5;
z4 =@(t) 1 + sqrt(t);
a1 =@(t) 4+(t-5).^2/4;
a2 =@(t) -10+(t-5).^2;
a3 =@(t) -t;
a4 =@(t) t.*sin(t)+6;
b1 =@(t) t+10;
b2 =@(t) -t-7;
b3 =@(t) 5;
b4 =@(t) (t.^2)./5;


%Each class consist of the function combined with the noise aswell as the
%last column indicating what class is represented.
Class1 = [x1(t)'.*(1+alpha*delta), y1(t)'.*(1+alpha*delta), z1(t)'.*(1+alpha*delta), a1(t)'.*(1+alpha*delta), b1(t)'.*(1+alpha*delta), zeros(length(t),1)];
Class2 = [x2(t)'.*(1+alpha*delta), y2(t)'.*(1+alpha*delta), z2(t)'.*(1+alpha*delta), a2(t)'.*(1+alpha*delta),b2(t)'.*(1+alpha*delta), ones(length(t),1)];
Class3 = [x3(t)'.*(1+alpha*delta), y3(t)'.*(1+alpha*delta), z3(t)'.*(1+alpha*delta), a3(t)'.*(1+alpha*delta),b3(t)'.*(1+alpha*delta), 2*ones(length(t),1)];
Class4 = [x4(t)'.*(1+alpha*delta), y4(t)'.*(1+alpha*delta), z4(t)'.*(1+alpha*delta), a4(t)'.*(1+alpha*delta),b4(t)'.*(1+alpha*delta), 3*ones(length(t),1)];

A = [Class1;
     Class2;
     Class3;
     Class4];

subplot(5,2,1)
hold on
```

```
56 title('1 och 2')
57 plot(Class1(:,1),Class1(:,2),'o')
58 plot(Class2(:,1),Class2(:,2),'o')
59 plot(Class3(:,1),Class3(:,2),'o')
60 plot(Class4(:,1),Class4(:,2),'o')
61 legend('Class 1','Class 2', 'Class 3', 'Class 4')
62
63 %Plots all the classes from each pair of dimension to give a
       comprehensible
64 %understanding of the dynamics of the classes.
65 subplot(5,2,2)
66 hold on
67 title('1 och 3')
68 plot(Class1(:,1),Class1(:,3),'o')
69 plot(Class2(:,1),Class2(:,3),'o')
70 plot(Class3(:,1),Class3(:,3),'o')
71 plot(Class4(:,1),Class4(:,3),'o')
72 legend('Class 1','Class 2', 'Class 3', 'Class 4')
73
74 subplot(5,2,3)
75 hold on
76 title('1 och 4')
77 plot(Class1(:,1),Class1(:,4),'o')
78 plot(Class2(:,1),Class2(:,4),'o')
79 plot(Class3(:,1),Class3(:,4),'o')
80 plot(Class4(:,1),Class4(:,4),'o')
81 legend('Class 1','Class 2', 'Class 3', 'Class 4')
82
83 subplot(5,2,4)
84 hold on
85 title('1 och 5')
86 plot(Class1(:,1),Class1(:,5),'o')
87 plot(Class2(:,1),Class2(:,5),'o')
88 plot(Class3(:,1),Class3(:,5),'o')
89 plot(Class4(:,1),Class4(:,5),'o')
90 legend('Class 1','Class 2', 'Class 3', 'Class 4')
91
92 subplot(5,2,5)
93 hold on
94 title('2 och 3')
95 plot(Class1(:,2),Class1(:,3),'o')
96 plot(Class2(:,2),Class2(:,3),'o')
97 plot(Class3(:,2),Class3(:,3),'o')
98 plot(Class4(:,2),Class4(:,3),'o')
99 legend('Class 1','Class 2', 'Class 3', 'Class 4')
100
101 subplot(5,2,6)
102 hold on
103 title('2 och 4')
104 plot(Class1(:,2),Class1(:,4),'o')
105 plot(Class2(:,2),Class2(:,4),'o')
106 plot(Class3(:,2),Class3(:,4),'o')
107 plot(Class4(:,2),Class4(:,4),'o')
108 legend('Class 1','Class 2', 'Class 3', 'Class 4')
109
110 subplot(5,2,7)
```

```
hold on
title('2 och 5')
plot(Class1(:,2),Class1(:,5),'o')
plot(Class2(:,2),Class2(:,5),'o')
plot(Class3(:,2),Class3(:,5),'o')
plot(Class4(:,2),Class4(:,5),'o')
legend('Class 1','Class 2', 'Class 3', 'Class 4')

subplot(5,2,8)
hold on
title('3 och 4')
plot(Class1(:,3),Class1(:,4),'o')
plot(Class2(:,3),Class2(:,4),'o')
plot(Class3(:,3),Class3(:,4),'o')
plot(Class4(:,3),Class4(:,4),'o')
legend('Class 1','Class 2', 'Class 3', 'Class 4')

subplot(5,2,9)
hold on
title('3 och 5')
plot(Class1(:,3),Class1(:,5),'o')
plot(Class2(:,3),Class2(:,5),'o')
plot(Class3(:,3),Class3(:,5),'o')
plot(Class4(:,3),Class4(:,5),'o')
legend('Class 1','Class 2', 'Class 3', 'Class 4')

subplot(5,2,10)
hold on
title('4 och 5')
plot(Class1(:,4),Class1(:,5),'o')
plot(Class2(:,4),Class2(:,5),'o')
plot(Class3(:,4),Class3(:,5),'o')
plot(Class4(:,4),Class4(:,5),'o')
legend('Class 1','Class 2', 'Class 3', 'Class 4')

%Writes a txt file containing the data
csvwrite('Rapport_Square_A5_K4.txt',A)
```

B.3 Linjär regression

```
clear weights
%% Import data and data preprocessing if necessary
%Imports data set
Data=csvread('Rapport_Square_A2_K2_Set3.txt');

% In case classes start at zero make them start at 1
if ~isempty(find(Data(:,end)==0))
    Data(:,end)=Data(:,end)+1;
end
%Degree of polynomial
degree=2;

%Normalizes the attributes
for k=1:length(Data(1,:))-1
      Data(:,k)=Data(:,k)-min(Data(:,k));
      Data(:,k)=Data(:,k)/max(abs(Data(:,k)));
```

```
17 end
18
19 % Removes datapoints in borders between classes with the k-NN algoritm
20 %Data=kNNCleanUp(Data,3);
21
22 numClass=length(unique(Data(:,end)));
23
24 %% Creates a testset and a trainingset
25 [testset, trainset]=testTrain(Data,0.2);
26
27 % Oversamples underrepresented classes in the trainingset by copying
       them
28 %trainset=overSampling(trainset);
29
30 %% Generates the weights and the classes using OneVsall
31 onevalltrain=OneVsAllv2(trainset);
32 onevalltest=OneVsAllv2(testset);
33 classes=zeros(length(testset(:,1)),length(onevalltest));
34 for k=1:length(onevalltrain)
35     train=onevalltrain{k};
36     weights(k,:)=lrw(train,degree);
37     classes(:,k)=(multiAndSum(testset(:,1:end-1),weights(k,:),degree));
38 end
39
40 %Creates the target matrix for the testset
41 target=zeros(length(testset),numClass);
42  for k=1:length(testset)
43      target(k,testset(k,end))=1;
44  end
45
46 % Sets a threshold for when to abstain from classifying a datapoint
47 %[classes, testtarget,valueReject, idxReject]=...
48 %classifyingThreshold(classes, testtarget,0.8);
49
50 % Makes the highest value in each row 1 and rest 0
51 classes = double(bsxfun(@eq, classes, max(classes, [], 2)));
52
53 %% Compares the classifications with the target and calculates error
54
55 error=length(find(classes~=target))/(2*length(testset));
56
57 for r=1:length(target(1,:))
58     precisionPerClass(r)=length(find(classes(:,r)==1 & target(:,r)==1))
          ...
59         /length(find(classes(:,r)==1));
60     recallPerClass(r)=length(find(classes(:,r)==1 & target(:,r)==1))...
61         /length(find(target(:,r)==1));
62 end
63 %% Plots the Line for the Classification boundry if there are 2 classes
       and 2 dimensions
64 if max(Data(:,end))== 2 && length(Data(1,:)) == 3
65     clf
66
67     %numerical = subplot(1,2,1)
68     figure(1)
69     clf
```

```
70      hold on
71      set(gca,'FontSize',20)
72      xnew=plotLineLinregA2K2V2(100,0,weights);
73      plotDataset(Data)
74
75      legend('Faktisk beslutslinje','Klass 1','Klass 2')
76      title('Faktiskt beslutslinje f r linjr regression')
77      axis([0 1 0 2])
78
79     %analytical = subplot(1,2,2)
80      figure(2)
81      clf
82      hold on
83      set(gca,'FontSize',20)
84
85      [x,y1,y2] = multiDimPlotter(weights(1,:),1,2,1,2,degree);
86      plot(x,y1,'k--','LineWidth',2)
87      plot(x,y2,'k--','LineWidth',2,'HandleVisibility','off')
88
89      [x,y1,y2] = multiDimPlotter(weights(2,:),1,2,1,2,degree);
90      plot(x,y1,'g--','LineWidth',2)
91      plot(x,y2,'g--','LineWidth',2,'HandleVisibility','off')
92
93      plotDataset(Data)
94
95      legend('Analytiskt polynom','Analytiskt polynom','Klass 1','Klass 2
            ')
96      title('Analytiska polynom f r linjr regression')
97
98      axis([0 1 0 2])
99     %linkaxes([numerical, analytical],'xy')
100
101 end
```

B.4 Winnow

```
1  clear weights
2  %% Import data and data preprocessing if necessary
3  % Import Data
4  Data = csvread('Rapport_Square_A2_K2_Set3.txt');
5
6  % In case classes start at zero make them start at 1
7  if ~isempty(find(Data(:,end)==0))
8      Data(:,end)=Data(:,end)+1;
9  end
10
11 %Normalizes Attributes
12  for k=1:length(Data(1,:))-1
13      Data(:,k)=Data(:,k)-min(Data(:,k));
14      Data(:,k)=Data(:,k)/max(abs(Data(:,k)));
15  end
16
17 % Removes datapoints in borders between classes with the k-NN algoritm
18 %Data=kNNCleanUp(Data,3);
19
20  %% User set Parameters
```

```
21  degree=2;
22  alpha=2;
23  epoch=400;
24  partToTrain=0.8;
25  nbrClasses = length(unique(Data(:,end)));
26
27 %% Initates variables and prepares for weight update
28
29 % Splits the data into a training part and a testing in given
      proportion
30 [TrainData, TestData] = testTrain(Data,partToTrain);
31
32 % Oversamples underrepresented classes in the trainingset by copying
      them
33 %trainset=overSampling(trainset);
34
35 % Calculates the number of constants in the polynomial depending on
      order
36 if ( degree == 1)
37      nbrConstants = length(Data(1,1:end-1)) + 1;
38 elseif(degree == 2)
39      nbrConstants = 1;
40     % Number of constants follow an arithmetic sum
41      for i = 1:length(Data(1,1:end-1))
42          nbrConstants = nbrConstants + i+1;
43      end
44 end
45
46 % Splits the training data into several matrices where only one class
      is
47 % not equal to zero at a time
48 OVA=OneVsAllv2(TrainData);
49
50 % Initiates random weights to be trained
51 for k=1:length(OVA)
52      weights(k,:)=rand(1,2*nbrConstants);
53 end
54
55 %% Trains the weigths for the algorithm
56 % Runs though all classes
57 for b=1:length(OVA)
58       localData=OVA{b};
59      % Creates a new vector with all the basic, square terms and mixed
60      % terms.
61       x_i=Vandermonde(localData,degree);
62      % Adds negative mirrored values to give possibility of polynomials
63      % with negative coefficents.
64       x_i=[x_i,-x_i];
65      % Loops over a set amount of epochs
66       for currentEpoch=1:epoch
67           for k=1:length(localData(:,1))
68              % Sets a hypothosis of classification of example, 0 or 1
69               if(weights(b,:)*x_i(k,:)' > 0)
70                               h = 1; else h = 0;
71               end
72              % Update weights, adjust if incorrect h , else do nothing
```

```
                weights(b,:)=weights(b,:).*alpha.^...
                    ((localData(k,end)-h).*x_i(k,:));
            end
        end
end
%% Classifies a matrix of test examples and a target vector
for k=1:nbrClasses
        x_j=Vandermonde(TestData,degree);
        x_j=[x_j, -x_j];
        classes(:,k)=x_j*weights(k,:)';
end

% Creates the target matrix
target=zeros(length(TestData),nbrClasses);
for k=1:length(TestData)
         target(k,TestData(k,end))=1;
end

% Sets a threshold for when to abstain from classifying a datapoint
%[classes, testtarget,valueReject, idxReject]=...
%classifyingThreshold(classes, testtarget,0.8);

% Sets the greatest value in a row to 1 and the rest to 0
classes = double(bsxfun(@eq, classes, max(classes, [], 2)));

%% Calculates error, recall and precision
% Calculates error
error=length(find(classes~=target))/(2*length(TestData));

% Calculates precision and recall per class
for r=1:length(target(1,:))
    precisionPerClass(r)=length(find(classes(:,r)==1 & target(:,r)==1))
        ...
        /length(find(classes(:,r)==1));
    recallPerClass(r)=length(find(classes(:,r)==1 & target(:,r)==1))...
        /length(find(target(:,r)==1));
end

% Removes NaN values from precision and recall vectors
precisionPerClass = rmmissing(precisionPerClass);
recallPerClass = rmmissing(recallPerClass);

%% Plots the Line for the Classification boundry if there are 2 classes
    and 2 dimensions
if max(Data(:,end))== 2 && length(Data(1,:)) == 3
    clf
    mid=length(weights)/2;
    weights1=weights(:,1:mid)-weights(:,mid+1:end);

    numerical = subplot(2,1,1)

    xnew=plotLineLinregA2K2(300,0,weights1);
    plotDataset(Data)

    legend('Numerical classification border','Class 1','Class 2')
    title('Numerical classification border')
```

```
127
128     analytical = subplot(2,1,2)
129     hold on
130
131     [x,y1,y2] = multiDimPlotter(weights1(1,:),1,2,1,2,degree);
132     plot(x,y1,'k--','LineWidth',2)
133     plot(x,y2,'k--','LineWidth',2,'HandleVisibility','off')
134
135     [x,y1,y2] = multiDimPlotter(weights1(2,:),1,2,1,2,degree);
136     plot(x,y1,'g--','LineWidth',2)
137     plot(x,y2,'g--','LineWidth',2,'HandleVisibility','off')
138
139     plotDataset(Data)
140
141     legend('Analytical classification border 1','Analytical
            classification border 2','Class 1','Class 2')
142     title('Analytical classification borders')
143
144     axis([0 1 0 2])
145     linkaxes([numerical, analytical],'xy')
146
147 end
```

B.5 Perceptron

```
1  clear weights
2  %% Import data and data preprocessing if necessary
3  % Import Data
4  Data = csvread('Rapport_Square_A2_K2_Set3.txt');
5
6  % In case classes start at zero make them start at 1
7  if ~isempty(find(Data(:,end)==0))
8      Data(:,end)=Data(:,end)+1;
9  end
10
11 %Normalizes Attributes
12  for k=1:length(Data(1,:))-1
13      Data(:,k)=Data(:,k)-min(Data(:,k));
14      Data(:,k)=Data(:,k)/max(abs(Data(:,k)));
15  end
16
17 % Removes datapoints in borders between classes with the k-NN algoritm
18 %Data=kNNCleanUp(Data,3);
19
20  %% User set Parameters
21  degree=1;
22  alpha=2;
23  epoch=400;
24  partToTrain=0.8;
25  nbrClasses = length(unique(Data(:,end)));
26 %% Initates variables and prepares for weight update
27 % Splits the data into a training part and a testing in given
       proportion
28 [TrainData, TestData] = testTrain(Data,partToTrain);
29
30 % Oversamples underrepresented classes in the trainingset by copying
```

```
    them
%trainset=overSampling(trainset);

% Calculates the number of constants in the polynomial depending on
    order
if ( degree == 1)
    nbrConstants = length(Data(1,1:end-1)) + 1;
elseif(degree == 2)
    nbrConstants = 1;
    % Number of constants follow an arithmetic sum
    for i = 1:length(Data(1,1:end-1))
        nbrConstants = nbrConstants + i+1;
    end
end

% Splits the training data into several matrices where only one class
    is
% not equal to zero at a time
OVA=OneVsAllv2(TrainData);

% Initiates random weights to be trained
for k=1:length(OVA)
    weights(k,:)=rand(1,nbrConstants);
end

%% Trains the weigths for the algorithm
% Runs though all classes
for b=1:length(OVA)
     localData=OVA{b};
     % Creates a new vector with all the basic, square terms and mixed
     % terms of a second order polynomial or simlpy adds a 1 for the
         first
     % order.
     x_i=Vandermonde(localData,degree);
     % Loops over a set amount of epochs
     for currentEpoch=1:epoch
         for k=1:length(localData(:,1))
             % Sets a hypothosis of classification of example, 0 or 1
             if(weights(b,:)*x_i(k,:)' > 0)
                             h = 1; else h = 0;
             end
             % Update weights, adjust if incorrect h , else do nothing
             weights(b,:)=weights(b,:)+alpha.*...
                 ((localData(k,end)-h).*x_i(k,:));
         end
     end
end
%% Classifies a matrix of test examples
for k=1:nbrClasses
         x_j=Vandermonde(TestData,degree);
         classes(:,k)=x_j*weights(k,:)';
end

% Sets the greatest value in a row to 1 and the rest to 0
classes = double(bsxfun(@eq, classes, max(classes, [], 2)));

```

```
%% Classifies a matrix of test examples and a target vector
for k=1:nbrClasses
        x_j=Vandermonde(TestData,degree);
        classes(:,k)=x_j*weights(k,:)';
end

% Creates the target matrix
target=zeros(length(TestData),nbrClasses);
for k=1:length(TestData)
         target(k,TestData(k,end))=1;
end

% Sets a threshold for when to abstain from classifying a datapoint
%[classes, testtarget,valueReject, idxReject]=...
%classifyingThreshold(classes, testtarget,0.8);

% Sets the greatest value in a row to 1 and the rest to 0
classes = double(bsxfun(@eq, classes, max(classes, [], 2)));

%% Calculates error, recall and precision
% Calculates error
error=length(find(classes~=target))/(2*length(TestData));

% Calculates precision and recall per class
for r=1:length(target(1,:))
    precisionPerClass(r)=length(find(classes(:,r)==1 & target(:,r)==1))
        ...
        /length(find(classes(:,r)==1));
    recallPerClass(r)=length(find(classes(:,r)==1 & target(:,r)==1))...
        /length(find(target(:,r)==1));
end

% Removes NaN values from precision and recall vectors
precisionPerClass = rmmissing(precisionPerClass);
recallPerClass = rmmissing(recallPerClass);

%% Plots the Line for the Classification boundry if there are 2 classes
    and 2 dimensions
if max(Data(:,end))== 2 && length(Data(1,:)) == 3
    %clf

    %numerical = subplot(1,2,1)
    figure(1)
    clf
    hold on
    set(gca,'FontSize',20)
    xnew=plotLineLinregA2K2(300,0,weights);
    plotDataset(Data)

    legend('Faktisk beslutslinje','Klass 1','Klass 2')
    title('Faktiskt beslutslinje f r Perceptron')
    axis([0 1 0 2])

    %analytical = subplot(1,2,2)
    figure(2)
    clf
```

```
137     hold on
138     set(gca,'FontSize',20)
139
140     [x,y1,y2] = multiDimPlotter(weights(1,:),1,2,1,2,degree);
141     plot(x,y1,'k--','LineWidth',2)
142     plot(x,y2,'k--','LineWidth',2,'HandleVisibility','off')
143
144     [x,y1,y2] = multiDimPlotter(weights(2,:),1,2,1,2,degree);
145     plot(x,y1,'g--','LineWidth',2)
146     plot(x,y2,'g--','LineWidth',2,'HandleVisibility','off')
147
148     plotDataset(Data)
149
150     legend('Analytiskt polynom','Analytiskt polynom','Klass 1','Klass 2
        ')
151     title('Analytiska polynom f r Perceptron')
152
153     axis([0 1 0 2])
154     %linkaxes([numerical, analytical],'xy')
155
156 end
```

B.6 Artificiella neurala nätverk

```
1  %% Import data and data preprocessing if necessary
2  %Imports data set
3  Data=csvread('Rapport_Square_A2_K2_Set3.txt');
4
5  % In case classes start at zero make them start at 1
6  if ~isempty(find(Data(:,end)==0))
7      Data(:,end)=Data(:,end)+1;
8  end
9
10 % Normalizes every attribut vector so that it's between -1 and 1
11 for k=1:length(Data(1,:))-1
12      Data(:,k)=Data(:,k)-min(Data(:,k));
13      Data(:,k)=Data(:,k)/max(abs(Data(:,k)));
14 end
15
16 % Removes datapoints in borders between classes with the k-NN algoritm
17 %Data=kNNCleanUp(Data,3);
18
19 %splits set into training and testing
20 [testset, trainset]=testTrain(Data,0.2);
21
22 % Oversamples underrepresented classes in the trainingset by copying
       them
23 %trainset=overSampling(trainset);
24
25 % Adds a constant or bias to the dataset
26 trainset=[ones(length(trainset),1),trainset];
27 testset=[ones(length(testset),1),testset];
28
29 %% Sets user parameters
30 learningrate=0.08;
31 % Number of classes
```

```
numClass=length(unique(Data(:,end)));
% Number of hidden neurons
numhidn=60;
% Number of epochs
numepoch=500;
% The transfer function, in this case the sigmoid function
transfun=@(x) 1./(1+exp(-x));

%% Initiates weights for both layers and a training target matrix

% Initial random weights of the hidden layer between -0.1 and 0.1
weightshid=rand(length(trainset(1,:))-1,numhidn)...
    .*ones(length(trainset(1,:))-1,numhidn)/5-0.1;

% Initial random weights of the output layer between -0.1 and 0.1
weightsout=rand(numhidn,max(trainset(:,end)))...
    .*ones(numhidn,max(trainset(:,end)))/5-0.1;

% Initiates a trainingtarget matrix with only zeros
trainingtarget=zeros(length(trainset),numClass);

%% Train the weights of the network
for i=1:numepoch
    for k=1:length(trainset)
        h=transfun(trainset(k,1:end-1)*weightshid);
        classes=transfun(h*weightsout);

        % Updates the training target vector with the class of the
            current
        % example
        trainingtarget(k,trainset(k,end))=1;

        %Calculates the responsibility
        deltaout=classes.*(1-classes).*(trainingtarget(k,:)-classes);
        deltahid=h.*(1-h).*(weightsout*deltaout')';

        % Updates weights
        weightsout=weightsout+learningrate.*h'*deltaout;
        weightshid=weightshid+learningrate.*trainset(k,1:end-1)'*
            deltahid;
    end
end

%% Classifies a matrix with test exmaples and creates target vector

%Runs the testset through the system and classifies the test example
    with a
%continous value
h=transfun(testset(:,1:end-1)*weightshid);
classes=transfun(h*weightsout);

%Creates a testtarget matrix where each row is a target vector
testtarget=zeros(length(testset),numClass);
for k=1:length(testset)
        testtarget(k,testset(k,end))=1;
end
```

```
85
86 % Sets a threshold for when to abstain from classifying a datapoint
87 %[classes, testtarget,valueReject, idxReject]=...
88 %classifyingThreshold(classes, testtarget,0.8);
89
90 %Sets the highest value on each row to 1 and rest to zero in the
       testset
91 classes = double(bsxfun(@eq, classes, max(classes, [], 2)));
92
93 %% Calculates the error, recall and precision
94
95 % Calculates error based on differences in the output and target
96 error=length(find(classes~=testtarget))/(2*length(testset));
97
98 % Calculates the precision and recall for each class
99 for r=1:length(testtarget(1,:))
100     precisionPerClass(r)=length(find(classes(:,r)==1 & testtarget(:,r)
          ==1))...
101         /length(find(classes(:,r)==1));
102     recallPerClass(r)=length(find(classes(:,r)==1 & testtarget(:,r)==1)
          )...
103         /length(find(testtarget(:,r)==1));
104 end
105
106 %% Plots the classification boundry in case of A2K2
107 clf
108 if max(Data(:,end)==2)
109     hold on
110     set(gca,'FontSize',30)
111     plotLineANNA2K2V2(300,0,weightshid,weightsout);
112     %plotSurfANNA2K2(300,0,weightshid,weightsout);
113     plotDataset(Data)
114
115     legend('Faktisk beslutslinje','Klass 1','Klass 2')
116     title('Faktiskt beslutslinje f r ANN')
117 end
```

B.7 En-Mot-Alla-metoden

```
1 %%
2 % This function takes a dataset with a range of classes and returns a
3 % cellvector where each element is a matrix where only one class is
4 % indicated as one and all others are set as zero
5 %
6 %       Input: - inData, a matrix with attributes for a number of
      examples
7 %                where the last column indicates the class and the
      rows
8 %                are individual examples
9 %
10 %      Output: - sets, a cell vector where each cell contains a matrix
11 %                of the same form as inData but where the classes in
      each
12 %                cell is one for the classes corresponding the the
      index
13 %                of sets and all others are classes are set to zero
```

```
14
15 function sets=OneVsAllv2(inData)
16
17 classes=inData(:,end);
18 idx=1;
19
20 for k=min(classes):max(classes)
21     tempset=inData;
22     tempset(:,end)=0;
23     tempset(find(classes ==k),end)=1;
24     sets{idx}=tempset;
25     idx=idx+1;
26 end
```

B.8 Lrw

```
1  %%
2  % lrw - Linear Regression Weights calculates the weights for the
       machine
3  % learning alogrithm, linear regression. The function calculates
       weights
4  % either for a first or second order polynomial with any given number
       of
5  % attirbutes. The returned weights can then be used to classify future
6  % examples
7  %
8  %       Input:      - trainingData, is an m x n matrix with all the
9  %                     training examples that the algoritm will base it'
       s
10 %                     classifications on. The matrix contains m
       examples
11 %                     and each example is described by n-1 attributes
       and
12 %                     the last column represents the class of the
       example
13 %                   - degree, is the order of polynomial that will be
       used
14 %                     to calculate the weights
15 %
16 %       Output:     - weights, is the weights that represents the
       surface
17 %                     that closest approximates all datapoints. These
18 %                     weights are used with a given threshold to decide
19 %                     which class a new example belongs to.
20 %%
21
22 function weights=lrw(trainingData,degree)
23
24 % Initiates a matrix with the constant terms of the function
25 vander=ones(length(trainingData(:,1)),1);
26
27 % Creates a matrix with all data points for a first order polynomial
28 if degree==1
29     vander=[vander, trainingData(:,1:end-1)];
30
31 % Creates a matrix with all data points for a second order polynomial
```

```
elseif degree==2
    for i=1:length(trainingData(1,:))-1
        vander=[vander,trainingData(:,i),trainingData(:,i).^2];
    end
    for j=1:length(trainingData(1,:)-2)
        for k=j+1:length(trainingData(1,:))-1
            vander=[vander, trainingData(:,j).*trainingData(:,k)];
        end
    end

% Error catch in case a degree input has been given besides 1 or 2
else
    disp('chose different degree, 1 or 2')
end
% calculates weights using linear regression
weights=vander\trainingData(:,end);
```

B.9 k-Nearest-Neighbour

```
%%
% A function that uses the k-NN algoritm to remove troubelsome
    datapoints.
% The function measures the 2-norm between all data points and if the
% majority of the k nearest datapoints to a given point belongs to a
% different class then the given data point is removed from the dataset
    .
%
%       Input:      - Data, is a m x n matrix with n-1 attributes and m
%                     examples. The last column represents the class of
     the
%                     given example.
%                   - k, is the number of Nearest Neighbours that the
%                     majority is decided on
%
%       Output:     - Data, is a reduced dataset where outlaying
    datapoints
%                     have been removed.

function Data=kNNCleanUp(Data,k)

%Normalizes Attributes
for j=1:length(Data(1,:))-1
    Data(:,j)=Data(:,j)-min(Data(:,j));
    Data(:,j)=Data(:,j)/max(abs(Data(:,j)));
end

idxCount=1;
while idxCount<=length(Data(:,1))
    distances=vecnorm(Data(idxCount,1:end-1)'-Data(:,1:end-1)');
    [kNN idxNN]=mink(distances,k+1);
    kNN(1)=[];
    idxNN(1)=[];
    if mode(Data(idxNN,end))~=Data(idxCount,end)
        Data(idxCount,:)=[];
    else
        idxCount=idxCount+1;
```

```
    end
end
```

B.10 Test Train

```
%%
% This functions seperates a whole set into two sets, one training set and
% one testset based on a parameter indicating how much of the whole should
% be in the training set.
%
% Input:            - Whole set to be split
%                   - How big part that should be training set [0-1] 1
%                     gives back the same set as the input
%
% Output:           - Training set for a machine learning algorithm
%                   - Test set for training algorithm
%%
function [trainSet, testSet]=testTrain(wholeSet,trainPart)

% Generates unique random indices to the amount of the set parameter
trainIdx=randperm(length(wholeSet(:,1)),...
    round(length(wholeSet(:,1))*trainPart));

% Create a new matrix containing only the rows of the randomized indices
trainSet=wholeSet(trainIdx,:);
% Removes the rows from the original set that are in the new matrix
wholeSet(trainIdx,:)=[];
% Set testSet as the remainder och the set
testSet=wholeSet;
```

B.11 Test Train Num

```
%%
% This functions seperates a whole set into two sets, one training set and
% one testset based on a user specified number of elements that should be
% in the training set.
%
%       Input:      - Whole set to be split
%                   - The number of elements you want in your training set
%
%       Output:     - Training set for a machine learning algorithm
%                   - Test set for training algorithm
%%

function [trainSet, testSet]=testTrainNum(wholeSet,trainNum)

% Generates unique random indices to the amount of the set parameter
trainIdx=randperm(length(wholeSet(:,1)),trainNum);
% Create a new matrix containing only the rows of the randomized indices
trainSet=wholeSet(trainIdx,:);
```

```
% Removes the rows from the original set that are in the new matrix
wholeSet(trainIdx,:)=[];
% Set testSet as the remainder och the set
testSet=wholeSet;
```

B.12 Vandermonde

```
% This function creates a list of values according to a specified order
    of
% polynomial, it creates square and mixed terms in the case of second
    order
% and adds a constant in the case of first order. The output is a
% vector where each element is a part of the polynomial with each
% row in A. Therefore if A is a vector then the output will be a vector
%
% The values will be ordered as following:
%   Linear  -       1,x(1),x(2),...,x(n)
%   Square  -       1,x(1),x(1)^2,x(2),...,x(n),x(1)*x(2),x(1)*x(3)+...
%
%   Input   - A     = Matrix where each row is coordinates to a point
%           - degree  = linear or square multiplication. 1 = linear, 2=
%           square
%%
function [ vander ] = Vandermonde(A,degree)
% creates the constant for any polynomial
vander=ones(length(A(:,1)),1);
% Creates the values for a first order polynomial
if degree==1
    vander=[vander, A(:,1:end-1)];
% Creates the values for a second order polynomail
elseif degree==2
    for i=1:length(A(1,:))-1
        vander=[vander,A(:,i),A(:,i).^2];
    end
    for j=1:length(A(1,:)-2)
        for k=j+1:length(A(1,:))-1
            vander=[vander, A(:,j).*A(:,k)];
        end
    end
% Error catch in case of degree not found in {1,2}
else
    disp('chose different degree, 1 or 2')
end
end
```