ALLERØD GYMNASIUM

SRP-opgave 2017

Elev id: 3x 24
Elev: Rasmus Hag Løvstad

Fag:	**Vejleder:**
Fysik	Christian Lundmand Jensen
Matematik	Stine Weisbjerg

Opgaveformulering:

Selvkørende biler

Gør rede for mindst én sensortype som anvendes på selvkørende biler. Du skal redegøre for de fysiske principper sensoren bygger på og hvorfor denne sensor anvendes på selvkørende biler.

Gør rede for den matematik som klassifikation og machine learning anvender. Du skal herunder beskrive matricer og egenvektorer og bevise udvalgte relevante sætninger. Du skal desuden lave et simpelt eksempel hvor egenskaberne af egenvektorer udnyttes.

Du skal med udgangspunkt i de beskrevne sensortypers udgangssignal udføre eksperimenter/simulationer som påviser sensornes relevans for selvkørende biler.

Omfanget: Opgaven skal være på 15-25 sider (med 1½ linjes linjeafstand, 2-2½ cm margin i både højre og venstre side og med 12-punktskrift).
Forside, resume, indholdsfortegnelse, bilag, evt. forord og kildefortegnelse *tælles ikke med*.
Bemærk figurer, grafer og formler og evt. appendiks *tælles med* i det samlede sideantal.

Opgaven skal afleveres senest den 21. december 2017, kl. 9:45 i Netprøver.dk, uploades som PDF-fil. Filen skal navngives således (fx eleven Jens Jensen i 3x: srpjensjensen3xag)

Studieretningsprojekt om selvkørende biler

Rasmus Hag Løvstad, 3.X

19. december 2017

Abstract

Autonomous cars have been one of the most iconic technologies of the decade, and have already proven to be a safer alternative to human drivers. In this paper, I have researched some of the technologies behind the self-driving car, being the CCD in a digital camera and logistic regression from Machine Learning. First, I explained how the physics in a CCD makes it able to capture photographs of the world using semiconductors and Bayer-filters. I then described both the theory and mathematics behind logistic regression, a tool used for classification in Machine Learning, including both two proofs and a presentation of matrix arithmetic. I could then use the math that I learned to develop a model from an example dataset, that ended up producing good results. At last, I combined both technologies in an experiment producing a model that can classify images of pedestrians, which potentially can be used in an autonomous car. Doing this, I could conclude that both the digital camera and machine learning can play an important role in self-driving cars.

Indhold

1 Introduktion

I løbet af de seneste tyve år har moderne teknologi ændret verden totalt. Computeren, mobiltelefonen og internettet har forbedret vores levekvalitet drastisk, og alle industrier fra medier til landbrug afhænger nu af teknologien. Også bilindustrien har nydt godt af det, og de fleste biler har indbygget GPS-system, bluetooth-anlæg og måske snart også autopilot. Googles firma Waymo har nu kørt over 3 mio. miles i deres egne selvkørte biler, med helt ned til kun 2 menneskelige afbrydelser per 10.000 miles [1]. Eftersom at op til 94% af alle bilulykker skyldes menneskelige fejl, kan et bybillede domineret af selvkørende biler derfor redde mange liv der ellers ville være gået tabt [2].

Når mennesker kører bil er den vigtigste sans vores øjne, der orienterer os om omverdenen. Dette kan man delvist erstatte med forskellige kameraer rundt om bilen der kan identificere bl.a. vejstriber og fodgængere. Jeg vil derfor prøve at beskrive hvordan digitalkameraet virker på et fysisk niveau, og hvordan det benyttes på en selvkørende bil.

Billederne skal behandles, bliver det oftest gjort af en machine learning algoritme, der kan opstille en model ud fra tidligere data[3]. Jeg vil derfor prøve at redegøre for hvordan noget af matematikken i machine learning virker gennem det der kaldes klassifikation.

Både sensorer og machine learning er dog blot værktøjer, indtil man kombinerer dem. Derfor har jeg til sidst tænkt mig at udføre et eksperiment for at demonstrere hvordan begge dele kan benyttes i en virkelig situation ved at prøve at klassificere billeder af mennesker, hvilket bl.a. kan benyttes til at identificere fodgængere og andre biler i den virkelige verden.

2 Digitalkamera

Digitalkameraer er en af de essentielle teknologier bag selkørende biler. De kan både fungere som et relativt billigt alternativ til mere komplicerede teknologier som LIDAR og radar, og kan samtidigt bruges til at identificere ting som vejstriber og læse vejskilte som andre teknologier ikke kan [3].

Det udgangssignal som et digitalkamera sender til bilen er et billede som består af et gitter af pixels, der hver især har et tal tilknyttet der beskriver hvor meget rød, grøn og blå farve der er i billedet. Dette udgangssignal kan derefter behandles af bl.a. machine learning, som derefter beslutter om der skal foretages en handling ud fra signalet.

Moderne digitalkameraer indeholder mange forskellige teknologier, men jeg har besluttet kun at beskrive hvordan selve sensoren i kameraet virker, den såkaldte *CCD*[4], der kan omdanne

de fotoner der rammer den til et elektrisk signal. For at forstå hvordan CCD'en kan gøre dette skal man vide noget om halvledermetaller.

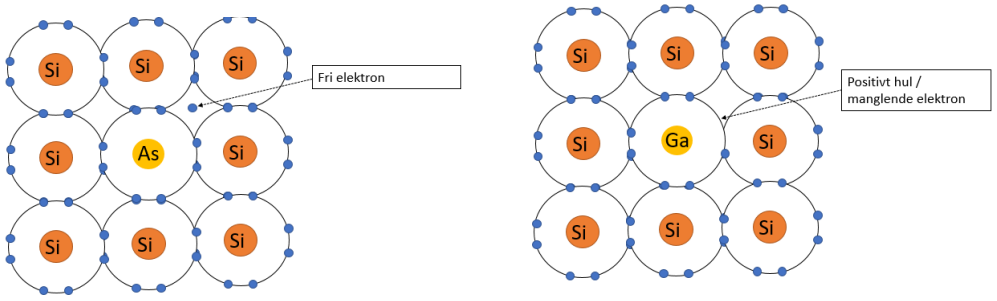
2.1 Halvledermetaller

Når vi betragter et stof som elektrisk ledende, så mener vi at stoffet har elektroner som kan bevæge sig frit mellem atomerne. Man kan beskrive det ved at forestille sig at atomerne "deles"om elektronerne. Metaller, som er meget ledende, har naturligt nemt ved at dele elektroner. Dette skyldes at de elektroner som ligger i metallernes yderste skal samtidigt ligger i det såkaldte *ledningsbånd*. Når et elektron ligger i ledningsbåndet, har det nok energi til at ligge i en orbital hvor det kan rejse mellem andre atomer. Dette danner en "sky"af elektroner som bevæger sig gennem et "gitter"af ioner[5], men hvor der i et neutralt ladet stykke metal er lige mange elektroner som protoner.

Ikke-metaller har også et ledningsbånd, men det ligger i modsætning til metallerne ikke sammen med valensbåndet. For at en elektron kan befinde sig i ledningsbåndet, skal det have et energiniveau der er meget højere end hvad der er normalt i valensbåndet.

Halvledermetaller er specielle fordi de ligger i mellemtrinnet mellem de to. Et halvledermetal har i princippet ingen elektroner i ledningsbåndet, men det kræver et meget lavere energiniveau for elektroner at excitere fra valensbåndet til ledningsbåndet. Man kan derfor hæve et halvledermetals ledningsevne ved at tilføje energi til metallet ved f.eks. at opvarme metallet eller lyse på det. Til gengæld betyder det også at et halvledermetal der har temperaturen 0 K fungerer som en isolator, da der er ingen frie elektroner i ledningsbåndet.

Mængden af energi der skal til for at en elektron exciteres til ledningsbåndet kaldes *båndgabet*, E_g. Jo lavere båndgab, jo mindre energi skal der til for at excitere elektronen til ledningsbåndet. Båndgabet for de to mest brugte halvledere Germanium og Silicium ligger på henholdsvis 0,7 eV og 1,1 eV [5]. Dette betyder at hvis en ren Silicium-halvleder skal absorbere en foton, så en elektron bliver exciteret, så skal fotonen have en bølgelængde på omkring 113 nm. Nogle gange er det nødvendigt at man manipulerer med en halvleders ledeevne. Dette kan opnås ved at introducere urenheder ind i halvledermetallets krystalstruktur, hvilket kaldes *doping*. Et hyppigt eksempel på dette er hvis man udskifter et Si-atom i en siliciumplade med et As-atom. As har en ekstra elektron i sit valensbånd i forhold til silicium, så derfor kan elektronen bevæge sig frit gennem halvlederen, hvilket forbedrer halvlederens ledeevne. Da elektronen er negativ kaldes dette *n-doping*, hvilket kan ses på figur 1a. Ved at dope med et Ga-atom, som har en elektron mindre i valensbåndet, kan man også forbedre ledeevnen. Her er der nu et hul i valensbåndet,



(a) *n-doping, hvor der er en fri elektron på grund af det tilføjede As-atom.*

(b) *p-doping, hvor der er et positivt hul på grund af en manglende elektron. Dette skyldes den tilføjede Ga-atom.*

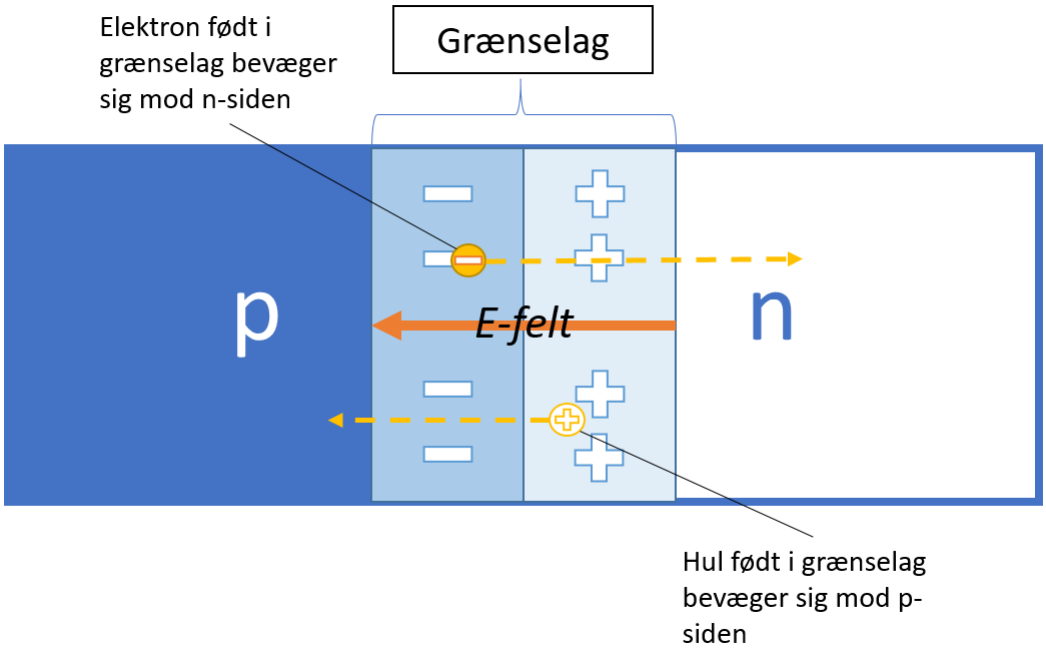
Figur 1: Forskellige former for doping af halvledermetaller. Her er silicium brugt som eksempel.

hvor der er plads til en elektron. Når en elektron udfylder dette hul, skabes der et nyt hul, og sådan bliver det ved. Derfor kan man ved denne type doping "fjerne"en potentiel elektron, hvilket tæller som en positiv ladning. Derfor kalder vi det *p-doping*, hvilket kan ses på figur 1b. Det er vigtigt at man husker at en dopet halvleder stadig er neutralt ladet, selvom der enten bliver introduceret eller fjernet en elektron, da det fremmede atom medbringer ligeså mange protoner som elektroner[5]. Når man nu kombinerer to modsat dopede halvledermetaller, kan man fremstille en *fotodiode*, hvilket er den vigtigste del af en CCD.

2.2 Fotodiode

Hvis man betragter en n-halvleder som havende et overskud af elektroner og en p-halvleder som havende et overskud af "huller"(manglende elektroner), kan det være rigtigt interessant at lægge to af sådanne halvledere ovenpå hinanden. Sker dette opstår der et såkaldt *grænselag* mellem de to, hvor der dannes et elektrisk felt. Dette skyldes at elektronerne på n-siden vil diffunderer til p-siden og rekombinerer med hullerne, hvilket gør antallet af huller på p-siden og elektroner på n-siden mindre. Hvis man husker analogien om at en halvlederen er et gitter af ioner der deles om et hav af elektroner, kan man regne ud at der i dette grænselag vil dannes en positiv og negativ side når elektronerne og hullerne udlignes. I dette grænselag vil derfor opstå et elektrisk felt, som går fra n-siden til p-siden. Alle huller der vil krydse fra p til n og elektroner der vil krydse fra n til p skal derfor have en kinetisk energi der er større end den elektriske potentielle energi som det elektriske felt udgør, den såkaldte *potentialbarriere*. Man kan beskrive grænselagets størrelse med følgende formel [6]:

$$W = \sqrt{\frac{2\epsilon}{q} V_0 \sqrt{\frac{1}{N_A} + \frac{1}{N_B}}} \tag{1}$$



Figur 2: En simpel model af en diode. Elektronen bevæger sig mod n-siden grundet den positivt ladet side af grænselaget. Hullet bevæger sig mod p-siden grundet den negativt ladet side af grænselaget.

hvor ϵ er materialets elektriske permativitet, q er elektronens ladning, V_0 er potentialbarrieren, N_A er koncentrationen af acceptor-atomer og N_D er koncentrationen af donor-atomer. Hvordan denne formel præcist fungerer er underordnet her, men det vigtigste er blot at man kan se at grænselagets størrelse afhænger af potentialbarrieren. Dette gør så at grænselaget vil prøve at opnå en ligevægt, hvor grænselaget er præcist stort nok til at elektroner og huller ikke kan krydse grænselaget tilfældigt. Hvis der til gengæld dannes et elektron-hul par i grænselaget vil det elektriske felt gøre så elektronen bevæger sig mod den positive side og hullet mod den negative side, hvilket er illustreret på figur 2. Hvis halvlederen er forbundet i *spærreretningen*, altså med p-siden sluttet til + og n-siden sluttet til -, så vil der løbe en lille strøm igennem systemet, hvilket kan måles[5]. Eftersom at fotoner med den rigtige bølgelængde kan danne et elektron-hulpar i dette grænselag, kan man måle hvor meget lys der rammer grænselaget ud fra den strøme der løber fra det. Dette er princippet i en fotodiode, hvor man desuden har placeret en mikrolinse så den fokuserer lyset ned mod grænselaget. [7].

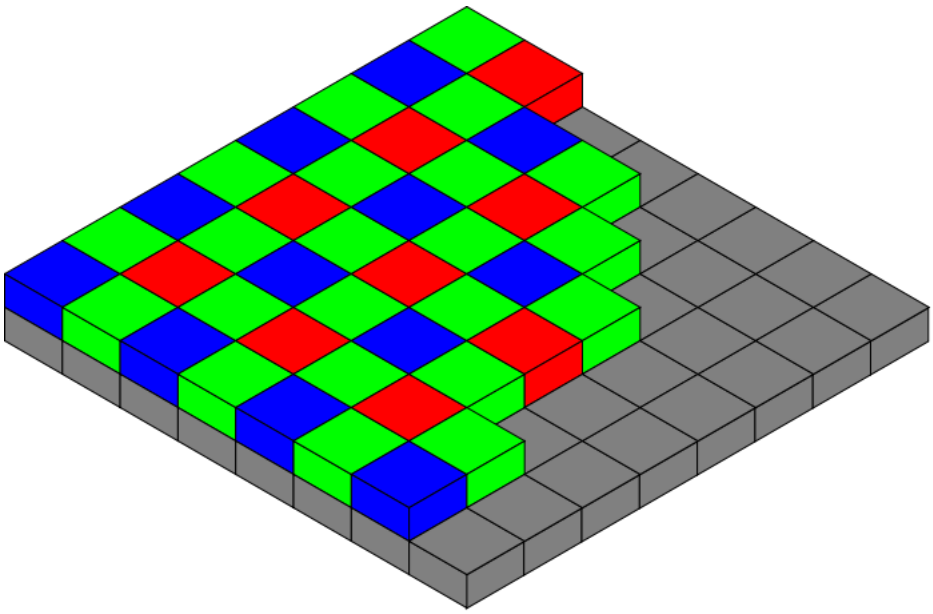
2.3 CCD

Teknologien bag en CCD bygger meget på teknologien bag fotodioden. CCD'en udnytter på samme måde som fotodioden grænselaget i en diode for at måle lysets intensitet. Dog har man

placeret en masse af disse dioder i et gitter, hvor hver diode repræsenterer 1 pixel. Dette betyder at man for at producere et billede i HD skal bruge et gitter på 1280×720 fotodioder. Eftersom at det tager tid at omdanne det elektriske signal til et digitalt billede og alle fotosensorerne skal have lige lang tid til at opfange fotoner, er hver fotosensor udstyret med en *potentialbrønd* der opbevarer de elektroner som bliver skubbet over i n-siden af dioden [8]. Når eksponeringen er slut, lukker kameraet for lyset til CCD'en, hvorefter man kan påbegynde at omdanne elektronerne i potentialbrøndene til et digitalt signal. Eftersom at man i en CCD-sensor kun har én komponent der kan oversætte diodernes analoge signal til digitalt signal (kaldet ADC), aflæses kun elektronerne i én potentialbrønd af gangen. Derfor har CCD'er en mekanisme til at rykke potentialbrønde i rækker, hvorefter hver brønd bliver rykket hen over ADC'en én af gangen. På den måde bliver alle brøndene aflæst, selvom det i tidlige CCD'er kunne være ret tidskrævende. Moderne CCD'er er udstyret med op til 4 læsere til hvert hjørne, hvilket gør det endnu hurtigere at aflæse. Der findes også en anden teknologi kaldet CMOS, hvor hver eneste pixel er udstyret med sin egen omformer, hvilket gør det en del hurtigere at aflæse [9]. Denne teknologi bliver bl.a. benyttet i de fleste moderne smartphones på grund af dens mindre strømforbrug.
For at en selvkørende bil skal kunne bruge et digitalkamera, skal den kunne tage video, og sende konstant information til bilens processor. Derfor skal hele denne process gøres mange gange i sekundet, inden den overhovedet kan blive behandlet. Heldigvis er denne teknologi blevet forsket i af kameraproducenter i årevis, og de fleste kameraer kan nu tage og behandle over 60 billeder i sekundet.

2.4 Farver

I den model af CCD'en som er beskrevet indtil videre er der ikke taget højde for en vigtig del af fotografering, nemlig farver. Vi ved at farver bliver repræsenteret gennem lys i forskellige bølgelængder, men vores dioder indsamler elektroner, hvilket kun bliver exciteret ved én specifik bølgelængde. Dette problem har man løst ved hjælp af to forskellige metoder. For det første har man forsket i en masse forskellige halvledermaterialer, så man nu har materialer med båndgab fra ultraviolet til infrarødt lys [5]. Dette gør så dioderne ikke kun bliver aktiveret af en enkelt bølgelængde, men nu kan være mere følsom for forskelligt lys. Herudover har man opfundet det såkaldte Bayer-filter, som kan isolere specifikke dele af lysets spektrum. Dette gør så man kan styre præcis hvilke farver der rammer hver sensor, ved kun at lukke fotoner igennem som har en specifik bølgelængde. Nu kan man få hver pixel til kun at opfange 1 farve, hvilket man senere kan kombinere til et samlet billede. Som man kan se på figur 3, så er der dobbelt så



Figur 3: Eksempel på hvordan et bayer-filter kan placeres over en CCD. Hvert gråt tern indikerer en pixel, som hvert har et filter der kun lader specifikke farver igennem. Kilde: [10]

mange grønne filtre som blå og røde. Dette skyldes at det menneskelige øje er mere sensitiv til grøn farve, så et billede med meget grøn farve bliver oplevet som mere detaljeret og med mindre støj. Når man har taget et billede med et kamera udstyret med et bayer-filter vil man få et billede der kun beskriver mængden af grøn farve i nogle pixels og mængden af rød og blå i andre pixels. Dette ligner ikke helt et billede endnu, så derfor skal vi behandle vores billede. Dette gøres ved hjælp af en algoritme der kan give et estimat af en pixels mængde af rød, grøn og blå farve ud fra dens nabopixels. På engelsk kaldes dette for *demosaicing*[11]. Et eksempel på et billede før og efter denne proces kan ses på figur 4. Selvom man måske skulle tro at denne proces ville ødelægge billedets kvalitet, så bliver de fleste billeder taget i sådan en høj kvalitet at man sjældent kan se forskel. De fleste kameraer kan desuden også tage billeder i det såkaldte RAW-format, hvor kameraet ikke har forbehandlet billedet endnu, så man selv kan bestemme hvilken algoritme man vil køre over den[11].

3 Machine Learning

Et af menneskets fordele i forhold til computere er vores evne til at lære ud fra erfaring. Vores handlinger er meget baseret på tidligere erfaringer, hvilket gør os mere tilpasningsdygtige i den virkelige verden. Det gør os også til bedre chauffører, da vi bl.a. har lært at genkende forskellige objekter omkring os som vejskilte og hvordan man skal reagere på dem [3].
Computere har normalt ikke denne evne. De gør kun hvad man fortæller dem, men eftersom at man ikke kan beskrive ethvert tænkeligt scenarie i virkeligheden, kan dette være svært. Det

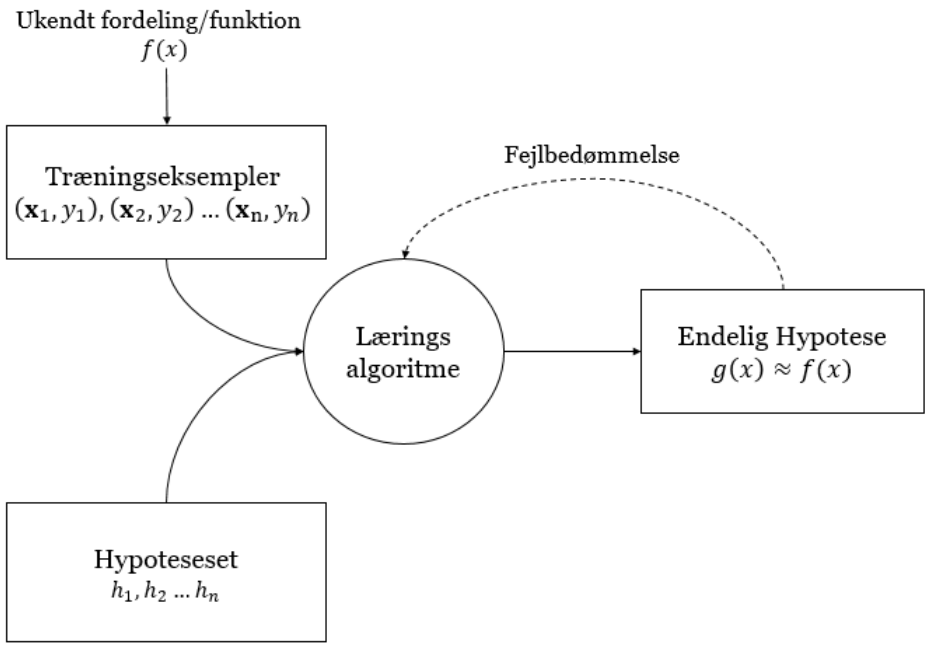
Figur 4: Det samme billede før og efter at blive behandlet af en demosaicing-algorimte. Kilde: [12]

er her at *Machine Learning* kommer ind i billedet. Machine Learning har taget inspiration i menneskets hjerne i den forstand at det kan lære fra tidligere erfaring i form af data. Ud fra denne data kan den derefter bestemme en model som kan benyttes til enten at estimere et udfald (klassifikation) eller et tal (regression)[13].

3.1 Læringsmodellen

Machine Learning er i sig selv blot et værktøj, og som en hammer er ubrugelig uden søm, er Machine Learning ikke noget værd uden data. Data er den absolut mest afgørende faktor på kvaliteten af ens endelige model, og er vores garanti for at modellen kan afspejle virkeligheden. Man kan dog ikke bare bruge alt data som man kan finde, for dataen skal være relevant. Når man vil lære fra data, så skal man finde et udfald som man vil forudse ud fra nogle givne parametre. Dette kan f.eks. være om en vin modtager gode anmeldelser ud fra dens pris, hvor den er fra, hvilken årgang den kommer fra osv. Hvis vi som mennesker får en dyr, fransk vin fra en god årgang i hånden ville vi måske estimere at den har fået gode anmeldelser. Vi har nemlig gode erfaringer med dyre franske vine fra denne årgang, og derfor vurderer vi at denne vin med god sandsynlighed også er god.

Maskiner tænker på samme måde, men eftersom at maskiner ikke kan drikke vin, og derfor aldrig har nydt en dyr fransk vin fra en god årgang, ved den ikke om den har fået gode eller dårlige anmeldelser. Hvis vi tilgengæld giver en masse anmeldelser fra nettet på forskellige vine, kan den finde en korrelation mellem hvordan vinens udfald (anmeldelser) afhænger af dens parametre (land, pris, årgang). Denne korrelation opstiller maskinen som en funktion, der med nok tid og data til sidst kan give et nogenlunde estimat af hvordan anmelderne vurderer vinen. [13]



Figur 5: Læringsmodellen. Ud fra en ukendt funktion har vi fået nogle parametre og udfald. Ved at kombinere dem med en hypotese fra hypotesesettet får man en hypotesefunktion der forhåbentligt estimerer $f(x)$. Herefter måler man hvor stor fejl der er mellem funktionens resultat og det forventede resultat, ud fra hvilken man vælger en ny hypotesefunktion og gentager.

Med matematiktermer kalder vi vinens parametre for $\mathbf{x}$ og dens anmeldelse for y. Eftersom at der er flere parametre til én anmeldelse er $\mathbf{x}$ en vektor, der indeholder hver parameter, så det er nemmere at regne med. Når man begynder at lære fra data, så antager man er der i naturen findes en ukendt funktion der med 100% sikkerhed kan fortælle hvilket udfald man får ud fra givne parametre. Det er ud fra denne funktion at alle vores y-værdier i vores dataset er dannet, og det er denne funktion vi gerne vil estimere. Denne funktion kaldes for *målfunktionen* $f(\mathbf{x})$.
Når vores algoritme prøver at estimere målfunktionen, sker det ved at den vælger en hypotese h fra vores hypoteseset $\mathcal{H}$, som indsættes i vores hypotesefunktion $g(\mathbf{x})$. En sådan hypotese kan f.eks. være at en vins anmeldelse afhænger meget af dens pris og næsten ikke af det land den kommer fra. Til sidst bliver vores hypotesefunktion vurderet op imod $f(\mathbf{x})$ ved at teste den på vores data. Jo mere præcist den kan vurdere vinens oprindelse, jo bedre er modellen. Ud fra hvor præcis modellen er kan man derefter vælge at justere sin hypotese, indtil man har fundet en hypotesefunktion der præcist kan estimere $f(\mathbf{x})$ ud fra den givne data[13] Dette kaldes læringsmodellen, og den er illustreret på figur 5. Det er ud fra denne model at alt indenfor Machine Learning sker.
Når man skal regne i Machine Learning er det rigtigt tit at man skal regne med meget data samtidigt. Derfor er det vigtigt først at have styr på grundprincipperne bag lineær algebra først.

3.2 Matricer

Matricer kan blive betragtet på forskellige måder alt efter hvad man skal bruge det til. En fysiker vil typisk tænke på en matrice som en transformation af et koordinatsystem, hvor en datalog måske vil tænke på en matrice som et skema af tal[14]. Indenfor matematikken benyttes begge synspunkter, men vi får kun brug for datalogens synspunkt.
Derfor beskriver vi en matrice som et skema af tal:

$$\mathbf{A} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1m} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{nm} \end{bmatrix}$$

For matricen $\mathbf{A}$ gælder at den har m rækker og n søjler. Derfor er det en $n \times m$ matrice [15]. Matricer benyttes typisk til at indeholde rigtig mange tal. Derfor er det smart at vi kan foretage nogle forskellige operationer på matricer.
Først og fremmest kan vi brug addition og substitution på en relativ intuitiv måde, ved blot at addere "plads for plads"[15]:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix} \tag{2}$$

Dette kræver dog at begge matricer har samme dimensioner. Man kan også gange en matrice med et tal (en skalar) ved at gange tallet ind på hver plads:

$$x \cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} xa & xb \\ xc & xd \end{bmatrix} \tag{3}$$

Det bliver dog lidt mere kompliceret når man skal til at gange to matricer med hinanden. Der er dog en tommelfingerregel som kan gøre det lidt mere overskueligt at forstå. Den går ud på at *den første matrices søjletal altid er lig den anden matrices rækketal i et matriceprodukt*[16]. Den sætning kan godt være lidt svær at forstå, så lad os bruge et eksempel.
Hvis man har en matrice $\mathbf{A}$ som har dimensionerne 3×4 og en matrice $\mathbf{B}$ som har dimensionerne 4×5 er følgende muligt:

$$\mathbf{A} \cdot \mathbf{B}$$

Dette skyldes at $\mathbf{A}$ har 4 søjler og $\mathbf{B}$ har 4 rækker. Dog er det ikke muligt at sige følgende:

$$\mathbf{B} \cdot \mathbf{A}$$

da $\mathbf{B}$ har 5 søjler og $\mathbf{A}$ har 3 rækker.

Når man så skal beregne matriceproduktet bruger man følgende regneregel:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a \cdot e + b \cdot g & a \cdot f + b \cdot h \\ c \cdot e + d \cdot g & c \cdot f + c \cdot h \end{bmatrix} \tag{4}$$

En måde at forstå dette på er ved at dele den første matrices rækker og den anden matrices søjler op i vektorer. Herefter kan de prikkes sammen i et mønster som kan ses her:

$$\mathbf{a}_1 = \begin{pmatrix} a \\ b \end{pmatrix}, \mathbf{a}_2 = \begin{pmatrix} c \\ d \end{pmatrix}, \mathbf{b}_1 = \begin{pmatrix} e \\ g \end{pmatrix}, \mathbf{b}_2 = \begin{pmatrix} f \\ h \end{pmatrix}$$

$$\mathbf{A} \cdot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{b}_1 & \mathbf{a}_1 \cdot \mathbf{b}_2 \\ \mathbf{a}_2 \cdot \mathbf{b}_1 & \mathbf{a}_2 \cdot \mathbf{b}_2 \end{bmatrix}$$

Alle matricer kan ganges med en kvadratisk enhedsmatrice og give sig selv, altså $\mathbf{A} \cdot \mathbf{E}_{n,n} = \mathbf{A}$. Enhedsmatricen har n dimensioner, og kendetegnes ved at den har 1-taller i diagonalen og 0 resten af stederne. Følgende er en enhedsmatrice i 2×2 dimensioner.

$$\mathbf{E}_{2,2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

En kvadratisk matrice har desuden tilknyttet en determinant. Determinanter kan blive vilkårligt svære at udregne alt efter matricens størrelse, så det vil jeg ikke gå nærmere ind på[15]. Det vigtigste vi skal huske er blot at hvis en matrices determinant er 0, så er matricen *lineær afhængig*. Dette betyder at mindst 1 af matricens søjler kan beskrives som en lineær kombination af de andre. Dette svarer til at sige at mindst en af matricens søjler er proportional med mindst en af de andre søjler.[16].

Ved at bruge disse regneregler kan vi begynde at forstå hvordan en *egenvektor* fungerer. Princippet bag egenvektorer ligger i at man kan gange nogle matricer med en særlig vektor, ud fra hvilket man blot skalerer vektoren. Med matematiktermer kan man beskrive det sådan her:

$$\mathbf{A}\mathbf{v} = \lambda \mathbf{v} \tag{5}$$

Her er $\mathbf{A}$ en matrice som har en egenvektor, $\mathbf{v}$ er en egenvektor til $\mathbf{A}$ og λ er den såkaldte *egenværdi*, der indikerer hvor meget egenvektoren bliver skaleret med. Det er ikke alle matricer som har en egenvektor. Her er et eksempel på en matrice uden en egenvektor, da den eneste vektor som kan indsættes på $\mathbf{v}$'s plads er nulvektoren:

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{v} = \lambda \mathbf{v} = \lambda \vec{0}$$

Om egenvektoren gælder der følgende regler: En matrice skal være kvadratisk før den kan have en egenvektor. Derefter er vi ikke interesseret i at en egenvektor er en nulvektor, da vi så ikke får nogen egenværdi. Den sidste regel er at matricen $\lambda\mathbf{E}_{n,n} - \mathbf{A}$ skal være lineært afhængig for at $\mathbf{A}$ har en egenvektor [17]. Denne sidste regel vil jeg nu prøve at bevise.

Bevis for regel om egenvektorer

Ud fra regel (5) ved vi at for at en egenvektor ud fra vores definition kan opfylde følgende udtryk:

$$\mathbf{Av} = \lambda\mathbf{v}$$

Man kan meget nemt være fristet til at isolere nulvektoren i dette udtryk ved at trække $\mathbf{Av}$ fra. Det giver følgende:

$$\Leftrightarrow \lambda\mathbf{v} - \mathbf{Av} = \vec{0}$$

Eftersom at en vektor kan defineres som en $1 \times n$ matrice, og en enhedsvektor ganget med en matrice giver matricen selv, kan vi sige følgende:

$$\Leftrightarrow \lambda\mathbf{E}_{n,n}\mathbf{v} - \mathbf{Av} = \vec{0}$$

hvilket vi kan faktorisere til

$$\Leftrightarrow (\lambda\mathbf{E}_{n,n} - \mathbf{A})\mathbf{v} = \vec{0}$$

Fordi vi har indsat enhedsmatricen har vi nu gjort $(\lambda\mathbf{E}_{n,n} - \mathbf{A})$ til en matrice. Det betyder at $\mathbf{v}$ er en del af matricens *nulrum*. Nulrummet kan defineres som det set af vektorer som ganget med en matrice giver nulvektoren. Man skriver at en vektor er del af nulrummet med følgende notation:

$$\mathbf{v} \in N(\lambda\mathbf{E}_{n,n} - \mathbf{A})$$

Nulvektoren selv er altid del af nulrummet, men eftersom at vi før har defineret egenvektoren til ikke at være en nulvektor, skal $\mathbf{v}$ altså ikke være $\vec{0}$.
Hvad kan vi bruge dette til? Hvis vi substiturer matricen $(\lambda\mathbf{E}_{n,n} - \mathbf{A})$ med matricen $\mathbf{B}$ for nemmere at kunne læse det kan vi prøve at gange de to matricer sammen:

$$\begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,n} \\ b_{2,1} & b_{2,2} & \dots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m,1} & b_{m,2} & \dots & b_{m,n} \end{bmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Hvis vi skriver matricen om til at indeholde søjlerne som vektorer får vi følgende udtryk:

$$\begin{bmatrix}\mathbf{b}_1 & \mathbf{b}_2 & \dots & \mathbf{b}_n\end{bmatrix}\mathbf{v} = v_1\mathbf{b}_1 + v_2\mathbf{b}_2 + \cdots + v_n\mathbf{b}_n = \vec{0}$$

Eftersom at egenvektoren ikke må være 0, må mindst én vektor i $\begin{bmatrix}\mathbf{b}_1 & \mathbf{b}_2 & \dots & \mathbf{b}_n\end{bmatrix}$ derfor kunne beskrives som en lineær kombination af de andre vektorer i matricen. Man kan finde denne kombination ved blot at trække $v_i\mathbf{b}_i$ fra på begge sider, hvorefter det der står på venstresiden er den lineære kombination der skal til for at danne højresiden.
Ud fra dette kan vi konkludere at λ kun er en egenværdi til $\mathbf{A}$ hvis $\det(\lambda\mathbf{E}_{n,n} - \mathbf{A}) = 0$ dvs. at $\lambda\mathbf{E}_{n,n} - \mathbf{A}$ er lineært afhængig. Hvis vi har en egenværdi må vi ifølge ligning (5) derfor også have en egenvektor, hvilket gør det muligt at finde egenværdier og egenvektorer ud fra en matrice[17].

Et regneeksempel der benytter denne sætning kunne f.eks. være at finde alle egenværdier og egenvektorer til matricen $A = \begin{bmatrix}1 & 2\\ 4 & 3\end{bmatrix}$. Vi kan starte med at finde følgende matrice:

$$\lambda\mathbf{E} - \mathbf{A} = \lambda\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} - \begin{bmatrix}1 & 2\\ 4 & 3\end{bmatrix} = \begin{bmatrix}\lambda - 1 & -2\\ -4 & \lambda - 3\end{bmatrix}$$

Vi kan nu finde determinanten af denne matrice for at bevise at λ er en egenværdi til $\mathbf{A}$. Dette kan vi fordi vi ved at hvis $\det(\mathbf{M}) = 0$, så er $\mathbf{M}$ lineært afhængig:

$$\det\left(\begin{bmatrix}\lambda - 1 & -2\\ -4 & \lambda - 3\end{bmatrix}\right) = (\lambda - 1)(\lambda - 3) - (-4)(-2) = \lambda^2 - 3\lambda - \lambda + 3 - 8$$

$$= \lambda^2 - 4\lambda - 5 = 0$$

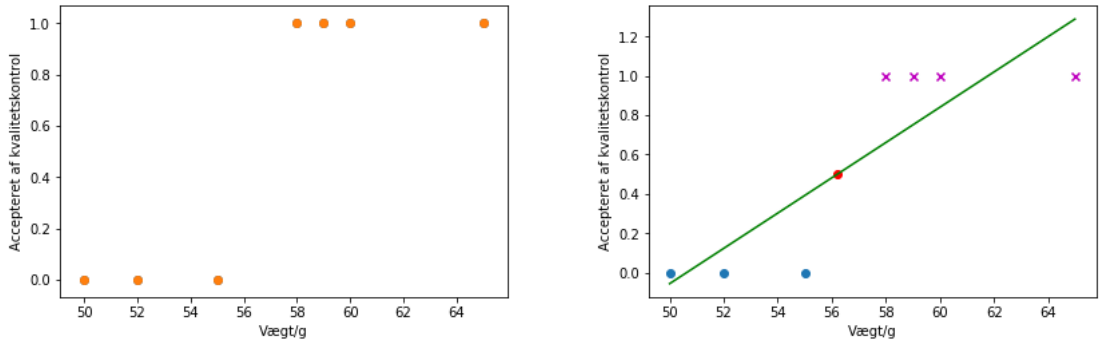
Dette er en andengradsligning som har løsningerne $\lambda = 5$ og $\lambda = -1$. Ud fra dette kan man benytte et CAS-værktøj til at isolere egenvektorerne:

$$(\lambda\mathbf{E} - A)\mathbf{v} = \vec{0}$$

$$\Leftrightarrow \mathbf{v} = \begin{bmatrix}t\\ 2t\end{bmatrix} \text{ når } \lambda = 5 \vee$$

$$\mathbf{v} = \begin{bmatrix}-u\\ u\end{bmatrix} \text{ når } \lambda = -1$$

Hvad t og u specifikt står for er underordnet, da en egenvektor stadig er en egenvektor uanset hvilken længde den har, så længe at forholdet mellem dens elementer er det samme.[17] Dette er alt hvad vi har brug for at vide om matricer inden vi skal benytte dem i Machine Learning.



(a) Produkter fra en opfundet fabrik. 1 på y- aksen betyder at produktet blev accepteret af sepunktet indikerer hvor linjen rammer 0,5 på y-aksen. (b) Lineær regression fittet på dataen. Grænkvalitetskontrol, 0 betyder at den blev afvist.

Figur 6

3.3 Logistisk Regression

En af de mest hyppige problemer som man gerne vil tackle med Machine Learning er det som kaldes klassifikation. I klassifikation kan parametrene være alle mulige slags tal, men udfaldet er diskret. Et eksempel på et klassifikationsproblem kan være at bedømme om en email er spam eller ej, eller om et produkt på en fabrik lever op til kvalitetskontrollen. Her kan man beskrive udfaldsrummet som sandt eller falsk, 0 eller 1 [18]. Parametrene kan dog stadig være reelle tal. Hvis man ser på figur 6a, så fordeler punkterne sig sådan, at alle produkter der når over en specifik vægt bliver accepteret. Så er spørgsmålet blot hvordan man skal bære sig ad med at finde disse punkter?

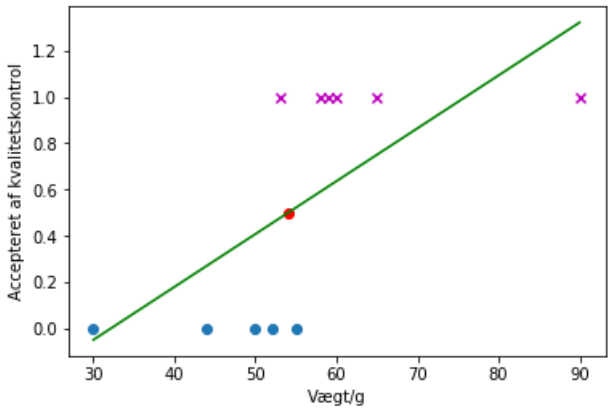
Én metode går ud på at bruge lineær regression på punkterne, hvilket kan ses på figur 6b. Ud fra denne funktion kan man klassificere punkter på følgende måde:

$$\text{Hvis } f(x) < 0.5 \text{ er udfaldet } 0$$

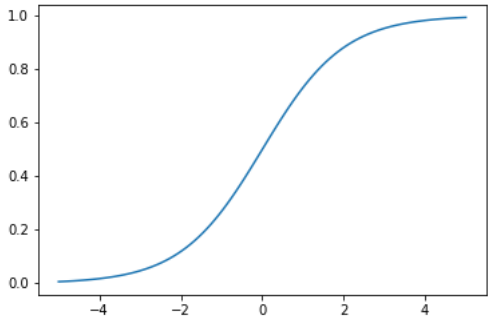
$$\text{Hvis } f(x) \geq 0.5 \text{ er udfaldet } 1$$

På det dataset som vises på figur 6, virker denne metode perfekt. I nogle dataset, som det på figur 7, kan linjen dog ikke gætte alle punkterne rigtigt, da der både er en større spredning og noget støj i dataen. Selvom at der er normalt at en model har en vis fejlmargin, så fortæller vores linje os ikke noget om hvor rigtig eller forkert som vores resultat er. Det er også underligt at $f(x)$ kan være over 1 og under 0, selvom vi aldrig ser det i vores data. Heldigvis er der en anden funktion vi kan bruge end en lineær funktion der passer meget bedre på vores data, nemlig en særlig logistisk funktion kaldet for sigmoid funktionen [18]:

$$h_\theta(\mathbf{x}) = \frac{1}{1 + \mathrm{e}^{-\theta^{\mathrm{T}}\mathbf{x}}}$$



Figur 7: Eksempel på et dataset med mere støj, hvor linjen ikke kan klassificere alle punkterne helt korrekt.



Figur 8: Sigmoidfunktionen plottet i et koordinatsystem

hvor θ er den såkaldte *vægt-vektor* som indeholder alle koefficienterne. Når man plotter følgende funktion i et koordinatsystem vil man lægge mærke til at funktionen går mod 0 når x går mod $-\infty$ og 1 når x går mod ∞. Dette betyder at x aldrig når under 0 og over 1. En konsekvens af dette er både at grænsepunktet fittes til dataen meget bedre, men også at funktionen faktisk også kan beskrive sandsynligheden for at udfaldet er 1, hvilket jeg kan bevise.

Bevis for sammenhæng mellem sigmoid-funktionen og sandsynlighed

Lad os starte med at definerer en stokastisk variabel X, der har udfaldsrummet $U = 0, 1$. X kan altså enten ske eller ikke ske. Ud fra denne information kan vi beregne oddsne for at X sker:

$$\text{odds} = \frac{P(X = 1)}{P(X = 0)}$$

Eftersom at en binær hændelse kun enten kan ske eller ikke kan ske, kan vi omskrive dette udtryk sådan her:

$$\text{odds} = \frac{P(X = 1)}{1 - P(X = 1)}$$

Nu kan vi tage den naturlige logaritme på begge sider:

$$\ln \text{odds} = \ln \frac{P(X=1)}{1-P(X=1)}$$

Hvis vi nu definerer en variabel a til at være den naturlige logaritme til oddsne, kan vi isolere $P(X=1)$, som vi fra nu af bare kalder p. Først opløfter vi det hele i e for at hæve den naturlige logaritme på højre side.

$$a = \ln \frac{p}{1-p}$$
$$\Leftrightarrow \mathrm{e}^a = \frac{p}{1-p}$$

Herefter ganger vi med (1-p) på begge sider

$$\Leftrightarrow \mathrm{e}^a(1-p) = p$$

$$= \mathrm{e}^a - \mathrm{e}^a p = p$$

Så plusser vi med $\mathrm{e}^a p$ på begge sider:

$$\Leftrightarrow \mathrm{e}^a = p + \mathrm{e}^a p$$

Faktoriserer:

$$= \mathrm{e}^a = (1 + \mathrm{e}^a)p$$

og deler med $(1 + \mathrm{e}^a)$

$$= \frac{\mathrm{e}^a}{1+\mathrm{e}^a} = p$$

Nu kan vi faktorisere nævneren:

$$= \frac{\mathrm{e}^a}{(\frac{1}{\mathrm{e}^a}+1)\mathrm{e}^a} = p$$
$$= \frac{1}{1+\mathrm{e}^{-a}} = p$$

Dette er sigmoidfunktionen. Hvis vi lader $\theta^T \mathbf{x}$ beskrive a får vi:

$$p = h_\theta(\mathbf{x}) = \frac{1}{1+\mathrm{e}^{-\theta^T \mathbf{x}}}$$

Hermed er sammenhængen mellem sigmoidfunktionen og sandsynligheden for en hændelse bevist[19].

Selve processen med at fitte denne funktion til et dataset er ikke helt nem. Allerførst skal

man sortere i sine data, så de står på en særlig form. Til klassifikation, som vi er igang med, så skal vi have alt dataen i en matrice $\mathbf{X}$ og alle udfaldene i en vektor $\mathbf{y}$.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_n \end{bmatrix}, \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

hvor hvert element $\mathbf{x}_i$ i $\mathbf{X}$ er en lodret vektor, der indeholder parametrene til punktet i. Når man skal fitte en lige linje i to dimensioner er vi vant til at bruge funktionen for en lige linje:

$$f(x) = ax + b$$

Når vi skal til at bruge d dimensioner, kan det dog hurtigt blive uoverskueligt.

$$f(x) = ax_1 + bx_2 + cx_3 + \cdots + zx_d + k$$

Derfor kan man beskrive det med matricer for at gøre det hele nemmere. Dette gør vi ved at opbevarer $x_1, x_2 \dots x_d$ i en vektor $\mathbf{x}$ og alle koefficienterne $a, b, c...$ i en vektor θ. Eftersom at vi skal have et reelt tal, skal vi enten prikke dem sammen eller betragte dem som matricer, og gange den transponerede θ^T med $\mathbf{x}$:

$$f(x) = \theta^T \mathbf{x} + k$$

Vi kan til sidst fjerne vores k ved at indsætte den som θ_0 og sætte x_0 til altid at være 1. Derfor vil vi altså til sidst kunne beskrive en lige linje som

$$f(x) = \theta^T \mathbf{x} = \theta_0 \cdot 1 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Dette beskriver grunden til at vi bruger en vægtvektor i sigmoidfunktionen, da den giver os en nem måde at holde styr på alle koefficienterne [18].

Hvis vi husker tilbage på læringsmodellen, så går Machine Learning ud på at man først vælger en hypotese ud fra et hypoteseset. Denne hypotese er faktisk vores θ. Når vi så har valgt vores koefficienter, indsætter vi dem i sigmoidfunktionen, og ser hvor godt den passer. Måden man ser hvor godt den passer på, er ved at bruge en såkaldt *cost-funktion*. Pointen bag cost-funktionen er at den ud fra en vægt-vektor kan fortælle hvor rigtig eller forkert hypotesefunktionen kan forudse et datapunkts udfald. Jo højere tal som cost-funktionen giver, jo dårligere estimere hypotesefunktionen det egentlige udfald.

Ved at minimere cost-funktionen, kan vi finde den hypotese der har den mindste fejl og derfor

passer bedst på vores data[18].

For at vi kan gøre dette, skal vi dog definere denne cost-funktion. For logistisk regression bliver den typisk defineret sådan her:

$$\text{cost}(\theta) = \begin{cases} -\log\left(h_\theta(\mathbf{x}_i)\right) \text{ hvis } y = 1 \\ -\log\left(1 - h_\theta(\mathbf{x}_i)\right) \text{ hvis } y = 0 \end{cases} \tag{6}$$

Dette betyder at jo tættere på det rigtige udfald som vores hypotesefunktion gætter, jo tættere på 0 bliver cost-funktionen. Jo mere forkert som hypotesefunktionen til gengæld er, jo større bliver cost-funktionen. Dette kan vi skrive på en enkelt linje sådan her [18]:

$$\text{cost}(\theta) = \left(-y \log\left(h_\theta(\mathbf{x}_i)\right)\right) + \left(-(1-y) \log\left(1 - h_\theta(\mathbf{x}_i)\right)\right)$$

hvilket har samme funktion som ligning (6). Indtil videre tester vores cost-funktion kun et enkelt datapunkt, men eftersom vi gerne vil have en overordnet idé om vores hypoteses præstation vil vi gerne teste hele vores dataset. Derfor kan vi finde vores gennemsnitlige cost gennem en funktion kaldet $J(\theta)$:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y \log\left(h_\theta(\mathbf{x}_i)\right) + (1-y) \log\left(1 - h_\theta(\mathbf{x}_i)\right)\right] \tag{7}$$

hvor m er antallet af datapunkter. Det er denne sætning vi gerne vil minimere når vi optimerer vores model. For at finde $\min_\theta J(\theta)$ bruges typisk en metode kaldet *gradient descent*[18]. Teorien bag gradient descent bygger på at vi gerne vil finde et globalt minimum til $J(\theta)$. Dette gøres ved at man først differentierer $J(\theta)$ til $J'(\theta)$, der beskriver $J(\theta)$'s hældning. Normalt når man optimerer plejer man blot at sætte $J'(\theta)$ til 0, og så løse efter θ. På grund af $J(\theta)$'s kompleksitet er dette dog endnu ikke lykkedes nogen endnu. Derfor må man lede efter dette minimum, skridt for skridt. Dette gøres ved at man i et hvilket som helst punkt beregner $J'(\theta)$. Hvis $J'(\theta)$ er positiv, bliver vi nødt til at gøre vægt-vektoren mindre, og hvis den er negativ må vi gøre vægt-vektoren større. Dette betyder at man ved at prøve forskellige justeringer proportionale med hældningens størrelse på θ til sidst kan finde en hypotesefunktion hvor $J'(\theta) \approx 0$ [18].

Ud fra dette kan vi udvælge en ny vægt-vektor sådan her:

$$\theta_{j,t} := \theta_{j,t-1} - \alpha J'(\theta) \tag{8}$$

hvor t angiver hvilken iteration af algoritmen vi er på,$\theta_{j,t}$ angiver vægt nummer j i vektor θ til iteration t, og α angiver en proportionalitetskonstant, der gør så θ ikke tager for store skridt af gangen, og muligvis går forbi minimum. α skal vælges inden man kører algoritmen, hvor en meget lille α giver en høj præcision, mens en stor α giver en hurtigere køretid. Normalt ville

Tabel 1: Forskellige specifikationer til biler fra enten Audi eller Toyota. Kilde: [21]

Producent	Længde/in	Bredde/in	Højde/in	Vægt/lbs	Hestekræfter	Pris/$
Audi	176.6	66.2	54.3	2337	102	13950
Audi	176.6	66.4	54.3	2824	115	17450
Audi	177.3	66.3	53.1	2507	110	15250
Audi	192.7	71.4	55.7	2844	110	17710
Audi	192.7	71.4	55.7	2954	110	18920
Toyota	158.7	63.6	54.5	2015	62	6488
Toyota	169.7	63.6	59.1	2280	62	6918
Toyota	169.7	63.6	59.1	2290	62	7898
Toyota	169.7	63.6	59.1	3110	62	8778
Toyota	166.3	64.4	53	2081	70	6938

jeg selv differentiere $J(\theta)$, men det viser sig at være en længere affære, som jeg har valgt at nedprioritere. Når man har gjort det får man i hvert fald følgende

$$\theta_{j,t} = \theta_{j,t-1} - \alpha \sum_{i=0}^{m} (y_i - h_\theta(\mathbf{x}_i)) x_j^{(i)} \tag{9}$$

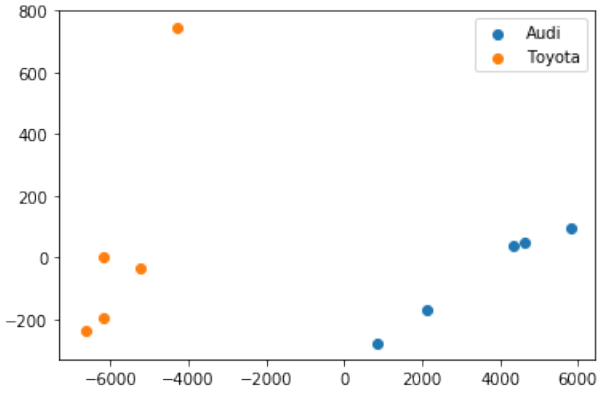
hvor $x_j^{(i)}$ er parameter j fra vektor $\mathbf{x}_i$ i $\mathbf{X}$. Denne version virker, men tager rigtig lang tid at udregne, eftersom at man udregner hver vægt enkeltvis. Derfor kan vi vektorisere dette udtryk til følgende[20]:

$$\theta := \theta - \alpha \mathbf{X}^T (\mathbf{y}^T - h_\theta(\mathbf{X}^T))^T \tag{10}$$

Dette giver os en ny vægt-vektor, som forhåbentligt er tættere op minimum end før. Denne formel gentages indtil at man er tilfreds med resultaterne.[18] Lad os benytte alt det vi har lært i et regneeksempel.

3.4 Regneeksempel med PCA og logistisk regression

Lad os starte med et simpelt dataset, der relevant nok handler om biler, der kan ses på tabel 1. Vi vil ud fra denne data finde en funktion der kan give et estimat af hvorvidt en bil er en Audi eller en Toyota ud fra en kombination af disse parametre. Det første man skal gøre er at undersøge om der er en sammenhæng mellem dataens udfald og parametre. Hvis man kun havde to parametre per datapunkt, kunne man plotte dem i et koordinatsystem og se om de grupperer sig efter deres udfald. Dette er dog svært, da vi har 6 dimensioner der skal kunne vises i et todimensionelt koordinatsystem. Heldigvis kan vi bruge en metode kaldet *Principal Component Analysis* (PCA). PCA kan reducere antallet af dimensioner i et dataset og samtidigt miste mindst muligt information ved hjælp af kovarians og egenvektorer. Ud fra dette kan man få et transformeret dataset der på en relativt troværdig måde kan repræsentere ens



Figur 9: PCA udført på dataset om biler, hvor datasettet bliver nedsat fra 6 til 2 dimensioner. Læg mærke til at der en form for gruppering blandt datapunkterne der svarer til deres bilmærker.

data. Selve algoritmen bag PCA er meget interessant, men eftersom at det blot kan betragtes som en "fætter"til machine learning, kan man blot nøjes med at benytte PCA som et værktøj uden yderligere uddybning. Hvis man derfor kører datasettet igennem en PCA-algoritme får man et transformeret dataset som er plottet i figur 9 [22]. Ud fra dette punktplot kan man se at de to bilmærker virker til at gruppere sig i hver sin side. Dette betyder at der er en form for korrelation mellem datapunkternes parametre og deres udfald, som endda virker til at være lineær. Derfor kan vi lave logistisk regression på vores datapunkter.

Lad os starte med at indsætte hvert datapunkt i en matrice:

$$\mathbf{X} = \begin{bmatrix} 176,6 & 176,6 & 177,3 & \dots & 166,3 \\ 66,2 & 66,4 & 66,3 & \dots & 64,4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 13950 & 17450 & 15250 & \dots & 6938 \end{bmatrix}, \mathbf{y} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

Her har alle Audier fået en y-værdi på 0 og alle Toyotaer fået en y-værdi på 1. Derfor kommer vores endelige funktion til at estimere sandsynligheden for en bil at være en Toyota. Vi skal herefter definere en starthypotese:

$$\theta = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \vec{0}$$

Nu har vi teknisk set en model, men ikke en særlig god en. For at den skal blive god, bliver vi nødt til at træne den ved hjælp af gradient descent. Lad os indsætte vores vægt og α på 0,05 i formel (10):

$$\theta := \vec{0} - 0,05\mathbf{X}^T(\mathbf{y}^T - h_\theta(\mathbf{X}^T))^T = \begin{pmatrix} -2,0 \\ -0,57 \\ 0,29 \\ -42,3 \\ -5,7 \\ -1156,5 \end{pmatrix}$$

Dette gør vi så endnu en gang:

$$\theta := \theta - 0,5\mathbf{X}^T(\mathbf{y}^T - h_\theta(\mathbf{X}^T))^T = \begin{pmatrix} 39,7 \\ 15,4 \\ 14,5 \\ 546,6 \\ 10,2 \\ 694,5 \end{pmatrix}$$

og sådan fortsætter man. Jeg har valgt at udføre dette 50 gange, hvilket til sidst giver mig vægtene:

$$\theta = \begin{pmatrix} 497.43 \\ 194.8425 \\ 191.5075 \\ 6685.9 \\ 80.475 \\ -1407.8 \end{pmatrix}$$

Nu har vi tilpasset vores model. Den kan vi nu teste både på vores egen data og på noget ny data. Som man kan se på tabel 2, så gætter vores model faktisk rigtigt i 100% af tilfældene. Dog skal det lige medtages at der er en ret begrænset mængde testdata som vi har til rådighed udenfor datasettet, hvilket gør så vi ikke kan komme med nogen god statistik for *hvor* godt modellen virker. Vi kan blot se at den både har formået at klassificere alle datapunkterne inde i datasettet og udenfor datasettet korrekt.

Som vi tidligere har sagt, så er data essentielt for modellens evne til at afspejle virkeligheden. Derfor er det lidt et problem at vi kun har ti datapunkter, da det sænker sandsynligheden for at den fejl som vi måler indenfor datasettet er tæt på den fejl modellen har udenfor datasettet.

Tabel 2: Testresultater for den færdigtrænede model

Bilmærke	Forventet værdi	Modellens resultat
Audi #1	0	0
Audi #2	0	0
Audi #3	0	0
Audi #4	0	0
Audi #5	0	0
Toyota #1	1	1
Toyota #2	1	1
Toyota #3	1	1
Toyota #4	1	1
Toyota #5	1	1
Biler uden for datasettet		
Toyota #6	1	1
Toyota #7	1	1
Toyota #8	1	1
Audi #6	0	0
BMW #1	?	0

Dette er svært for os at bevise med så lidt testdata, men hvis vores testdataset var større, ville man nok kunne se en forskel [13].

4 Forsøg

Vi har undersøgt både hvordan digitalkameraet kan opfange et billede af verden omkring det, og hvordan Machine Learning kan estimere funktioner og fordelinger. Hvordan dette relaterer sig til selvkørende biler kan dog måske være svært at forudse, så derfor har jeg tænkt mig at udføre et forsøg, der kan demonstrere hvordan en selvkørende bil kan benytte disse to værktøjer under kørsel.

Jeg vil gøre dette ved træne en model, der ud fra billeder taget med et digitalkamera, kan undersøge om der er et menneske tilstede på billedet. Dette kan have en hel reel praktisk applikation i en selvkørende bil, da det er vigtigt for bilen at kunne opdage hvorvidt at der er en fodgænger foran bilen. Desuden benytter dette eksperiment både et digitalkamera og machine learning, som begge er ting vi allerede har redegjort for[3].

4.1 Dataindsamling

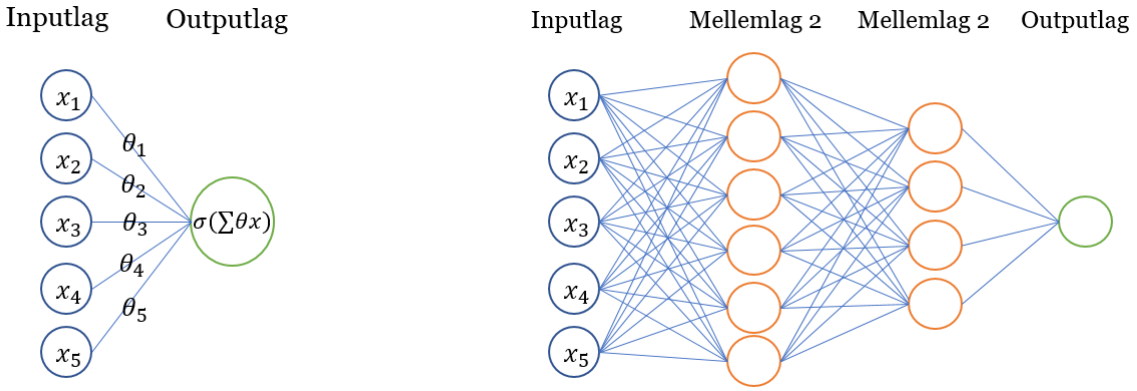
Før vi kan træne noget netværk skal vi bruge data, og gerne meget af det. Dette har jeg opnået ved at bruge en eftermiddag på Allerød Gymnasium, hvor jeg har taget så mange billeder af mennesker som jeg kunne få lov til. Herefter har jeg også taget billeder som ikke indeholder mennesker, da vores model skal kunne kende forskel mellem de to typer billeder. Hvert billede blev taget af en Huawei P10 mobiltelefon, er i farver og har dimensionerne 3968×2976, og da jeg var færdig med at tage billeder havde jeg 77 billeder af mennesker og 25 billeder uden mennesker.

4.2 Databehandling

Til at behandle dataen har jeg valgt at træne et såkaldt *convolutional neural network*, der er en form for neuralt netværk. Et neuralt netværk har den fordel at den er en *universal function approximator*, hvilket betyder at den kan estimere meget komplicerede funktioner. Dette er smart fordi den derfor kan bruges til mange flere opgaver end f.eks. logistisk regression som f.eks. billedgenkendelse, talegenkendelse, medicinske formål, finans, spamfiltrering og mere[23]. Dette skyldes at den selv kan bygge sit eget abstraktionsniveau ud fra sit input. Hvis et menneske skal klassificere et billede af en kat, ville man måske tælle antallet af knurhår, om den har en hale, eller et ansigt der ligner en kat. Et neuralt netværk er aldrig blevet lært hvordan en hale eller knurhår ser ud, den modtaget bare en inputvektor med data. Den kan til gengæld med nok data og computerkraft selv finde genkendende træk ved en kat, der hjælper den til at klassificere ny data. Disse træk kan være ligesom mennesket knurhår, haler og lign, men det kan også være noget nyt som modellen kun selv forstår. Vi kan som mennesker aldrig helt forstå hvad det er for nogle træk som modellen benytter, kun se resultatet. Derfor kalder man typisk et neuralt netværk for en "black box"[24].

Det viser sig, at et neuralt netværk som det beskrevet ikke er synderligt mere kompliceret end simpel logistisk regression. Faktisk kan man sige at et neuralt netværk *er* logistisk regression, bare gjort mange gange og kombineret med sig selv.

På figur 10a kan man se en almen logistisk regression. Her har man noget data som hver ganges med en vægt og køres igennem sigmoidfunktionen. Dette er godt til at bestemme simple sammenhænge og løse klassifikationsproblemer hvor to grupper punkter kan adskilles med en lineær funktion. På figur 10b ses derimod et simpelt neuralt netværk med tre lag. Dette netværk kan klassificere meget mere præcist end den logistiske regression, men kræver til gengæld meget



(a) Logistisk regression med 1 lag.

(b) Neuralt netværk med 3 lag.

Figur 10

mere data og computerkraft at træne.

Hvis vi skulle bruge et ordinært neuralt netværk til vores billedgenkendelse, så skulle vi have et inputlag der er $3968 \times 2976 = 11.808.768$ elementer langt. At skulle foretage udregninger på dataset af denne størrelsesorden på min computer ville tage længere end min levetid. Derfor bruger man et trick der hedder *kernel convolutions*, som både kan hjælpe med at genkende træk som kanter og hjørner i billedet, og samtidigt skalere billedet ned til en mindre størrelse. Convolutions har på samme måde som logistisk regression vægte som indstilles gennem træning af netværket, hvilket gør så de kan opfange alle mulige mønstre i forskellige abstraktionslag. I et netværk der skal genkende mennesker kan man forestille sig at det første lag af convolutions opdager kanter og hjørner, et andet lag opdager hoved-lignende objekter, arme og ben, et tredje lag opdager øjne og munde. Til sidst vil det neurale netværk til sidst vurdere om der er tale om et menneske eller ej[25].

Problemerne ved neurale netværk er som sagt at de kræver meget data og meget computerkraft, hvilket begge ting er noget jeg mangler. Derfor har jeg valgt at bruge en model som Google allerede selv har trænet kaldet Inception. Inception er blevet trænet over en masse billeder fra internettet i to uger på en meget kraftig computer, hvilket har gjort den i stand til at klassificerer en masse forskellige billeder. Den har desuden også udviklet en masse convolutions og vægte som generelt er nyttige i billedgenkendelse. Dem kan vi gøre brug af ved hjælp af *transfer learning*, en metode til at omtræne det sidste lag i et neuralt netværk så det virker med vores egne billeder [26].

Dette har jeg gjort ved hjælp af et værktøj kaldet TensorFlow. TensorFlow tillader mig nemt at omtræne Inception med de billeder som jeg giver den. Herefter giver den mig en model som jeg kan benytte til at klassificere nye billeder. Præcist hvilke kommandoer jeg har skrevet kan ses her: [27]. Jeg har i bilag 1 vist eksempler på forskellige testresultater, men overordnet set kunne

den (næsten) gætte alle mine testresultater rigtigt. I alt kunne den gætte alle testbillederne med rigtige mennesker på rigtigt, men den kunne godt komme i tvivl når der var menneskelignende objekter i billeder som ikke forestillede mennesker. I bilag 1 kan man f.eks. se at et billede af en dukke blev klassificeret som et menneske, selvom det ikke er et rigtigt menneske. Man kan også se at modellen blev lidt i tvivl når den skulle klassificere en chimpanse, som har menneskelignende træk, men ikke er et rigtigt menneske.

Jeg tror at der er en ret stor usikkerhed i denne model i form af at den kun er trænet på omkring 100 billeder. Når man skal træne store netværk der kan klassificere mange slags billeder med høj sikkerhed er det vigtigt at have rigtig meget forskelligt data. Jeg har f.eks. kun trænet billeder taget på AG, så modellen er måske blevet vant til at der altid er en grålig baggrund, og ville klare det dårligt i f.eks. en skov. Hvis denne model også skulle blive benyttet på en selvkørende bil ville det derfor nok være nødvendigt at have trænet det på billeder af fodgængere og ikke bare tilfældige elever og lærerer på et gymnasium. Den skal være trænet i alt slags terræn og lysniveau, og indeholder billeder af mennesker fra alle mulige vinkler. Dette betyder nok at man for at have en god model gerne skal have 1000+ billeder med og uden mennesker i for at lave en optimal model. Hvis det til gengæld er lykkedes en, kan man nu opdage fodgængere med en relativ høj præcision som er nødvendig for selvkørende biler, hvilket kan føre til en meget mere sikker kørsel i gaderne.

5 Konklusion

Selvkørende biler indeholder mange teknologier som alle bidrager til deres kørsel. De er udstyret med sensorer, som f.eks. digitalkameraets CCD, hvis fysik og betydning for bilen er blevet redegjort for. Dette inkluderede en gennemgang af halvledermetaller og dioder sammen med en præsentation af de såkaldte Bayer-filtre. Disse sensorer giver et udgangssignal som skal behandles, hvilket udføres af forskellige matematiske modeller, trænet af machine learning. Derfor er både teorien bag matricer og machine learning generelt blevet redegjort, hvilket inkluderede et bevis af en regneregel om egenvektorer og et kort regneeksempel. Denne teori kunne herefter benyttes til at forklare om teorien bag klassifikation, en kategori indenfor machine learning, hvilket også indeholdt et bevis og et regneeksempel.

Til sidst blev CCD'ens sammenhæng med selvkørende biler påvist gennem et eksperiment der viste hvordan man ved hjælp af machine learning kunne benytte CCD'ens udgangssignal, billeder, til at klassificere hvorvidt et menneske befandt sig foran sensoren. Ud fra dette kunne det konkluderes at en CCD kan være en relevant sensor på en selvkørende bil, da det kan hjælpe

den med bl.a. at identificere fodgængere og evt. vejskilte og vejstriber.

Litteratur

[1] Waymo, "Waymo, on the road." https://waymo.com/safetyreport/. Hentet: 07-12-2017.

[2] NHTSA, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey." https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115. Udgivet: 02-2017, Hentet: 07-12-2017.

[3] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, and J. Pazhayampallil, "An empirical evaluation of deep learning on highway driving," *Cornell University Library*. Udgivet: 17-04-2017, Hentet: 07-12-2017.

[4] Wikipedia, "Digital camera." https://en.wikipedia.org/wiki/Digital_camera. Hentet: 08-12-2017.

[5] P. Holck, J. Kraaer, and B. Merci Lund, *ORBIT A htx*. Systime, 2009. 1. udgave, 2. oplag. ISBN-13: 978-87-616-1398-1.

[6] C. Honsberg and B. Stuart, "Solving for depletion region." http://www.pveducation.org/pvcdrom/solving-for-depletion-region. Hentet:18-12-2017.

[7] G. S. University, "Photodiode slide." http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/photdet.html. Hentet: 10-12-2017.

[8] A. U. Institut for Fysik og Astronomi, "Sådan virker en ccd." http://phys.au.dk/forskning/forskningsomraader/ole-roemer-observatoriet/teknisk-information/saadan-virker-en-ccd/. Hentet: 10-12-2017.

[9] B. Hammack, "Ccd: The heart of the digital camera (how a charge coupled device works)." https://www.youtube.com/watch?v=wsdmt0De8Hw. Udgivet: 15-05-2012, Hentet: 10-12-2017.

[10] Wikimedia, "Bayer pattern on sensor." https://commons.wikimedia.org/wiki/File:Bayer_pattern_on_sensor.svg. Hentet: 11-12-2017.

[11] M. Pound and S. Riley, "Capturing digital images (the bayer filter) - computerphile." https://www.youtube.com/watch?v=LWxu4rkZBLw&t=73s. Udgivet: 22-02-2015, Hentet: 10-12-2017.

[12] "Digital camera sensors (kun brugt illustration)." http://www.cambridgeincolour.com/tutorials/camera-sensors.htm. Hentet: 11-12-2017.

[13] Y. S. Abu Mostafa, M. M. Ismail, and H. T. Lin, *Learning From Data - A short course*. 2012. ISBN 13: 978 1 60049 006 4.

[14] G. Sanderson, "Essence of linear algebra (playliste)." https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab. Hentet: 13-12-2017.

[15] G. R. Ebbesen, "Matricer og lineære ligningssystemer." https://www.uvmat.dk/paradig/not/n242-1.pdf. Hentet: 13-12-2017.

[16] J. Carstensen, *Lineær algebra*. Systime a/s, 1994. 1. udgave, 1. oplag, ISBN: 87 7783 507 7.

[17] S. Khan, "Eigen-everything (playliste)." https://www.khanacademy.org/math/linear-algebra/alternate-bases/eigen-everything/v/linear-algebra-introduction-to-eigenvalues-and-eigenvectors. Hentet 14-12-2017.

[18] A. Ng, "Machine learning - andrew ng, stanford university." https://www.youtube.com/playlist?list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN. Hentet 14-12-2017.

[19] B. Foltz, "Statistics 101: Logistic regression, logit and regression equation." https://www.youtube.com/watch?v=NmjT1_nClzg. Hentet 15,12,2017.

[20] "Logistic regression vectorization example, stanford university." http://deeplearning.stanford.edu/wiki/index.php/Logistic_Regression_Vectorization_Example. Hentet 16-12-2017.

[21] R. Srinivasan, "Automobile dataset." https://www.kaggle.com/toramky/automobile-dataset. Hentet: 19-12-2017.

[22] Wikipedia, "Principal component analysis." https://en.wikipedia.org/wiki/Principal_component_analysis. Hentet 16-12-2017.

[23] Wikipedia, "Artificial neural network." https://en.wikipedia.org/wiki/Artificial_neural_network. Hentet 17-12-2017.

[24] S. Riley and B. Martinez, "Deep learning - computerphile." https://www.youtube.com/watch?v=l42lr8AlrHk. Udgivet 19-04-2016, Hentet 17-12-2017.

[25] S. Riley and D. M. Pound, "Neural network that changes everything - computerphile." https://www.youtube.com/watch?v=py5byOOHZM8. Udgivet 20-03-2016, Hentet 17-12-2017.

[26] J. Shlens, "Train your own image classifier with inception in tensorflow." https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html. Udgivet 09-03-2016, Hentet 17-12-2017.

[27] "Tensorflow for poets." https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/. Hentet: 19-12-2017.

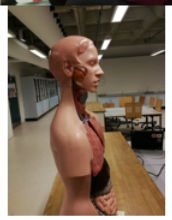
Bilag

Bilag 1: Testresultater fra neuralt netværk.

Menneske: 1,00

Menneske: 1,00

Menneske: 1,00



Menneske: 1,00

Ikke Menneske: 1,00

Ikke Menneske: 0,99

Ikke Menneske: 0,99

Ikke Menneske: 0,79