Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Datateknologi	Vårsemesteret 2021 Åpen
Forfattere: Eirik Hvitsten, Magnus Gulbrandsen og Sigurd Grøvdal Hansen	
Fagansvarlig: Tomasz Wiktorski Veileder: Tomasz Wiktorski	
Tittel på bacheloroppgaven: Garmin applikasjon for analyse av helsedata Engelsk tittel: Garmin application for analysis of health metrics	
Studiepoeng: 20	
Emneord: Maskinlæring, Garmin, Aterosklerose	Sidetall: 76 + vedlegg/annet: 9 Stavanger 15. mai 2021

Sammendrag

Denne bacheloroppgaven går ut på lage en applikasjon som skal brukes på en Garmin-smartklokke. Applikasjonen bruker de to maskinlæringsmetodene lineær regresjon og bestemmelsestre til å komme fram til om en person har aterosklerose eller ikke. I oppgaven blir det også gått mer inn på andre metoder som kunne blitt brukt, samt Garmin, aterosklerose og ikke minst hvordan denne applikasjonen har blitt og hvorfor den er blitt slik. Det skal også være lett for andre å bruke, endre og å laste ned programvaren på sin egen klokke.

Innholdsfortegnelse

Liste av figurer

1 Introduksjon

1.1 Oppgavebeskrivelse

Oppgaven går ut på å utvikle en applikasjon til Garmin sine klokker ved hjelp av deres rammeverk. Rammeverket tillater utviklere å lage sin egen applikasjon til sitt eget bruk, eller til eventuell publisering. Det er forksjellige typer applikasjoner som kan lages, alt fra utseendeendringer til hele applikasjoner. I oppgaven skal det utvikles en applikasjon som har fokus på målingene som enhetene tar inn.

Oppgaven er basert på tidligere forskning, som viser at det kan være mulig å avdekke aterosklerose hos personer under trening. Applikasjonen skal derfor benytte maskinlæringsmodeller for å gjenkjenne mønster i hjerteraten. Ved å se nærmere på ulike perioder, kan det tidvis være et skille mellom personer med og uten aterosklerose, som er interessant å se nærmere på.

Applikasjonen som utvikles skal være brukervennlig både for forskningen og for personene som bruker klokken. Det skal derfor prøves ulike utseender for å finne et optimalt ett. Det skal også gjøres enkelt å implementere maskinlæringsmodellene, slik at de stadig kan oppdateres med nye datasett. Samtidig skal applikasjonen være enkel å oppdatere.

1.2 Mål med oppgaven

1.2.1 Presise resultater

Applikasjonen skal evne å skille mellom utøvere med og uten aterosklerose med en lav feilmargin. Ved bruk av maskinlæringsalgoritmer, vil det alltid være rom for feil. Algoritmene resulterer i estimater, som i det store og hele er kalkulerte gjetninger. I frykt for at det skal være ett enten eller mål, altså enten så estimerer man riktig eller så er det feil, legges det vekt på å øke presisjonen og redusere feilmarginen.

1.2.2 Effektiv integrering av modeller

Applikasjonen skal evne å integrere maskinlæringsmodeller på en effektiv måte. Ettersom det skal analyseres datasett, og det skal utvikles en applikasjon for å estimere en helse-relatert tilstand, vil bruk av maskinlæringsmodeller være uunngåelig. Målet blir derfor å velge modeller som er optimale, både i henhold til bruk, men også til integrering på applikasjonen. Det skal være enkelt og raskt

å analysere datasett, for så å oversette modellene til bruk på klokken, ettersom det stadig vil være endringer. Desto færre prosesser som må settes i gang, desto bedre.

1.2.3 Applikasjonen skal kunne påbygges

Applikasjonen skal kunne utvides og ekspanderes, med tanke på forskning som skjer parallelt. Med tanke på at forskning rundt temaet er pågående, vil det med stor sannsynlighet være endringer i fremtiden. Det er derfor viktig at det tas hensyn til fleksibilitet når applikasjonen utvikles. Dette er både rettet mot applikasjonen og ressurser som benyttes ved siden av, med tanke på koding, modeller, integrasjon og bruk.

1.2.4 En brukervennlig applikasjon

Applikasjonen skal være enkel å bruke og lett å forstå. Det skal være mulig for en person i alle aldere å bruke applikasjonen uten problemer. Det medfører at designet må være enkelt både utseendemessig og funksjonelt, gjennom enkle og tydelige farger, samt enkle funksjoner. Applikasjonen er også forskningsrelatert, som gjør at den må være vell så brukervennlig med tanke på oppdatering og vedlikehold.

1.2.5 Lavt batteriforbruk

Applikasjonen skal være effektiv med tanke på strømforbruk. De aller fleste har nok vært i en situasjon hvor det har vært behov for strøm på en elektronisk enhet, men det har manglet. Det kan være frustrerende. Det er derfor viktig at applikasjonen benytter optimale algoritmer for å unngå at strømmangel skyldes dårlig implementering.

1.3 Motivasjon for oppgaven

Motivasjonen for oppgaven ligger i både interessen for trening og velvære og interessen for dataanalyse. Det er stor interesse rundt trening og velvære, og samtlige har brukt treningsklokker tidligere. Av den grunn er det interessant å se nærmere på målingene som klokkene tar opp, og hvordan disse kan bearbeides.

Maskinlæring er et tema som er i stadig utvikling, og mulighetene innenfor temaet vokser stadig, i takt med teknologiens utvikling. Det er derfor høyst interessant å se nærmere på maskinlæringsmodeller, for å lære mer om hvordan de funker og potensielt anvende de.

I tillegg er det interessant om det hadde vært mulig å bidra til et forskningsrelatert helseproblem, ved å utvikle en applikasjon som kan hjelpe til.

1.4 Oppbyggning

Denne oppgaven er strukturert på denne måten:

Kapittel 2 Forklarer bakgrunnsinformasjon om garmin og om aterosklerose.

Kapittel 3 Går mer inn i detalj på hva som kan brukes til å løse oppgaven og hvordan dette blir brukt.

Kapittel 4 Ser på hvordan programvaren har blitt til og hvordan den kan brukes.

Kapittel 5 Her vises hvordan programvaren fungerer og dens resultat.

Kapittel 6 Målene blir gjennomgått og oppgaven blir konkludert.

2 Bakgrunn

Dette kapittelet sier noe om firmaet Garmin og hva de har utgitt, samt hvordan de ulike utgivelsene deres brukes. Det blir også snakket om hva aterosklerose er og hvorfor det er så skummelt.

2.1 Om Garmin

Garmin er et av de fremste selskapene innenfor GPS teknologi. De har spesialisert seg innenfor flere felt under dette temaet, blant annet GPS-navigasjon for bil, båt og fly. GPS-enhet i bil er det som kan ansees som deres mest kjente produkt fram til nyere tid.

Figur 1: GPS. [1]

Med utviklingen innenfor hva en mobiltelefon og lignende små enheter kan oppnå, har en dedikert GPS-enhet i bil blitt mindre og mindre populært. Med dette har Garmin tydelig skiftet fokuset over på mer bærebare enheter. Hovedproduktet innen denne kategorien er smartklokker. Med sitt opphav i analyse av hjertefrekvens til utøvere, mest kjent som pulsklokke, har smartklokker den dag i dag en god del flere egenskaper enn forgjengeren.[7]

[1] Humberto Möckel, CC BY-SA 3.0
https://upload.wikimedia.org/wikipedia/commons/f/f9/Garmin_N%C3%BCvi_200.jpg

Figur 2: Garmin Forerunner 935.

En av hensiktene med smartklokker er å gjøre treningshverdagen så enkel som mulig, gjennom å utvikle enheter med presise mål, effektiv strømstyring og uavhengighet. Med uavhengighet legges det vekt på at produktet ikke skal oppfattes som manglende uten en smart-telefon integrering, men som et fullverdig produkt uten. Selv om uavhengighet er et viktig aspekt, er også integrasjon av smart-telefon et stort fokus. Med dette kan en kommunisere fra klokken og over internett, laste ned applikasjoner og analysere data på andre enheter. For å imøtekomme kravene som forbrukere kan stille til slike produkter, kan Garmin enten ansette flere utviklere som kan lage god programvare til klokkene sine, eller la andre lage programvaren selv. Dette løste de da med å publisere sin egen «Software Development Kit» (SDK). I tillegg til at man kan tilpasse enheten til eget bruk, har de også en butikk besående av utviklede applikasjoner.

2.1.1 SDK

Et «Software Development Kit», eller SDK, er et rammeverk, bestående av ulike utviklerverktøy, som tillater alle å kunne utvikle applikasjoner for en gitt enhet. Verktøyene kan omfange alt fra kodeeksempler, biblioteker, dokumentasjon, APIer og programvare. De er ment som byggeblokker for å hjelpe utviklingen av for eksempel applikasjoner til mobiler med visse standardiserte grensesnitt. En god SDK vil inneholde de nødvendige verktøyene for å utvikle applikasjoner for en spesifikk enhet og dens økosystem.[16]

Med å lage dette tillater utviklerne alle andre personer på ulike steder på kloden til å tilpasse

programvaren til sitt eget bruk. Der et firma må prioritere det de anser som det mest viktige, kan en fremmed utvikler lage det en selv har lyst til. En annen måte å se det på er at man lar folket velge selv hvordan ens programvare skal brukes, gjennom å gjøre endringer til programvaren. For å putte dette i perspektiv, kunne et fysisk eksempel på nytten av et slikt rammeverk, være;

> Når et forlag gir ut en bok, vil ikke boken være mulig å endre, så fort den er laget. La oss si en rekke mennesker gjerne skulle endret hvordan boken representeres, ved å gjøre skriftstørrelsen større, eller endre rekkefølge på kapitlene. Endringene til boken er ikke mulig å gjennomføre uten å gi ut nye. Det er her en SDK kommer inn i bildet. Det som forlaget kunne valgt å gjøre for å endre dette, ville vært å lage et rammeverk rundt hvordan boken ble laget, og gi folket tilgang på dette. Da kan man selv gå inn å gjøre endringer til eget bruk, uten å behøve og produsere nye enheter.

Ved at selskapet utvikler et rammeverk, som tillater andre utviklere til å bidra med å tilpasse produktet til andre brukere, kan dette øke verdien til smartklokkene og til selskapet.

2.1.2 Garmins SDK, Connect IQ

Garmin tilbyr en SDK, som de har valgt å kalle Connect IQ. Enhetene som Garmin tillater en å utvikle applikasjoner til, er hovedsakelig klokker, men også noen type GPSer. Connect IQ inneholder en del forskjellige verktøy for å hjelpe utviklere:

- Monkey C, Garmins egenutviklet programmeringsspråk.
- Integrasjon med populære IDEer som Eclipse og Visual Studio Code.
- Dokumentasjon, nybegynnerveiledning og kodeeksempler.
- Connect IQ Device Simulator, en simulator der man kan simulere egenlagde applikasjoner på alle av Garmins klokker. Simulatoren inneholder en rekke innstillinger for å endre på klokken og teste egen kode, uten å måtte eie hver klokke man vil utvikle på. Det er også mulig å kjøre aktivitetsfiler, av ".fit"-typen, for å for eksempel teste ut applikasjonen på en spesifikk treningstur.

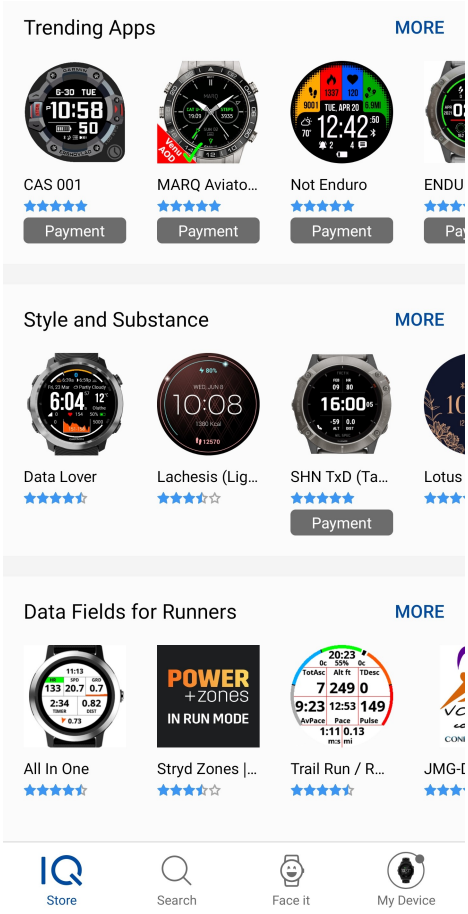
Figur 3: Connect IQ Device Simulator.

- Toybox, Garmins API som tillater en å lett kommunisere med klokken. Her er det mulig å hente, endre og legge på data, kommunisere over Bluetooth og bruke diverse biblioteker på klokken.

Figur 4: Toybox.

Når det skal utvikles en applikasjon i Connect IQ, kan man velge mellom 5 ulike typer; "Watch

Faces", "Data Fields", "Device Apps", "Widgets". Disse lar en påvirke blant annet koblingen mellom telefon og klokke, hvordan hovedskjermen ser ut og hvordan helsedata prosesseres under trening. Programvaren man har lagd kan også lastes opp og deles med resten av Garmin brukerne gjennom "Connect IQ Store". Connect IQ Store er tilgjengelig på Android og iOS mobiltelefoner, og fungerer som en applikasjonsbutikk. Man registrerer Garminklokken sin og en kan da finne et bredt utvalg av kompatible applikasjoner. For å legge applikasjonen på klokken er det bare å laste den ned på Connect IQ Store på mobilen, så synkroniseres klokken ved neste mulighet, og applikasjonen er lagt til.



Figur 5: Connect IQ Store.

I denne oppgaven utvikles det en applikasjon av typen "Data Fields" for Garmin sine klokker, Garmin Forerunner 935 og Garmin Fenix 6. Det kan være utfordrende å bli kjent med et nytt rammeverk, derfor benytter Connect IQ seg av utviklerspråket Monkey C, til å forenkle prosessen.

2.1.3 Applikasjons typer

2.1.3.1 Device Apps

"Device Apps" er applikasjoner som kan kjøres på Garmin klokken. Det er den type applikasjon med meste fleksibilitet og tilgang til all data og informasjon på klokken. En kan sammenligne den med en applikasjon på mobilen i forhold til egenskapene. Applikasjonene kan kommunisere med en mobiltelefon over Bluetooth, og videre til internett via telefonen. De har også tilgang til for eksempel ANT sensoren og akselerometeret. Klokken kan kjøre én applikasjon om gangen, da den kan registrere data på en ".fit" fil som en aktivitet. Filen kan senere undersøkes og brukes til å analysere aktiviteten og lastes opp på Garmin sin egen Connect side, der man får en god oversikt over informasjon på filen. Applikasjoner kan gjøre alt fra å registrere en løpetur, til Maps og til grunnleggende spill.

Figur 6: Maps.

På grunn av all tilgangen til klokken "Device Apps" har, krever det mer batteriforbruk og minne ved å kjøres, noe som må tas hensyn til ved utviklingen av slike applikasjoner. De har heller ikke muligheten til å kjøres i bakgrunnen, noe som begrenser bruken den om en vil for eksempel kjøre en annen applikasjon i tillegg.

2.1.3.2 Watch Faces

En "Watch Face" er urskiven på klokken. Man kan her endre på og tilpasse utseende til å vise det en mengde av informasjon på klokken. Utseende kan variere alt fra en simpel analog eller digital klokke til et vidt spekter av informasjon, som f.eks. puls, klokkeslett, kalorier brent, høydemeter, batteristatus eller kompass.

Figur 7: Watch Face.

"Watch Faces" kjøres kontinuerlig på klokken, og er av den grunn en stor kilde til batteriforbruk. På grunn av dette er "Watch Faces" den applikasjonstypen med mest restriksjoner. Det er blant annet en modus som heter "Sleep Mode", som applikasjonen automatisk havner i etter en tidsperiode hvor den ikke er brukt. I "Sleep Mode" er applikasjonen begrenset til å kun kunne oppdateres en gang i minuttet, og kan ikke bruke tidtakere eller animasjoner. Applikasjonen går ut av denne modusen når brukeren løfter klokken for å se på den. Mulighetene for hvordan utseende ser ut er mange, og kan i tillegg endres av slutt brukeren på Connect IQ Store. [10]

2.1.3.3 Data Fields

Å utvikle en "Data Fields" applikasjon tillater en å endre, eller legge til ønskede mål under treningen. De er tilgjengelige når en "Device App" kjøres, og kan bli sett på som tillegg til applikasjoner. De egner seg til å gjøre grunnleggende kalkulasjoner på data som allerede blir registrert via "Device App"-en, men kan også brukes til mer kompliserte bruksområder. Siden "Data Fields" er mer et

tillegg enn en egen applikasjon, er det restriksjoner på bruk av for eksempel kommunikasjon over bluetooth og via mobil, gps- og sensor-manipulasjon.

Et eksempel på en endring via "Data Field" kan være;

> I stedet for å vise frem antall kalorier brent, kan man velge å vise frem hvor mange øl man har gjort seg fortjent til.

Et eksempel på å legge til mål kan være;

> Vise frem gjennomsnittlig hjerterate de siste 20 sekundene. Det kan også gjøres endringer på hvordan dette presenteres på klokken.

Figur 8: Beer Counter.

2.1.3.4 Widgets

"Widgets" er en lettvektig versjon av "Device Apps", og er tilgjengelig gjennom å bla oppover eller nedover fra klokkeuret. De er ofte brukt til å kommunisere via telefonen for å trekke data fra internett eller applikasjoner på telefonen som værdata eller kalender. "Widgets" er ment for mer kortvarig interaksjon og har derfor noen restriksjoner. Denne applikasjonstypen settes på pause etter en tidsperiode og kan ikke registrere data fra aktiviteter. Den kan også være interaktiv og for eksempel brukes til å registrere hvor mye vann man drikker i løpet av dagen. Eksempler på "Widgets" er et lett tilgjengelig kompass, eller en grunnleggende kalkulator. [8]

Figur 9: Weather Widget.

2.1.4 Monkey C

Monkey C er et objekt-orientert språk, utviklet av Garmin, som har i oppgave å gjøre utviklingen av en applikasjon så enkel som mulig. På grunn av en smartklokkes naturlige liten størrelse, er minnebruk et stort fokus. På en klokke med 16 kb minne for bruk av en applikasjon trengs det nøye betraktning for å balansere minnebruken. Denne bekymringen er en av grunnene til at Garmin utviklet Monkey C, et skreddersydd språk for sine applikasjoner, slik at de kan kjøres minneeffektivt og frigjør utviklere fra den bekymringen. For å ikke påvirke tilgjengeligheten og brukervennligheten, så har Monkey C mange likheter med andre kjente språk, noe som gjør det enkelt å lære. [12]

```java
public int addNumbers(int a, int b) {
    int result = a + b;
    return result;
  }
```

Java-kode eksempel.

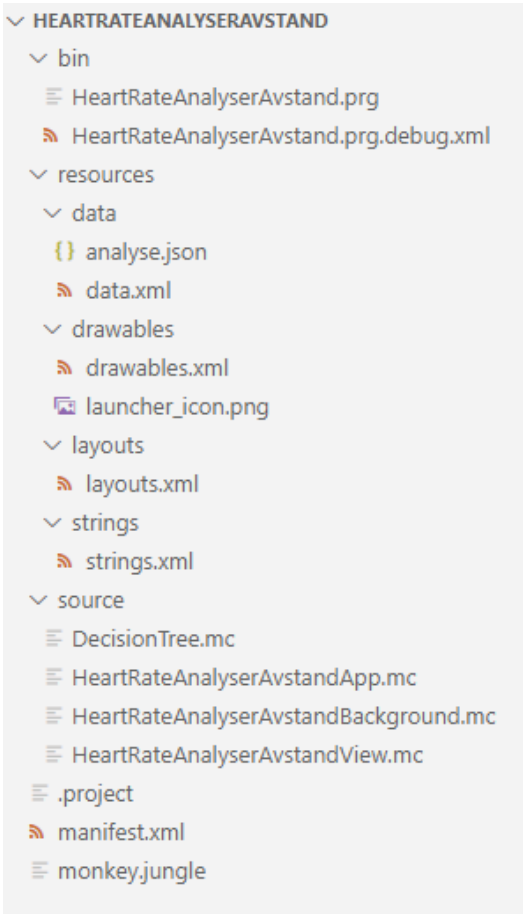
```python
def add_numbers(a, b):
    result = a + b
    return result
```

Python-kode eksempel.

```
function addNumbers(a, b) {
    var result = a + b;
    return result
}
```

Monkey C-kode eksempel.

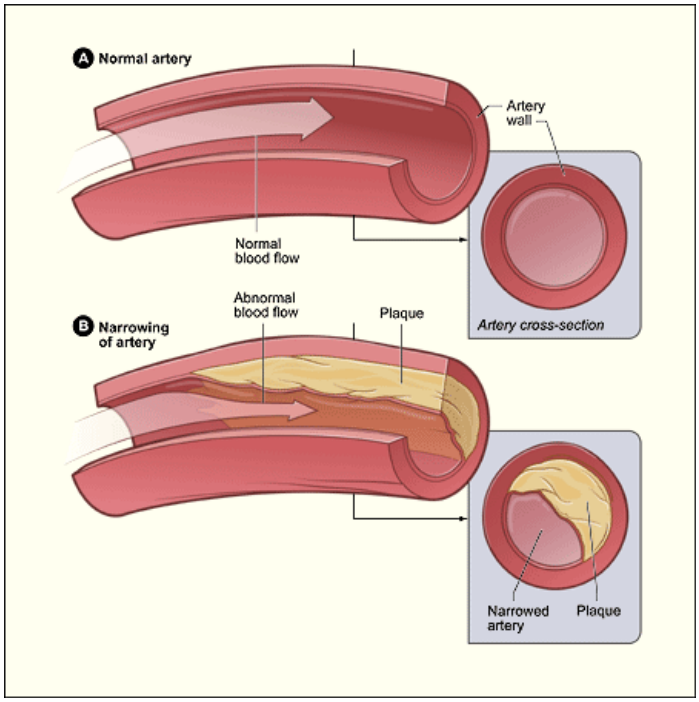
I samarbeid med rammeverket, kan utvikleren velge selv hvilken eller hvilke enheter som det skal utvikles til, og man blir gitt hjelpelinjer og skjelettkode slik at man enkelt finner veien. Det er også en standard mappestruktur som gjelder for alle klokker.



Figur 10: Mappestruktur.

2.2 Om Aterosklerose

Aterosklerose, også kjent som åreforkalkning, er en tilstand hvor fettpartikler (lipider) og betennelsesceller, som består av hvite blodceller og glatte muskelceller, reduserer blodårenes evne til å frakte blod rundt i kroppen. Dette er en underliggende tilstand som kan føre til hjerte- og karsykdommer, hjerteinfarkt og hjerneslag.



Figur 11: Forskjellen mellom en frisk arterie og en der det samler seg opp plakk.[2]

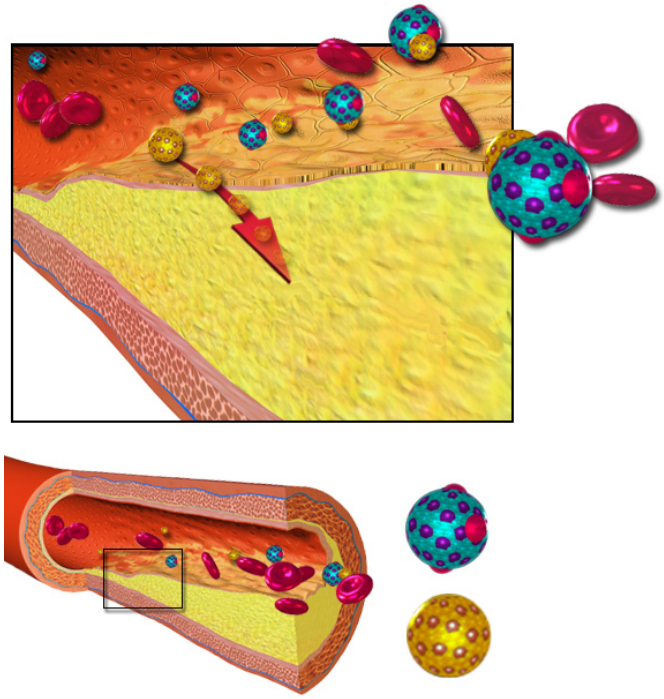
For å unngå at dette kan skje er det viktig med en rask, jevn, laminær blodgjennomstrømning. Når blodet flytter jevnt på seg, bidrar det til å holde åreveggene og det innerste cellelaget i åren, altså endotelet, glatt, elastisk og ikke minst friskt. Der årene deler seg oppstår det lettere turbulens i blodgjennomstrømningen, som igjen gjør det lettere at det oppstår aterosklerose og man kan da se de første tegnene på dette der. For å oppsummere kan man da finne tidlige tegn på aterosklerose der blodet er turbulent og ikke har en jevn hastighet.

I det tidlige stadiet av aterosklerose kan man se små lysegule fettstreker i endotelet. Disse oppstår når endotelet blir skadet og det samler seg opp betennelsesceller og fettpartikler under det innerste

[2]NHLBI, Public domain
https://upload.wikimedia.org/wikipedia/commons/b/b3/Atherosclerosis_diagram.png

laget i blodåren, intima. For å unngå at dette skjer er det viktig å ikke ha for mange fettpartikler eller fremmede, skadelige stoffer i blodet, aktiverte blodplater, aktive betennelsesceller og som nevnt, turbulent blodgjennomstrømning. På dette tidlige stadiet er det heldigvis reversibelt og disse strekene kan gå bort.

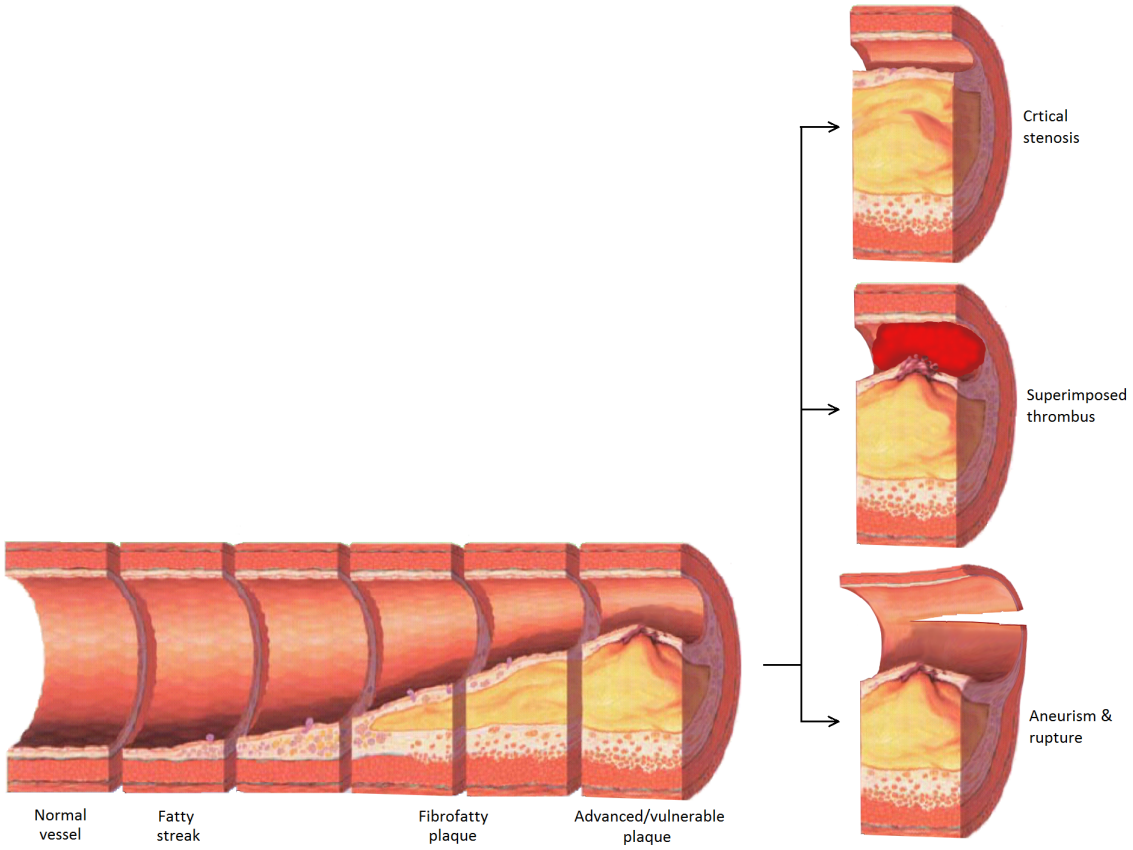
Fettpartiklene dette kan bestå av er kolesterol, triglyserider, fosfolipider og kolesterolestere. Når det kommer til fettpartikler i blodet, er det som oftest for mye kolesterol som er en av syndebukkene. Disse blir da fraktet rundt i blodet sammen med andre fettpartikler i lipoproteiner. Disse lipoproteinene vil da samle seg sammen inn i arterieveggen hvis endotelet ikke fungerer optimalt eller hvis det er alt for mange i blodet. Når dette skjer, vil hvite blodceller gjøres om til makrofager, så til skumceller når makrofagene tar til seg partikler av lipoproteinene. Dette gjør så at skumcellene tar til seg mye kolesterol og når cellene dør vil de etterlate seg rester inne i intima, disse avleiringene kalles for det aterosklerotiske plakket. Når dette skjer blir de glatte muskelcellene påvirket og det vil bli laget en fibrøse kappe som er bindevev som dekker innsamlingen av fettpartikler og betennelsesceller. I plakket foregår det fettdeponering. Plakket vil påvirkes av fritt kolesterol, som har en korrelasjon med at det kan være betennelse i et plakk.



Figur 12: Lipoproteiner som samler seg sammen. [3]

[3]BruceBlaus CC BY 3.0 30.sep 2013

Dette plakket endrer seg stadig, men endringen skjer veldig sakte over lang tid. For at det skal samles opp mer plakk må enten forholdene i blodåren forverres eller holde seg på det nivået det var på da oppsamlingen begynte. Så hvis forholdene ikke endrer seg til det bedre, vil det bli mer og mer plakk som til slutt kan sperre for blodet i åren. Det er også fullt mulig å ha flere steder med plakkdannelse i samme arterie. En annen ting er at glatte muskelceller fra media, som er det mellomste laget i arterien, etter hvert vil komme inn i både plakket og i kappen. Noe annet som kan påvirke aterosklerosen i stor grad under hele prosessen, er aktiverte blodplater. Når forholdene i blodåren eller endotelet ikke er optimalt, kan blodplatene bli akkurat det, aktiverte. Da vil de bli med å stimulere betennelsescellene med at de er ekstra klebrige og kan da feste seg til betennelsescellene.



Figur 13: De ulike stadiene av aterosklerose og hva det kan ende med.[4]

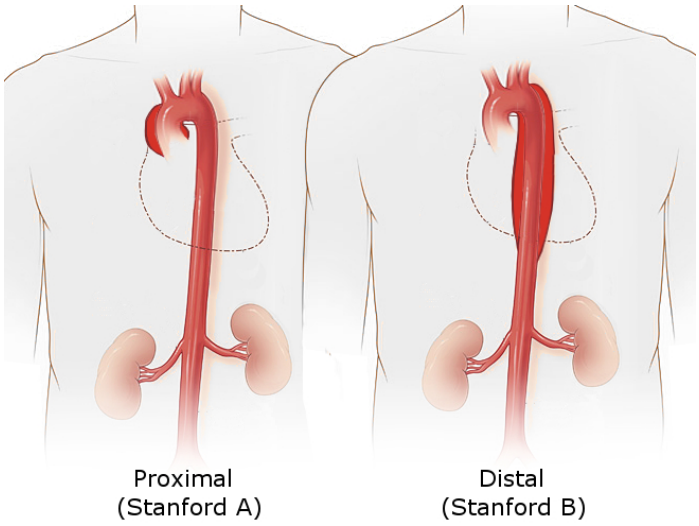
Plakket kan bli stabilt når kappen rundt er tykk og er dekket av endotelet. Da er plakket helt

https://en.wikipedia.org/wiki/Atherosclerosis#/media/File:Blausen_0227_Cholesterol.png

[4]Nicholas Patchett 27. mars 2015 CC BY-SA 4.0

https://en.wikipedia.org/wiki/Atherosclerosis#/media/File:Late_complication_of_atherosclerosis.PNG

skjermet fra blodet. Hvis kappen er tynn, er plakket sårbart. Dette skjer når det er betydelig mer betennelsesceller og fett i forhold til kappen. Da kan det oppstå skader og slitasje på endotelet. Når dette skjer, er ikke lenger innholdet i plakken like godt beskyttet. Som en konsekvens av dette kan kappen få en rift i seg, da kan innholdet i plakket komme ut i åren og lage en blodpropp, som igjen kan tette åren helt. En annen ting som kan skje er at plakket kan fylles med blod, så kommer deler av plakket ut i blodstrømmen sammen med blodplater og kan senere tette igjen mindre arterier. Noe annet som kan skje er at selve veggen i blodåren bli svekket såpass mye at det blir dannet lommer eller poser med blod langs selve blodåren (disseksjon og aneurismer).



Figur 14: Disseksjon på aortaen.[5]

Det kan være ganske vanskelig å oppdage at man har aterosklerose. Det er fordi at blodårene må få en ganske så stor kul av plakk før man kan merke symptomer på dårligere blodtilførsel. De mest rammede stedene av aterosklerose er kransarteriene til hjertet, arteriene som går mot hjernen, arteriene fra lysken mot kneet og bukens hovedpulsåre. [36]

Noen av symptomene på aterosklerose kan være hjerneslag, hjertekrampe, hjerteinfarkt og koldbrann [2]. En annen kan være smerter og tretthet i benmuskulaturen som dukker opp under bevegelse og forsvinner når man slapper av. Denne diagnosen heter claudicatio intermittens, og kan bli kalt for røykeben. Dette viser en direkte sammenheng med trening, da man vil merke dette ekstra godt når belastningen øker [31].

Hvis man er sent i utviklingen av aterosklerose kan dette behandles med utblokking av den gjeldende

[5]Nicholas Patchett 27. mars 2015 CC BY-SA 4.0
https://upload.wikimedia.org/wikipedia/commons/2/2b/Aortic_dissection_types.jpg

arterien. Etter at dette er gjort settes det som oftest inn noe som forsterker blodåren slik at den ikke skal implodere etter behandlingen [21].

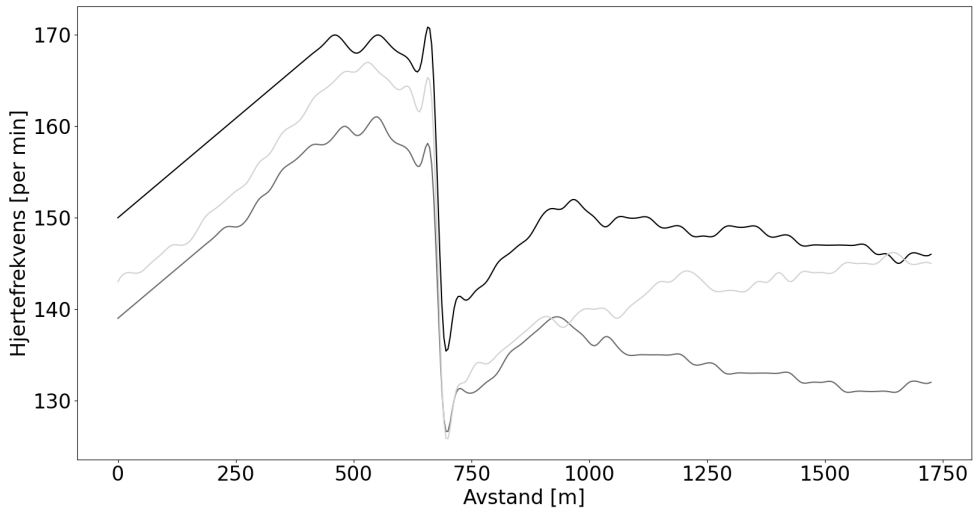
Siden symptomene på aterosklerose kan være veldig vanskelig å oppdage, og når de oppdages kan det være alvorlige ting som deriblant blodpropp, er det viktig å være føre var når det kommer til dette. De som er mest i risiko er de som røyker, har diabetes, er overvektig, har forhøyet kolesterolnivå eller har høyt blodtrykk. Det er også mulig at gener spiller en rolle, så hvis noen i nær familie har høyt kolesterol eller blodtrykk, kan det være lurt å besøke en lege som kan sjekke deg. Så det beste som kan gjøres for å forhindre at aterosklerose utvikler seg er å ha et sunt kosthold og mosjonere tilstrekkelig [21].

3 Dataanalyse

I dette kapittelet blir det sett på hva som er bakgrunnen til hvordan dette prosjektet forsøker å finne ut om noen har aterosklerose eller ikke. Noe annet er hvordan dette skal gjøres og med hvilke modeller. Det blir også sett på noen andre modeller som muligens kunne blitt brukt, men blir ikke det av ulike grunner. Til slutt blir noen andre applikasjoner som eksisterer fra før sammenlignet med den i dette prosjektet.

3.1 Tidligere forskning

Ettersom det kan være vanskelig å oppdage aterosklerose hos noen, er det blitt forsket på tilstanden relatert til trening. Det er blitt utført et sykkelritt, hvor utøvere med og uten aterosklerose har vært delaktig i rittet. Rittet bestod av ulike segmenter som var av interesse for forskningen i forhold til å studere hjerteraten sin utvikling. Disse ble loggført og sett nærmere på. Utøverene som deltok er av alle aldere, kjønn og fysisk tilstand, og kan plasseres under tre grupper. Dette er vist på figur 15. Dataene som skiller gruppene ble dannet ved å se på gjennomsnittet av alle målingene for hver av utøverene innenfor hver gruppe. Det vil si at det ikke bare er en person man ser utviklingen av hjerteraten til nedenfor, men gjennomsnittet av flere. Av den grunn vil det forekomme variasjon. [33]



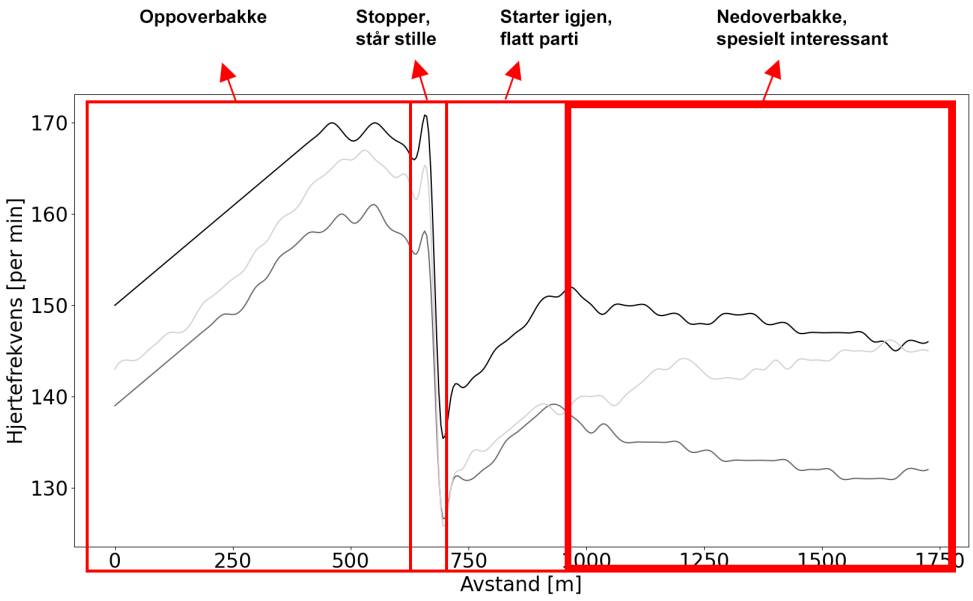
Figur 15: Tinghaug ritt.

Den første gruppen er de utøverene som ikke har påvist åreforkalkning. Gruppen de faller under representeres av fargen svart. Ved senere anledning vil gruppen representeres ved den samme svarte fargen på forskjellige grafer, og betegnes som gruppen: Normal.

Den andre gruppen er de utøverene som har påvist åreforkalkning. Denne gruppen representeres på grafen ovenfor gjennom den mørkegrå fargen. Til forskjell fra denne gruppen og den siste gruppen, som begge har påvist åreforkalkning, har disse utøverene en gjennomsnittlig lavere eller normal innsats anledningsvis. Senere vil gruppen representeres ved mørkegrå farge og betegnes som: Safe.

Den siste gruppen er utøvere som også har påvist åreforkalkning, i likhet med gruppen Safe. Denne gruppen skiller seg ut ved å ha en høyere innsats under deler av rittet. Gruppen er representert ovenfor ved den lysegrå fargen, som er det samme den vil bli representert ved senere. Denne gruppen betegnes som: Press.

I all hovedsak er det hjerteraten sin oppførsel som er i fokus, og det er nettopp fordi åreforkalkning gjør det vanskeligere for kroppen å føre blodet rundt. Ved å se på utviklingen til hjerteraten ved ulike segmenter, kan man potensielt skille mellom gruppene, som er formålet med applikasjonen. Med segmenter, menes forskjellige perioder med ulike forhold. Forholdene som varierer kan være å starte og stoppe eller å sykle i oppover og nedover bakke. Nedenfor, på figur 16, er de ulike segmentene for den litt under 2km lange distansen markert. Perioden av spesiell interesse er markert nøyere, ettersom det er denne perioden som viser et tydeligere skille blant gruppene.



Figur 16: Tinghaug ritt med markerte perioder.

Som vist ovenfor var det i utgangspunktet ikke så altfor stor forskjell i utviklingen av hjerteraten i de aller fleste segmentene. Det er et lite sprang mellom de i forhold til hva den faktiske hjerteraten var, men det kunne skyldes en rekke ulike årsaker. Det kunne naturligvis skyldes fysisk form, men også andre variabler som alder, vekt og høyde. Av den grunn var det ikke så altfor mye av interesse i de segmentene.

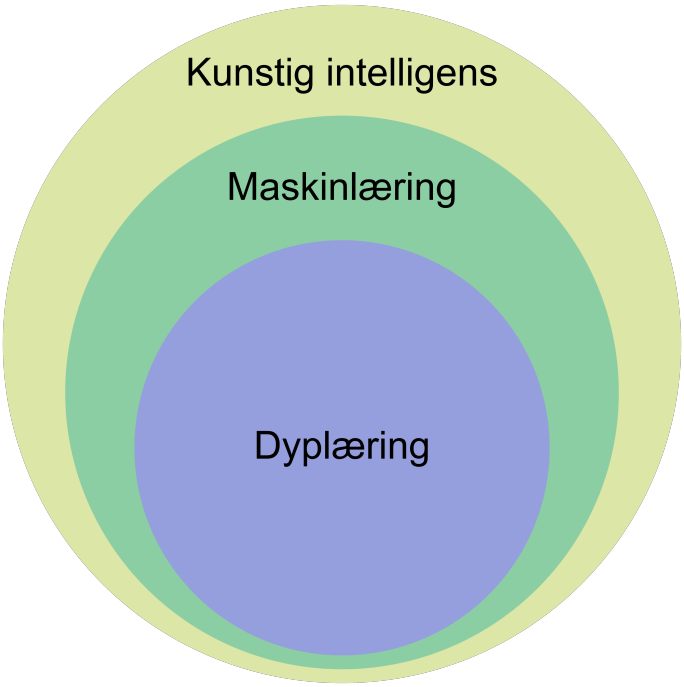
Perioden som er uthevet ovenfor var en periode på rundt 800m hvor det ble et skille i utviklingen av hjerteraten blant gruppene. Det som foregikk før denne perioden var et lite stopp på toppen av en slak bakke, før utøverene gav seg ut på 800m med slak nedover bakke. Skillet i hjerterate utviklingen gjaldt spesielt blant utøverene som falt under Press kategorien, gitt ved den lysegrå fargen. Hjerteraten deres viste seg å øke over perioden, mens utøverene innenfor de to andre gruppene hadde en hjerterate som sank over perioden.

Videre skal ulike modeller for analyse vurderes og utvikles. Dette skal gjøres basert på det tydeligste skillet blant gruppene, som er det fremhevede perioden på figur 16. I tillegg til å gjøre vurderingen av modeller basert på utviklingen ovenfor, vil det være hensiktsmessig å legge til grunne at det kan være andre segmenter av interesse i fremtiden, samt andre variabler som kan potensielt kan inkluderes.

3.2 Modeller for analyse

Formålet med applikasjonen er å skille mellom de ulike gruppene som er presentert. Dette skal gjøres gjennom å benytte den loggførte dataen relatert til den tidligere forskningen. For å utnytte dataen til å skille mellom gruppene, kan det benyttes maskinlæring.

Maskinlæring er en del av fagfeltet som kalles kunstig intelligens. Kunstig intelligens går ut på å gi datamaskiner evnen til å gjenspeile menneskelig intelligens. Det er et fagområdet som har vært i utvikling allerede så tidlig som på 1950-tallet. Parallelt med utviklingen innenfor teknologi, har også interessen og kunnskapen relatert til kunstig intelligens vokst. Med kunstig intelligens følger det mange spørsmål i forhold til muligheter i fremtiden. Innenfor kunstig intelligens finnes det to hovedområder som danner grunnlag for modeller til analyse og utvikling. Disse områdene er maskinlæring og dyplæring, og er vist på figur 17. [23]



Figur 17: Maskinlæringsdeler.

Maskinlæring består i hovedsak av algoritmer som tar inn data eller målinger og lærer av de. Hensikten med å lære av et datasett er å gjøre maskinen kapabel til å ta inn nye datasett og estimere resultater basert på det maskinen har lært. Datasettet algoritmene lærer av kan kategoriseres som merket eller umerket. Om datasettet er merket betyr det at det allerede er visst hvilken gruppe målingene tilhører. En enkel form for maskinlæring som kan benytte merkede datasett er beslutningstre. Et beslutningstre er som et tre hvor hver gren inneholder informasjon. Ved at datasettet er merket ved treet til enhver tid hvor mange fra hver gruppe som falt under ulike grener. Om det er et umerket datasett, vet man ikke med sikkerhet hvilken gruppe målingene tilhører. En form for maskinlæring som kan benytte dette datasettet kan være Support-Vector Machine eller SVM. SVM sin oppgave kan være å gi mening ut av noe som tilsynelatende kan være et veldig rotete datasett. Dette gjøres gjennom å se dataen fra ulike perspektiver. Mer om dette i kapittel 3.2.3.

Som vist på figur 17, finnes dyplæring på innsiden av maskinlæring. Det er en form for avansert maskinlæring, hvor metodene tas et steg lenger med tanke på kompleksitet. Mens maskinlæring skal kunne gjenspeile menneskelig læring ved å ta data inn og gi et svar tilbake, så skal dyplæring kunne gjenspeile et menneskes evne til å lære. Måten en menneskehjerne går gjennom utallige operasjoner for å gjøre enhver ting til enhver tid, er måten dyplæring går frem med å lære, ved å ha utallige måter å komme frem til et svar. Dette gjøres gjennom det som kalles nevrale nettverk. Som navnet,

dyplæring, tilsier så tar det mye data for at et nevralt nettverk skal operere optimalt, og læringen er minst mulig påvirket av mennesker. Det ses nærmere på dyplæring i kapittel 3.2.5. [35]

Valgene av modeller er mange. Valget kan variere i stor grad kun basert på hvordan datasett man har å jobbe med. Ikke bare det, men andre faktorer som størrelsen på datasettet og kompleksitet kan spille en stor rolle. På toppen av dette skal dette kunne ses i sammenheng med bruk innenfor en applikasjon, som også må tas hensyn. Av den grunn skal disse modellene bli vurdert videre;

Regresjon
Beslutningstre
Support-Vector Machine
ARIMA
Nevrale nettverk

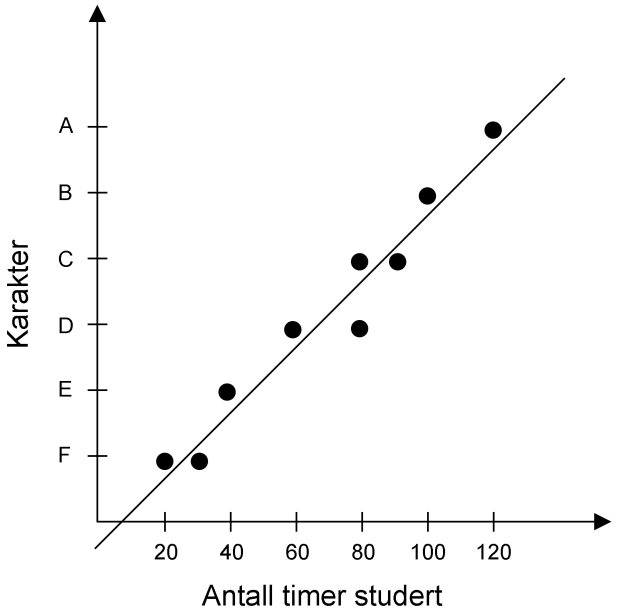
3.2.1 Regresjon

Regresjon er ofte forbundet med statistikk, og det er nettopp der regresjon har sitt opphav. Regresjon er brukt til å forstå forholdet mellom ulike variabler. Videre kan det brukes til å forstå hvilke faktorer som styrker forholdet og eventuelt hva man kan se bort ifra, ettersom det ikke har noe effekt. Av variablene som analyseres, regnes en av de som den avhengige variabelen. Dette er variabelen som prøves å forstås, og som utfallet av analysen vil ha noe å si for. Det kan både være en enkelt eller flere variables som er av relasjonsmessig interesse for den avhengige variabelen. Disse variablene kalles uavhengige variabler. Som oftest kan ikke en avhengig variabel beskrives av kun en enkelt uavhengig variabel. Ofte så er det mange variabler som kan påvirke utfallet. Om det er en enkelt uavhengig variabel eller flere, skiller de to underkategoriene simpel og multipel regresjon. Det finnes mange ulike typer regresjon, avhengig av variabler og metoder som brukes, men videre ses det på skillet mellom en lineær og ikke-lineær form for regresjon [14]

Lineær regresjon er den enkleste formen for regresjon, og går ut på å estimere verdien av et datasett gjennom en rett linje. Grunnen til at det er den enkleste formen for regresjon er fordi datasettet som modellen ønskes å brukes på er gjerne forutsigbart eller kontinuerlig. Et eksempel for å tydeliggjøre hvordan en lineær regresjonsmodell kan se ut kan være forholdet mellom antall timer studert i et fag mot karakteren i faget.

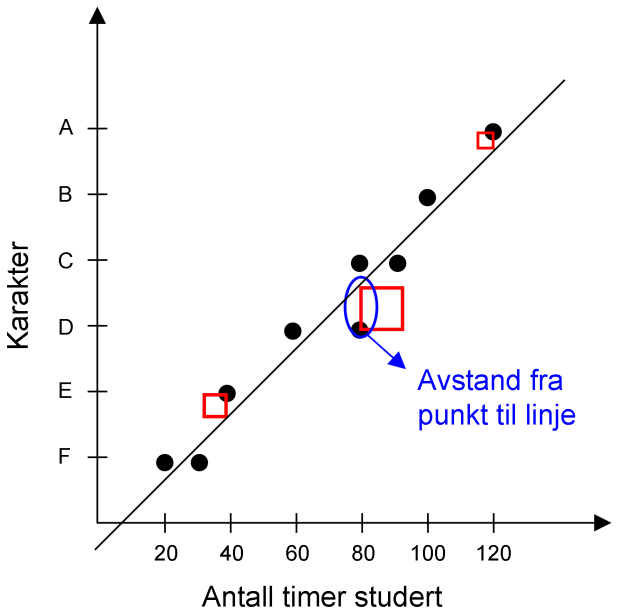
Ved å bruke lineær regresjon til å etablere forholdet mellom antall timer studert i et fag mot karakteren man fikk har man en simpel lineær regresjon. Grunnen til at akkurat de variablene ble valgt til å illustrere er fordi de er nokså forutsigbare, som gjør at en lineær fremgangsmåte er passende. Om

man ønsker å stille spørsmålet; "Hvor mange timer må jeg bruke for å få en A?", kan man se hvilke av variablene er den avhengige og den uavhengige. Den uavhengige variabelen er antall timer brukt, etter som det er årsaken til estimatet. Den avhengige variabelen er karakteren som er resultatet av estimatet. Ved å sette sammen vilkårlig data for å illustrere poenget, kan målingene se ut som på figur 18.



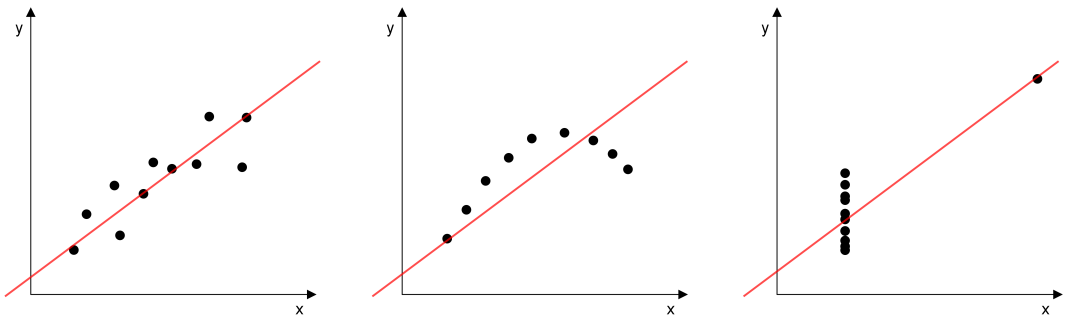
Figur 18: Eksempel på en forutsigbar graf.

Ovenfor kan man tydelig se at forutsigbarhet, eller en viss relasjon mellom variablene. Det lineære regresjonsmodeller nå ønsker å gjøre er å generere en rett linje for å best mulig estimere den avhengige variabelen basert på de uavhengige variablenes input. Metoden som lineær regresjon bruker for å generere linjen kalles minste kvadraters metode. Metoden går ut på å se på avstanden fra hvert punkt til en linje, lage kvadrater ut av avstanden og summere kvadratene. Linjen som genererer den laveste summen gir et estimat hvor det er minst avstand fra punktene til linjen, sett i sin helhet. Dette er illustrert på figur 19.



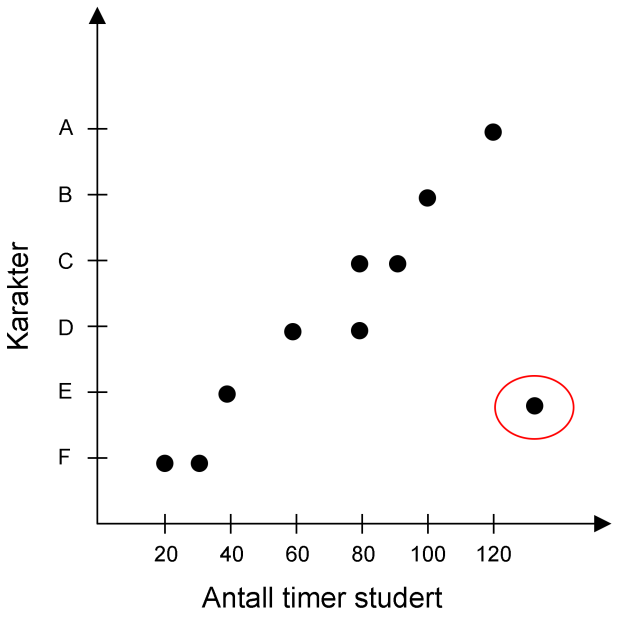
Figur 19: Minste kvadraters metode.

Datasettet har som regel veldig mye å si for å kunne fastslå om regresjonen gir et godt estimat eller ikke. Dette faller i stor grad ned på forutsigbarheten. Det at datasettet må være forutsigbart kan også bety at det må være relativt konsistent med tanke på målingene. Det trengs ikke mer enn litt uforutsigbarhet før tolkningen av den regressive linjen kan være veldig misledende. På figur 20, ser man et eksempel på hvordan den samme linjen kan tolkes på fire vidt forskjellige måter, avhengig av målingene. Det er også verdt å nevne at størrelsen på datasettet kan ha stor betydning for det nevnte tolkningsproblemet. Med mindre datasett vil uforutsigbar data ha større påvirkning.



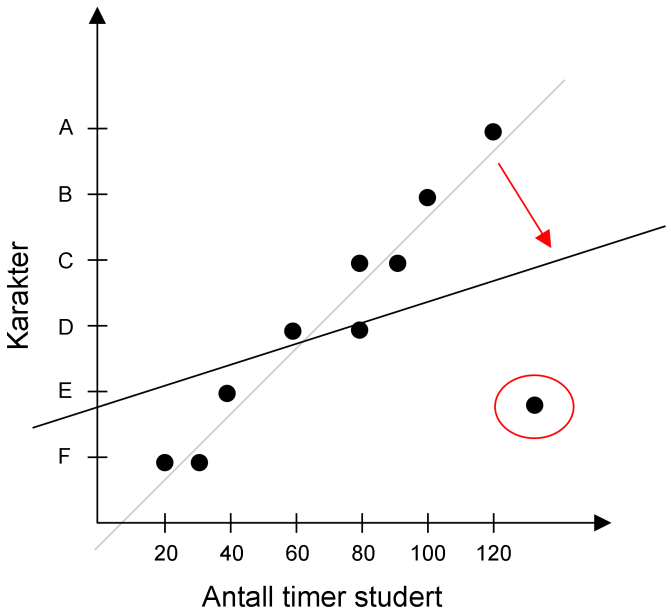
Figur 20: Ulike tolkningsproblemer basert på estimat.

På bakgrunn av tolkningsproblemet, finnes det måter å sørge for at datasettet representeres på best mulig måte. Det kan anses som metoder for å forberede dataen. Den viktigste forberedelsen er naturligvis å være nøye på om datasettet kan regnes som forutsigbart eller ikke. Evalueringen kan tilsynelatende lene mot et forutsigbart datasett, men noen uteliggende målinger påvirker modellen i stor grad. Disse type uteliggende målinger kalles for "noise" eller støy, og er vist i relasjon med det tidligere eksemplet på figur 21. [4]



Figur 21: Uteliggende måling.

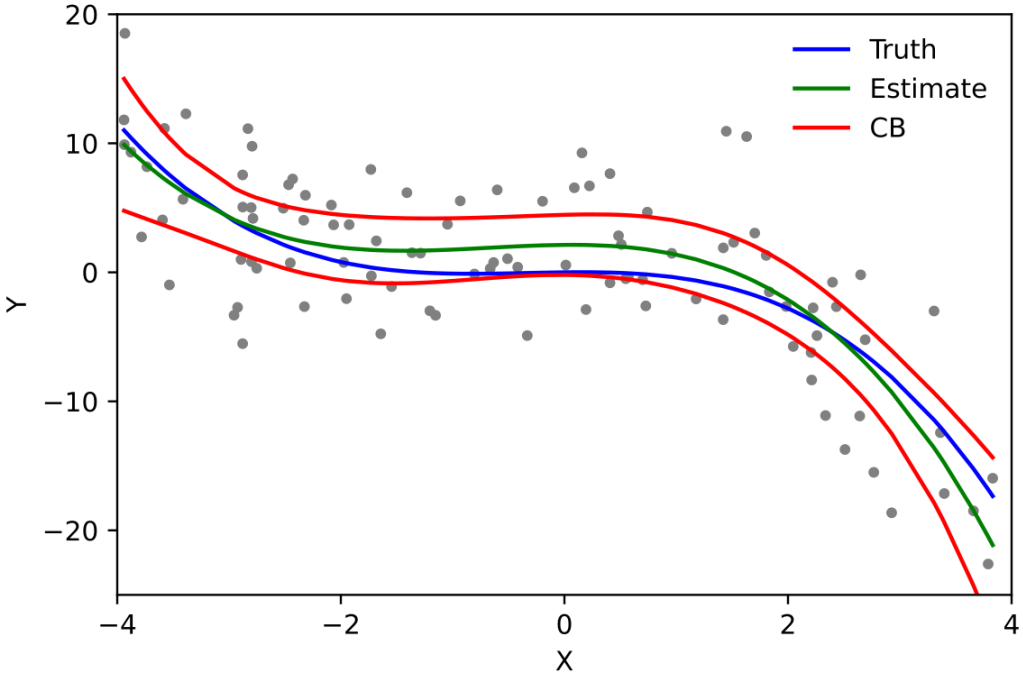
Om de uteliggende målingene viser seg å ødelegge, selvom målingene i stor grad peker en vei, kan det være en god ide å se bortifra slike målinger, for å gi modellen den beste representasjonen basert på forutsigbarheten. Nedenfor, på figur 22, kan man se den enorme forskjellen i estimert regresjon, selvom dette er et vilkårlig, lite datasett generert for å illustrere yttertilfeller.



Figur 22: Nytt estimat med uteliggende måling.

Avslutningsvis kan den lineære modellen utvikles videre ved å se på flere forhold. De aller fleste relasjoner som dette har flere variabler som påvirker de. Når det kommer til eksempelet ovenfor, kan det tenke seg å inkludere en rekke andre variabler som kan påvirke karakteren. Det kan være store generelle forhold som søvn, kosthold eller sosialisering, eller kanskje man ønsker å se på relasjonen mellom studietid, karakter og koffeininntak.

Ikke-lineær regresjon eller polynomial regresjon er typisk en videreføring av en lineær regresjon. Det kan være tilfeller hvor det er tydelig sammenheng mellom variabler, gjennom å se på mønsteret, men mønsteret lar seg ikke godt nok beskrives gjennom en rett linje. I disse tilfellene tyr man til en ikke-lineær fremgangsmåte, og resultatet av det kan se ut som på figur 23. Ved ikke-lineære problemer ser man på hvor betydelig hver variabel er for utfallet gå høyere eller lavere ordre for de ulike leddene. Dette øker kompleksiteten, og det er viktig å balansere hvor høyt og lavt man går. Ettersom datasettet til oppgaven passer godt til en lineær modell, vil ikke denne fremgangsmåten bli sett nærmere på. [22]



Figur 23: Eksempel på polynomial regresjon. [6]

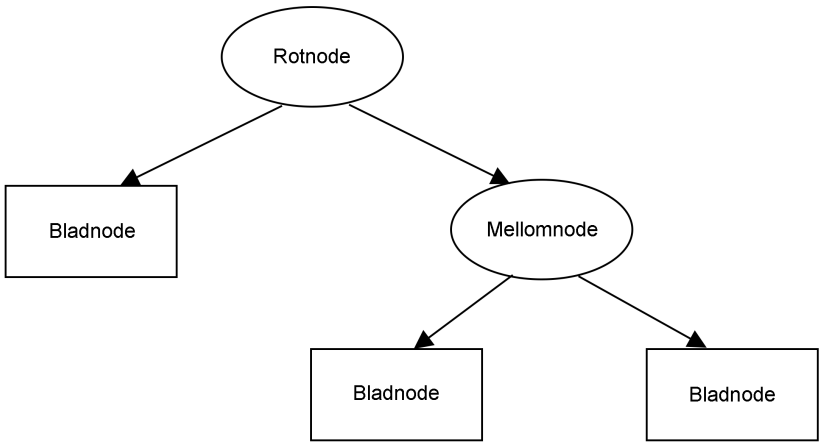
Relatert til applikasjonen er fordelen med lineær regresjon at det er en av de enklere modellene. Selv om datasettet som analyseres er ganske lite, lar det seg enkelt gjøre å estimere hjerteraten over tid, spesielt i spesifikke perioder, ettersom datasettet som jobbes med er relativt forutsigbart. I tillegg lar det seg gjøre å estimere hjerteraten til enhver tid, gjennom linjens funksjon, som kan være gunstig i forhold til applikasjons-bruk og analyse. Ulempen med å bruke modellen er at hjerterate kan i stor grad variere, og uteliggende data kan enkelt påvirke estimatet. Dette kan bli spesielt tydelig ettersom datasettet som jobbes med er i det minste laget.

På bakgrunn av at det er en enkel fremgangsmåte for å estimere utviklingen av hjerteraten, samt at det kreves minimalt med minne, ble denne modellen tatt i bruk.

[6] Skbkekas, CC BY 3.0 9. april 2009
https://en.wikipedia.org/wiki/Linear_regression#/media/File:Polyreg_scheffe.svg

3.2.2 Beslutningstre

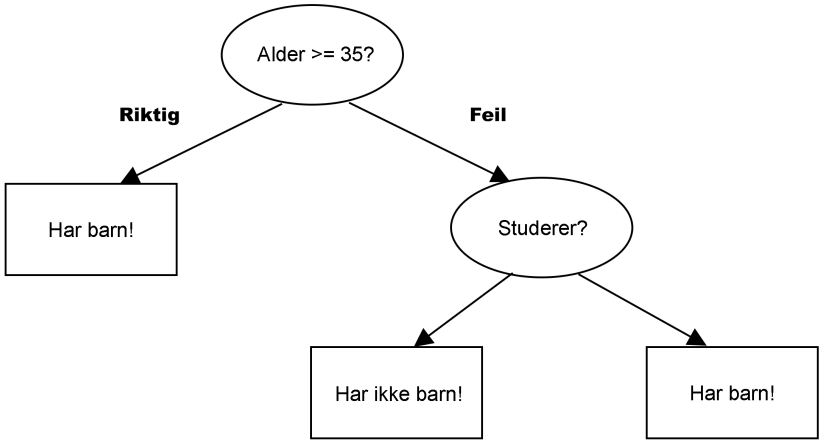
Et beslutningstre er et maskinlæringsalgoritme som består av en rekke beslutninger bygd opp i formen av et tre. Mer konkret så består treet av noder og grener, hvor grenene representerer veien videre i treet basert på beslutningen. Hvordan treet ser ut og hva nodene inneholder er avhengig av datasettet modellen lærer fra. Modellen er godt egnet til klassifiseringsproblemer, på bakgrunn av dens evne til å lære, og er langt mer leselig enn de aller fleste algoritmer. Treet består av tre typer noder, som er rotnoden, bladnoden og mellomnoder. Rotnoden definerer toppen av treet, og er der hvor traverseringen av treet begynner. Bladnodene befinner seg i enden av treet, uavhengig av hvilken vei man traverserer gjennom det. En typisk trestruktur med definerte noder ser ut som på figur 24.



Figur 24: Beslutningstre struktur.

Rotnoden og mellomnodene består av en sammensetning som gjør opp beslutningen som tas for å gå videre i treet. Sammensetningen består av en variabel av interesse og en terskel for hvordan man faller innenfor variabelen. Sammensetningen kan f.eks være alder og terskelen kan være 18. Dette bestemmes av treet da det bygges, ettersom det lærer av datasettet den får. Ved bladnodene får man resultatet. Resultatet er da treet sitt estimat for det datasettet man gir den basert på datasettet den lærte av. I lag med resultatet finner man også vektene som hører til bladnodene. Disse forteller en om hvor mange fra hver gruppe eller klasse som havnet ved bladnoden. Ved å se på treet fra tidligere, kunne det sett ut som på figur 25, med den veldig enkle vilkårlige dataen som

finnes på figur 26 som eksempel. I det vilkårlige eksempelet, er klassifiseringene; foreldre eller ikke, altså har/har ikke barn. [1]



Figur 25: Beslutningstre eksempel.

Typisk ønsker man at det er entydig, ved at det kun representanter fra en gruppe, men det er ikke alltid tilfelle. Det vil oftest være tilfeller hvor det er flere fra ulike grupper, ettersom det er store individuelle forskjeller blant mennesker. Disse forskjellene måles da treet bygges ved hjelp av algoritmer for å måle hvor rent datasettet er.

Når det kommer til prosessen med å bygge treet, består den av å svare på; "Hvilke variabler har størst påvirkning?". Dette må svares på, ettersom de skal plasseres først eller øverst i treet. Som nevnt finnes det algoritmer for å måle hvor rent et datasett er. Med hvor rent datasettet er, menes hvor forutsigbart det er. En annen måte det kan ses på er hvor tydelig dataene peker i en retning. For å måle dette er det to varianter som brukes for beslutningstre, og det er ved bruk av entropi eller Gini Index. For å illustrere tankene bak byggeprosessen, skal det ses nærmere på entropi.

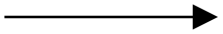
Formelen for entropi ser slik ut, hvor p_i representerer sannsynligheten eller forekomstene av en gruppe i et datasett. T representerer kolonnen som analyseres.

$$E(T) = \sum_{i=1}^{C} -p_i \cdot log_2(p_i) \tag{1}$$

Svaret som formelen gir er mellom 0 og 1, hvor 0 antyder et rent datasett, noe som er ønskelig, mens 1 representerer høy urenhet. Eksempelvis, kan man ta ut en kolonne fra tabellen på figur 26, for å

regne ut renheten basert på tallene som er registrert. [34]

Navn	Alder	Barn?	Studie?
Hans	41	1	0
Trine	22	0	1
Bjørn	31	1	0
Petter	27	0	1
Katrine	35	1	0



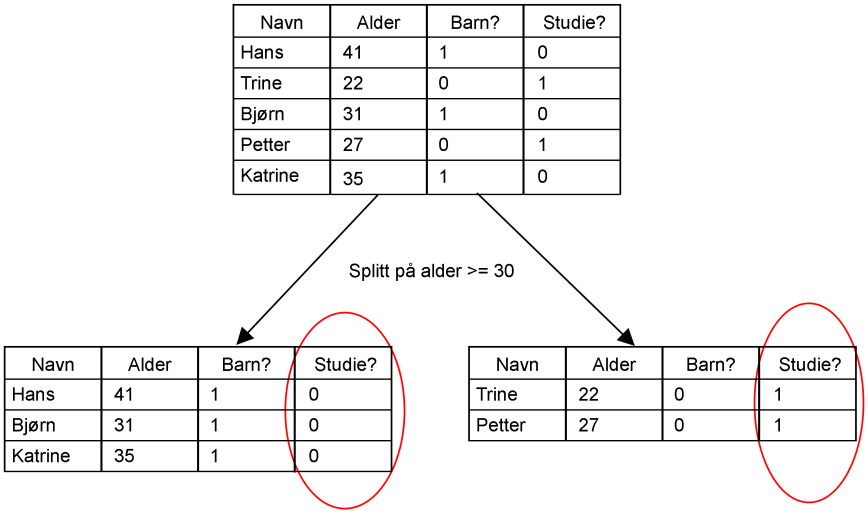
Studie?
0
1
0
1
0

Figur 26: Entropien til en kolonne ses nærmere på.

Ved å se på forekomstene av hver klasse, kan man se at $2/5$ av personene studerer, mens $3/5$ av personene ikke studerer. Ved å benytte formelen ovenfor, ville entropien blitt;

$$E(T) = (-\frac{2}{5} \cdot log_2(\frac{2}{5})) + (-\frac{3}{5} \cdot log_2(\frac{3}{5})) = 0.97 \tag{2}$$

Som tyder på et veldig urent datasett. Det holder ikke for et beslutningstre å kun se på disse enkle tabellene, men må se det i en helhet istedet, ettersom det kan være et stort tre med ulike variabler som skal bygges. Det interessante målet for beslutningstre er er derfor "Information Gain", som benytter entropi, til å beslutte hvordan det optimale treet ser ut. For å illustrere det, er det vist på figur 27 hvordan den tidligere tabellen kan splittes basert på en relatert variabel, som i dette tilfellet var alder.



Figur 27: Hvordan man kan måle fremgang.

Ved å regne ut entropien i hver av de nye tabellene, kan man se at renheten har økt betraktelig i begge. Siden størrelsen på datasettet er såpass lite, er entropien lik 0 i begge de nye tabellene, som vil si at de er helt rene. "Information Gain" består stort sett av å se på alle slike mulige tilfeller for splittelser for å finne ut hva den kan lære mest av. Formelen for det ser slik ut;

$$IG(T, A) = E(T) - \sum_{v \in A} (\frac{T_v}{T} \cdot E(T_v) \tag{3}$$

Kort sagt regnes først den entropien i de nye tabellene med korrekt vektleggelse i forhold til antall data i de nye tabellene. Disse trekkes fra den originale entropien. Dersom resultatet er positivt betyr det at treet vet mer enn før, ergo at renheten er større. Fra eksempelet har man E(T) = 0.97. Videre summeres opp entropien i de nye tabellene. For den venstre tabellen blir det;

$$(\frac{3}{5} \cdot ((-\frac{3}{3} \cdot log_2 \frac{3}{3}) + (-\frac{0}{3} \cdot log_2 \frac{0}{3})) \tag{4}$$

For begge tabellene er entropien 0, altså helt rent. Det vil si at;

$$IG(T, A) = 0.97 - (0 + 0) = 0.97 \tag{5}$$

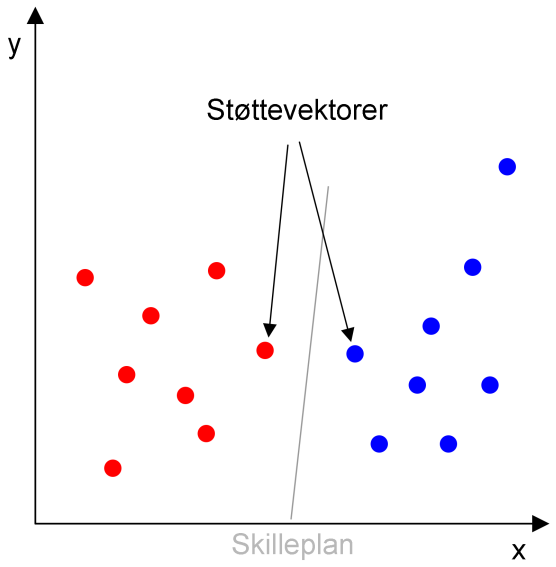
Det viktigste i forhold til hva "Information Gain" gir er at den er positiv. Om den er positiv betyr det at det har blitt større orden på datasettet. Desto større, desto mer har man lært. [17]

Fordelen med å benytte beslutningstre i applikasjonen er det er blant de enklere modellene for maskinlæring. Beslutningstre er spesielt enkle å lese og forstå, noe som gjør den ekstra anvendelig. I tillegg gjør modellen det mulig å benytte seg av klassifikasjon, som er veldig passelig i dette tilfellet. I likhet med lineær regresjon, lar det seg også gjøre å ta hensyn til flere variabler, som kan være høyst nødvendig, med tanke på individuelle forskjeller. Ulempen som følger med modellen ligger i byggeprosessen av treet. Det er noe som Monkey C ikke har et bibliotek for, og som derfor må løses på egenhånd.

På bakgrunn av at det er en modell som er lett anvendelig, fleksibel og ikke minst leselig, ble denne modellen tatt i bruk senere.

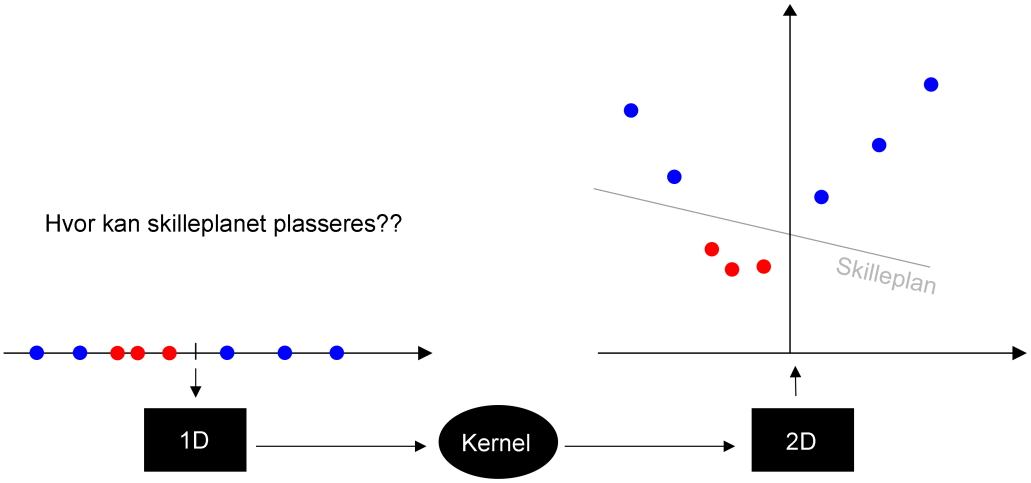
3.2.3 Support-Vector Machine

SVM står for Support-Vector Machine. Maskinlæringsmodellen baserer seg på å klassifisere data ved å separere de ved bruk av et plan. Planet oppstår ved å finne den største avstanden fra de nærmeste punktene til planet. Denne avstanden kalles marginen. Hver side av planet representerer da en klassifikasjon. Fra ett todimensjonelt perspektiv ser det ut som på figur 28. Når nye datapunkter kommer inn, plasseres de i rommet, og desto lengre unna planet punktet er, desto sikrere er klassifikasjonen. [3]



Figur 28: Skille mellom klassene.

Datasettet som man har å jobbe med kan være vanskeligere å klassifisere enn det som er vist ovenfor. Ovenfor er det et tydelig skille mellom gruppene, mens datasettet kan ofte være vanskeligere å skille. Måten SVM håndterer dette er å skille ved å gå opp en dimensjon til modellen har lært seg å skille de godt nok. På figur 29 vises hvordan et nokså rotete en-dimensjonelt datasett kan la seg skilles ved å gå opp en dimensjon. Dette kan gjøres igjen og igjen til skilnaden er stor nok.



Figur 29: Fra 1D til 2D via "Kerneling".

Styrken til Support-Vector Machine modellen ligger i det som kalles "Kernel Machines" eller "Kernel Functions". "Kernel Machines" definerer en rekke algoritmer for mønster analyse. Oppgaven til disse algoritmene består av se etter mønstre av ulike typer, som korrelasjon eller klassifikasjon. Det er en rekke ulike "Kernel Functions" som kan brukes, som gjør SVM veldig fleksibel. Av den grunn kan modellen funke godt både for strukturerte og ustrukturerte data, selv om det er en fordel med førstnevnte. Til slutt passer det i utgangspunktet godt til prosjektet, ettersom modellen foretrekker mindre datasett. Ulempen som følger med denne modellen ligger i kompleksiteten som følge av styrkene. Det å velge riktig "Kernel Function" kan være utfordrende. I likhet med nevrale nettverk, kan også SVM være vanskelig å både forstå og oversette. [29]

I utgangspunktet kunne SVM vært aktuelt til prosjektets problemstilling, basert på måten modellen fungerer og mindre faktorer som at det foretrekker mindre datasett. Den er dog litt for komplisert og tidkrevende, som gjør at den ikke ble sett nærmere på videre.

3.2.4 ARIMA

ARIMA ("autoregressive integrated moving average") er en tidsseriemodell som blant annet bruker korrelasjon til å forutse hvordan en graf kan ende opp med å se ut. For å forstå hva dette er, burde man forstå hva en AR-modell er for noe. Det er en modell som kan for eksempel forutse hvor mye strøm noen kan komme til å bruke neste måned med å se på hvor mye som ble brukt en annen måned. Da kan modellen blant annet se på forbruket forrige måned eller for ett år siden. Denne modellen er god til å forutse i sykluser og siden denne personen ikke bruker like mye strøm hvert år, kommer denne modellen sannsynligvis til å ta noe feil [25]. Det er her MA-biten kommer inn, den sier noe om hvor mye feil antagelsen var forrige måned, og tar dette i betrakning når den regner ut strømbruken for denne måneden [26]. Dette blir også kalt for en ARMA-modell [27]. Denne modellen fungerer også best på sykluser, så for å se på noe annet kan ARIMA-modellen bli brukt.

I denne modellen kan man tenke se at en graf over tid med en viss stigning, som sier noe om antall klokker solgt. Man vil igjen forutse hva den neste verdien blir og det er her I-biten blir brukt med at den lager en ny graf som er nokså lik den andre, bare at den ikke har den jevne stigningen. Dette gjøres med at den sammenligner den neste verdien med den forrige verdien, eller en annen verdi d steg tilbake i tid og plasserer dette i den nye grafen. Når den nye grafen er konstruert brukes ARMA-modellen på den, og man finner så ut den neste verdien og transformerer denne inn i den originale grafen. [24]

Dette er en modell som kan egne seg godt til å finne ut hva den neste verdien kan være, men hvis det den skal gjette på ikke har noen klar syklus og/eller ikke har et jevnt stigningstall kan gjetningen bli nokså feil.

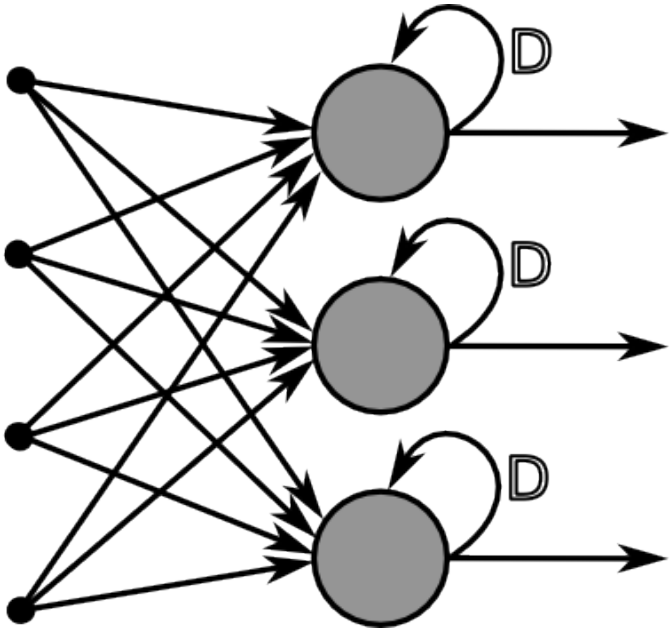
ARIMA er en modell som brukes til å finne ut hva den neste verdien kan bli, og er ikke noe som finner ut hvor godt et resultat stemmer overens med noe annet, som blant annet er noe som blir gjort i dette prosjektet. En annen ting er at modellen bruker tidligere data til å finne ut hva den neste blir, som muligens kunne blitt brukt til å finne ut hva pulsen til en med PRESS er etter x meter. Ulempen er at denne modellen i dette eksempelet ville vært veldig utsatt for eksterne variabler som blant annet bakker og klima. Dette gjør da at det kan bli vanskelig å implementere denne modellen til en som er brukendes.

3.2.5 Nevrale Nettverk

En annen måte å analysere dataen på er å bruke nevrale nettverk. Forskjellen på nevrale nettverk og maskinlæring er hovedsakelig måten de lærer på. En maskinslæringsalgoritme vil få data tildelt

av noen, og for å gjøre den bedre må den få hjelp av et menneske. En algoritme som bruke nevrale nettverk vil bruke dataen den får inn og vil etter hvert lære av sine egne feil, og det uten å få hjelp av et menneske [13]. Noen typer nevrale nettverk er da RNN og CNN.

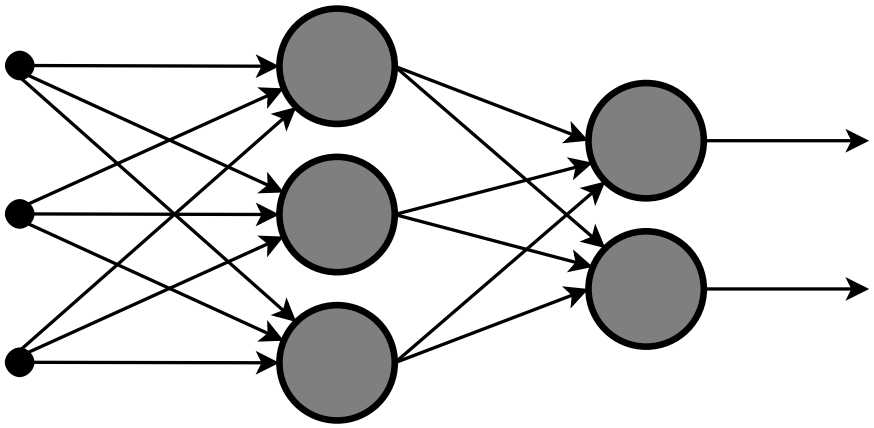
RNN eller "recurrent neural network" er et nevralt nettverk som kan "huske". Det blir brukt til å ta i mot vilkårlige inndata og som prosesserer det slik at det klarer å kalkulere seg frem til hva som skal forstås og sammenligne det med det som har blitt kalkulert før. Dette fungerer veldig bra der ting skal gjøres etter hverandre, som for eksempel å gå gjennom ei sjekkliste eller å lese og forstå ei setning. RNN kan også bli brukt til å forutse hva en setning skal bli for noe når den blir skrevet, for eksempel til autokorrekt på en mobiltelefon.



Figur 30: RNN. [7]

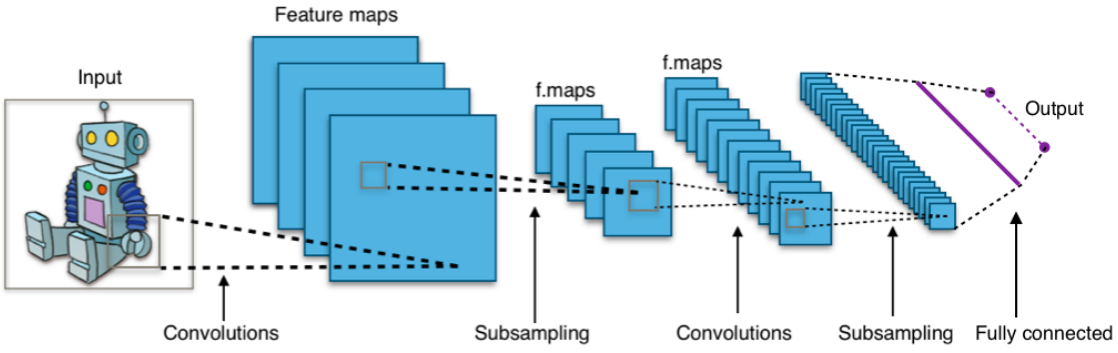
CNN eller "convolutional neural network" er et annet eksempel på nevrale nettverk, det som er spesielt med dette er at det lager flere ulike filtre som brukes når data blir prosessert. Det som gjør dette så bra, er at for å komme frem til en del av noe, blir den delen gått gjennom av flere filtre som sammen kommer frem til ett bedre svar. Akkurat dette er spesielt effektivt når det er snakk om bilder.

[7] Chrislb, CC BY-SA 3.0
https://upload.wikimedia.org/wikipedia/commons/d/dd/RecurrentLayerNeuralNetwork.png



Figur 31: CNN. [8]

Fordelen med å bruke slike nettverk er at de kan finne ulike mønstre på tvers av flere variabler og så klare å komme frem til en god gjetning som kan stemme godt overens med det meste. For å finne de beste mønstrene og deretter få de mest presise gjetningene, må det analyseres store mengder med data. I denne oppgaven er datamengden noe begrenset og da er det en mulighet for at nevrale nettverk kan finne rare mønster som tilfeldigvis stemmer overens med dataen som blir brukt til å trene det opp. En måte å komme rundt dette på er å vektlegge spesifikke data gjennom treningen mer enn andre. Akkurat dette er noe som kan ramme RNN, da det potensielt er svært vanskelig å vekte de ulike minnene det har opparbeidet seg om ting, som da kan føre til en upresis eller feil gjetning [32]. En av ulempene til CNN er at kompleksiteten med flere filtre har en pris, deriblant at det kan kreve svært mye av en datamasking å lære opp nettverket, og siden det er nokså komplekst bruker det også en del tid til å gå gjennom alt det skal gå gjennom.



Figur 32: Et hendelsesforløp CNN kan ha. [9]

[8] Offnfopt, CC BY-SA 3.0
https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Multi-Layer_Neural_Network-Vector-Blank.svg

Noen ulemper med nerale nettverk er at de kan kreve store datasett og mye prosessering for å bli nøyaktige og at det er veldig vanskelig, om ikke umulig for oss mennesker å begrunne hvorfor datamaskinen har kommet frem til den konklusjonen den har [9]. I tillegg så ville det vært komplisert å implementere den til å fungere til dette prosjektet med tanke på den tiden som har vært til rådighet. Dette er altså neon av grunnene til at det ikke blir brukt her.

3.3 Sammenlikning med andre applikasjoner

Oppgaven ønsker å se nærmere på hjerteraten til brukeren under aktivitet. Det som ønskes å se nærmere på er utviklingen, som f.eks; Hvor raskt hjerteraten stiger eller synker og hvor høyt eller lavt den går. Dette skal ses i sammenheng med den fysiske formen brukeren er i, samt individuelle trekk, for å kunne gjenkjenne unormal oppførsel ved hjerteraten. Prosjektet har derfor ulike utregninger for å se på trenden av treningsøkten i samsvar med fysisk form, og i tillegg andre ressurser, som tabeller, tilgjengelig for å kunne plassere personen under en kategori.

Før utviklingen av applikasjonen var det interessant å se nærmere på hva som finnes på markedet allerede. Ettersom Garmin har sitt eget rammeverk for utvikling av applikasjoner i tillegg til et marked, eksisterer det mange type applikasjoner fra før av. Blant de som har valgt å dele kilde-koden med alle, var det interessant å se på fremgangsmåter og løsninger som ble valgt i forhold til liknende problemstillinger som vi stod over.

3.3.1 Åpen-kilde applikasjoner

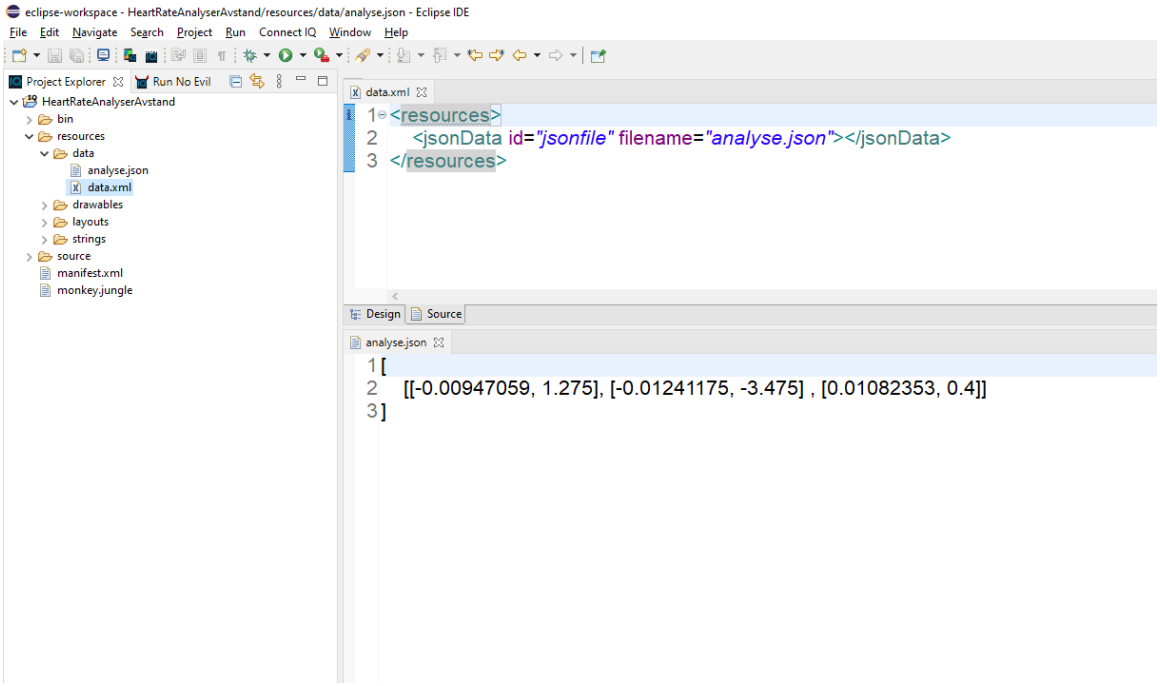
3.3.1.1 Kaloriteller Det første eksempelet på et annet prosjekt er en kalori-måler. I løpet av treningsøkten, får man opp ulike varianter av mat med et kalori-innhold som tilsvarer antall kalorier brent i løpet av treningsøkten. Det gjøres en 2% differanse sjekk, for å sørge or at differansen er liten nok til å bytte, men foruten om det er det få utregninger. Det brukes også en look-up tilnærming, basert på informasjon om maten. Denne informasjonen lagres i et dictionary. Applikasjonen som skal lages bruker en liknende tilnærming gjennom å estimere basert på kunnskap som finnes. Som vist på figur 33 er tabellen hardcodet, som vil si den er mindre tilbøyelig til endringer.

[9] Aphex34 CC BY-SA 4.0 16. des 2015
https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png

```
25
26  protected var equivalents = {
27      // https://www.nutritionix.com/
28      "1" => {
29          "1" => "$1$ raspberry",
30          "+" => "$1$ raspberries",
31      },
32      "2" => {
33          "1" => "$1$ strawberry",
34          "+" => "$1$ strawberries",
35      },
36      "3" => {
37          "1" => "$1$ M&M",
38          "+" => "$1$ M&M's",
39      },
40      "4" => {
41          "1" => "$1$ Skittle",
42          "+" => "$1$ Skittles",
43      },
44      "8" => {
45          "1" => "$1$ french fry",
46          "+" => "$1$ french fries",
47      }
```

Figur 33: Hardcodet dictionary.

For deres applikasjon kan det være en passende tilnærming, ettersom den type informasjon sjelden endrer seg. Applikasjonen som utvikles i prosjektet ønsker å bruke en liknende tilnærming ved å hente kunnskap fra utsiden, men ta det et lite steg lenger ved å gjøre det mer fleksibelt. Det vil si at det skal være enkelt å bruke, oppdatere og vedlikeholde informasjonen fra utsiden på klokken. Gjennom å oppdatere en json-fil på klokken, som vist på figur 34, lar det seg enkelt gjøre å implementere nye funn eller oppdaterte datasett.



Figur 34: Bruk av ressurser til å hente informasjon.

Totalt sett en relativt ulik applikasjon, men en veldig lik tilnærming til den som ble benyttet med tanke på å oversette maskinlæringsmodellen fra et annet språk til Garmin sitt rammeverk. [19]

3.3.1.2 Halvmaraton løpeoversikt Et annet eksempel er et prosjekt som det i utgangspunktet er en god del ulike versjoner av. Denne typen prosjekt inneholder all informasjonen du skulle ønske med tanke på aktiviteten som gjøres. I dette tilfellet omhandler det løping, og enda mer spesifikt for et halvmaraton løp. Av den grunn inneholder den utregninger som omhandler fart og tid, men inkluderer også steg-frekvens og hjerterate. Utregningene som omhandler fart innebærer den gjennomsnittlige farten de siste 60 sekundene, og de siste 10 sekundene, for å kunne sammenlikne. Her kontrolleres målinger fra de siste 60 sekundene til enhver tid gjennom bruk av en liste, og dette manipuleres for å få tak i de siste 10.

Applikasjonen som skal lages bruker en liknende tilnærming når det kommer til å periodisere informasjonen som eksisterer under en treningsøkt. I likhet med deres applikasjon gjøres dette gjennom å benytte optimale ressurser i forhold til minnet og effektivitet. Det området som applikasjonen som utvikles ønsker ta et steg videre henger sammen med funksjonaliteten. Applikasjonen som utvikles skal i tillegg ta hensyn til at det skal kunne byttes mellom ulike former for analyse, samt ulike tidsvinduer under analysen. Dette er på bakgrunn av individualiseringsbehovet, hvor det gjelder å

være fleksibel. Ulike individuelle forskjeller kan forsterkes ved ulike perioder, som gjør at behovet for å justere kan være stort.

I det store bildet gjelder det for applikasjonen som utvikles å være fleksibel i forhold til bruk og funksjonalitet. Dette er noe de fleste applikasjoner slipper unna, med tanke på at de utvikles for et spesifikt formål. Enda formålet er klart for applikasjonen, kan det være flere veier til målet. [18]

3.3.2 Oppsummering

Det å analysere hjerteraten gjøres i likhet med andre prosjekter gjennom relativt ukomplekse likninger. Man kan like vel differensiere mellom dette prosjektet og andre, ved at det legges opp til at applikasjonen best mulig kan benytte en rekke ulike parametere for å gjenkjenne individet bak aktiviteten best mulig.

For å kunne fastslå om hjerteraten oppfører seg som den skal, er applikasjonen knyttet til forskning utenfor selve applikasjonen. I tillegg benyttes det maskinlæringsalgoritmer, for å best mulig kunne lære å gjenkjenne mønstrene for forventet oppførsel. Dette er noe som ikke er så altfor vanlig i samtlige andre prosjekter. Denne dataen lagres i et JSON format, som applikasjonen henter informasjon fra. Dataen oppdateres også stadig, og derfor er det et ekstra fokus på automatisering av oppdateringen. Gjennom alle ledd, fra analysen til applikasjonen til bruk av applikasjonen på klokken, skal det enkelt kunne justeres og oppdateres for å ha optimal analyse til enhver tid.

Sammenliknet med de åpen-kilde applikasjonene som finnes ute på markedet, prøver applikasjonen å ta det et lite steg lenger ved å ta i bruk kilder og modellen fra utsiden og integrere de.

4 Utvikling

I dette kapittelet blir det sett på hvorfor programvaren har blitt sånn som den har blitt. Dette innebærer hvorfor simuleringen er slik den er og hva som kan gjøres for å skille mellom ulike mennesker og deres form. Det blir også gått mer inn i detalj hvordan modellen har blitt bygd opp og hvordan den fungerer. Til slutt blir det sett på hvordan det blir for sluttbrukeren og om dette kan påvirke batteritiden på klokken.

4.1 Identifisering av måleperiode

Grunnen for at simuleringen ikke begynner rett etter pausen på toppen av Tinghaug, er fordi at når de fortsetter å sykle igjen stiger pulsen deres ganske raskt. Dette medfører at det kan bli vanskelig å finne ut hvor mye pulsen stiger eller synker, om den endrer i det hele tatt. Derfor er det mye enklere å begynne simuleringen når deltakerne startet på nedoverbakken og vi kan derfor sammenligne deres pulsendring i mye større grad. Dette er fordi at deltakerne da har en mye mer forutsigbar og fast endring i pulsen. En annen ting er at det ser ut som at bakken begynner ved et kryss, som gjør det mye enklere å klippe opp fitfilene slik at simuleringen begynner på samme plass på tvers av filene vi bruker.

Figur 35: Tinghaug.

I simuleringen blir pulsen til deltakerne sammenlignet med referansegrafen (figur 15) hver femtiende

meter. Grunnen for at det akkurat er den avstanden er ganske lett, nemlig at det er et ganske fint og rundt tall. Det er heller ikke for ofte, slik at det ikke blir for mange målinger å holde styr på, og det er heller ikke for få målinger slik at det blir vanskelig å se personlige variasjoner i pulsen til hver deltaker.

Analysen er over etter 800 meter. Dette er fordi at det er ganske nøyaktig den avstanden fra krysset til en sving der deltakerne måtte bremse, svinge og begynne å trå for å få opp hastigheten igjen. Om simuleringen skulle fortsette etter dette, måtte den ha tatt hensyn til at at pulsen til deltakerne begynner å stige når han eller hun må ta i litt for å få opp igjen farten. Hvis simuleringen hadde stoppet etter for eksempel 900 meter hadde resultatene blitt mye mer feil.

I simuleringen ble det først prøvd å bruke sekund til å bestemme når den skulle registrere målingene, altså at nye målinger blir registrert per x antall sekund. Dette fungerte ikke helt optimalt da ulike deltakere bruker ulik tid til å nå svingen, som igjen fører til feil i analysen. Det ble så konkludert med at det var mer presist å bruke meter syklet isteden for sekund siden bakken er like lang for alle. I referansegrafen er det også brukt meter langs x-aksen som gjør det enklere å sammenligne data mellom grafen og deltakerne.

4.2 Individualisering av en modell

Modellene som er blitt brukt ser på en spesifikk periode hvor det viser seg å være en ulik utvikling av hjerteraten blant gruppene. Modellene retter fokus mot utvikling, som vil si at hvor høy eller lav hjerteraten er på starten ikke vil påvirke utfallet. I fremtiden kan det vise seg å være andre perioder eller trender som skal ses nærmere på, hvor forskjellen blant individer har mer å si. Det kan derfor være verdt å ta hensyn til at ikke alle har lik vekt, alder og fysisk form. De tre tingene kan hver for seg påvirke utviklingen en person har i hjerteraten sin under en treningsøkt, og derfor kunne det være lurt å kunne skille mellom dem.

Vekten spiller en rolle i kraften som brukes av utøveren under økten. Hvor mye kraft som brukes i ulike perioder, kan være verdt å se nærmere på, ettersom det kan påvirke hvordan hjerteraten ser ut. Dette kan være spesielt viktig når vi sammenlikner ulike personer. Garmin sine "Watch App" applikasjoner får ut hvor mye watt som brukes. Denne kan deles på vekten, for å få ut det som betegnes som "Power" som da er:

$$Power = \frac{watt}{vekt} \tag{6}$$

Siden modellene som er blitt laget i dette prosjektet bruker lineær regresjon og bestemmelsestrær

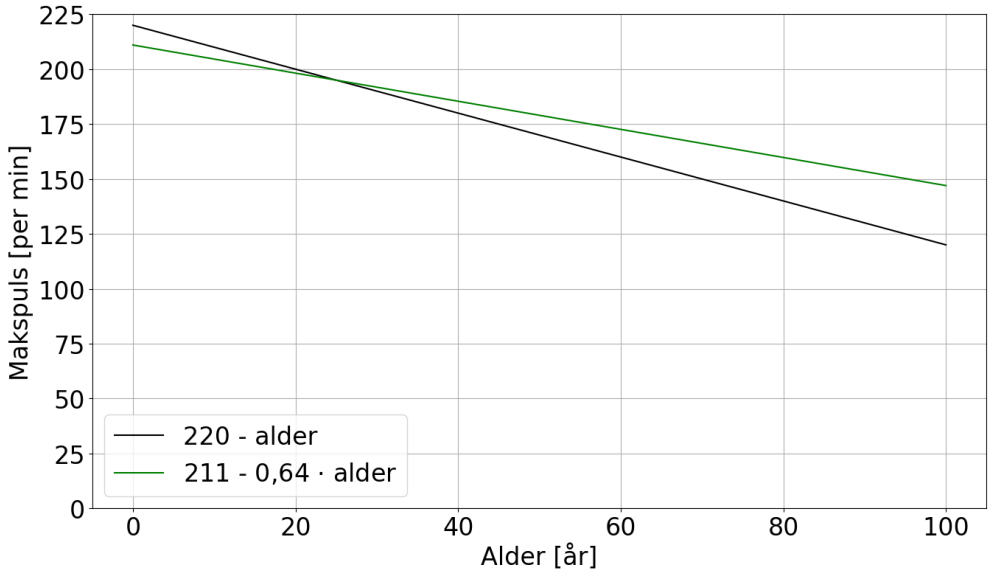
burde det ikke være for komplisert å bruke power til å bidra til å finne en passende gruppe til en deltaker. Spesielt når det kommer til bestemmelsestrær så burde dette være relativt enkelt å ta i bruk. Vekt er også noe som påvirker noen hvilepuls. Hvis man er overvektig er det også en mulighet at makspulsen er lavere enn hos noen som har en normal vekt. Fedme gjør også at hvilepulsen blir høyere [30].

Alder er noe som kan være relativt enkelt å ta hensyn til. Ettersom man blir eldre, vil den maksimale hjerteraten synke. Hvor mye den synker med avhenger av andre variabler, som fysisk form og genetikk, samt andre variabler som værforhold. Generelt over, ser man like vel et tydelig skille mellom den maksimale hjerteraten av en person i 20-åra og en person i 60-åra. Når man blir eldre, vil også hvilepulsen synke noe med årene. Det enkleste estimatet for maksimal hjerterate er å ta 220 og trekke fra alder, men det er også en annen måte som er litt mer presis. Det er å ta alder multiplisert med 0,64 og trekke fra 211 [28]. Det leder til de enkle formlene;

$$Maksimalhjerterate = 211 - alder \cdot 0,64 \tag{7}$$

$$Maksimalhjerterate = 220 - alder \tag{8}$$

Disse formlene vil da lede til denne grafen. Her vil forskjellene på de komme godt til syne da formel 8 vil synke mye raskere enn formel 7.



Figur 36: Viser forskjellen mellom de to formlene for makspuls.

Fra den maksimale hjerteraten kan videre informasjon utledes som melkesyreterskler og hjerterate soner under trening. Ved å bruke en modell som sammenlikner hjerterate direkte, i stedet for å se på utviklingen, vil det være viktig å ta hensyn til akkurat dette.

Ikke alle personer er like godt trent, noe som kan påvirke hvor fort en person kan gå fra relativt høy puls, til en som er mye lavere. For eksempel så vil pulsen hos en som er i god fysisk form synke med 20-30 pulsslag ett minutt etter en makspulstest, mens hos en som er i dårlig form kan den synke med under 15 slag [15]. I dette prosjektet blir analysen kjørt når deltakerne «hviler» i en nedoverbakke, så da kan pulsen synke ganske mye hvis de faktisk hviler i denne bakken. Siden pulsen er det som er brukt her til å analysere, så kunne det vært viktig å ta hensyn til, spesielt hvis de som bruker dette prosjektet er i en helt annen form enn de deltakerne som analysen er bygget på.

Det er også andre ting som kan påvirker pulsen til mennesker, som blant annet blodprosent, varme, alkohol, koffein, røyking og stress [5]. Hvis en person har lite blod i kroppen, må hjertet pumpe oftere for å klare å sirkulere nok blod til kroppen. Varme gjør slik at blodkar utvider seg, og da må det pumpes mer siden det er et større volum å fylle. Alkohol i blodet gjør slik at hjertet må jobbe raskere, og det samme skjer når man inntar koffein, om det så er fra energidrikker, kaffe eller noe lignende. Røyking over lengre tid vil øke hvilepulsen hos noen med 0,35 slag i minuttet og om man har en vane med å røyke en pakke med 20 røyk om dagen, vil dette gjøre slik at hjertet slår sju flere slag i minuttet enn om man ikke røyket. Dette er også noe av grunnen til at røyking øker risikoen for hjerte- og karsykdommer, siden hjertet må anstrenge seg mer [5]. Noe annet som påvirker pulsen er som sagt stress. Når man blir stresset får man adrenalin ut i blodomløpet, og dette påvirker hjertet med at det både slår hardere og raskere [20].

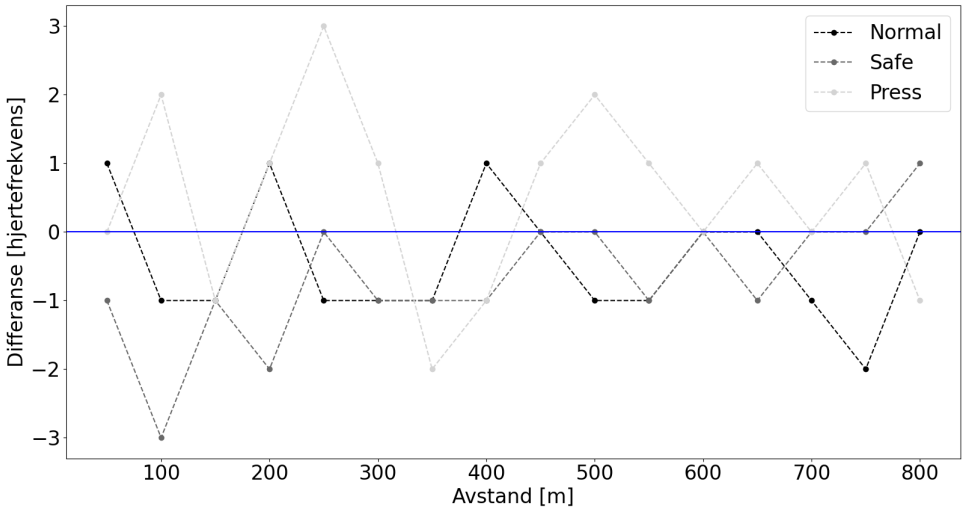
Alt dette er ting som kan påvirke modellen som er laget i dette prosjektet. Hvis det skal brukes til å avsløre problemer med blodomløpet, så kan det være en god ide å ta hensikt til at pulsen til folk er ulik og den er lett påvirkelig av ulike ting, om det er noe man kan for selv eller ikke.

4.3 Oversetting av en modell fra python

4.3.1 Lineær regresjon

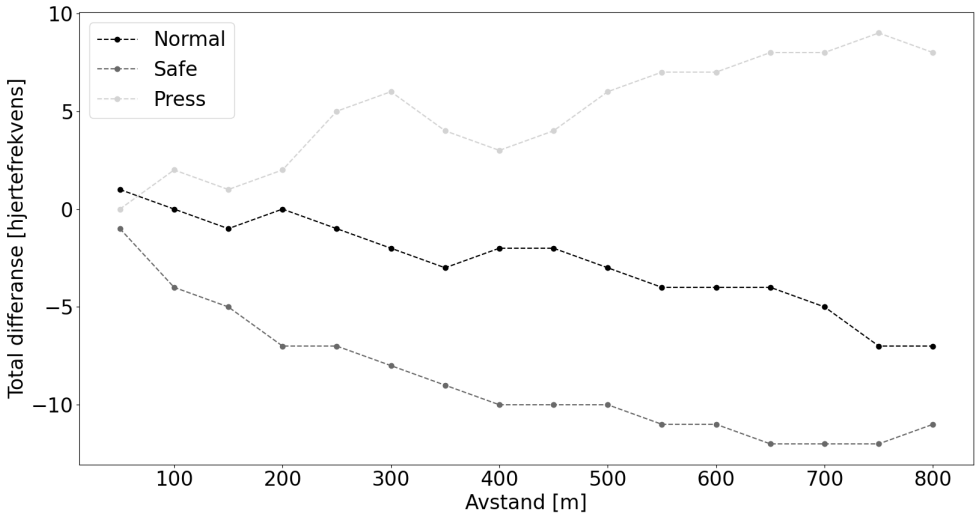
Siden er utviklingen av hjerterate over tid som er av interesse, ble dataene analysert gjennom lineær regresjon. For å gjøre dette, måtte hjerterate målingene, sammen med avstanden, hentes ut. Først og fremst er hjerterate veldig individuelt, som vil si at den kan generelt være høyere eller lavere, samt variere mer eller mindre hos ulike personer. Det ble det valgt å se på differansen mellom målingene, for å hindre forskjellen mellom høy eller lav hjerterate. For å minske variabiliteten, ble

det differansen mellom målingene sett på i sin helhet over en periode på 50 meter. Resultatet av dette var figur 37. Det er inkludert en blå linje langs midten, for å indikere null økning. Man er interessert i om dataene ligger over eller under midtpunktet. Fra grafen ligger de fra Press gruppen for det meste over, mens de som tilhører Normal og Safe ligger for det meste under.



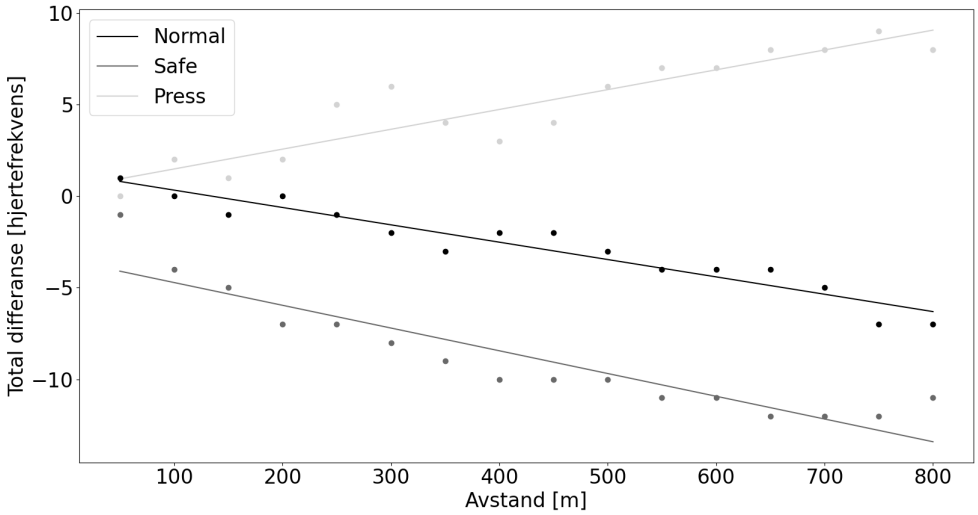
Figur 37: Individuell utvikling hver periode.

Denne dataen ble analysert videre, gjennom å se på den totale differansen ved hver nye periode. Det vil si at de foregående periodene ble inkludert, før den nye perioden legges til de foregående. Dette gjøres for å se på hvordan utviklingen ser ut over en lengre periode, ikke bare utviklingen for hver periode. Som følge av dette, ser man på figur 38 at det er et tydelig skille i hvordan hjerteraten utvikler seg over tid hos de ulike gruppene.



Figur 38: Total utvikling over tid

Det ble foretatt en lineær regresjons-analyse av dataene fra figur 38 for å komme frem til en formel for å kunne forutse hva som er den forventede utviklingen av hjerteraten er ved hver meter. Under, på figur 39, er resultatet av regresjons-analysen inkludert. Linjen viser hva som kan forventes i forhold til punktene som er på grafen.



Figur 39: Lineær regresjonsanalyse.

Linjen representerer en formel for den forventede utviklingen av hjerterate ved hver meter, og den

vises nedenfor. Her beskriver formel 9 Normal, formel 10 Safe og formel 11 Press.

$$y = -0.009x + 1.275 \tag{9}$$

$$y = -0.012x + -3.475 \tag{10}$$

$$y = 0.01x + 0.4 \tag{11}$$

Applikasjonen benytter resultatet av regresjons-analysen ved at det blir lagret i et XML format på klokken, som applikasjonen kan hente informasjon fra. Mer om dette i kapittel 4.5. Ved å foreta en regresjons-analyse behøves ikke mer data enn konstanten og koeffisienten for å anslå om brukeren tilhører en av gruppene. I likhet med analysen, ser man på perioder av 50 meter. I løpet av de 50 meterne, tar applikasjonen inn hjerteratemålingene som kommer, og sammenlikner med forrige måling for å se om den synker eller øker. Ved slutten av hver periode, legges differansene sammen, for å se den totale utviklingen av perioden. Som nevnt kan hjerteraten periodevis variere, selvom den tilhører en viss trend, derfor blir den totale differansen ved en periode lagt til de foregående periodene. Dette kan f.eks se slik ut, hentet fra en simulering;

Trend per 50 meter: [5, 2, 2, 2, 2]
Total trend: [5, 7, 9, 11, 13]

For å se hvilken gruppe personen tilhører, sammenliknes utviklingen av den totale differansen over distanse med estimatet på tilsvarende tid for hver av de tre funksjonene. Den funksjonsverdien som den totale differansen er nærmest, er den gruppen den tilhører.

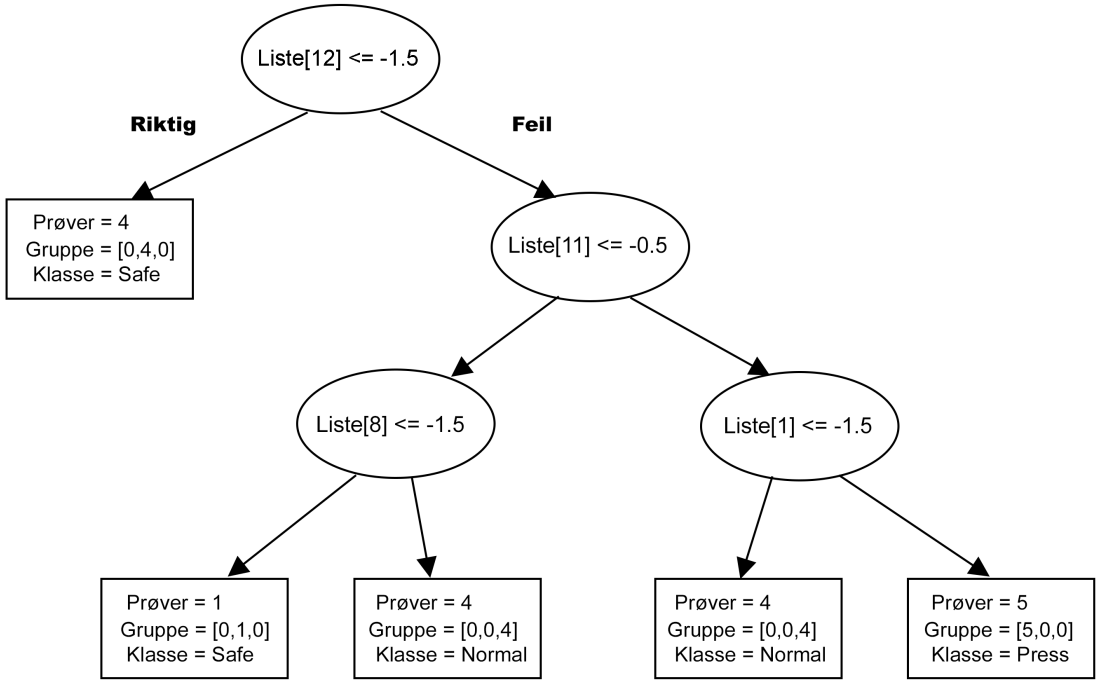
4.3.2 Beslutningstre

Formålet med å bruke beslutningstre er å kunne plassere brukeren i en gruppe basert på trend dataen over en liten distanse. Av den grunn måtte beslutningstreet som skulle bygges basere seg på trend dataene som ble skaffet gjennom de simulerte testene fra tidligere. Et sett med trend data som treet lærer fra kan se slik ut, og det er tydelig hvordan treet blir bedre jo mer data som er anskaffet.

Trend: [-3, -5, -4, -4, -6] -> Gruppe: Normal
Trend: [-2, -3, -3, -5, -6] -> Gruppe: Normal
Trend: [-4, -6, -8, -11, -12] -> Gruppe: Safe
Trend: [-5, -7, -9, -11, -13] -> Gruppe: Safe
Trend: [5, 7, 9, 11, 13] -> Gruppe: Press

Trend: [3, 3, 4, 5, 7] -> Gruppe: Press

Med dataene som skulle brukes på plass, manglet det bare en måte å bygge beslutningstreet på. Python sitt sklearn bibliotek består av ulike maskinlæringsmodeller, bl.a lineær regresjon som ble brukt tidligere, derfor ble det naturlig å bygge et beslutningstre ved hjelp av det biblioteket. Dataene ble samlet i et leselig filformat, for så å bli brukt til å lage treet i python. Treet endte opp med å se ut som i figur 40.



Figur 40: Beslutningstre basert på trend-data.

Etter dataen var samlet inn og beslutningstreet var bygget, gjenstod det å bruke modellen til å analysere brukere av applikasjonen. Treet måtte derfor bli oversatt fra python til Monkey C. For å minske minnebruk, ble dette gjort gjennom å lage og skrive funksjonen direkte til applikasjonen. For å lage funksjonen, måtte treet først gjøres om til et leselig format. Formatet som ble valgt var et streng format, som ble seende ut som på figur 41

```
|--- 10 <= 0.50
|   |--- 12 <= -1.50
|   |   |--- 8 <= -2.50
|   |   |   |--- weights: [1.00, 0.00, 0.00] class: lysegraa
|   |   |--- 8 >  -2.50
|   |   |   |--- weights: [0.00, 4.00, 0.00] class: moerkegraa
|   |--- 12 >  -1.50
|   |   |--- 10 <= -2.50
|   |   |   |--- weights: [0.00, 1.00, 0.00] class: moerkegraa
|   |   |--- 10 >  -2.50
|   |   |   |--- weights: [0.00, 0.00, 5.00] class: svart
|--- 10 >  0.50
|   |--- weights: [4.00, 0.00, 0.00] class: lysegraa
```

Figur 41: Strengrepresentasjon.

Funksjonen som ble lagd ble også skrevet i streng format, for å kunne skrive den direkte til en fil i applikasjonen. Det første som måtte gjøres var å identifisere de viktige elementene i utskriften fra figur 41. Det viktigste derfra var liste indexen, som representerer hvilken tidsperiode man ser på. De andre viktige elementene var verdien ved liste indexen, altså verdien som hjerteraten til utøveren under bruk skulle sammenliknes med. Til slutt ble også vektene tatt i bruk, som sier hvor mange innenfor hver gruppe som falt under denne noden. På figur 42 vises hvordan dette ble tatt vare på og brukt videre. Funksjonen er lagt ved i vedlegg A.1

```
Value : 0.50, Index: 10
Value : -1.50, Index: 12
Value : -2.50, Index: 8
weights: [1.00, 0.00, 0.00] class: lysegraa
Value : -2.50, Index: 8
weights: [0.00, 4.00, 0.00] class: moerkegraa
Value : -1.50, Index: 12
Value : -2.50, Index: 10
weights: [0.00, 1.00, 0.00] class: moerkegraa
Value : -2.50, Index: 10
weights: [0.00, 0.00, 5.00] class: svart
Value : 0.50, Index: 10
weights: [4.00, 0.00, 0.00] class: lysegraa
```

Figur 42: Nøkkelinfo.

Siden dette er et tre som må bygges fra bunnen av, var det viktig å identifisere høyden til nodene. Basert på utskriften, kan høyden til noden identifiseres gjennom å se på antall tabulator-strenger før man kommer til selve informasjonen. Resultatet av det kan man se på figur 43, og koden i vedlegg A.2.

```
('|--- 10 <= 0.50', 0)
('|   |--- 12 <= -1.50', 1)
('|   |   |--- 8 <= -2.50', 2)
('|   |   |   |--- weights: [1.00, 0.00, 0.00] class: lysegraa', 3)
('|   |   |--- 8 >  -2.50', 2)
('|   |   |   |--- weights: [0.00, 4.00, 0.00] class: moerkegraa', 3)
('|   |--- 12 >  -1.50', 1)
('|   |   |--- 10 <= -2.50', 2)
('|   |   |   |--- weights: [0.00, 1.00, 0.00] class: moerkegraa', 3)
('|   |   |--- 10 >  -2.50', 2)
('|   |   |   |--- weights: [0.00, 0.00, 5.00] class: svart', 3)
('|--- 10 >  0.50', 0)
('|   |--- weights: [4.00, 0.00, 0.00] class: lysegraa', 1)
```

Figur 43: Tre høyde.

Ettersom funksjonen må skrives som en streng i form av if/else-er er det viktig at funksjonen bygges

i enden og beveger seg innover mot roten. Når det er snakk om en node som ikke er roten og ikke er en bladnode, er det viktig å vite om noden har noen barn. Basert på utskriften, kan de to barnene til noden være nokså separert. Av den grunn ble nodene som hørte sammen satt sammen i par, slik at det var lett å vite hvor man skulle se for å finne barnenoder. Dette er vist på figur 44, samt lagt ved i vedlegg A.3.

```
The node at index: 0 has a pair at index: 11
The node at index: 1 has a pair at index: 6
The node at index: 2 has a pair at index: 4
WEIGHT at index: 3
WEIGHT at index: 5
The node at index: 7 has a pair at index: 9
WEIGHT at index: 8
WEIGHT at index: 10
WEIGHT at index: 12
```

Figur 44: Parvis.

Til slutt itereres den parvise listen med tupler, bestående av par-indexer og høyde. Den itereres baklengs, for å starte i bladnoden. Funksjonsstrengen ved hver node lagres i et dictionary, med sin index som nøkkel, slik at når dens foreldre ser at den har en node som barn kan den søke opp indexen dens. Ved å gjøre dette, kan funksjonen bestående av en rekke if/else-er bygges. Sammen med litt formatering endte koden opp som i vedlegg A.4, og funksjonen ble seende ut som figur 45.

```
function decTree(list){
    if (list[10] <= 0.5){
        if (list[12] <= -1.5){
            if (list[8] <= -2.5){
                System.println("weights: [1.00, 0.00, 0.00] class: lysegraa");
            }else {
                System.println("weights: [0.00, 4.00, 0.00] class: moerkegraa");
            }
        }else {
            if (list[10] <= -2.5){
                System.println("weights: [0.00, 1.00, 0.00] class: moerkegraa");
            }else {
                System.println("weights: [0.00, 0.00, 5.00] class: svart");
            }
        }
    }else {
        System.println("weights: [4.00, 0.00, 0.00] class: lysegraa");
    }
}
```

Figur 45: Funksjon.

Det eneste som gjenstod etter an funksjonen var bygget var å skrive den til prosjektet. Strengen ble skrevet til en .mc fil som ble lagret inne i prosjektet klar til bruk. For å sørge for at prosjektet er oppdatert, ble det også bygget på nytt, slik at den nye filen kan tas i bruk. Funksjonen som gjør dette vises i vedlegg A.5.

4.4 Brukervennlighet

Utseende programvaren har på klokken er ganske rett frem. Den har i grunn fire forskjellige bokser, der hver boks sier noe som er relevant for brukeren. Boksen oppe til venstre viser hjerterytmen til brukeren. Boksen rett under viser hvor langt brukeren har beveget seg siden analysen begynte. Begge de to boksene oppdateres jevnlig, men distanse-boksen stopper når analysen er ferdig. De to boksene til høyre viser resultatet til analysen. Den øvre viser resultatet etter lineær regresjon og oppdateres hver 50 meter passert, eller hva som den konstante variabelen *AVSTANDDELTA* er definert som i **JSON-filen**. Boksen under viser resultatet bestemmelsestreet får etter hver gang det har blitt kjørt.

Når analysen er igang, er fargen på boksene til høyre lysegrå og de til venstre er svarte (se figur 2).

Det er også mulig å endre klokkeinnstillingene til en mørk modus. Da blir all tekst som er svart om til hvit og de andre fargene forblir de samme (se figur 46). Når brukeren enten har beveget seg 800 meter eller har trykket på runde-knappen en gang blir fargen på de to og grønne og er da ferdig. Boksen som viser distanse blir også grønn da. Hvis brukeren trykker på runde-knappen før et beslutningstre er blitt testet, vises resultatet som *N/A*. Etter dette kan brukeren se hva resultatet til de to forskjellige modellene ble og etter hvilken distanse analysen ble gjort ferdig på.

Figur 46: Applikasjonen i mørk modus.

Tanken bak dette er at brukeren ikke skal trenge å se på resultatene underveis i analysen og at dette ikke er viktig før etter at analysen er helt ferdig. Så når den først er ferdig skal det være lett å se at den faktisk er ferdig uten å tenke for mye. Det er også lettere å lese hva som står når fargen har litt mer kontrast mot bakgrunnen.

Måten programvaren fungerer på er at brukeren må starte en aktivitet, om det så er løping, sykling eller noe annet, og samtidig ha valgt at dette programmet skal vises blant de ulike "Data Field"-ene.

Noe som ikke er fullt utnyttet i programmet er runde-knappen. Dette er den eneste knappen som kan oppdages av et "Data Field". De andre knappene kan sette et bakgrunnslys på skjermen, velge hvilket "Data Field" som skal vises eller om aktiviteten skal settes på pause.

Siden "Data Field" kun kan oppdage og bruke et trykk på runde-knappen, kan dette implementeres slik at programmet kan endre på noen variabler hvis brukeren trykker to ganger på runde-knappen

innenfor noen sekunder. Dette kan da brukes hvis brukeren vil at modellen for lineær regresjon skal oppdatere seg oftere eller sjeldnere med å endre på *AVSTANDDELTA*. Dette doble knappetrykket kan da rullere mellom variablene 25, 50 og 100. Noen andre idéer er at et det kan endres på hvor lenge analysen skal holde på eller hvor mange trær som skal brukes til bestemmelsestre-modellen.

4.5 Opplasting og oppdatering av modellen

For å ha en fremtidssikker programvare ble det satt fokus på opplasting og oppdatering, samt tilpassing. Å kunne endre på parametere og analyse metoder er her en essensiell del, og siden Monkey C ikke er et veldig utbredt og vanlig brukt programmeringsspråk, måtte det til med en mer tilgjengelig løsning.

En Garmin klokke kan lese fra diverse filer lagret i programvaren, så løsningen ble å lage en JSON-fil.

For å oppdatere programvaren ble det laget et Python-skript. Skriptet tar inn JSON-filen og oppdaterer den tidligere versjonen. Så bygges programvaren og resultatet er en ".prg" fil som kan overføres rett til klokken eller lastes opp til Garmin Connect IQ.

```
1 [
2     50,
3     800,
4     [[-0.00947059, 1.275], [-0.01241175, -3.475] , [0.01082353, 0.4]],
5     0.6
6 ]
```

JSON-fil.

Her er JSON-filen som blir brukt i oppgaven. Linje 2 er *AVSTANDDELTA* og linje 3 er avstanden som må passeres for at bestemmelsestreet skal fungere. Linje 4 er de ulike funksjonene som blir brukt til lineær regresjon, der der det første tallet inne i hver hakeparentes er stigningstallet og det andre er konstantleddet. På linje 5 er størrelsen på bufferen som blir brukt i lineær regresjon-modellen.

Connect IQ er Garmins applikasjonsbutikk. Enhver bruker kan laste opp applikasjoner til denne butikken. En ideell løsning på å oppdatere programvaren på alle klokkene hadde da vært å lastet den opp her. Problemet som oppstår da er at den må verifiserer og godkjennes av Garmin, noe som kan ta alt fra noen dager til uker. Dette vil da skape unødige forsinkelser og tregheter i prosessen for brukeren. For å løse dette kan man laste opp en applikasjon som en betaapplikasjon, noe som vil gjøre den tilgjengelig for kun den brukeren som laster den opp.

Det ble da vurdert et par forskjellige løsninger på dette problemet.

4.5.1 Metode 1: Felles Garmin konto

Alle er koblet til samme Garmin konto. Da vil en kun trenge å gå inn på Connect IQ og oppdatere appen. Problemet med dette er at brukerinfo som vekt, kjønn osv. blir samme for alle klokkene. Og det er vanskelig å skille mellom de ulike aktivitetene, ettersom man ikke kan se hvilken spesifikk klokke den er fra.

4.5.2 Metode 2: Oppdatering via en felles forskningskonto

Alle har sin egen Garmin konto. For å oppdatere og laste ned appen må man logge seg på den brukeren som har betaappen, synkronisere klokken, oppdatere appen og logge seg av. Man må så logge seg på sin egen bruker og synkronisere klokken.

Det endte her opp med metode 2. Metode 1 støter på problemer som vanskeligheter for å analysere data i framtiden. Dette er på grunn av at alle aktivitetene vil bli lastet opp på samme bruker, uten å kunne vite hvilke bruker den hører til.

Det blir heller ikke mulighet for å kunne bruke individuelle brukerdata som kjønn, høyde, vekt osv. ettersom det er satt på den delte Garmin kontoen.

Av disse grunnene ble metode 2 den valgte framgangsmetoden.

4.6 Batteriforbruk

Under utviklingen av programvaren som skal kjøre på klokken, var hele tiden batteribruk i bakhodet. Det var meningen at alt skulle være raskt og ting ikke skulle brukes unødvendig mye og ofte. I applikasjonens mappestruktur (figur 10) er det en fil som heter "HeartRateAnalyserAvstandView.mc" og i denne filen er det meste av programvaren som er utviklet av oss. Denne filen inneholder også standard-funksjoner som kjøres enten hvert sekund, når et knappetrykk blir registret og liknende. En av disse funksjonen er "compute()", den kjøres ganske regelmessig, omtrent hvert sekund. Funksjonen leder også til en annen funksjon som bestemmer hvilke andre funksjoner som skal kjøres hvis visse kriterier blir møtt. Poenget her er det ikke skal gjøres kalkulasjoner for ofte, for å ikke bruke for mye av klokkens ressurser slik at batteriet ikke blir tappet for raskt. For å oppnå dette er det som sagt visse kriterier som skal møtes:

```
if (info.elapsedDistance - startDistanse - avstandPassert >= AVSTANDDELTA){
    avstandPassert += AVSTANDDELTA;
    distanseAnalysert = info.elapsedDistance - startDistanse;
    calculateTrendAvstand(hrTrend);
    linRegression();
    hrTrend = [];
}
```

Én av flere if-sjekker.

Her er noen av de kriteriene som skal møtes for at et par av de andre funksjonene skal kjøres. I dette tilfellet er *AVSTANDDELTA* lik 50, som vil si at denne if-sjekken vil kjøres gjennom hver 50 meter som passeres. Før denne sjekken og etter, er det bare små kalkulasjoner og noen andre if-sjekker.

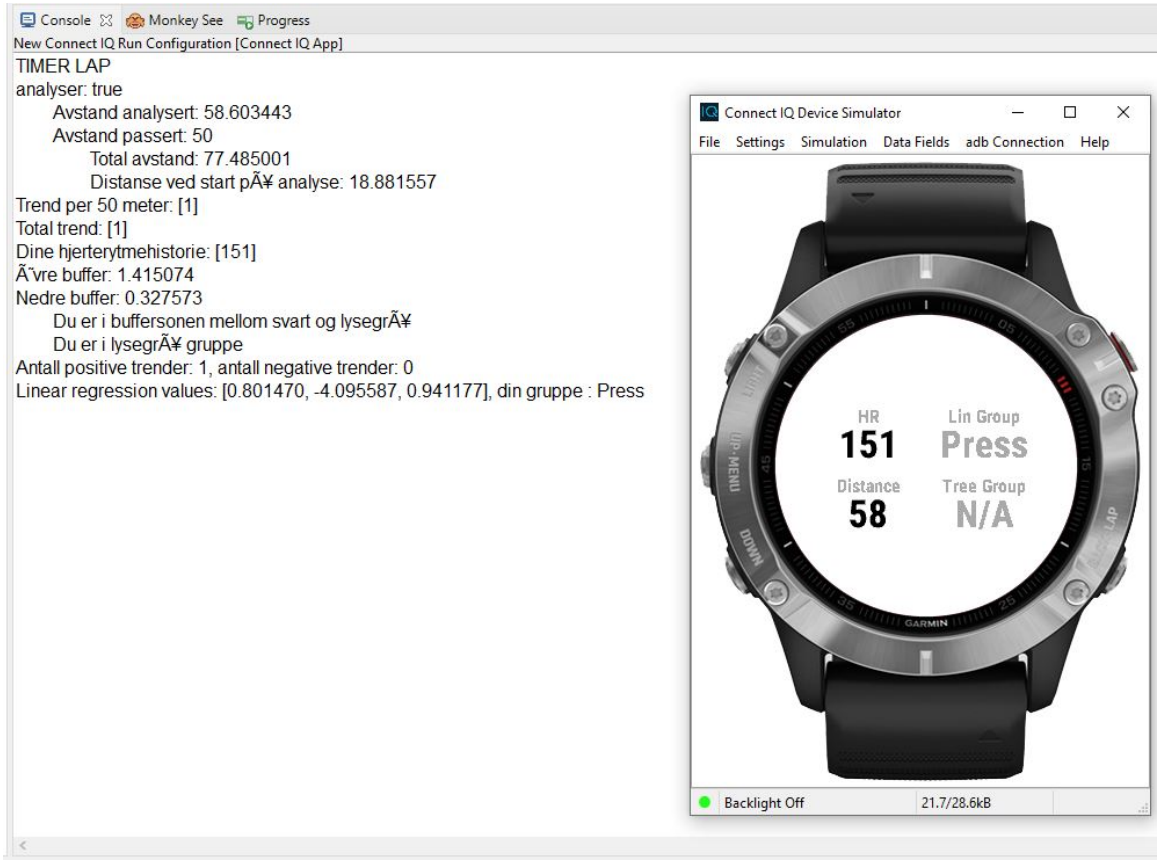
Generelt i koden er også doble for-løkker unngått og store deler av koden er bestemt av ulike if-sjekker. I denne filen er det også flere steder man kunne brukt en *switch/case* istedenfor flere *if/else*. Dette er ungått siden det bruker mye mer minne enn den mer tradisjonelle if-else [6]. Noe annet som er ungått så langt det lar seg gjøre, er å deklarere variabler som brukes i flere funksjoner flere ganger.

5 Evaluering

Etter at applikasjonen var utviklet, gjenstod det å teste den. Det som skulle testes var å plassere en person innenfor en av gruppene basert på modellene som ble brukt. For både den lineære modellen og beslutningstreet er hensikten å plassere personen i riktig gruppe så ofte som mulig. Måten de to modellene ble testet på var gjennom å simulere rittet til personer vi kjente tilstanden til. I søken på det beste estimatet mulig, måtte ulike variabler modifiseres for å se om presisjonen kunne økes. I tillegg skulle batteriforbruket testes, for å være sikker på at fornuftige valg er blitt tatt under utviklingen av applikasjonen.

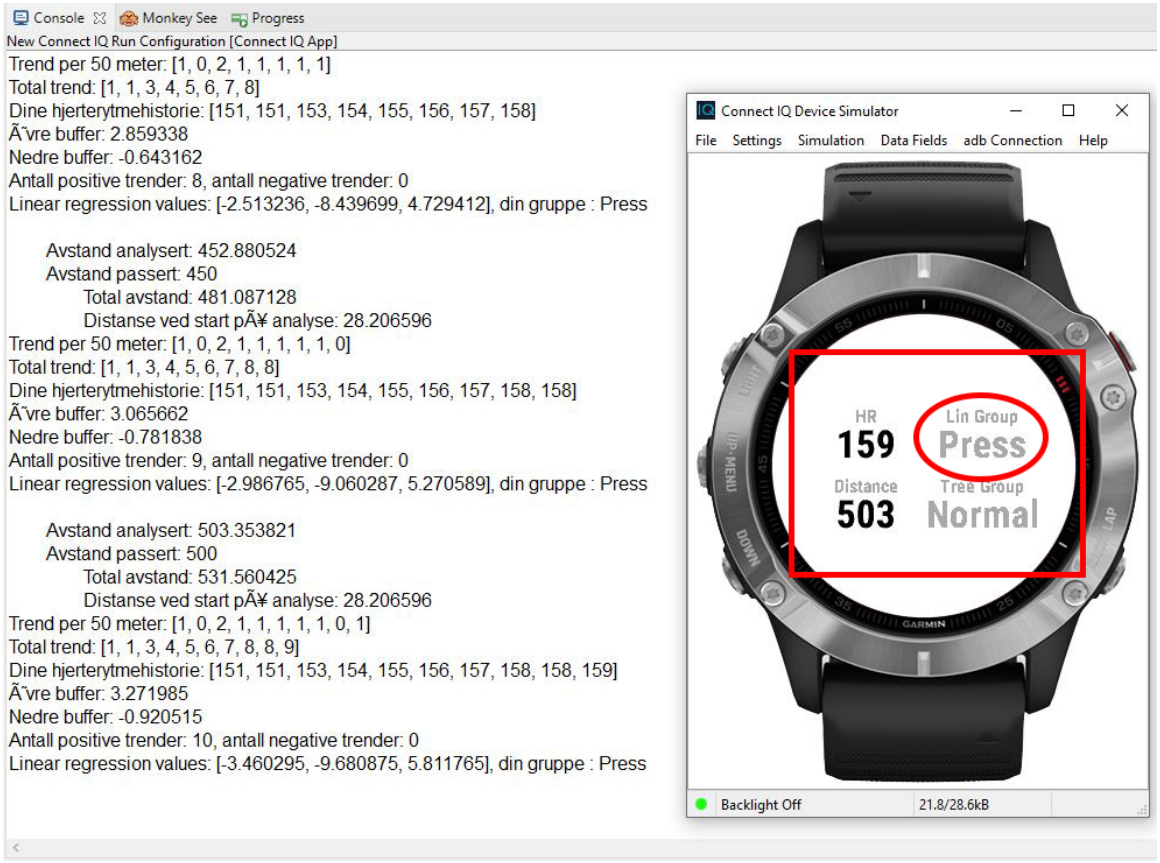
5.1 Lineær regresjon modell

Basen for den lineære modellen ble testet ved å se på trenden til utøveren over tid. Denne sammenliknes med estimatet, gjennom å bruke formlene som er lagt ved i kapittel 4.3.1. Til å begynne med er ikke estimatet til å regne med, ettersom det er såpass få målinger, men nedenfor på figur 47 kan man se hvilken gruppe personen er estimert til å tilhøre tidlig i simulasjonen.



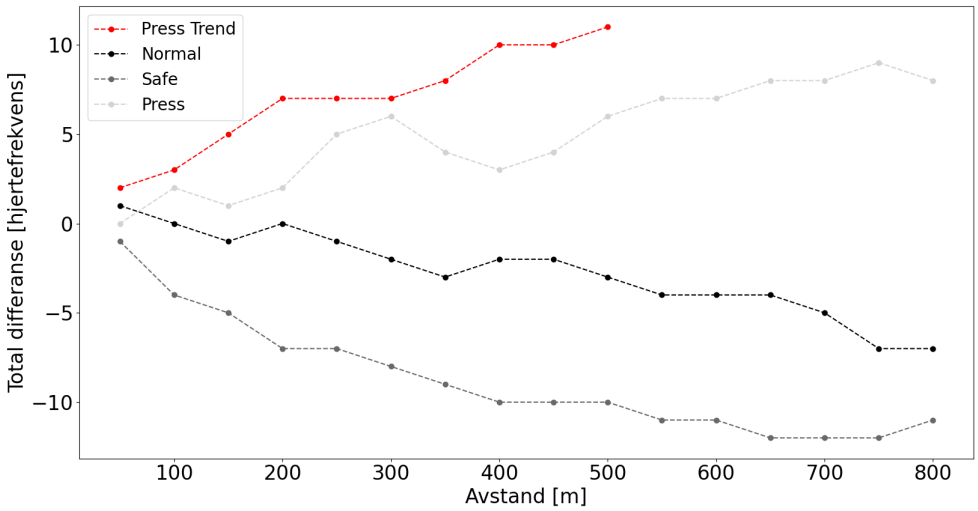
Figur 47: Starten av simulering.

Desto lengre man kommer ut i simuleringen desto bedre blir estimatet for hvilken gruppe personen tilhører, ettersom det er større skilnad mellom de. Ved å ta en nærmere titt på dataen halveis gjennom simulasjonen, kan man se hvordan det ligger an. Nedenfor på figur 48, er personen kommet litt over halveis ned bakken.



Figur 48: Midten av simulering.

Den foreløpige utviklingen kan ses i sammenheng med dataen fra analysen som modellen stammer fra. Ved å se på den totale utviklingen over tid på figur 49, kan man se hvordan personen i dette tilfellet ligger an til å tilhøre gruppen Press.



Figur 49: Hvordan det ser ut halveis.

Ved hver periode tar de tre funksjonene inn det foreløpige tidspunktet, og gir ut estimatet for den gruppen. Resultatet av dette er vist på figur 48, men nedenfor ser man hvordan likningen brukes helt nøyaktig. Her beskriver formel 12 Normal, formel 13 Safe og formel 14 Press.

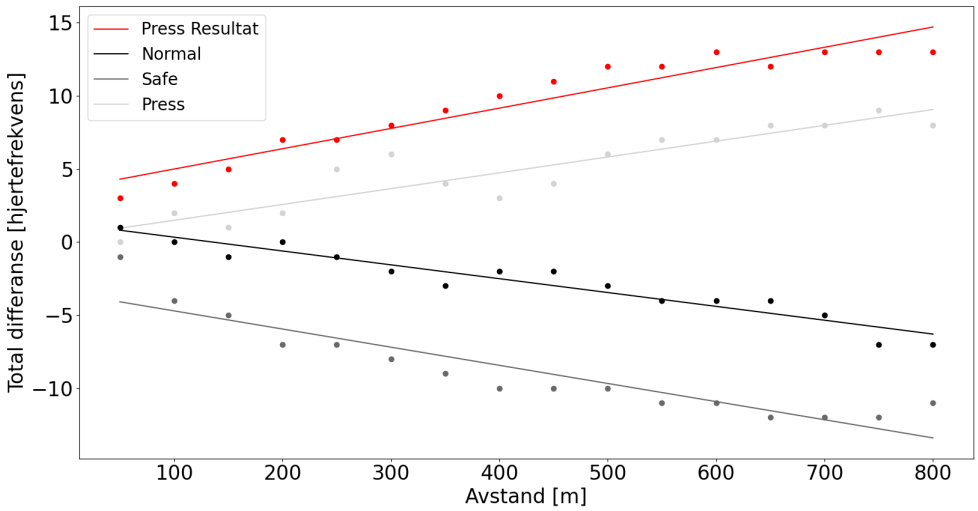
$$(-0.009 \cdot 5080) + 1.275 = -3.225 \tag{12}$$

$$(-0.012 \cdot 500) - 3.475 = -9.475 \tag{13}$$

$$(0.01 \cdot 500) + 0.4 = 5.4 \tag{14}$$

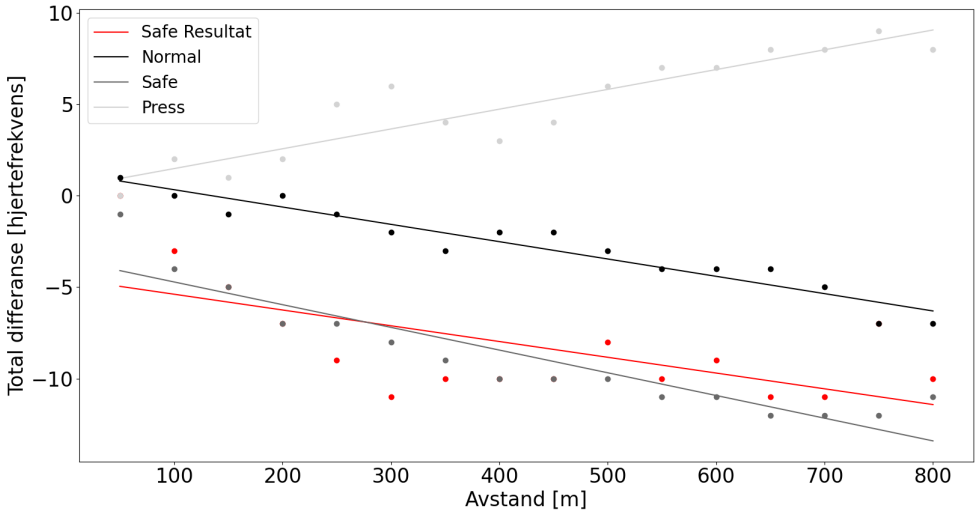
Trenden = 11, som vil si at den er nærmest 5.4 som tilhører Press

For å illustrere resultatet av hvordan trenden så ut for en person, ble det valgt å lage estimater for hver person, og plotte disse sammen med analysen for å se hvor personen burde ende opp. Resultat av en person innenfor Press gruppen vises på figur 50, hvor det forventes at hjerteraten stiger over tid.



Figur 50: Regresjonsanalyse av trend-data for en Press klassifisert person.

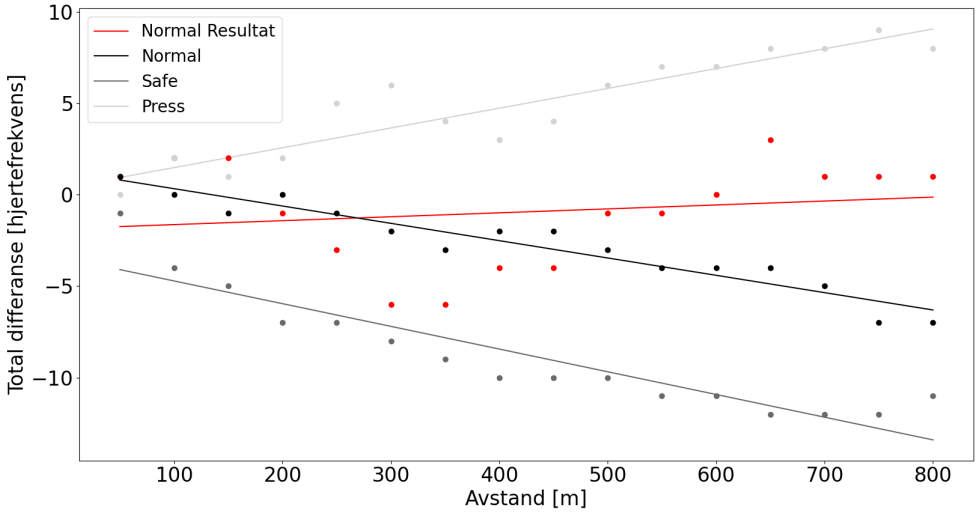
Resultat av den person innenfor Safe gruppen vises på figur 51, hvor det forventes at hjerteraten synker en god del



Figur 51: Regresjonsanalyse av trend-data for en Safe klassifisert person.

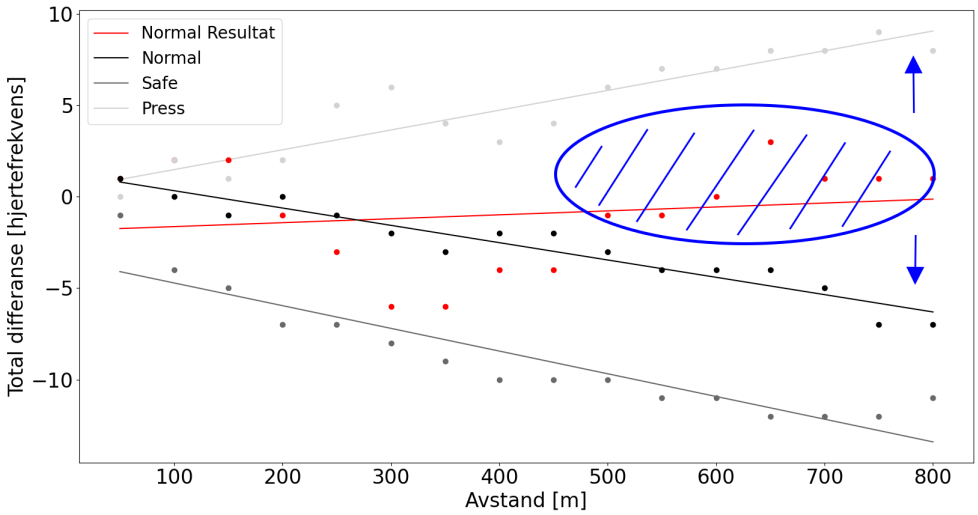
Til slutt resultatet av en person innenfor Normal gruppen på figur 52, hvor det forventes at hjert-

eraten synker, men hjerteraten varierer en god del.



Figur 52: Regresjonsanalyse av trend-data for en Normal klassifisert person.

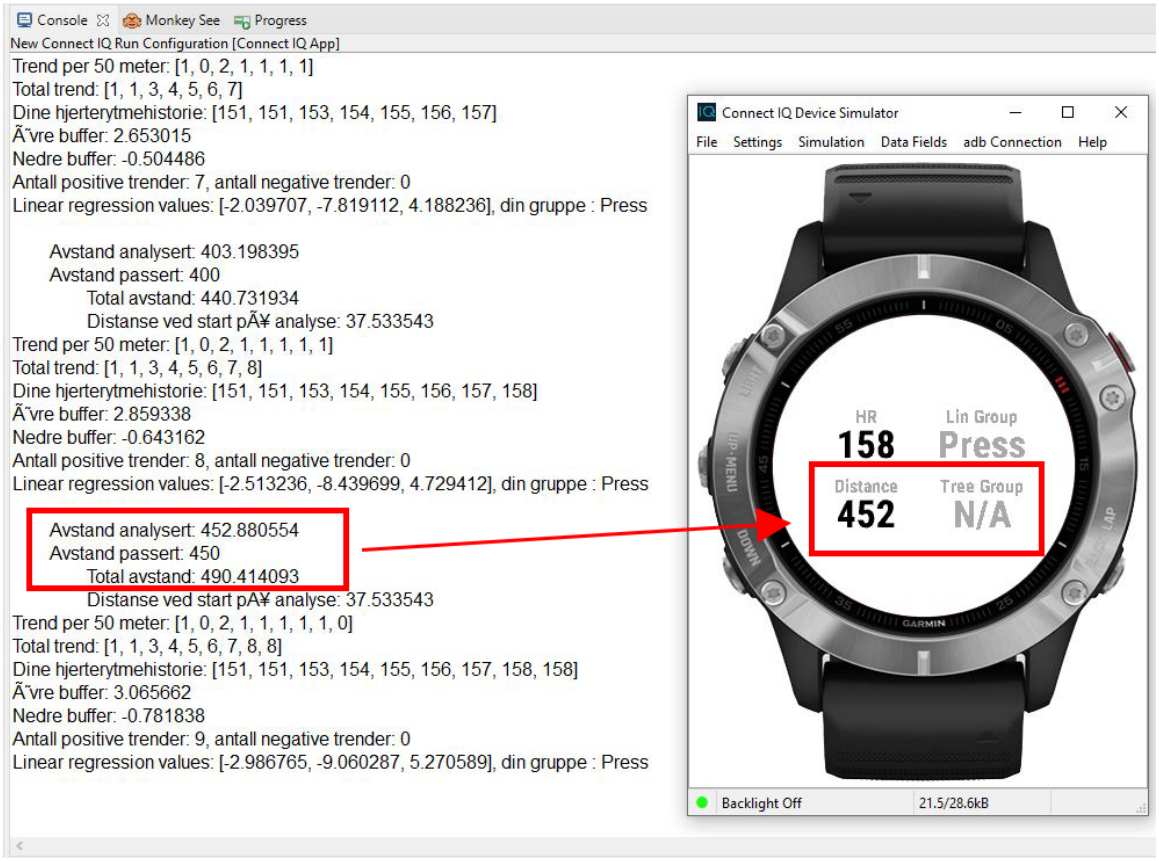
Som man kan se på figur 52, kan det å anslå hvilken gruppe personen tilhører kun basert på utviklingen være problematisk. Dette skyldes at estimatet for de tre gruppene er basert gjennomsnittet av alle personene. Eksempelvis, ved å kun se på et par enkeltmålinger, kan det se ut som trenden er stigende, selv om den i det store bildet har vært synkende mesteparten av tiden. For denne analysen er det viktig å ta hensyn til at målinger kan variere og det kan være vanskelig å estimere. Dersom utviklingen er tilsynelatende flat, altså at den havner mellom to estimat, telles det antall perioder i en synkende trend mot antall perioder en økende trend. På figur 53 illustreres buffersonen hvor det er vanskelig å tyde hvilken gruppe personen tilhører.



Figur 53: Regresjonsanalyse av trend-data for en Normal klassifisert person med buffersone.

5.2 Decision Tree modell

Fremgangsmåten for testingen av beslutningstreet foregikk nokså likt som ved testingen av den lineære funksjonen. Ved å simulere fit filer ved den aktuelle tidsperioden, med personer som allerede var kategorisert, var det mulig å sammenlikne med hva beslutningstreet anslo. Ettersom beslutningstreet bygges basert på hvor mye data det tar inn, betyr det at man ikke kan få ut et estimat før det er blitt samlet nok data under simulasjonen. Under, på figur 54, ser man hvordan det ikke lar seg gjøre å få ut et estimat enda. Listen av data er ikke lang nok.

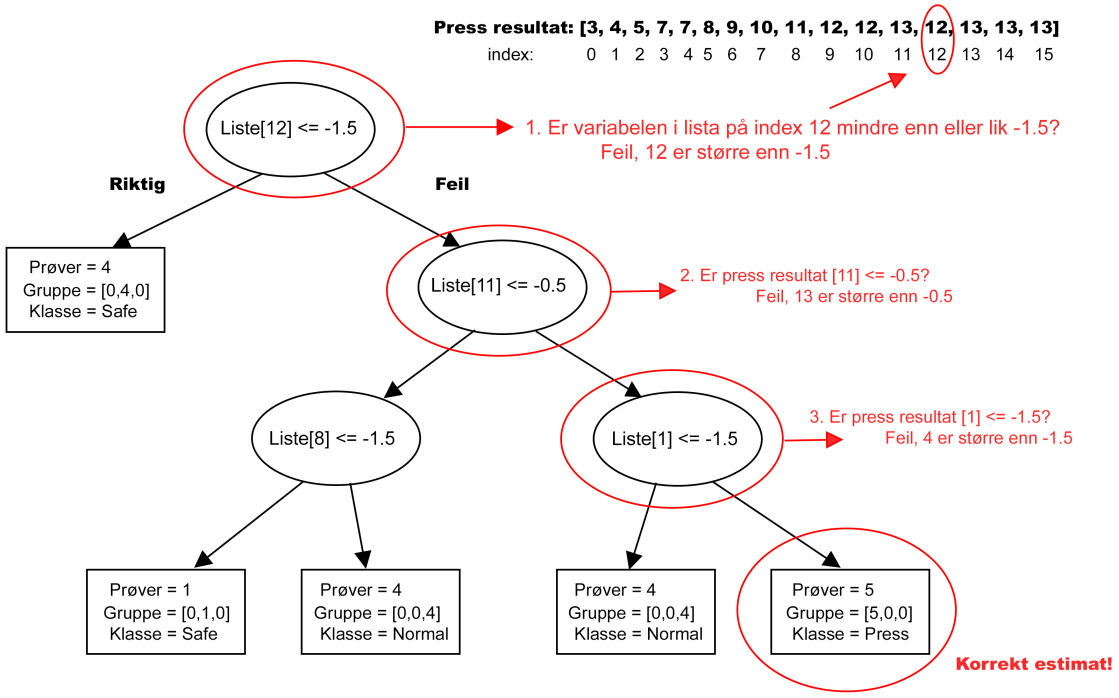


Figur 54: Manglende estimat.

Når utøveren kommer til punktet hvor det er samlet inn nok data, sendes listen med data gjennom funksjonen som estimerer hvilken gruppe personen tilhører. Det er samlet inn nok data etter 800m, som tilsvarer distansen til bakken det gjøres en analyse av. På simulatoren kan man se et estimat ved 800m, som vist på figur 55.

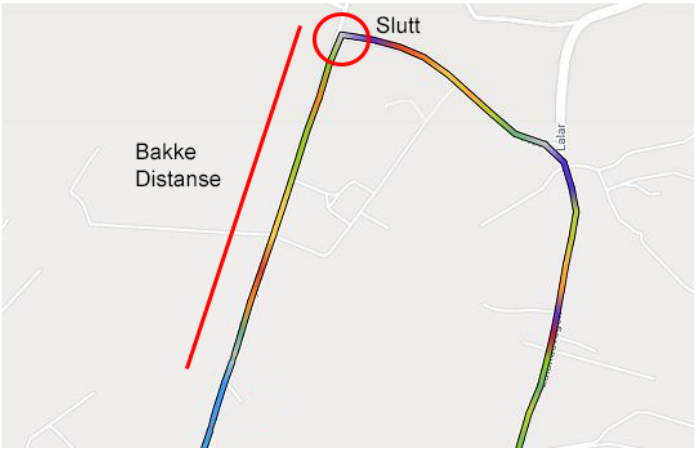
Figur 55: Endt simulering.

Fordelen med beslutningstre er at de er veldig leselige, og av den grunn fungerer funksjonen relativt rett frem. Når brukeren kommer til slutten av distansen som ønsker å analyseres, tar funksjonen inn trend mønsteret og traverserer gjennom treet til det kommer frem til et predikat for hvilken gruppe man tilhører. På figur 56 kan man se hvordan funksjonen brukes, illustrert i tre-format.



Figur 56: Beslutningstreet som traverseres.

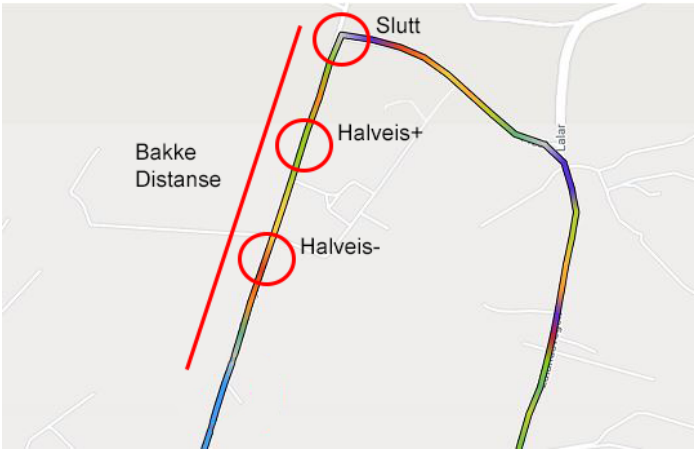
Ulempen med beslutningstreet viste seg å være at det kun lar seg gjøre å benytte den modellen en gang i løpet av distansen, altså ved slutten, som vist på figur 57.



Figur 57: Analysen ved bruk av ett tre.

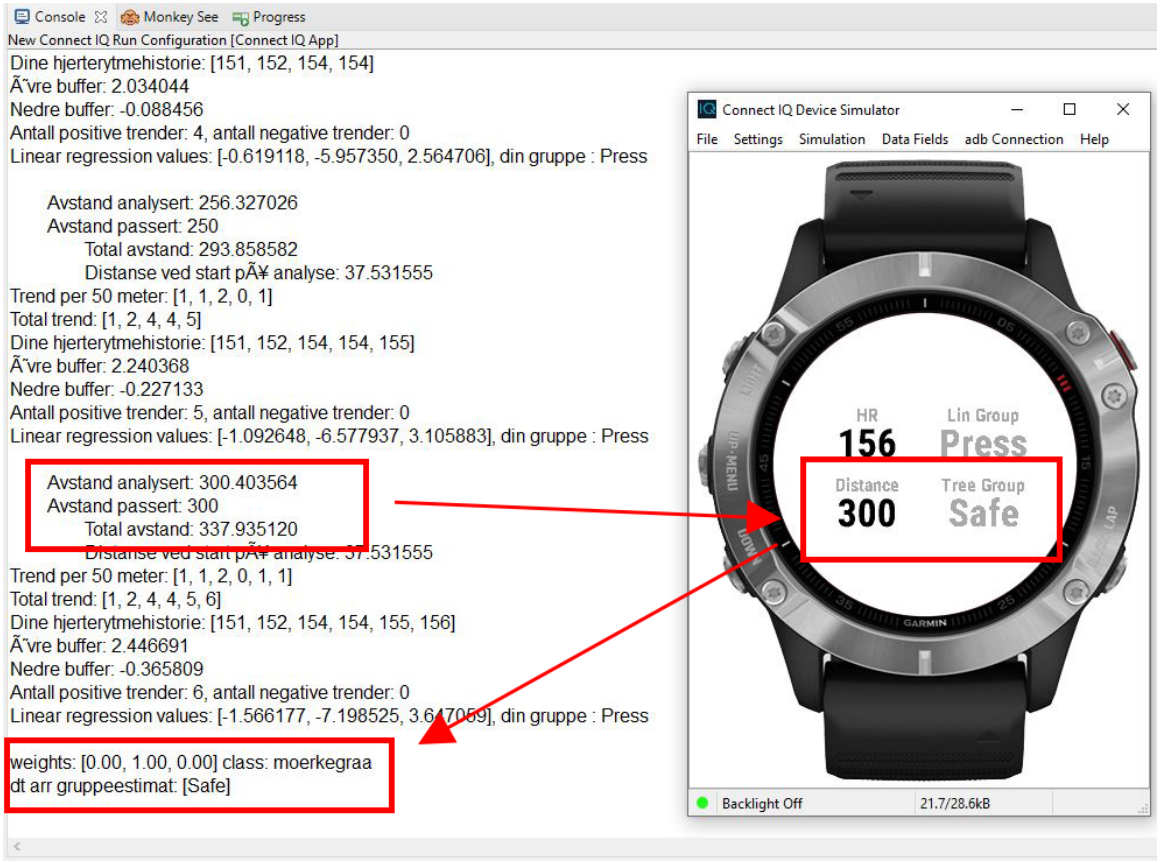
Målingene som er tilgjengelig kan være begrenset, og som følge av det kan også målingene være tvetydige. Av den grunn kan det ene estimatet ved veis ende komme til kort. Av den grunn ble det

benyttet samme fremgangsmåte som tidligere, til å lage flere trær over ulike perioder. Med det blir det mulig å få ut fler estimater, som kan være med på å påvirke utfallet. Det var nå mulig å få ut estimater for periodene på figur 58 i løpet av distansen som analyseres.

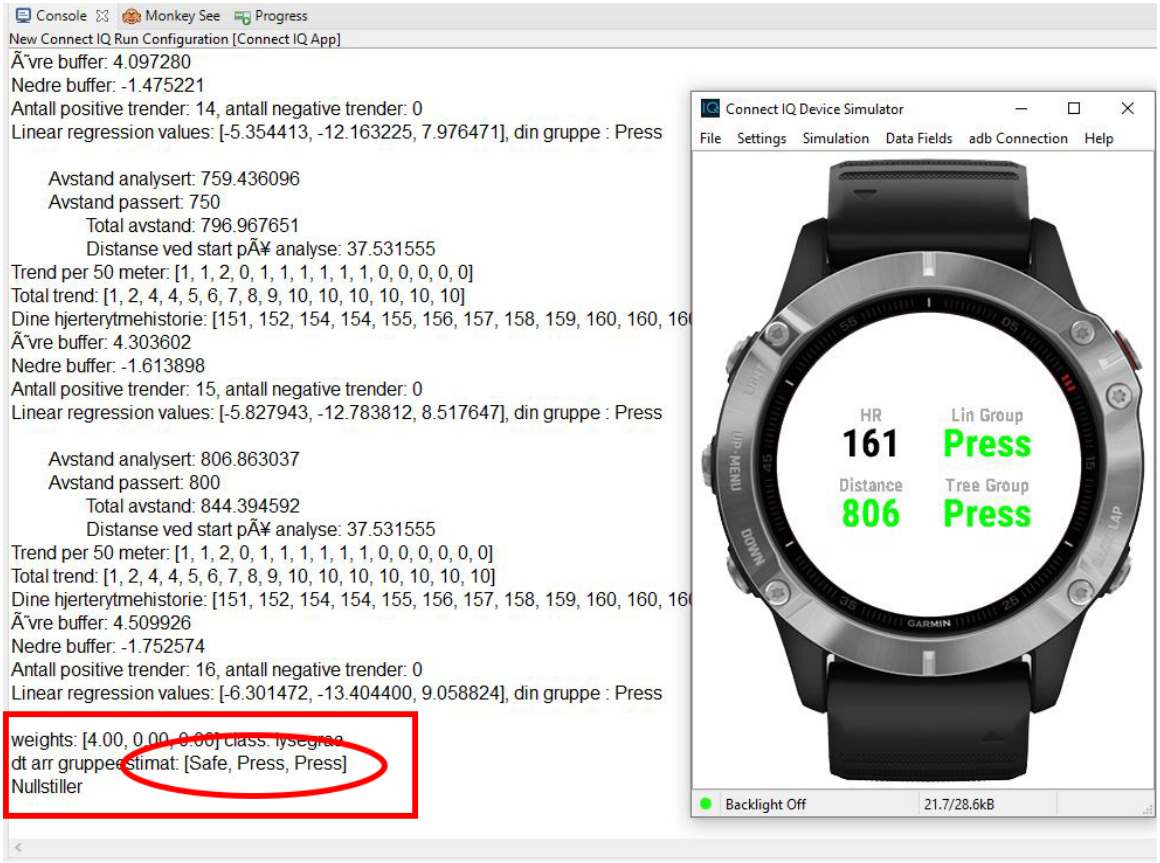


Figur 58: Analysen ved bruk av flere tre.

Som følge av å ha flere punkter å analysere, lar det seg gjøre å få et estimat tidligere før man er ved endt distanse. Estimatene legges ved i en liste, slik at alle estimatene potensielt påvirke det endelige utfallet. På figur 59 vises det hvordan det nå gis ut et estimat etter 300m, som er under halveis av den totale distansen.



Figur 59: Starten ved bruk av flere tre.



Figur 60: Slutten ved bruk av flere tre.

5.3 Batteriforbruk

En lang sykkeltur med en pulsklokke kan kreve en god mengde av batteriet, det var derfor viktig å forsikre at applikasjonen ikke forårsaket en stor økning i batteriforbruk. Siden det ble avgjort å gjøre målinger hver femtiende meter, vil det ikke gjøres kalkulasjoner alt for ofte. Noe som ble hypotisert til å gjøre slik at applikasjonen ikke ville ha en stor påvirkning på batteriforbruket.

Det ble derfor gjort en test som varte 1 time og 30 minutter der to klokker ble tilbakestillt til fabrikkinstillingene og "Dataregistrering" satt til "Hvert sekund".

Figur 61: Dataregistrerings innstillingene.

Applikasjonen ble så lastet ned på kun én av klokkene. Testen startet med full batterikapasitet, med en klokke på hver hånd. Det ble testet med den optiske hjerterytme sensoren på selve klokken og ikke pulsbelte for å forhindre noen form for forstyrrelser, over Bluetooth og ANT, med hverandre som kan påvirke batteribruket.

Figur 62: Begge klokkene med 100% batteri.

Klokkene ble ladet opp til 100% batterimengde. Applikasjonen ble brukt periodevis i 800 meters intervaller for å få testen den fullt ut.

Figur 63: Klokken til venstre har ikke applikasjonen, klokken til høyre har applikasjonen.

Halvveis under testen var batteriforbruket nokså likt. Etter testen var batteriforbruket på klokken med applikasjonen 0.3% mer enn klokken uten.

Figur 64: Klokken til venstre har ikke applikasjonen, klokken til høyre har applikasjonen.

Etter testen ble det konkludert at applikasjonens batteriforbruk var innenfor en akseptabel mengde. Flere kalkulasjoner vil naturligvis kreve mer batteriforbruk, og det ble gjort bevisste grep for å forhindre for mange av dem, som vist i kapittel 4.6

6 Konklusjon

Her blir de ulike målene for prosjektet gått gjennom og til slutt blir oppgaven konkludert.

6.1 Måloppnåelse

6.1.1 Presise resultater

Målet om presisjon var nok alltid det mest utfordrende målet, men kanskje også det viktigste for nytten av applikasjonen sin del. Basert på resultatene av å simulere fit-filene som ble utdelt utover i prosjektet, hvor utøverenes status allerede var visst, ble feilprosenten større enn ønsket innledningsvis. Som følge av det vil vi konkludere med at målet om presisjon ikke er nådd.

Det kan være en rekke ulike faktorer som spiller inn når det kommer til hvorfor målet ikke ble nådd. Den største kommer som følge av å undervurdere hvor store de individuelle forskjellene kan være. Når det er sagt tror vi det var korrekt å velge ut noen av de enklere maskinlæringsmodellene, og ha fokus på få variabler til å begynne med. I forhold til å holde feilmarginen lav burde nok fler variabler blitt tatt i bruk, større datasett å lære fra, samt vurdere andre alternativer til maskinlæringsmodeller.

6.1.2 Effektiv integrering av modeller

Målet om å integrere maskinlæringsmodeller gikk ut på å gjøre veien fra modell til bruk på applikasjonen så kort og enkel som mulig. Det ble benyttet to ulike modeller som begge krevde biblioteker fra andre programmeringsspråk til å bygge og analysere. Den ene modellen krevde lagringsplass for å benytte det som ble lært fra modellen, mens den andre kunne benyttes direkte ved å skrive den til applikasjonen. På bakgrunn av dette vil vi konkludere med at målet om en relativt effektiv integrasjon ble nådd.

Det var en periode et ønske om å optimalisere enda mer, ved å bruke script fra andre programmeringsspråk direkte på applikasjonen, men det ville ikke applikasjonens programmeringsspråk støtte. Det gikk likevel fint å skrive til applikasjonen fra Python, samt bygge filen på nytt, slik at prosessen med å oppdatere modellen alltid avsluttes ved å oppdatere applikasjonen også. Ulempen i forhold til integrasjonsprosessen er at applikasjonens lokale lokasjon varierer fra datamaskin til datamaskin. Av den grunn krever det at man klarer å lokalisere applikasjonen. Når det er gjort er resten tatt hensyn til, og av den grunn mener vi at dette ikke er en så stor ulempe.

6.1.3 Applikasjonen skal kunne påbygges

Målet om fleksibilitet og hvorvidt det ble oppnådd er i stor grad knyttet opp mot modellvalg, integrasjon og generell koding. Selv om modellvalgene er av de enklere maskinlæringsmodellene, tar begge hensyn til sammenheng mellom variabler og deres effekt. Med tanke på integrasjonen av modellene, ble prosessen relativt vellykket med tanke på effektivitet, men prosessen er nok ikke så fleksibel som vi skulle ønsket. Vi vil konkludere med at målet er oppnådd til en viss grad.

Vi lærte utover i prosjektet at det er store individuelle forskjeller fra person til person, og det ble derfor enda viktige å se på modeller som tillot å ta hensyn til dette. Den første modellen som ble valgt, altså den lineære var definitivt mer rettet mot enkelt bruk og få variabler, men regresjon evner å se på flere variabler og sammenhengen. Valget av den siste modellen, beslutningstre, var i større grad rettet mot bruk av flere variabler. Det ble lagt opp til bruk av flere variabler i modellene, men fleksibiliteten til integrasjonsprosessen er manglende. Det vil si at det må nok gjøres justeringer, hvertfall ved å se på flere enn en variabel.

6.1.4 En brukervennlig applikasjon

Målet om brukervennligheter i utgangspunktet vanskelig for oss å bedømme, ettersom det er vi som har utviklet applikasjonen. Det er dog minimalt med aktivitet som behøves for å bruke applikasjonen, ettersom "Data Field"-applikasjoner er designet for å være enkle og kompakte. Gjennom å teste forskjellige utseender basert på forskjellige fargekombinasjoner, kom vi frem til noe vi mener er enkelt, nøytralt og presist. Derfor kan vi, basert på vår egen evaluering, konkludere med at applikasjonen er brukervennlig.

Det ble naturligvis lagt mest fokus på personen som bruker klokken, slik at deres opplevelse er optimal. Til slutt mener vi det ble en brukervennlig opplevelse, men med det i bakhodet, skulle vi gjerne forhørt oss med flere andre enn hos selv for å se hva andre synes. Med tanke på den forskningsrelaterte brukervennligheten, er det noe som ble problematisk på bakgrunn av at Garmin må verifisere applikasjonen. Det ble laget en løsning hvor applikasjonen kan oppdateres og distribueres via nett, som gjør den relativt brukervennlig.

6.1.5 Lavt batteriforbruk

Målet om en god implementering av modellene og koding for å effektivisere strømforbruket så mye som mulig ble stort sett gjort gjennom effektiv koding. Ved å ha strømforbruket i bakhodet under utviklingen, er det enklere å unngå tunge løkker og mye minnebruk. Ved å se på resultatet av

testene, kan vi konkludere med at målet ble nådd.

6.2 Videre utvikling

Variasjonen på hjerterytmemønsteret fra person til person skapte en del problemer med å finne en god løsning på hvilke gruppe de hørte til. Ved videre forskning kan det finnes andre mønster enn kun hjerterytme. Det kan her være en idé å se på andre variabler, om det blir tilgjengelig, som for eksempel alder, kjønn, høyde og vekt. Med funn av andre variabler kan også da applikasjonen utvides for å ta hensyn til disse.

Hensynet til andre typer variabler kan for eksempel bli gjort i JSON-filen "analyse.json", der en bestemmer hvilke variabler en vil bruke. Det må da enten legges til en del ekstra kode for hver type variabel, eller ha en kode som ikke "diskriminerer" på hvilke type variabel den kjører.

Mønsteret som blir sett på er nokså spesifikt, og videre utvikling kunne sett på om det er andre områder under en treningstur, eller treningsturen i sin helhet, som kan inkluderes i analysen. Det kan her fort bli litt for mange bevegelige deler å ta hensyn til, og komplisere applikasjonen til den grad at minnebruk og batteriforbruk blir en større faktor å ta hensyn til. Med en minne-grense på 32 kB per "Data Field" på Garmin Forerunner 935 [11], er det ikke alt for mye rom til ekspansjon.

Med flere variabler, kan det også legges til funksjoner som at brukeren kan velge hvilke variabel som blir analysert under kjøring. Dette kan gjøres ved å bruke runde-knappen, der flere knappetrykk innen kort tid kan endre på variabelen.

Siden Monkey C ikke har samme like stor grad av bibliotekstilgang som andre programmeringsspråk, måtte det i oppgaven oversettes fra andre språk, som for eksempel Python. Det kan her videreutvikles ved å oversette flere maskinlæringsmetoder, som for eksempel SVM. Andre programmeringsspråk kan også bli tatt hensyn til, for å utvide fleksibiliteten og tilgangen til applikasjonen.

Referanser

[1] Decision trees explained, 2020. URL https://towardsdatascience.com/decision-trees-explained-3ec41632ceb6. [Lest: 06.04.21].

[2] H. Arnesen. Aterosklerotisk sykdom, 2018. URL https://sml.snl.no/aterosklerotisk_sykdom. [Lest: 05.03.21].

[3] N. Bambrick. Support vector machines: A simple explanation, 2016. URL https://www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html. [Lest: 04.05.21].

[4] J. Brownlee. Linear regression for machine learning, 2020. URL https://machinelearningmastery.com/linear-regression-for-machine-learning/. [Lest: 15.03.21].

[5] I. Brøndum. Hva vet du om pulsen?, 2019. URL https://iform.nu/sunn-livsstil/hva-vet-du-om-pulsen. [Lest: 08.05.21].

[6] CedricD. Performances dos and donts, 2018. URL https://forums.garmin.com/developer/connect-iq/f/discussion/6267/performances-dos-and-donts?fbclid=IwAR278q_Fve4Gb6-0-FiDoyM6R1_DIzSD83D63kDbo8lhRxrHNgqZafwYVfA. [Lest: 13.05.21].

[7] Companies History. Garmin ltd. history, profile and history video, 2021. URL https://www.companieshistory.com/garmin/. [Lest: 06.05.21].

[8] DC RAINMAKER. Garmin connect iq: An in-depth introduction to the platform you can now use today, 2019. URL https://www.dcrainmaker.com/2015/01/connect-iq-intro.html. [Lest: 11.05.21].

[9] N. Donges. 4 reasons why deep learning and neural networks aren't always the right choice, 2019. URL https://builtin.com/data-science/disadvantages-neural-networks. [Lest: 09.05.21].

[10] GARMIN LTD. App types, 2021. URL https://developer.garmin.com/connect-iq/connect-iq-basics/app-types/. [Lest: 24.04.21].

[11] GARMIN LTD. Device reference, 2021. URL https://developer.garmin.com/connect-iq/reference-guides/devices-reference/#forerunner%C2%AE935. [Lest: 10.05.21].

[12] GARMIN LTD. Hello monkey c!, 2021. URL https://developer.garmin.com/connect-iq/monkey-c/. [Lest: 24.04.21].

[13] K. Goyal. Machine learning vs neural networks: What is the difference?, 2020. URL https://www.upgrad.com/blog/machine-learning-vs-neural-networks. [Lest: 13.04.21].

[14] Great Learning Team. What is regression? definition of regression?, 2020. URL https://www.mygreatlearning.com/blog/what-is-regression/. [Lest: 15.03.21].

[15] O. J. Halvorsen. Sjekk ditt eget pulsfall, 2021. URL https://www.dn.no/trening/sjekk-ditt-eget-pulsfall/1-1-1903376. [Lest: 08.05.21].

[16] IBM Cloud. Api vs. sdk: What's the difference? URL https://www.youtube.com/watch?v=kG-fLp9BTRo. [Lest: 06.05.21].

[17] J. Lutes. Entropy and information gain in decision trees, 2020. URL https://towardsdatascience.com/entropy-and-information-gain-in-decision-trees-c7db67a3a293. [Lest: 09.05.21].

[18] D. Muino. Hmfields, 2015. URL https://github.com/dmuino/HMFields. [Lest: 03.05.21].

[19] M. Mullie. Calories equivalent - a garmin connect iq data field, 2019. URL https://github.com/matthiasmullie/connect-iq-datafield-calories-equivalent. [Lest: 03.05.21].

[20] Norsk Helseinformatikk. Hjertebank og urolig hjerte, 2020. URL https://nhi.no/symptomer/hjerte-og-kar/hjertebank. [Lest: 05.03.21].

[21] Norsk Helseinformatikk. Åreforkalkning, aterosklerose, 2020. URL https://nhi.no/kroppen-var/sykdomsprosesser/areforkalkning/?page=all. [Lest: 05.03.21].

[22] A. Pant. Introduction to linear regression and polynomial regression, 2019. URL https://towardsdatascience.com/introduction-to-linear-regression-and-polynomial-regression-f8adc96f31cb. [Lest: 05.05.21].

[23] R. Reynoso. A complete history of artificial intelligence, 2019. URL https://www.g2.com/articles/history-of-artificial-intelligence. [Lest: 11.05.21].

[24] ritvikmath. Time series talk : Arma model [Video], 2019. URL https://www.youtube.com/watch?v=3UmyHed0iYE.

[25] ritvikmath. Time series talk : Autoregressive model [Video], 2019. URL https://www.youtube.com/watch?v=5-2C4eO4cPQ.

[26] ritvikmath. Time series talk : Autoregressive model [Video], 2019. URL https://www.youtube.com/watch?v=voryLhxiPzE.

[27] ritvikmath. Time series talk : Autoregressive model [Video], 2019. URL https://www.youtube.com/watch?v=HhvTlaNO6AM.

[28] Ø.. Sandbakk. Ny formel for beregning av makspuls, 2014. URL https://www.trening.no/utholdenhet/ny-formel-for-beregning-av-makspuls. [Lest: 08.05.21].

[29] StatQuest with Josh Starmer. Support vector machines part 1 (of 3): Main ideas!!! [Video]. URL https://www.youtube.com/watch?v=efR1C6CvhmE&t=744s.

[30] A. Strandheim. Hvordan påvirker fedme pulsen?, 2015. URL http://unikard.org/hvordan-pavirker-fedme-pulsen. [Lest: 08.05.21].

[31] M. Vetrhus and H. Arnesen. Claudicatio intermittens, 2019. URL https://sml.snl.no/claudicatio_intermittens. [Lest: 08.05.21].

[32] M. West. Explaining recurrent neural networks, 2020. URL https://www.bouvet.no/bouvet-deler/explaining-recurrent-neural-networks. [Lest: 13.04.21].

[33] T. Wiktorski, M. F. Bjørkavoll-Bergseth, and S. Ørn. Data-centered analysis of physiological metrics during a strenuous physical exercis, 2015. URL 06-biomedts-data-centered-analysis.pdf. Upublisert.

[34] Will. Decision tree flavors: Gini index and information gain, 2016. URL http://www.learnbymarketing.com/481/decision-tree-flavors-gini-info-gain/. [Lest: 09.05.21].

[35] A. Wolfewicz. Deep learning vs. machine learning - what's the difference?, 2021. URL https://levity.ai/blog/difference-machine-learning-deep-learning. [Lest: 11.05.21].

[36] K. Ytrehus. Aterosklerose, 2020. URL https://sml.snl.no/aterosklerose. [Lest: 05.03.21].

Vedlegg A Kode

A.1 Nøkkelinfo

```
file = open("distDT.txt", "r")

lines = file.readlines()

info_tuples = []
for line in lines:
    strip_line = line.strip()

    # GRUPPER I LISTE, SE ETTER HOYDE BASERT PA |---
    split_by_space = strip_line.split(" ")

    # Get the actual information
    if 'weights' in strip_line:
        idx = split_by_space.index('weights:')
        sep = " "
        info = sep.join(split_by_space[idx : len(split_by_space)])
        print(info)
        info_tuples.append(info)
    else:
        list_idx = None
        if split_by_space[-3].isnumeric():
            list_idx = split_by_space[-3]
        else:
            list_idx = split_by_space[-4]
        value = split_by_space[-1]
        print("Value : " + value + ", Index: " + list_idx)
        info_tuples.append((int(float(list_idx)), float(value)))
```

kode/nokkelinfo.py

A.2 Finne høyde

```
file = open("distDT.txt", "r")

lines = file.readlines()

tuples = []
```

```
for line in lines:
    strip_line = line.strip()

    # GRUPPER I LISTE, SE ETTER HOYDE BASERT PA |---
    split_by_space = strip_line.split(" ")

    height_counter = 0
    for string in split_by_space:
        if string == '|---':
            # Store the lines of string together with their height as tuples
                # Add to a list to keep track
            if height_counter == 0:
                tup = (strip_line, height_counter)
                tuples.append(tup)
            else:
                node_height = int(height_counter / 3)

                tup = (strip_line, node_height)
                tuples.append(tup)
            break;
        else:
            height_counter += 1
```

kode/finn_hoyde.py

A.3 Danne par

```
def find_index(list, start_index, cur_height):
    for i in range(start_index+1, len(list)):
        if(list[i][1] == cur_height):
            return i

# For each tuple, find it's pair if it has one
    # Does not have one if the string contains 'weights'
checked_indexes = []
pairs = []
for index, tup in enumerate(tuples):
    streng = tup[0]
    hoyde = tup[1]

    # SOK ETTER WEIGHTS I STRENGEN
    if 'weights' in streng:
        # BLAD NODE, GJOR NOE MED DEN..
```

```
17          #print(streng)
18          print("WEIGHT at index: " + str(index))
19      else:
20          # VANLIG NODE, LAG IF ELSE FUNKSJON
21
22          # Find the pair-node
23          if index not in checked_indexes:
24              first_node = index
25              second_node = find_index(tuples, index, hoyde)
26
27              # Mark the indexes as checked
28              checked_indexes.append(first_node)
29              if second_node != None:
30                  checked_indexes.append(second_node)
31
32              print("The node at index: " + str(first_node) + " has a pair at index: "
    + str(second_node))
33
34              pairs.append((first_node, second_node, hoyde+1))
```

kode/dann_par.py

A.4 Lage funksjonen

```
1  # 1: Build a dictionary with all the information needed
2  # 2: Start from the back of the list
3      # Build each if / else statement
4          # Use the dictionary to know what to place inside the if / else
5  function_text = {}
6  for i in range(len(pairs)-1, -1, -1):
7      # Find the values of our nodes
8      first_index = pairs[i][0]
9      second_index = pairs[i][1]
10     list_index = info_tuples[first_index][0]
11     value = info_tuples[first_index][1]
12
13     # Get the height for tab-usage
14     height = pairs[i][2]
15
16     # Find the values of the children nodes
17     if_child = info_tuples[first_index+1]
18     else_child = info_tuples[second_index+1]
19
```

```
20      # Write if / else statement
21      # IF
22      func_text = ""
23      func_text += "\t"*height
24      func_text += "if (list["+str(list_index)+"] <= "+str(value)+"){\n"
25      if 'weights:' in if_child:
26          func_text += "\t"*(height+1)
27          func_text += "System.println(\""+if_child+"\");"
28          func_text += "\n"
29      else:
30          #func_text += "\t"*(height+1)
31          func_text += function_text[first_index+1]
32          func_text += "\n"
33      func_text += "\t"*height
34      func_text += "}"
35      # ELSE
36      func_text += "else {\n"
37      if 'weights:' in else_child:
38          func_text += "\t"*(height+1)
39          func_text += "System.println(\""+else_child+"\");"
40          func_text += "\n"
41      else:
42          #func_text += "\t"*(height+1)
43          func_text += function_text[second_index+1]
44          func_text += "\n"
45      func_text += "\t"*height
46      func_text += "}"
47
48      # Save node in the dictionary
49      function_text[first_index] = func_text
50
51  # Build the base of the function
52  func_text = "function decTree(list){\n"
53  func_text += function_text[0]
54  func_text += "\n}"
```

kode/lag_funksjon.py

A.5 Skriv til applikasjon

```
1  # WRITE TO A NEW .MC FILE AND PUT INSIDE THE APP
2  # REBUILD
3  def bygg_fil(prosjekt_location, project_name, key, text_to_write):
```

```
    # Change to directory
    dir = project_location + project_name + "/resources/data"
    os.chdir(dir)

    # Write to file, overwrites the one that is there
    file = open("DecisionTree.mc", "w")
    file.write(text_to_write)
    file.close()

    os.system(
        "monkeyc -y " + key + " -f " + project_location+project_name+"/monkey.jungle
    " + "-o " + project_location+project_name+"/bin/"+project_name+".prg"
    )

# File locations
project_location = "C:/Users/magnu/eclipse-workspace/"
project_name = "SeveralDataFields"
key = "C:/Users/magnu/keys/developer_key"

# Make it into a class
class_text = "class DecisionTree {\n"
class_text += "\t" + func_text
class_text += "\n}"

bygg_fil(project_location, project_name, key, class_text)
```

kode/skriv_til_app.py

Vedlegg B Kildekode

B.1 GitHub

Hele programmet og med annen relevant data og informasjon er laster opp til GitHub.

https://github.com/EirikHvitsten/Bachelor