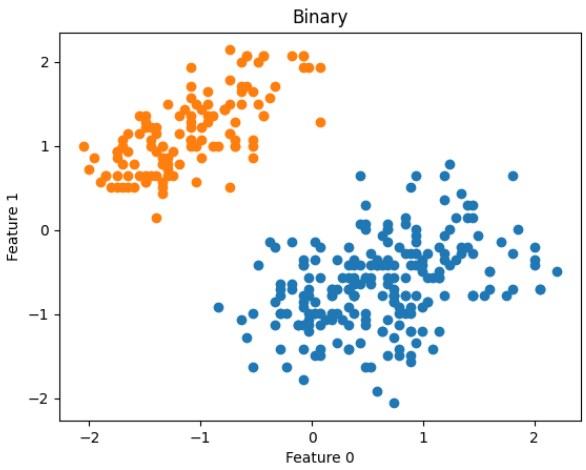
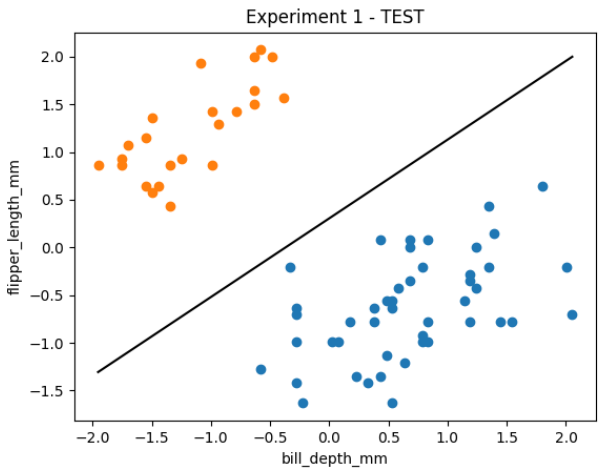
14.11.2024

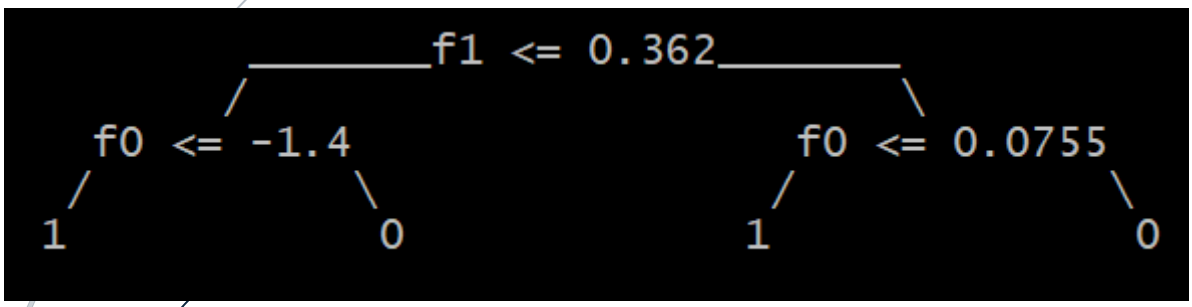
Supervised Learning

Karaktersatt oppgave 2

DTE2602 – Introduksjon Maskinlæring og AI







Eirik Tennøfjord
STUDENT

Innhold

1. Introduksjon

I dag er hører vi stadig oftere om at AI er i ferd med å ta over mange forskjellige jobber. Vinklingen på hvordan dette vil slå ut samfunnsmessig er ofte varierende. Det er ingen hemmelighet at de store automatiserte omveltningene gjør utslag på hvordan samfunnet utvikler seg. Bare innenfor programmering ser vi at tendensen øker når det gjelder å lage kode [1].

Spesielt med den utviklingen som har vært innenfor vision-system, så har en del av produktiviteten innenfor deler av industrien økt. Et eksempel på dette kan være sortering av fisk. I disse tilfellen bruker man et «kamera», som da kan sortere basert på hvilken art det er, samt anslå størrelse på fisken [2].

I denne rapporten skal vi se på noen metoder man kan benytte innenfor supervised learning, som handler om klassifisering. Når vi snakker om supervised learning, referer vi til en metode innenfor maskinlæring, som bruker innsamlede data til å lære opp en algoritme til å eksempelvis kategorisere [3].

Videre vil vi i denne rapporten undersøke hvordan man ved å benytte ulike algoritmer, kan klassifisere tre forskjellige pingvinarter, basert på et datasett som viser art, øy, nebb-lengde, nebb-dybde, vinge-lende, kroppsvekt og kjønn. I rapporten vil vi fokusere på numerisk data (nebb-lengde, nebb-dybde, vinge-lengde og kroppsvekt).

En av de metodene vi vil teste for å kunne klassifisere hvilken pingvinart vi har data om, er beslutningstrær. Dette er en metode som bruker spørsmål til å skille data fra hverandre. Eksempelvis «Er vinge-lengden over eller under 185mm?».

Den neste metoden vi skal se på i denne rapporten er perceptron. Perceptron brukes for å lineært separere to klasser, i vårt tilfelle skille arter fra hverandre.

I denne rapporten vil vi se nærmere på hvordan de forskjellige egenskapene til perceptron og beslutningstrær kan variere fra lineært separerbare datasett, og de som ikke er lineært separerbar. Videre vil vi etter hvert sammenligne resultater for de forskjellige klassifiseringsmetodene.

2. Teori

Det finnes mange typer maskinlæring, men som regel kan vi dele disse inn i tre hoveddeler. Disse hoveddelene er:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Grovt sett er supervised learning en metode å lære basert labeled/merket datasett. Her har hvert av datasettene et «riktig svar», som man kan sjekke opp mot. Algoritmen prøver å finne sammenheng mellom data som kommer inn, og data som blir sendt ut [6].

Unsupervised learning er læring uten label/merking. Her vil man prøve å finne mønster/struktur uten at man å vite hvordan mønsteret eller gruppene er [6].

Reinforced learning benytter en agent som gjennom interaksjon med et miljø, prøver ulike handling. Når en handling er gjennomført vil man få en belønning eller straff. Ved å benytte denne formen vil man prøve å samle opp mest belønning over tid [6].

I denne rapporten vil vi gå videre inn på supervised learning.

2.1 – Supervised Learning

Supervised learning er er en type maskinlæring som lærer seg mønster eller strukturer basert på datasett med tilhørende labels. Med labels mener vi en riktige svar. Ved å benytte et datasett som er labeled/merket vil vi kunne lære opp systemet til å forutsi utgangen, når den blir presantert med ukjent data.

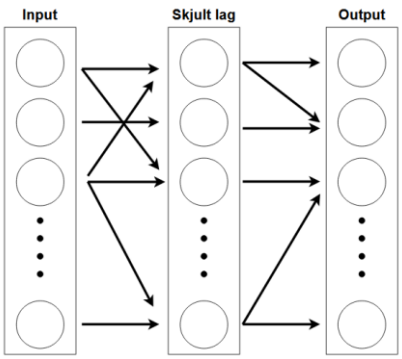
I supervised learning er det derfor viktig at vi kan trene opp modellen vår, slik at man med ulike inngangsdata kan lære seg hvordan utgangene skal være.

Med inngangsdata mener vi features/egenskaper i datasettene, slik som tidligere nevnt med pingviner som har forskjellige nebb-størrelse og vekt. Ved å benytte datasett for allerede innsamlet data, hvor man kjenner output (eksempelvis pingvinart), så vil man kunne lære opp systemet.

Det er også viktig å kunne teste at modellen fungerer, derfor bør man ikke sende alle datasettene inn som trening, men holde tilbake en del, for så å teste den trente modellen. For å splitte data mellom trening og testing er det ofte brukt et forhold på 80%/20% [4].

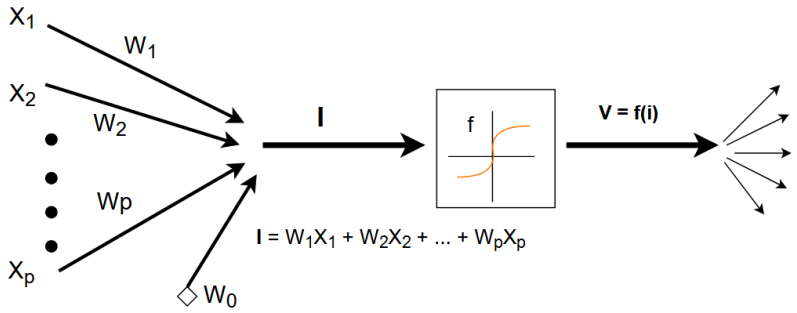
2.2 – Nevrale nettverk

Den mest vanlige metoden innenfor supervised learning, er det vi kaller for nevrale nettverk. Dette er en type AI, som prøver å etterligne den menneskelige hjernen. Den er bygd opp av flere nevroner, som er sammenkoblet i et nettverk. Ved å justere på vekter som ligger i nettverket mellom nevronene, så kan man få det modellerte nettverket til «å lære». Kunnskapen ligger lagret i selve vektene mellom nevronene.



Bilde 2.2.1 – Eksempel på hvordan et nevralt nettverk kan være sammenkoblet.

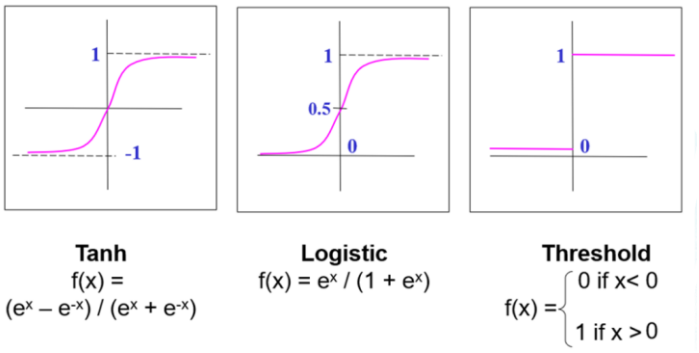
I bildet over ser vi eksempel på hvordan et nevralt nettverk kan være sammenkoblet. Her har vi et input lag, et skjult lag og et output lag. Hver sirkel representerer et nevron, og hver pil representerer en sammenkobling mellom nevroner. Det er altså vektingen på disse sammenkoblingen som får et nevralt nettverk til å lære. Vi får inn en eller flere input verdier, som ved hjelp av vekting kan tippe resultatet [5].



Bilde 2.2.2 – Hvordan et nevron kan være satt opp.

I bildet over ser vi hvordan inputene fra X blir justert med vektingene fra W. Til sammen danner disse verdien I, som blir benyttet i aktiveringsfunksjonen som er tegnet som en firkant i tegningen over. Ut ifra aktiveringsfunksjonen få vi V, som er resultatet aktiveringsfunksjonen. I tegningen over kan vi se at det også er tegnet inn en «W0», dette er en egen vekting som vi kaller bias. Denne kan gjøre modellen mer fleksibel. Bias kan til en viss grad påvirke skjæringspunktet. Bias kan forskyve verdiene I, slik at man er mer gunstig når man benytter threshold funksjonen. Dersom I er for stor kan bias være negativ, dersom I er for liten så kan bias være positiv. Bias blir også lært opp, på samme måte som de andre vektene [7].

2.3 – Aktiveringsfunksjon



Bilde 2.3.1 – Hentet fra UiT slide. Viser tre vanlige aktiveringsfunksjoner.

Her er en oversikt over noen av de mest vanlige aktiveringsfunksjonene. Tanh kan returnere verdier mellom -1 og 1. Logistic (sigmoid) kan returnere verdier mellom 0 og 1. Threshold funksjonen er som en if/else funksjon, dersom den er mindre enn 0, så vil den returnere «0», og dersom den er 0 eller større, så vil den returnere «1». I denne rapporten vil vi fokusere på en enkel threshold funksjon [5].

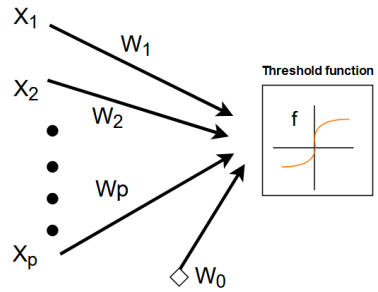
For å justere vektene er dette vanskelig å gjøre manuelt, derfor er det mest effektivt om man benytter multivariabel ikke-linær regresjon, også kjent som «least-squares method». Her vil man da måle forskjellene mellom forventet svar og faktisk svar. For å måle feilen benytter vi:

$$E = \sum(Y_i - V_i)^2$$

Feilen blir da summen av faktisk svar minus forventet svar i andre. Ved å benytte denne feilen kan vi da justere vektene på bakgrunn av dette. Y i formelen over er typisk en vektor med labels/fasit [5].

2.4 – Perceptron

Perceptron er den enkleste formen for nevrale nettverk. Når vi snakker om en perceptron er dette et nevralt nettverk, som inneholder kun ett nevron. Dette var grunnlaget for mye av utviklingen innenfor AI. En perceptron kan man bruke for å klassifisere data mellom to ulike klasser. Her vil man da benytte aktiveringsfunksjonen «threshold». Denne typen virker best på lineært separerbar data, som vil si dersom vi hadde hatt et 2D plot, så ville man kunne se hvor man kunne strakt en linje, for å skille mellom forskjellige klasser [5].



Bilde 2.4.1 – Threshold funksjon i perceptron

For justering av vekter i en perceptron gjør vi følgende. Dersom vi har et visst antall X-verdier, og et gitt antall W-verdier. Her er X input data, og W er vektingene. Vi kan benytte følgende formel:

$$W[i] = W[i] + a * (fasit - V) * X[i]$$

I denne formelen beregner vi en ny vekt for «W[i]», vi benytter en fasit som sier hva resultatet skal være, V er det resultatet perceptron har «tippet» at det er, X[i] er input data, og a er læringsraten som skal ligge mellom 0 og 1. For å finne en ny vekting tar vi utgangspunkt i den eksisterende verdien til W[i], deretter legger til læringsraten * (fasit – tippet resultat) * Input verdien. Ved å justere på størrelsen til læringsraten, så vil vi kunne justere hvor fort perceptron lærer [5].

2.5 – Gini Impurity

Gini impurity (GI) er et matematisk verktøy som hjelper til med å bygge gode beslutningstrær. For å kunne bygge gode beslutningstrær, må man vite noe om hvor god sannsynligheten er for at man har riktig. Dette kan man gjøre ved hjelp av GI. Ved å beregne hvor høy sannsynligheten er for å kunne gjette riktig, så kan vi snu om på dette å presentere hva sannsynligheten er for å gjette feil.

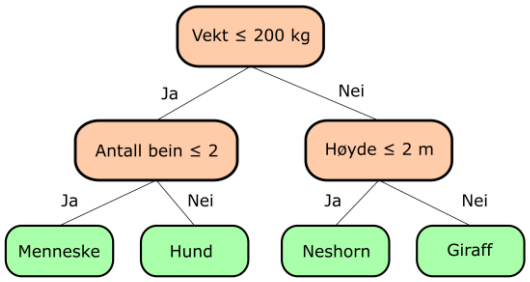
$$GI = 1 - \sum_{i} P_i^2$$

Her er GI, gini impurity, P er andelen av eksempler som tilhører en et unikt element vi vil klassifisere. For å finne hvor stor sannsynligheten er for å gjette riktig kan vi summere opp antallet for hver unik klasse vi vil klassifisere, og dele på det totale antallet med datasett, og til slutt opphøye i andre. Da vil vi kunne få en måling på hvor urent det er.

2.6 – Beslutningstrær

Beslutningstrær er en type metode hvor man starter i rot noden, og deretter stiller spørsmål for å splitte datasettene. Når disse datasettene blir splittet, så vil man lage nye noder/branches i beslutningstreet. Nodene som inneholder spørsmål, og splitter datasettene kaller vi grennoder.

Dersom beslutningstreet er ferdig med klassifiseringen, og har ikke flere spørsmål, så blir denne noden kallet for løvnode (leafnode).



Bilde 2.6.1 – Hentet fra jupyter notebook filen til karaktersatt oppgave 2

I bildet over kan vi se hvordan man ved hjelp av beslutningstrær kan skille de forskjellige klassene. Vi starter i rot noden, og stiller spørsmål om vekt, da blir datasettene splittet i to. For det ene settet stiller vi spørsmål om høyde, som resulterer i to løv noder (neshorn og giraff). For det andre datasettet stiller man spørsmål om antall bein, og da klarer man å skille mellom to klasser til, som genererer to løv noder (menneske og hund)

3. Metode

For å utvikle modeller for klassifisering har vi benyttet flere forskjellige metoder innenfor supervised learning. Dette inkluderer perceptron og beslutningstrær. Selve arbeidet er gjennomført ved å benytte Python, og ved hjelp av pyplot (matplotlib) biblioteket og binarytree har man kunnet visualisere en del av de resultatene rapporten har kommet frem til.

I denne rapporten er det laget klart for fem forskjellige eksperimenter. I koden kan man velge mellom de forskjellige eksperimentene ved hjelp av input i python.

```
Experiments that can be simulated:

1. Perceptron: Separate Gentoo from the other two species.
   Features to be used: bill_depth_mm and flipper_length_mm. Show plot and accuracy.

2. Perceptron: Separate the Chinstrap species from the other two.
   Features to be used: bill_length_mm and bill_depth_mm. Show plot and accuracy.

3. Create a decision tree to separate the Gentoo penguin species from the other two.
   Features to be used: bill_depth_mm and flipper_length_mm. Measure accuracy and visualize the decision tree.

4. Create a decision tree to separate the Chinstrap species from the others.
   Features to be used: bill_length_mm and bill_depth_mm. Measure accuracy and visualize the decision tree.

5. Repeat the experiments several times with random shuffling and splitting.
   Each time, create a decision tree based on all 4 features in the dataset to distinguish between all three species.
   Measure accuracy.
```

Bilde 3.0.1 – Hvordan man kan velge eksperiement i programmet.

3.1 – Datasett

For innlesing av datasett i forsøk med perceptron og beslutningstrær ble filen «palmer_penguins» lest inn. Deretter måtte man renske filen for ukomplett data, siden flere av radene var merket «NA». Videre ble de datasettene (X) vi ønsket for de forskjellige eksperimentene hentet ut, sammen med label (y-marise). De datasettene som er benyttet i denne rapporten er de numeriske, samt label som var navnet på pingvin arten. Etter dette ble datasettet (X) normalisert ved hjelp av z-score, som da vil sentrere data rundt 0, og gi et standardavvik på 1. Til slutt ble label (y) konvertert til index for å representere pingvinartene.

For perceptron ble label-matrisen (y) også konvertert til et binært sett (0 og 1).

Etter dette ble datasettet splittet om til et treningssett og et testsett. For å dele opp disse benyttet man i denne rapporten 80/20 prinsippet, som skal være en enkel pekepinne for hvordan man kan dele opp datasettet. Det samme ble gjort for label (y) [4].

- X_train: Inputdata som blir benyttet til trening
- y_train: Fasiten til inputdata til trening
- X_test: Inputdata som blir benyttet til testing etter treng modell.
- y_test: Fasiten til inputdata til testing

3.2 – Perceptron

Når datasettene er splittet opp, så kan vi begynne treningsprosessen for perceptron klassen. For å gjøre dette så sender vi inn treningsdataen som vi laget til i datasettet, samt y-matrisen. Da vil perceptron klassen trenes opp basert på disse datasettene, og videre kunne justere de forskjellige vektene basert på om man gjetter riktig eller feil. For å kunne trene perceptron på en god måte, bruker man også epoker for å simulere flere ganger. Dersom man ikke har noen feil, så vil man konkludere med at perceptron har konvergert, altså at den har en «god nok» løsning.

For å sjekke om modellen faktisk fungere så blir denne i ettertid testet ved å sende inn X-test og y-test fra datasettet vi splittet opp. For å sjekke hvor godt funksjonen fungere er det implementert en funksjon som heter «accuracy», som sjekker hvor mange korrekte man har fra test-settet.

3.3 – Beslutningstrær

For beslutningstrær blir det gjennomført den samme datasplittingen som for perceptron, bare at her trenger man ikke å gjøre om y-data til binært (0 eller 1). Her kan man beholde indexene for alle tre klassene (0-2).

For å trene opp beslutningstreet sender man inn på samme måte som perceptron treningsdata (x_train og y_train). Ved hjelp av disse trener beslutningstreet seg på hva som er det beste spørsmålet å stille (funksjonen best_split_feature_value). Da vil beslutningstreet bruke en slags brute force, for å finne de spørsmålene som gir best total gini impurity.

For å visualisere det ferdige beslutningstreet kan vi bruke print funksjonen på selve «DecisionTree» klassen. Da vil man få ut en trestruktur, som viser hvilken index egenskapen man skal bruke i spørsmålet sitt, samt hvilken verdi man skal bruke.

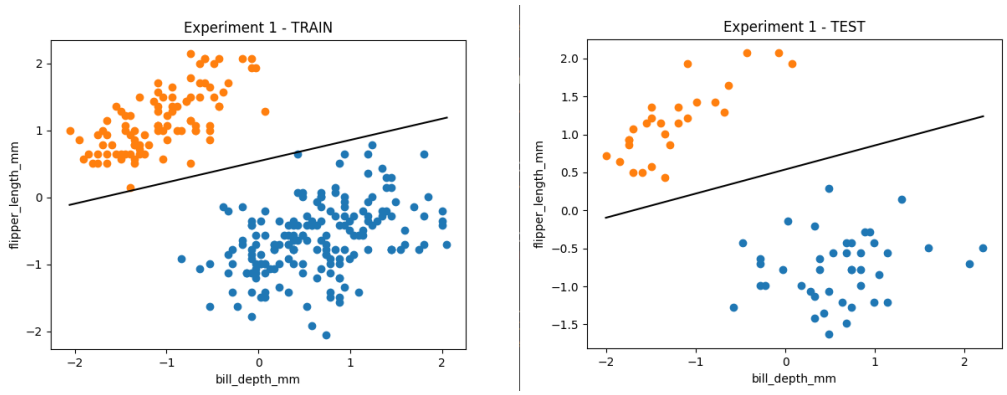
Også her bekrefter man hvor effektivt det er ved å bruke testdata (X_test og y_test) for å generere en accuracy verdi, som sier hvor effektiv modellen er.

4. Resultat

4.1 - Eksperiment 1:

I det første eksperimentet skal vi benytte perceptron til å prøve å skille ut en av artene fra de to andre. Vi leser inn data fra «palmer_penguins.csv». Deretter splitter vi datasettet som beskrevet i metode kapittelet. I dette eksempelet vil vi skille arten «Gentoo» fra de to andre. Egenskapene (features) vi bruker for dette eksperimentet er «bill_depth_mm» og «flipper_length_mm». For å

visualisere resultatet har vi laget til en «decision boundry» linje for å illustrere hvordan man skiller klassene.

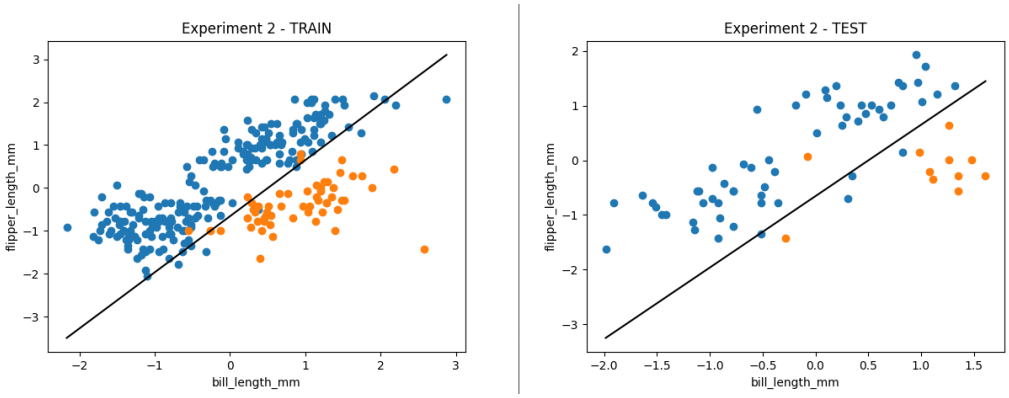


Bilde 4.1.1 – Viser hvordan treningsdata og testdata blir linjert skilt i eksperiment 1.

I dette eksperimentet kan vi se ut ifra bildet at det skal være godt mulig å skille ut den ene klassen ved å tegne en lineær linje. I dette eksperimentet ble i utgangspunktet satt til 1000 epoker, men det konvergerte lenge før hver gang. I dette eksperimentet brukte vi 80/20 regelen for datasplitt, og en bias på 0. Nøyaktigheten varierte mellom 0.96 og 1.0.

4.2 - Eksperiment 2:

I andre eksperiment skal vi prøve å skille arten «Chinstrap» fra de to andre, ved å benytte egenskapene «bill_length_mm» og «bill_depth_mm». Metoden var også her perceptron.

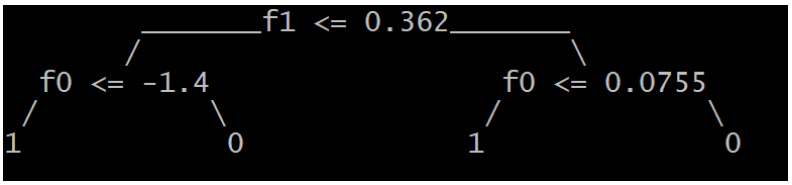


Bilde 4.2.1 – Viser hvordan treningsdata og testdata blir linjert skilt i eksperiment 2.

I eksperiment 2 kunne vi på forhånd se at det blir mer utfordrende. Her er det mye vanskeligere å skille de to settene ved å tegne en lineær linje. I eksempelet over er nøyaktigheten på 0.97, men testene varierer fra 0.80 til 0.97. I dette eksperimentet ble det benyttet en bias på 1, og 1000 epoker. I dette eksperimentet ble det nesten aldri konvergering, og man måtte kjøre gjennom alle epokene. For trening og test, så ble det også her benyttet 80/20.

4.3 - Eksperiment 3:

I dette eksperimentet skulle vi i stedet for å benytte perceptron bruke beslutningstrær for å skille klassene. I dette eksperimentet skal vi prøve å skille «Gentoo» fra de to andre artene ved hjelp av egenskapene «bill_depth_mm» og «flipper_length_mm».

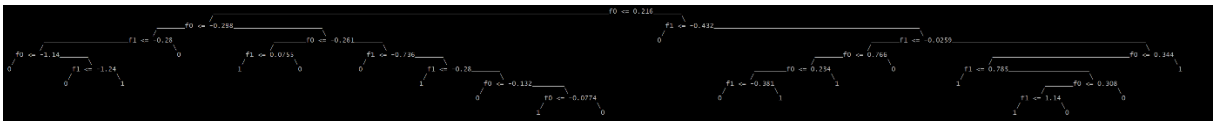


Bilde 4.3.1 – Viser Decision Tree for eksperiment 3.

Vi kan se ut ifra bildet over så er spørsmålet fra index 1 det beste å stille når man er i root. Verdien man vil sjekke er 0.362. Videre ser vi at i dette tilfellet blir de splittet videre til to grennoder, før man til slutt ender opp med fire løv noder. I dette eksperimentet fikk man som regel en accuracy på 1.0, som er 100% korrekt. Også her ble det benyttet 80/20 for datasettet.

4.4 - Eksperiment 4:

I dette eksperimentet skal vi se hvor effektivt vi klarer å skille «Chinstrap» fra de to andre artene ved å benytte egenskapene «bill_lenght_mm» og «bill_depth_mm».


Bilde 4.4.1 – Viser Decision Tree for eksperiment 4.

Bildet over er noe utydelig, men det viser allikevel at det blir noe mer komplisert enn forrige eksperiment. Her er det mange flere spørsmål involvert for å generere gode beslutningstrær. Nøyaktigheten i forsøkene ligger mellom 0.90 og 0.96. Datasettet er splittet 80/20.

4.5 - Eksperiment 5:

I det siste eksperimentet vil vi gjøre forsøkene mange ganger for å finne en total statistikk, som viser den totale nøyaktigheten. I dette eksperimentet vil vi benytte alle de fire numeriske featurene som er i datasettet, samt skille mellom alle de tre artene. Også her ble det benyttet et 80/20 split på datasettet. For å få et nokså godt gjennomsnittlig resultat, så ble det kjørt 1000 simuleringer.

```
The average is: 0.9622318840579588
```

Ved tusen simuleringer fikk vi en nøyaktighet på ca. 0.96

5. Diskusjon og konklusjon

I det første eksperimentet kunne vi lett se ut ifra plottet som ble vist at man skal kunne tegne en lineær linje, for å skille de to klassene. Dette førte til at testingen som regel konvergerte før alle epokene var gjennomført. Samtidig var den ikke alltid 100% nøyaktig, som som skyldes variasjon i hvilken datapunkter som ble benyttet til trening og testing.

Forsøk nummer to ble mer utfordrende for perceptron metoden. Her var det ikke mulig å tegne en lineær linje som skilte den ønskede arten fra de andre. Dette gjorde at treningen aldri konvergerte, som resulterte i at man hele tiden ville kjøre gjennom alle epokene. Det også noe feil i alle testene som ble gjennomført og ingen hadde av testene hadde 100% korrekt.

For forsøk nummer tre, som var med beslutningstrær, var nøyaktigheten som regel 1.0 (100%). Dette skyldes at ved de egenskapene som lå til grunn, enkelt kunne skille artene fra hverandre. I de fleste resultatene var det én root node, to gren noder og fire løv noder. I printen kunne man se hvilken feature index som var brukt for hvert spørsmål, samt hvilken verdi som ble brukt for å splitte opp datasettet.

I eksperiment nummer 4 kunne vi se at beslutningstreet hadde oftere en høyere nøyaktighet enn hva perceptron hadde. Perceptron hadde store variasjoner i nøyaktigheten, noe som er avhengig av hvilken datapunkter som blir benyttet for test og trening. Når perceptron varierte fra 0.80 til 0.97, så lå beslutningstreet mellom 0.90 og 0.96 i nøyaktighet. Dette er nok fordi man ikke klarer å lineært separere artene ved å benytte perceptron.

I det siste eksperimentet ser vi på hvordan resultatet for et beslutningstre er ved å bruke mange simuleringer. Dette gir en bedre nøyaktighet for målingene. I dette eksperimentet fikk man en nøyaktighet på 0.96, når man brukte alle egenskapene, samt skilte mellom alle de tre artene. Dette gir et godt bilde på hvordan man kan bruke et beslutningstrær til å skille mellom flere ulike klasser, noe som ikke er mulig ved perceptron, siden man ikke kan skille alle tre artene ved hjelp av én lineær linje.

Videre arbeid for dette eksperimentet kan være:

- Kunne laget til en rekursiv meny, så man slipper å kjøre python scriptet på nytt for hver gang.
- Kunne lagt til flere aktiveringsfunksjoner. Det ville vært spennende å se hvilke resultat de forskjellige aktiveringsfunksjonene kunne gitt.
- Hadde også vært spennende å prøvd ut SVM (Support Vector Machine), for å se hvordan denne kunne klassifisere de ulike klassene.

6. Kilder

[1] – Stokke, Ole: En fjerdedel av koden hos Google blir nå AI-generert. Hentet 23. November fra https://www.kode24.no/artikkel/en-fjerdedel-av-koden-hos-google-blir-na-ai-generert/82156162

[2] – Optimar. Hentet 23. November fra https://optimar.no/solutions/product/species-recognition

[3] – Google. Hentet 23. Novemer fra https://cloud.google.com/discover/what-is-supervised-learning

[4] – *Ahmed*: The Motivation for Train-Test split. *Medium*. Hentet 23. November 2024 fra https://medium.com/@nahmed3536/the-motivation-for-train-test-split-2b1837f596c3

[5] - UiT Norges Arktiske Universitet (2024, 23. November) Nevrale nettverk og perceptrons – Del 1 Introduksjon. Canvas. https://uit.instructure.com/courses/34928/modules/items/1026565

[6] – Kumar, Sendeep: Supervised vs Unsupervised vs Reinforcement. Aitude. Hentet 24. November 2024 fra https://www.aitude.com/supervised-vs-unsupervised-vs-reinforcement/

[7] - UiT Norges Arktiske Universitet (2024, 23. November) Torsdagsmøte 20241107. Canvas. https://uit.instructure.com/courses/34928/modules/items/1088128