

1 Introduksjon

Maskinlæring (ML) handler om å finne mønster i data som er nyttige for å løse praktiske problemstillinger, som Grokking [1, kap. 8] deler inn i tre hovedkategorier; *Supervised Learning*, *Unsupervised Learning* og *Reinforcement Learning*. Her beskrives førstnevnte som «one of the most common techniques in traditional machine learning», altså er slike teknikker en velbrukt tilnærming i ML. Det finnes mange ulike algoritmer innen *Supervised Learning*, som *Decision Trees*, *K-Nearest Neighbor*, *Artificial Neural Networks*, *Support Vector Machines*, *Logistic Regression*, *Lasse Regression* og *Naïve Bayes*, for å nevne noen [1], [2]. Kotsiantis [2] viser at flere av disse kan brukes både for regresjons- og klassifiseringsoppgaver, men er gjerne mest egnet for én av dem. Hver metode har styrker og svakheter, og vil være bedre egnet for noen typer data og problemstillinger enn andre.

For å skape gode maskinlæringsmodeller må vi altså være godt kjent med egenskaper ved ulike metoder, algoritmer og tilnærninger til ML. I denne oppgaven skal vi se nærmere på to typer algoritmer for veiledet læring, *Decision Tree* (DT) og *Perceptron*, og hvordan disse kan brukes i klassifisering. Algoritmene skal implementeres fra grunnen av, kun med støtte fra grunnleggende biblioteker for datahåndtering og visualisering.

De nevnte algoritmene skal trenes og testes på datasettet *palmerpenguins* [3], som inneholder observasjonsdata for tre ulike pingvinarter. Målet med modellen er at den knytter riktig klasse (art) opp mot andre egenskaper, ved hjelp av andre egenskaper i datasettet. Gjennom denne øvelsen skal vi ta for oss hvordan disse algoritmene er bygd opp, ulike egenskaper ved dem og hvordan de presterer under praktisk anvendelse på et reelt datasett.

2 Teori

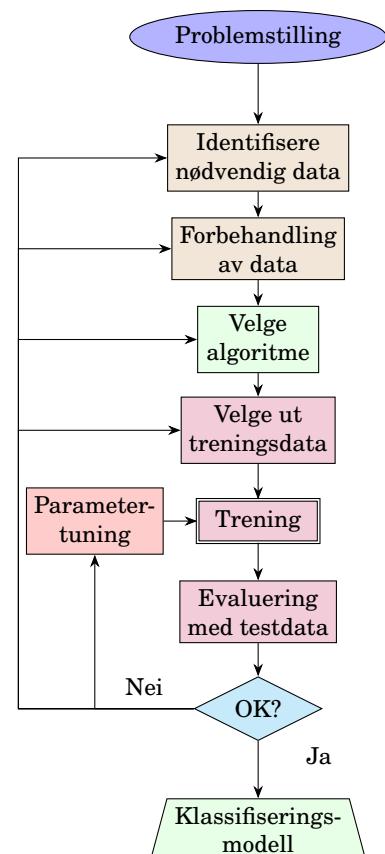
2.1 Veiledet læring

Det som skiller *Supervised Learning*, eller veiledet læring, fra andre tilnærninger til ML, er at i veiledet læring brukes kjente sammenhenger mellom egenskaper (*features*) og en målverdi (*label*) [2, s. 249]. Målverdiene kan være av kategorisk eller kontinuerlig art, som henholdsvis gir regresjons- eller klassifiseringsproblemer. Maskinlæringsalgoritmen bruker det kjente datasettet til å danne et sett regler som kobler inndata, egenskaper, til utdata, målverdiene. Dette er en prosess som i vitenskapsteorien er kjent som *induksjon*, altså at man danner generelle modeller på bakgrunn av enkeltobservasjoner. I motsetning til *deduktive* slutninger er ikke en *induktiv* slutning logisk bindende, men uttrykker en sannsynlighetsovervekt [4]. Dette gjelder videre for veiledet læring, altså vil reglene som dannes aldri kunne uttrykke absolutt kunnskap, og kvaliteten til modellen vil være tett knyttet til datagrunnlaget.

I en induktiv erkjennelsesprosess er vi avhengig av å basere slutningene våre på riktig data. Det vil si at datagrunnlaget må kunne gi informasjon om et generelt tilfelle innen ønskede rammer, og de utvalgte egenskapene vi måler må evne å svare på den gitte sammenhengen. Med andre ord må data være representativ og det må være en korrelasjon mellom egenskaper og målverdi. Om vi ønsker å lage en modell over hvilken mat folk liker, er det kanskje lite nyttig å se på personers høyde eller øyenfarge. Og om vi ønsker at modellen skal være generell for alle mennesker, nytter det ikke å basere seg på data fra en begrenset aldersgruppe eller geografisk lokasjon. Om det er en kausal sammenheng mellom egenskap og målverdi er imidlertid ikke like avgjørende. Kausale sammenhenger vil selvfølgelig skape sterke modeller, men utgangspunktet er gjerne at vi ikke kjenner til slike sammenhenger på forhånd. Det er her både styrken og svakheten til induktive prosesser ligger, nemlig at vi ikke trenger å kjenne kausalitet for å tilegne oss kunnskap gjennom logisk deduksjon, men kan bruke erfaringsbasert kunnskap fra observasjoner til å skape modeller som kan estimere informasjonen vi er ute etter.

En styrke i veiledet læring, er at vi kombinerer vår logiske og erfaringsbaserte kunnskap med maskinens evne til å finne mønster. Vi velger altså på forhånd ut et sett med data som vi har grunn til å tro kan inneholde mønster som er av generalisierbar verdi. Vi dumper ikke tilfeldig data i en «algoritmekvern» og håper på det beste, men bruker tid på å velge ut et passende datasett med egenskaper som har potensielle til å gi gode resultater. En kan ta en *brute force* tilnærming, der en samler og sammenligner tilfeldig valgt data og håper på det beste. Dette vil ikke gi et godt grunnlag for veiledet læring, og kreve mye tid til håndtering av støy, pre-prosessering av data og maskinressurser for å finne kandidater til en datamodell [2, s. 250].

Som figur 1 viser, er flere av stegene i veiledet læring «manuelle» og relatert til datautvelgelse, databehandling og preprosessering. Trening, evaluering og tuning av algoritmen er bare en liten del prosessen. Fremgangsmåten bygger som nevnt helst på kunnskap og forståelse av datasettet, men prosessen er også eksperimentell. Når én modell har blitt trent og evaluert, brukes denne *feedbacken* til å ta et valg. Hvis vi ser at modellen presterer svakt, må det vurderes å gå ett eller flere steg tilbake i prosessen, illustrert av pilene på venstre side i figuren. Må hyperparametre justeres, egner algoritmen seg dårlig til å løse problemstillingen,

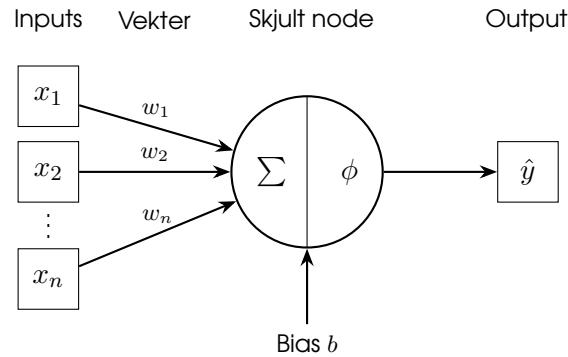


Figur 1: Veiledet læring [2, s. 250]

er kvaliteten på datasettet svakt, eller må vi velge andre egenskaper? Først når denne prosessen har ledet til en god modell med akseptabel evne til å gjøre gode prediksjoner, kan den tas i bruk som et verktøy til å klassifisere ukjent data.

2.2 Perceptron

Den første maskinlæringsmodellen vi skal se nærmere på er *Perceptronet*. Modellen simulerer et biologisk nevron, der stimuli (inputdata) vektes av det kunstige nevronets egenskaper og bidrar til aktivering gjennom perceptronets aktiveringsfunksjon [1, kap. 9]. Som et resultat av denne aktiveringene sender perceptronet ut et signal (utdata). Som et biologisk nevron kan Perceptronet ta inn data fra mange kilder, men resultatet er én enkelt variabel som utgjør modellens respons.



Figur 2: Modell av Perceptron

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n \quad (1)$$

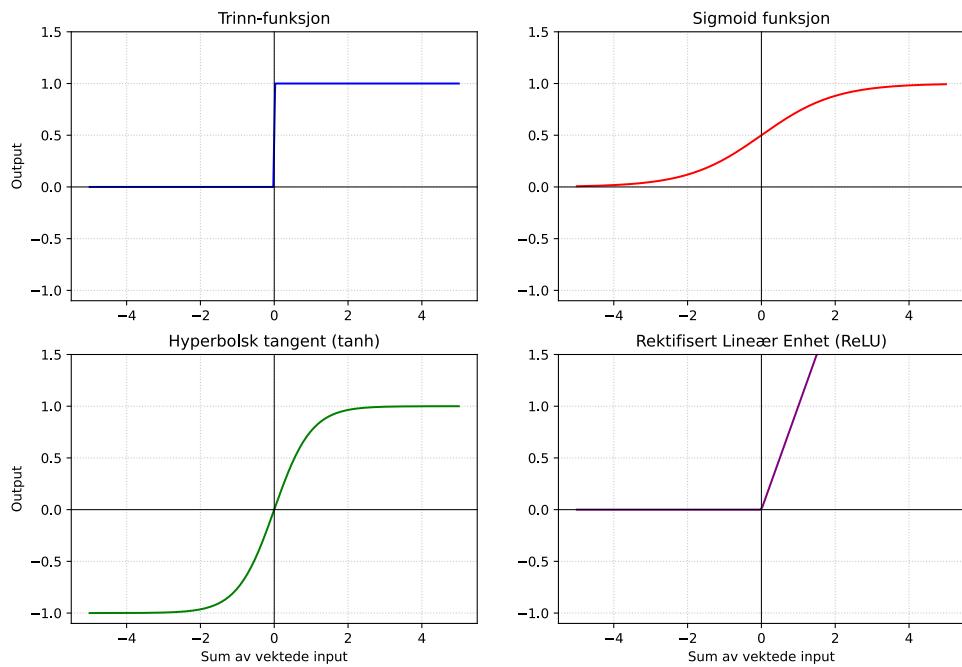
$$\mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \in \mathbb{R}^n \quad (2)$$

$$b \in \mathbb{R} \quad (3)$$

$$\hat{y} = \sigma(W^T \cdot \mathbf{x} + b) \quad (1)$$

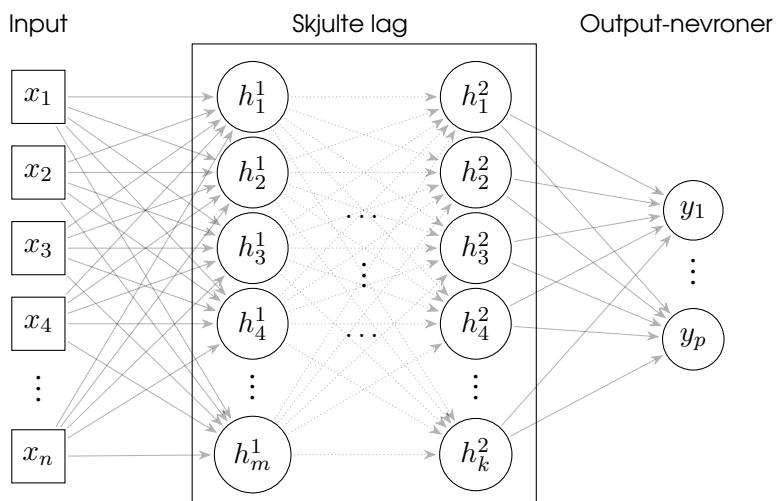
$$(2)$$

$$\sigma(z) = \begin{cases} 1 & \text{hvis } z \geq 0 \\ 0 & \text{hvis } z < 0 \end{cases} \quad (3)$$

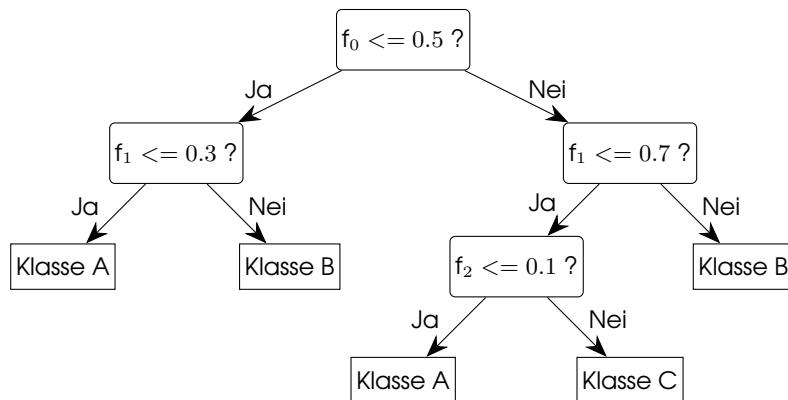


Figur 3: Aktiveringsfunksjoner

2.2.1 Nevrale nettverk (ANN)



2.3 Beslutningstrær (DT)



2.3.1 Gini impurity (GI)

3 Metode

Målet vårt er som nevnt å bli bedre kjent med maskinlæringsmodellene *perceptron* og *beslutningstrær*. Vi skal gjøre dette gjennom en rekke eksperimenter, der flere versjoner av perceptron og beslutningstrær trenes og testes med det samme datasettet. Et enkelt perceptron kan bare gjennomføre binær klassifisering, så vi må tilpasse datasettet til dette. For å ha mulighet til å illustrere perceptronets beslutningsgrense i et 2D-plot, vil vi også begrense inndata til to egenskaper for noen av modellene. En oversikt over modellene som skal trenes er i tabell 1. Videre gjøres det rede for stegene som har blitt gjennomført, i henhold til prosessen i veiledet læring som vist i figur 1.

Nr.	Modell	Ant. egenskaper	Ant. målklasser
1	Perceptron	2	2
2	Perceptron	2	2
3	PerceptronOVA	4	4
4	Beslutningstre	2	2
5	Beslutningstre	2	2
6	Beslutningstre	4	4

Tabell 1: Modeller som skal trenes

3.1 Steg 1: Identifisere nødvendig data

Datasetssett som er brukt i denne oppgaven har blitt et populært datasetssett for opplæring, illustrasjon og testing innen maskinlæring og *Data Science* [3]. Det inneholder observasjonsdata av tre ulike pingvinarter, samlet av en forskingsstasjon på Antarktis. Datasetssettet har x observasjoner av åtte ulike egenskaper (*features*). En oversikt over de ulike egenskapene er i tabell 2.

Egenskap	Datatype	Forklaring
species	Kategorisk (str)	Navn på art.
island	Kategorisk (str)	Navn på øy for observasjon.
bill_length_mm	Numerisk (float)	Lengde på nebb i mm.
bill_depth_mm	Numerisk (float)	Dybde på nebb i mm.
flipper_length_mm	Numerisk (float)	Lengde på luffer i mm.
body_mass_g	Numerisk (int)	Individets vekt i gram.
sex	Kategorisk (str)	Kjønn.

Tabell 2: Målte egenskaper i datasetssettet *palmerpenguins*

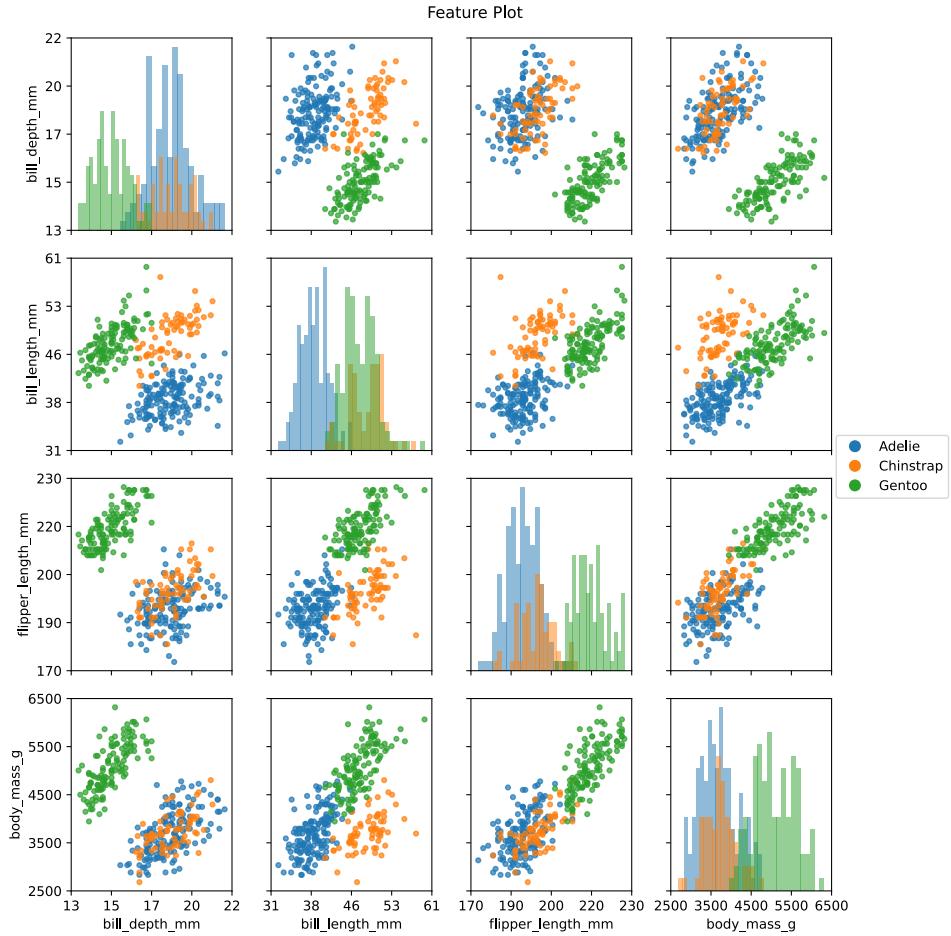
Vi ser at datasetssettet inneholder tre kategoriske egenskaper, i tillegg til fire numeriske, kontinuerlige egenskaper. Art (*species*) er som nevnt målverdien vi er ute etter. Av de øvrige egenskapene skal vi kun bruke de numeriske, og velger bort kjønn (*sex*) og øy (*island*). Dette gjøres hovedsaklig på bakgrunn av at vi kun ønsker å benytte kontinuerlige, numeriske egenskaper for dette eksperimentet.

I tabell 3 vises en enkel analyse av verdiene i datasetssettet for de utvalgte egenskapene. Her vises antall gyldige verdier, gjennomsnitt, standardavvik, minimum og maksimumsverdier, samt antall unike verdier for klassevariabelen.

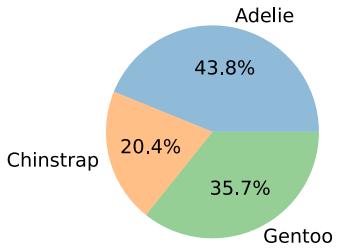
stats	species	bill_length	bill_depth	flipper_length	body_mass
N gyldig	344	342	342	342	342
Gj.snitt		43.92	17.15	200.92	4201.75
SD		5.46	1.97	14.06	801.95
Min		32.1	13.1	172	2700
Maks		59.6	21.5	231	6300
Unike	4				

Tabell 3: Innledende analyse av egenskaper.

For å få oversikt over mulige sammenhenger mellom ulike egenskaper og klassen de tilhører, har vi laget en matrise av grafer der hver egenskap plottes mot hver av de andre egenskapene i figur 4. Langs diagonalen vises histogrammet for den aktuelle egenskapen. Spredningsdiagrammene gir gode indikasjoner på hvilke egenskaper som egner seg godt til å skille de ulike artene fra hverandre.



Figur 4: Sammenhenger mellom egenskaper i datasettet.



Figur 5: Representasjon.

Figur 5 viser representasjonen av hver art i datasettet. Vi ser at fordelingen ikke er jevn, og vi har færre observasjoner av *Chinstrap* å lære fra. Dette kan lede til skjevhets i beregninger, noe som må tas høyde for.

Gjennom den innledende undersøkelsen av datasettet, vurderer vi å ha nødvendig data til å trenne en klassifiseringsalgoritme som kan skille arter fra hverandre. Figur 4 viser tydelig klustering av de ulike artene. Gode egenskaper for å skille **Gentoo** fra de to andre artene kan være egenskapene *bill_depth_mm* og *flipper_length_mm*, og gode egenskaper for å skille **Chinstrap** fra de andre kan være *bill_depth_mm* og *bill_length_mm*.

Siden hensikten med denne øvelsen primært er å se på implementasjon og egen-skaper ved utvalgte algoritmer innen veiledet læring, er vi ikke så opptatt av å vurdere observasjonenes validitet og generaliserbarhet. Hvis målet hadde vært å løse en praktisk problemstilling, ville det vært viktig å vurdere om datasettet er representativt for miljøet den trente modellen skulle operere i. Det ville også vært lurt å samle ytterligere testdata for å sjekke om modellen også ville prestert godt under andre forhold. Her vil vurderingene våre av modellens prestasjoner begrenses til hvor godt den opererer innenfor rammene av dette datasettet.

3.2 Steg 2: Forbehandling av data

Før datasettet kan brukes til trening av en veiledet læringsalgoritme, må vi sørge for at datasettet har et konsekvent og lesbart format. Vi har noen klassiske utfordringer i datasettet vårt, med ulike dataformater, manglende data og variabler med svært ulik skala. Særlig er verdiene for gram mye større enn millimetermål på pingvinenes nebb (se tabell 3 og 2). Manglende observasjonsdata er merket *NA* i datasettet, og som vi ser i tabell 3 mangler verdier for de utvalgte egenskapene ved to av observasjonene. Disse har vi valgt å fjerne. Videre har vi brukt *Z-score* for å normalisere verdiene, slik at ulik skala ikke skal gi skjevheter i vektningen av forskjeller mellom observasjoner.

3.3 Steg 3: Velge algoritme

I dette arbeidet har vi et litt annet utgangspunkt enn i en typisk problemstilling, der algoritmene som skal undersøkes er valgt ut på forhånd. Det er likevel naturlig å fremheve egenskapene til de valgte algoritmene, som oppsummert i tabell 4.

Modell	Styrker	Svakheter	Viktige hensyn
P	- Enkel	- Databegrensninger	- Data må skaleres
	- Ressursvennlig	- Ustabil konvergens	- Dataegenskaper
	- Lineær separasjon	- Kun binære klasser	- Læringsrate
P-OVA	- Flere klasser	- Databegrensninger	- Balanserte klasser
	- Enkel	- Konflikter	- Ressursbruk avhengig av antall klasser
DT	- Fleksibel		
	- Ikke-lineære sammenhenger	- Overtilpasning	- Dybde og beskjæring
	- Velger beste egenskaper	- Ustabil struktur - Egenskapsskjevhets	- Håndtere egenskapsskjevhets

Tabell 4: Fremhevede egenskaper ved algoritmene. (P står for *perceptron*)

3.3.1 Perceptron

Perceptronet er en enkel modell som krever lite maskinressurser både under trening og kjøring. Prediksjoner tar utgangspunkt i én enkelt matrisemultiplikasjon mellom vekter og input. Det gir også en lineær økning av kompleksiteten når antall egenskaper øker. I vårt datasett har vi få egenskaper, som holder ressursbruken lav. Perceptronet fungerer imidlertid kun for data som kan separeres lineært, og et enkelt perceptron kan kun gjennomføre binær klassifisering. Modellen er også sensitiv på egenskaper med ulik skala. Under treningen er det viktig at læringsraten settes på et nivå som er tilpasset egenskapenes skala, slik at oppdateringene er små nok til å finne beslutningsgrensen. Perceptronet har ustabil konvergens, og kan finne ulike løsninger når den trenes på samme data flere ganger. Om datapunktene er godt spredt fra hverandre, vil det finnes mange like gode beslutningsgrenser, og en lokal beste løsning for treningssettet trenger ikke være den beste generelle løsningen.

For vårt datasett er det viktig at vi skalerer egenskapene slik at ulik skala ikke skaper en skjevhetsmotér til én egenskap. Som vi så tidligere, er flere av egenskapene i datasettet kandidater for lineær separasjon mellom klassene, så perceptron kan være en egnet algoritme for å skille disse klassene.

3.3.2 Perceptron One-vs-All

I løpet av dette arbeidet valgte vi å inkludere et svært enkelt kunstig nevralgt nettverk, med ett perceptron for hver målkasse. Dette betyr at de samme grunnleggende egenskapene for perceptronet gjelder også for dette nettverket, som at observasjonsdata må være lineært separerbart mellom klassene. Siden hvert perceptron i nettverket trenes til å indikere når input svarer til én av målklassene, oppstår det en konflikt når flere perceptron gir positivt svar. Dette må håndteres for at ikke den samme klassen velges hver gang flere perceptroner aktiveres. Ressursbruk er som ved perceptronet knyttet til antall egenskaper som brukes, men i tillegg økes kompleksiteten når antall målkasser øker, da vi trenger ett perceptron for hver målkasse.

3.3.3 Beslutningstre

Beslutningstrær er svært fleksible, kan håndtere uskalerte data, ikke-lineære sammenhenger og har en struktur som er enkel å forstå. De er ikke begrenset til binær klassifisering. Algoritmer for beslutningstrær hjelper oss også å velge ut de egenskapene og spørsmålene som er best for å skille klasser fra hverandre. Et beslutningstre er altså ikke like avhengig av forprosessering av data som perceptron. Vi vil likevel bruke samme forprosessering for beslutningstrær og perceptron, for å gjøre det lettere å sammenligne resultatene.

Algoritmen for å bygge beslutningstreet tar utgangspunkt i alle unike verdier i datasettet. Det medfører en skjevhetsfordel for egenskaper med mange unike

verdier. En binær egenskap som kjønn har færre mulige måter å deles på enn kontinuerlige variabler. Her vil vi bare bruke egenskaper som er kontinuerlige, som gjør at vi i stor grad unngår denne skjevheten. Med beslutningstrær er det også viktig å passe på at de ikke blir overtilpasset datasettet. Om ett sett med inputverdier alltid svarer til samme målverdi i datasettet, vil det alltid være mulig å få 100% nøyaktighet om det stilles like mange spørsmål som antallet observasjoner. Dette betyr ikke at beslutningstreet vil prestere godt på ukjente data. Et overtilpasset beslutningstre har laget flere regler som bare stemmer overens med treningsdata, og ikke har en mer generalisert verdi. For å motvirke overtilpasning er det viktig å beskjære treeet slik at det ikke blir for komplekst. Det gjør vi ved å velge gode hyperparametre for beslutningstreet som passer for trening på det aktuelle datasettet.

3.4 Steg 4: Skille data til trening og testing

For å motvirke overtilpasning og ha mulighet til å sjekke om den trenete modellen er i stand til å håndtere nye data, er det viktig at vi har tilgang både til test- og treningsdata. Vi har bare tilgang på ett datasett, som betyr at vi er nødt til å dele opp datasettet, slik at noe data holdes igjen for å teste modellen. Vi har brukt både tilfeldig sampling og *K-fold* for å dele datasettet. *K-fold* ble brukt i forbindelse med justering av hyperparametre, mens tilfeldig sampling ble brukt for trening og testing av endelig modell etter justering av hyperparametre. Ved tilfeldig sampling ble 80% av datasettet valgt til trening (20% til testing). For deling med *K-fold* ble datasettet tilfeldig delt inn i 5 (omtrent) like store deler, der én gruppe ble valgt somtestdata og resten ble brukt til treningsdata. Denne prosessen ble gjentatt til hver del ble brukt somtestdata én gang.

3.5 Steg 5 - 7: Trening, evaluering og justeringer

Denne delen av prosessen ble gjennomført i en tilbakekobblingssløyfe, med automatiske delprosesser.

For å finne de beste hyperparameterne for modellene ble det implementert en algoritme for å trenne og skåre modellene med ulike hyperparametre, for flere tilfeldig valgte delinger av trenings- og testdata. Algoritmen velger aktuelle hyperparametre tilfeldig fra gitte intervaller. Trening og testing skjer så med de valgte hyperparametrene, etter valgt metode for splitting av datasettet, et gitt antall ganger. Snittet av skårene etter testingen for de valgte hyperparametrene lagres. Nye hyperparametre velges, og prosessen gjentas til antall ønskede iterasjoner er nådd. Data fra disse testene plottes så i diagrammer, ett for hver hyperparameter, som viser modellens snittskårer for ulike verdier av hver parameter.

Skårene fra disse testene ble brukt for å evaluere ulike versjoner av modellene. Valg for hyperparametre til den endelige modellen ble gjort med mål om best presisjon. For beslutningstrærne var det også et mål å holde kompleksiteten så lav som mulig, uten å ofre for mye presisjon, for å unngå overtilpasning.

Etter endelig valg av hyperparametre ble det gjennomført en siste tilfeldig deling av test- og treningsdata (80% til trening), og det ble gjennomført tester av modellens prestasjon.

Modell	Søk nr.	Hyperparameter	Verdier
Perceptron 1	1 - 100 kombinasjoner	epochs	100, 200, ..., 600
		Læringsrate	[0.01, 0.15] ₁₀₀ (d^1)
		Maks presisjon	0.90, 0.91, ..., 1.0

Tabell 5: aa

$$\left\{ x \mid x = a + \frac{b-a}{c-1} \cdot n, n \in \{0, 1, \dots, c-1\} \right\}$$

3.6 Måling av modellens prestasjon

For å måle modellenes prestasjon har vi sett på ...

3.7 Implementasjon

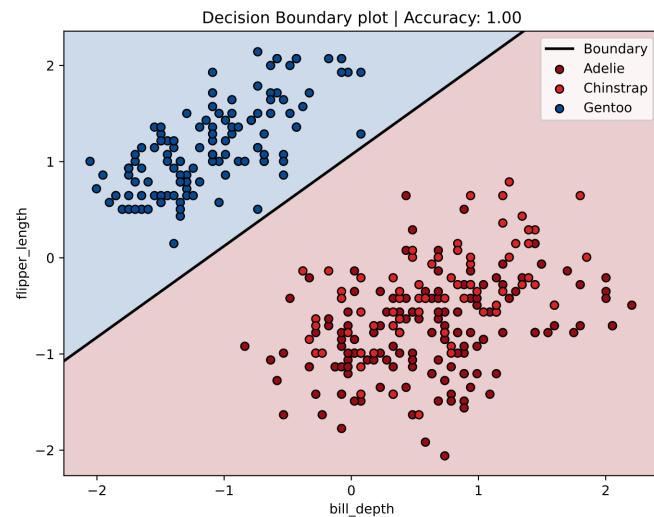
Selve implementeringen av algoritmene er gjort i *Python*, med hjelp av bibliotekene *Numpy* og *matplotlib* [5]–[7].

Kode for forbehandling av data er implementert i modulen `data_tools`

4 Resultat

4.1 Perceptron 1

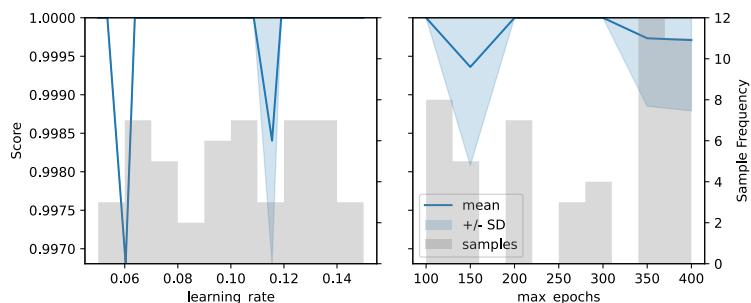
I det første eksperimentet brukte vi egenskapene *flipper_length* og *bill_depth* til å trenne et perceptron til å skille arten **Gentoo** fra de to andre artene, **Adelie** og **Chinstrap**. For disse to egenskapene var **Gentoo** observasjonene tydelig adskilt fra de to resterende artene. Det gjorde det enkelt for perceptronet å finne en lineær separasjon mellom gruppene. Siden



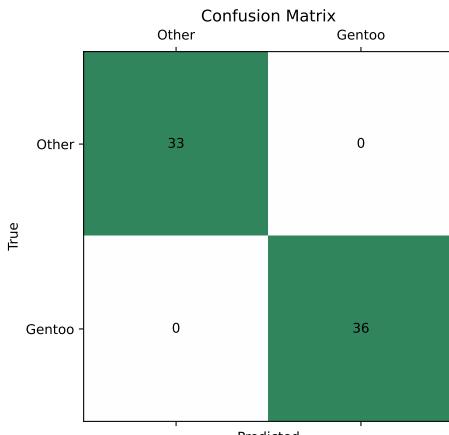
Figur 6: Perceptron 1 - beslutningsgrense

perceptronet ble trent på 2 egenskaper, kan vi enkelt fremstille inndata og beslutningsgrensen i et 2D plot, som vises i figur 6.

Vi startet med et utgangspunkt med en læringsrate på 0.1 og et maksimalt antall treningsiterasjoner på 300. Problemstillingen i dette eksperimentet var enkelt for modellen å tilpasse seg, så vi trengte i liten grad å gjøre justeringer. Det ble likevel kjørt et hyperparametersøk, og resultatene i figur 7 viser at modellen presterer nesten 1.0 i presisjon for alle testede hyperparametre. Plottene viser gjennomsnittlig score for hyperparameteren som en blå linje, der standardavvik på 1 er skravert i samme farge rundt linjen. De grå søylene i bakgrunnen viser antall samplings for hyperparameterverdier i det området. Siden parametrene velges tilfeldig fra en liste eller et intervall, er distribusjonen spredt. Se tabell 1 for valgte intervaller. Scorene er *b-skårer*.



Figur 7: Perceptron 1 - Hyperparametersøk

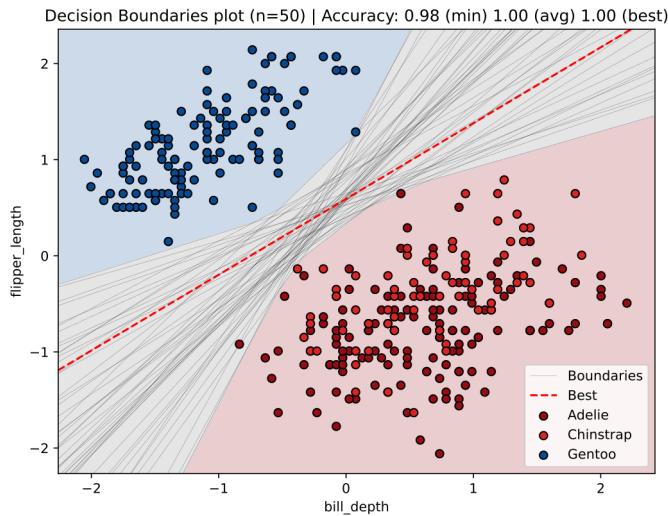


Figur 8: Perceptron 1,
Forvirringsmatrise

Modellen presterer godt og klarer å klassifisere alle individene i testdatasettet korrekt, som vi ser i forvirringsmatrisen i figur 8. Dette gir toppscore 1.0 på både presisjon, sensitivitet, nøyaktighet og spesifisitet.

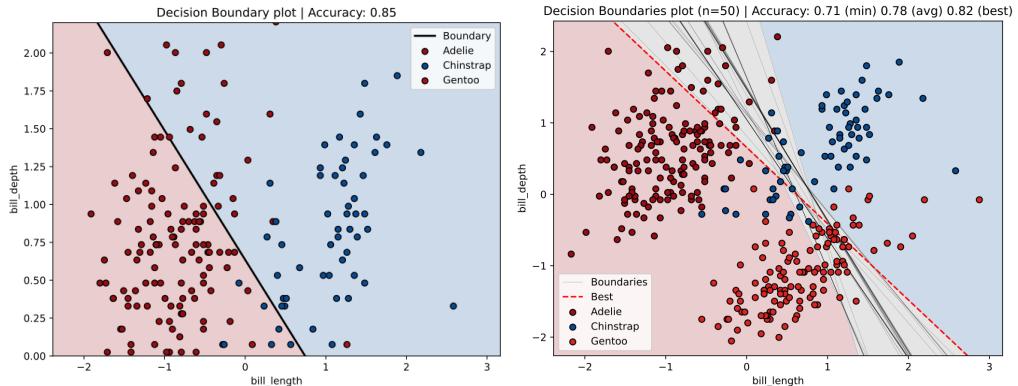
En svakhet ved perceptronet er at om flere beslutningsgrenser skiller data like godt, har den ingen mulighet til å vurdere hvilken linje som er best å velge. I dette scenariet, med god spredning mellom gruppene, finnes det mange svært ulike beslutningslinjer som skiller data like godt. Da kan vi risikere å trenere modellen til å bruke en beslutningslinje som ligger svært tett mot grensen til en av gruppene. Det kan gi større risiko for feil klassifisering av ukjente datapunkter. Intuitivt kan vi tenke at en linje som ligger mot midten av grenseområdet vil være mer nøyaktig i møte med ny data. At perceptronet ikke konvergerer til én løsning, vises i figur 9. Her har et

perceptron blitt trent på samme testdata med samme hyperparametre 50 ganger. Beslutningslinjen i hver iterasjon er plottet som en tynn linje, og en av linjene med høyest presisjon er rød. Hele det grå feltet i midten representerer området et perceptron ville klassifisert et datapunkt enten som den første eller andre gruppen, avhengig av tilfeldigheter under treningen.



Figur 9: Perceptron 1 - Beslutningslinjer under identiske treningsforhold

4.2 Perceptron 2

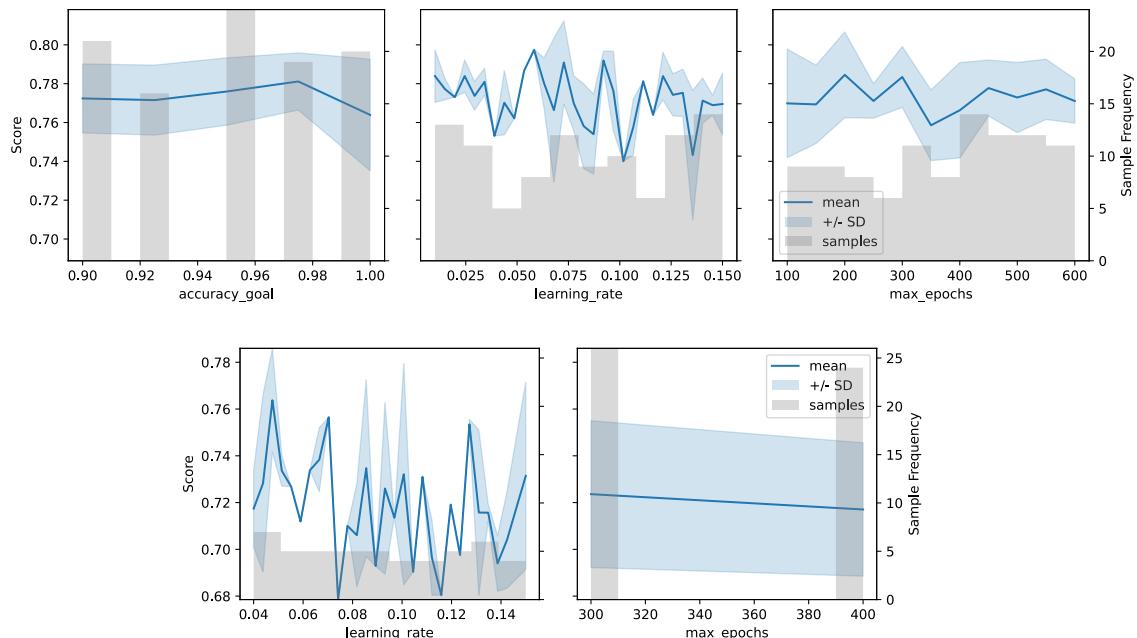


Figur 10: Perceptron 2 - Beslutningslinjer

I det andre eksperimentet skulle vi skille **Chinstrap** fra de to andre artene ved hjelp av egenskapene *bill_depth* og *bill_length*. Vi ser i figur 10 at det her ikke er mulig å skille datapunktene på en naturlig måte med en rett linje. Det nederste

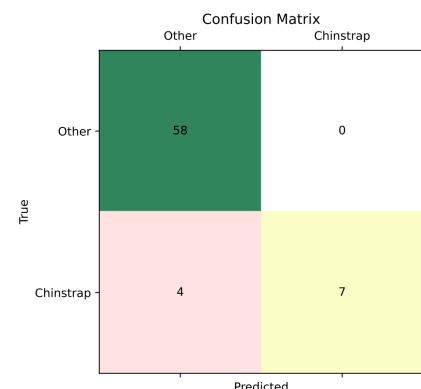
plottet i figuren viser ulike forsøk på å tegne den beste linjen fra 50 modeller, gitt det samme utgangspunktet, med justerte hyperparametre.

For å finne de beste hyperparameterne ble det gjennomført to randomiserte søk i aktuelle intervaller for hver hyperparameter. Resultatene av søket vises i figur 11. Det første søker vises av de tre øverste plottene. I det andre søker har vi satt presisjonsmålet til 1.0 for alle testene. Vi ser at *b-skåren* er mest sensitiv for endringer i læringsraten. Skåren svinger mye ved små endringer i læringsrate, og det er vanskelig å se et tydelig mønster. Når datapunktene fra de to gruppene ligger så tett, uten en klar lineær separasjon, vil små endringer i beslutningslinjens stigning kunne gi store utslag, der flere punkter tildeles nye klasser. I det andre søker testes verdier for læringsraten i små steg på ca 0.00379. Det er et visst repeterende mønster her, men men svingningene er relativt små, og utgjør i underkant av ± 0.1 i *b-skåre*. Det er ikke så mue å hente på å tune hyperparametre her, og det ser ikke ut til at vi vinner særlig på å øke antall treningsiterasjoner. Med *max_epochs* på 200 til 300, og en læringsrate på rundt 0.1, ser ut til å være fornuftige valg.



Figur 11: Perceptron 2 - Hyperparametersøk

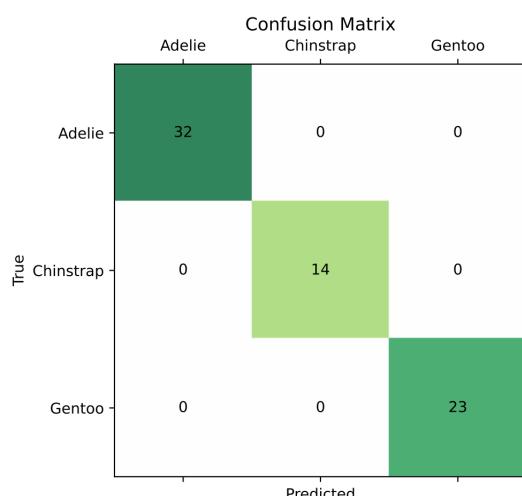
Forvirringsmatrisen i figur 12 viser at modellen har litt større utfordringer med å klassifisere testdata like godt som modellen i det første eksperimentet. Den klarer seg fint i den største gruppen, med en *b-skåre* på 0.93. Klassifisering av **Chinstrap** er svakere, med en *b-score* på 0.69. Samlet har de en skåre på 0.81. *Presi-*



Figur 12: Perceptron 2 - Forvirringsmatrise

sjonen for **Chinstrap** predikasjonene er høy, beregnet til 0.86. Det indikeres av forvirringsmatrisen, som viser at det er ingen falske positive klassifiseringer for denne gruppen. Sensitiviteten er derimot ikke like høy, beregnes fra matrisen til å være 0.57. Modellen har altså bare klart å finne i underkant av 60% av individene som hører til denne klassen. Tilgang på observasjoner av klassen **Chinstrap** er betydelig lavere enn samlede observasjoner fra de andre klassene. Med kun 10 datapunkter for klassen i test-datasettet, gir det et svakt datagrunnlag til å vurdere en mer generell prestasjon av modellen. Plottet av beslutningsgrensen og mangel på god (lineær) separasjon av de to gruppene, gjør likevel at vi vurderer at denne modellen ikke vil prestere godt i klassifisering av ukjent data.

4.3 Perceptron One-vs-All



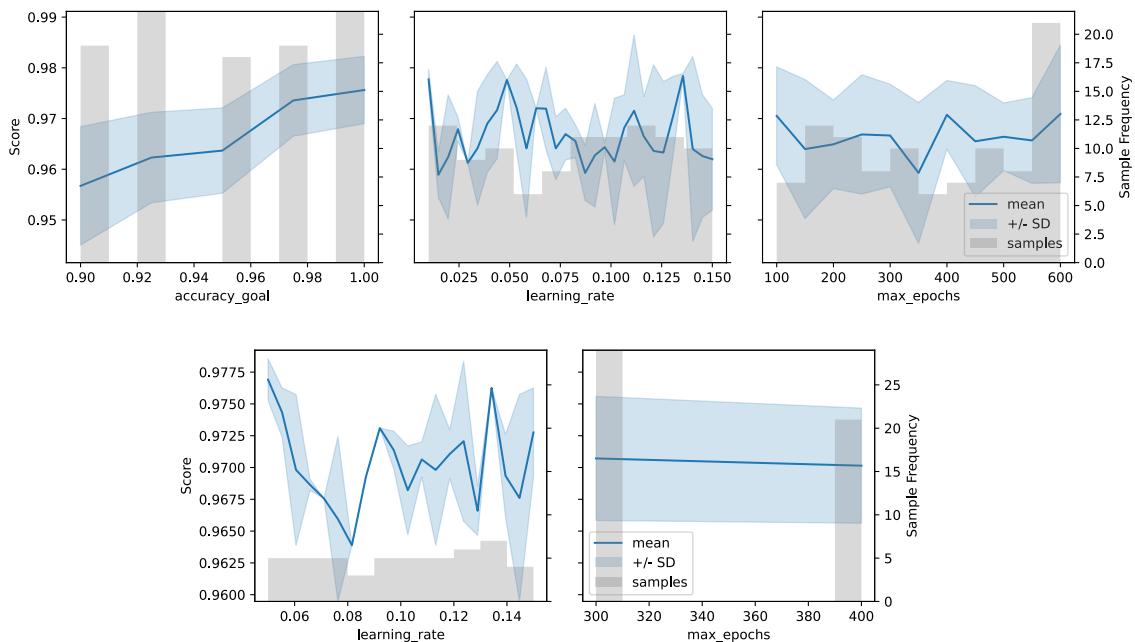
Figur 13: Perceptron OVA - Forvirringsmatrise

med single perceptron, og skårene svinger uregelmessig, særlig for endringer i læringsraten. Siden resultater for alle testene ligger over 0.95, vil de fleste valg fra disse verdiområdene være gode. Maks treningsiterasjoner kan settes i nedre ende for å spare ressurser, sammen med en læringsrate mot midten av testområdet. Vi har valgt maks iterasjoner på 300 sammen med en læringsrate på 0.95.

For det siste perceptron-eksperimentet har vi implementasjonen av et svært enkelt kunstig nevratløt nett av perceptroner. Denne modellen takler klassifisering av flere klasser, så her er ikke målklassene delt inn i en binær gruppe. Alle de numeriske egenskapene i datasettet er valgt som inndata.

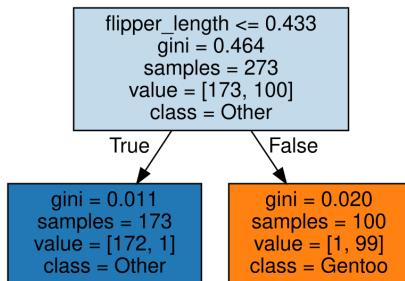
Forvirringsmatrisen i figur 13 viser gode resultater for klassifisering av testdata, der alle individene har blitt klassifisert riktig. Dette gir altså høyeste skåre, 1.0, for alle mål på treffsikkerhet og sensitivitet.

Resultater av hyperparametersøkene for denne modellen er i figur 14. Ikke overraskende ser vi noen av de samme tendensene som for eksperimentene



Figur 14: Perceptron OVA - Hyperparametersøk

4.4 Beslutningstre 1 (DT 1)

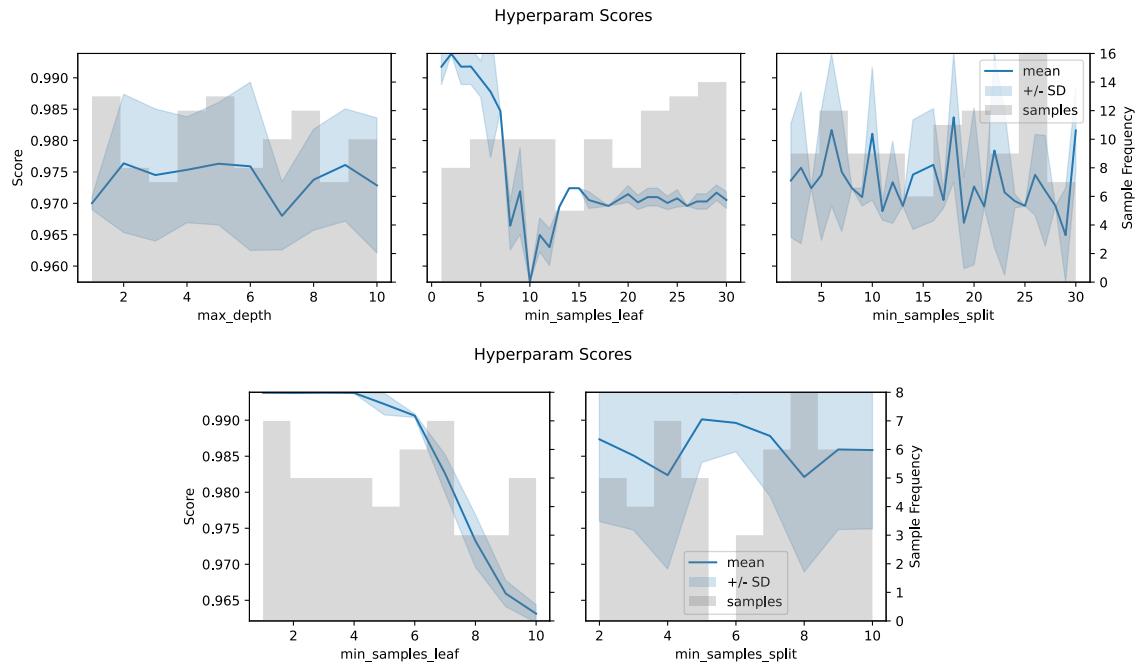


Figur 15: Beslutningstre 1 (DT 1)

Det første eksperimentet med beslutningstrær er gjennomført med samme input og mål som det første eksperimentet med perceptron, **Gentoo** skulle skilles fra andre klasser ved hjelp av *flipper_length* og *bill_depth*. Uten beskjæring får vi et tre som klassifiserer datasettet «perfekt». For å unngå unødvendig ressursbruk og kompleksitet, samt motvirke overtilpasning, har vi beskjært treet både under generering *pre-pruning* og etter det er ferdig generert *post-pruning*. Etter trening og beskjæring fikk vi treet i figur 15. Treet viser statistikk for hver node, med tall for *gini-impurity*, antall individer av hver klasse i noden og hvilket spørsmål som brukes for å splitte data ved noden. Farge og klasse tildeles noden ut fra klassen som dominerer ved noden. Fargen er sterkest ved *gini* = 0 og går gradvis mot hvit når *gini* går mot 0.5.

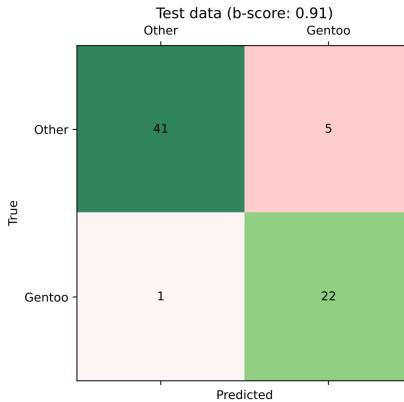
Beskjæring under generering av treet har vi gjort med hyperparametene i 16, som stopper videre forgrening om det er for få individer igjen i noden, treet blir for dypt, eller aktuelle delinger gir for få individer i en løvnode. Plottet viser at

b-skårene er over 0.96 for alle hyperparameterne som har blitt undersøkt. For å unngå overtilpasning har vi valgt å holde oss til hyperparametre som reduserer kompleksiteten til treet, samtidig som vi ser at presisjonen er på et akseptabelt nivå. Vi har fokusert mest på parameterne som setter krav til antall individer før og etter splitting, og satt maks dybde til et høyere tall, som en øvre grense det ikke forventes at algoritmen når. Fra søkedata og manuell kontrolltesting av ulike verdier ble `min_samples_split` satt til 10 og `min_samples_leaf` til 5. Dette er ikke de verdiene som skårer høyest på søkeret, men med en skåre på over 0.95 er vi allerede i faresonen for overtilpasning, så vi ønsker ikke et tre som er dypere enn hva det er grunnlag for å lære på bakgrunn av tilgjengelig data.



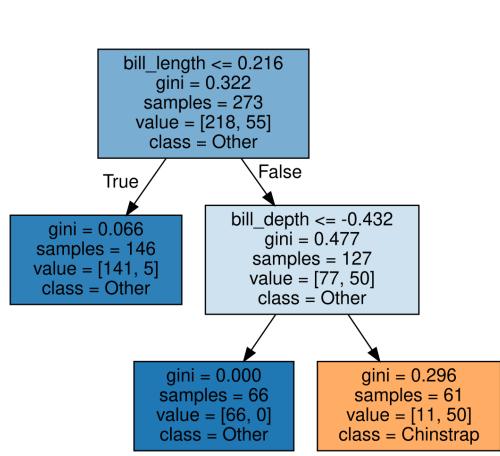
Figur 16: DT 1 - Hyperparametersøk

I figur 17 ser vi forvirringsmatrisen til det ferdigtrente beslutnignstreet, målt opp mot prestasjon i klassifisering av testdatasettet. Dette gir en *b-skår* på 0.97 for **Gentoo**, 0.98 for den andre gruppen og totalt en skår på 0.97.

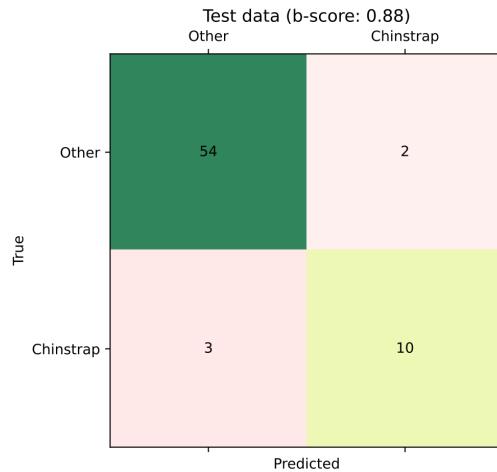


Figur 17: DT 1
- Forvirringsmatrise

4.5 Beslutningstre 2 (DT 2)



(a) Beslutningstre 2 (DT 2)



(b) Beslutningstre 2 - Forvirringsmatrise

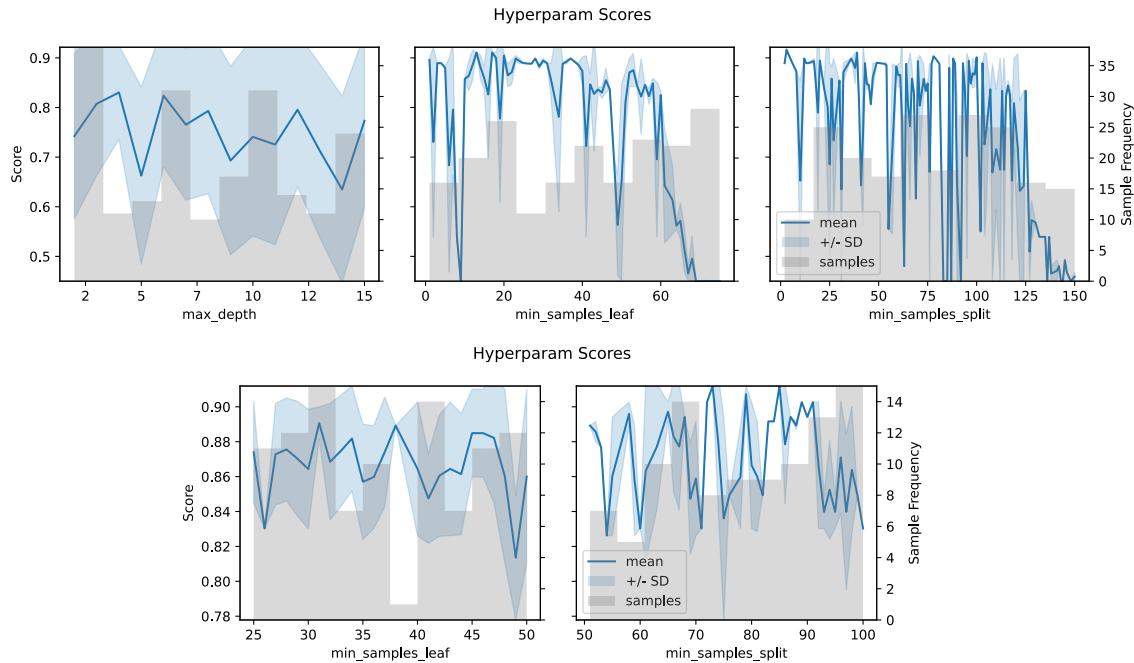
Figur 18: Beslutningstre 2

Det andre beslutningstreet fulgte samme design som for eksperimentet med *Perceptron 1*, målet var å skille **Chinstrap** fra de andre artene med verdier for egenkapene *bill_depth* og *bill_length*. Resultatet etter beskjæring er i figur 18a.

Figur 18b viser at nesten alle individer har blitt klassifisert korrekt. Dette gir *b-skårene* 0.80 for **Chinstrap**, 0.96 for den andre gruppen og 0.88 totalt for alle predikasjonene.

Hyperparametersøket i figur 19 viser lavere skårer enn i forrige eksperiment. Her har vi også inkludert et større spenn av verdier, så høyere varians er naturlig. For hyperparametersøk 2, andre linje av figuren, ser vi at presisjonen er relativt høy,

selv ved større verdier av beskjæringsparametrene. Verdiene svinger uregelmessig, som kan indikere at de fleste valgene gir relativt like resultater. På bakgrunn av søk og tester ble *min_samples_split* satt til 100 og *min_samples_leaf* til 50.



Figur 19: Beslutningstre 2 - Hyperparametersøk

4.6 Beslutningstre 3 (DT 3)

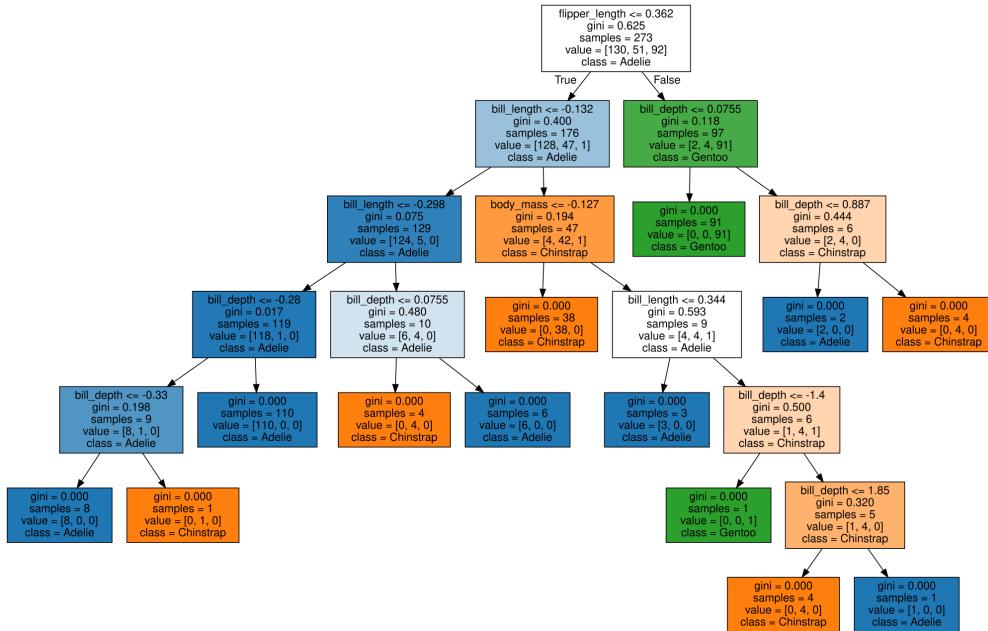
Det siste beslutningstreet bruker alle egenskaper som inndata og alle klassene fra datasettet er med som målverdier. For dette treet vil vi også demonstrere beskjæringens påvirkning på trestrukturen. En modell trent uten satte beskjæringsparametre er vist i figur 20.

Dette treet er perfekt tilpasset treningsdata, der alle nodene har blitt delt til *gini impurity* er 0. Dette er ikke et godt tegn, da det indikerer at treet er overtilpasset. Ved bruk av hyperparametre kan vi stoppe genereringen av treet, før det blir unødig komplekst og for godt tilpasset treningsdata. Etter beskjæring med hyperparametre ser treet ut som i figur 21.

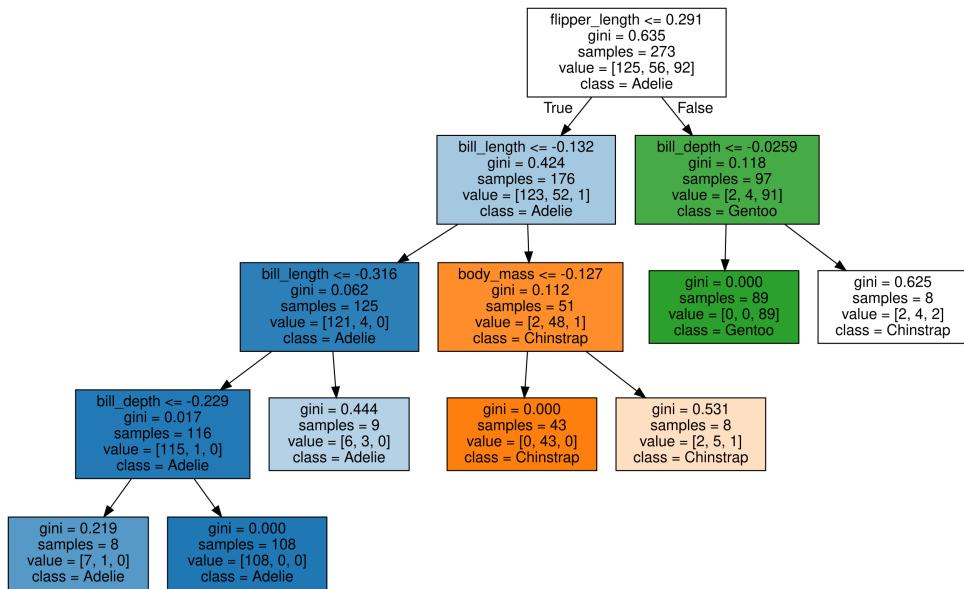
Dette begynner å se bra ut, men vi ser at et helt deltre på venstre side gir samme klasse for alle løvnoder. Dette er unødvendig, og kan fjernes gjennom *post-pruning* etter at treet er generert. Da står vi igjen med det ferdig beskjærte treet i figur 22.

Resultater av hyperparametersøket for Beslutningstre 3 er i figur 24. Vi har valgt 16 for *min_samples_split* og 8 for *min_samples_leaf*.

Modellens ytelse illustreres av forvirringsmatrisen i ??.



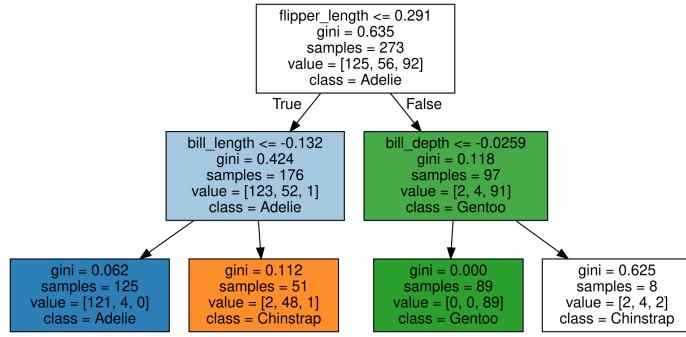
Figur 20: DT 3 uten beskjæring



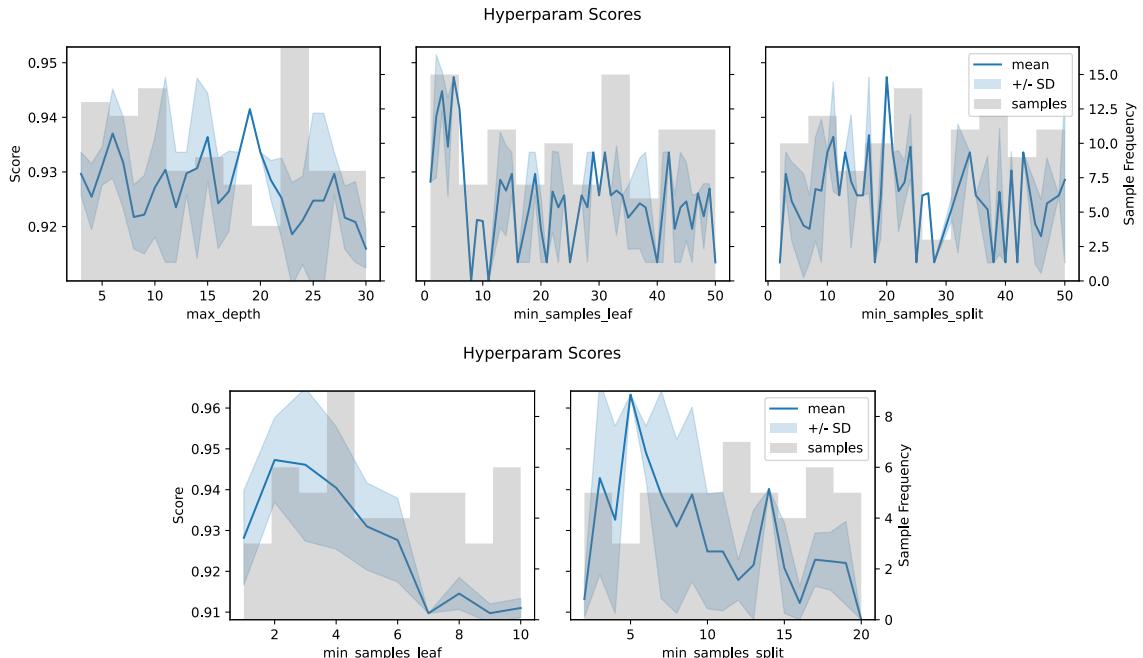
Figur 21: DT 3 før *post-pruning*

5 Diskusjon og konklusjon

Vi har sett



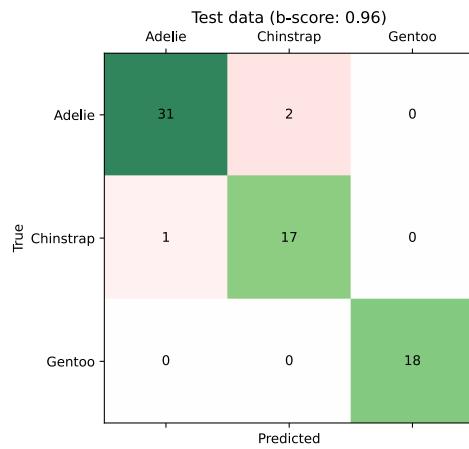
Figur 22: DT 3 etter post-pruning



Figur 23: DT 3 - Hyperparamettersøk

Referanser

- [1] R. Hurbans, *Grokking artificial intelligence algorithms*. Shutter Island, NY, USA: Manning, 2021.
- [2] S. Kotsiantis, «Supervised Machine Learning: A Review of Classification Techniques.,» *Informatica (Slovenia)*, årg. 31, s. 249–268, jan. 2007.
- [3] A. M. Horst, A. P. Hill og K. B. Gorman, *palmerpenguins: Palmer Archipelago (Antarctica) penguin data*, R package version 0.1.0, 2020. DOI: 10.5281/zenodo.3960218. [Online]. Hentet fra: <https://allisonhorst.github.io/palmerpenguins/>.



Figur 24: DT 3 - Forvirringsmatrise

- [4] K. E. Tranøy, *induksjon – filosofi*, i *Store Norske Leksikon*, 2024. [Online]. Hentet fra: https://snl.no/induksjon_-_filosofi Lastet ned: 15.11.2024.
- [5] G. Van Rossum og F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.
- [6] C. R. Harris, K. J. Millman, S. J. van der Walt *et al.*, «Array programming with NumPy,» *Nature*, årg. 585, nr. 7825, s. 357–362, sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Hentet fra: <https://doi.org/10.1038/s41586-020-2649-2>.
- [7] J. D. Hunter, «Matplotlib: A 2D graphics environment,» *Computing in Science & Engineering*, årg. 9, nr. 3, s. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.