

DTE-2602 | H24

# Karaktersatt oppgave 2

Supervised Learning

Bjarte Flø Lode

24. november 2024



# 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*, *Lasso 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.

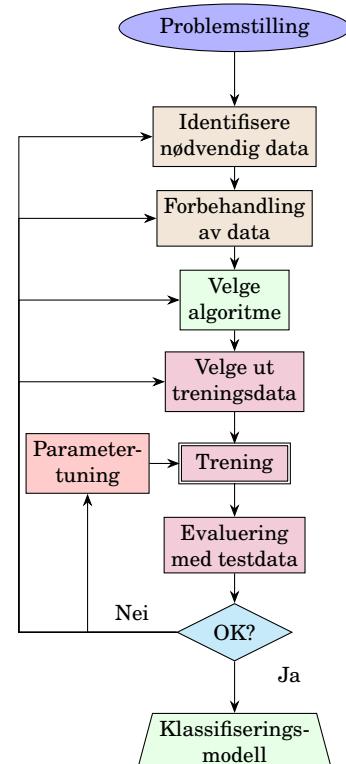
For å skape gode maskinlæringsmodeller må vi altså være godt kjent med egenskaper ved ulike metoder, algoritmer og tilnærminger til ML. I denne oppgaven skal vi se nærmere på to typer algoritmer for veiledet læring, *Decision Tree* (DT) og *Perceptron*, og hvordan disse kan brukes i klassifisering. Algoritmene 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 tilnærminger 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 kjen-



Figur 1: Veiledet læring (2, s. 250)

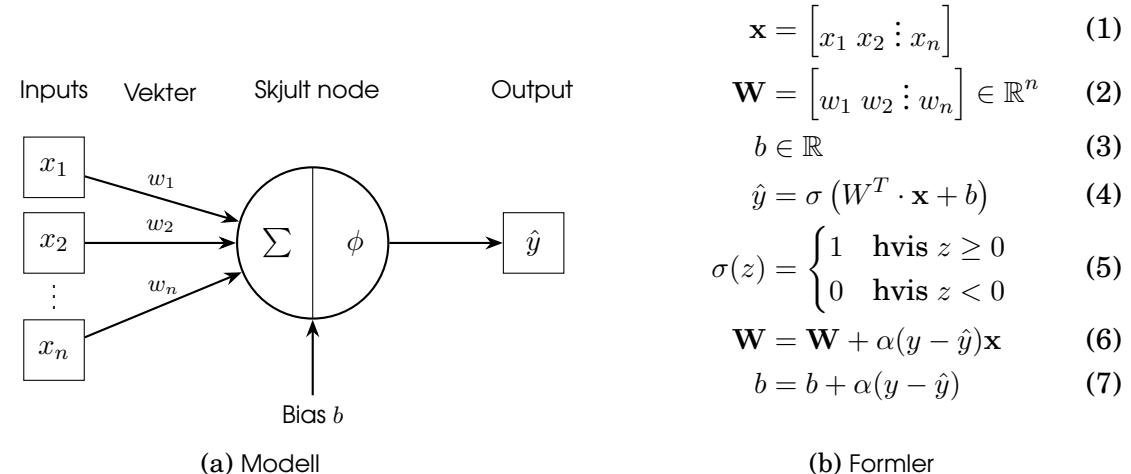
te datasettet til å danne et sett regler som kobler inndata, egenskaper, til utdata, målverdiene. Dette er en prosess som i vitenskapsteorien er kjent som *induksjon*, altså at man danner generelle modeller på bakgrunn av enkeltobservasjoner. I motsetning til *deduktive* slutninger er ikke en *induktiv* slutning logisk bindende, men uttrykker en sannsynlighetsovervekt [4]. Dette gjelder videre for veiledet læring, altså vil reglene som dannes aldri kunne uttrykke absolutt kunnskap. For at denne induktive læringen skal fungere, må datagrunnlaget være representativt og ha relevante egenskaper som korrelerer med målverdiene. Dataene trenger ikke nødvendigvis ha kausale sammenhenger - styrken i veiledet læring er at vi kan finne mønstre uten å kjenne de underliggende årsakssammenhengene på forhånd.

Som figur 1 viser, er flere av stegene i veiledet læring «manuelle» og relatert til datautviegelse, databehandling og preprosessering. Trening, evaluering og tuning av algoritmen er bare en liten del prosessen. ProsesSEN er iterativ og eksperimentell - resultater fra en modell brukes til å vurdere om hyperparametre må justeres, om algoritmen er egnet,

eller om datagrunnlaget må forbedres. Først når modellen oppnår akseptabel prediksjonsnøyaktighet, kan den brukes på ukjente data.

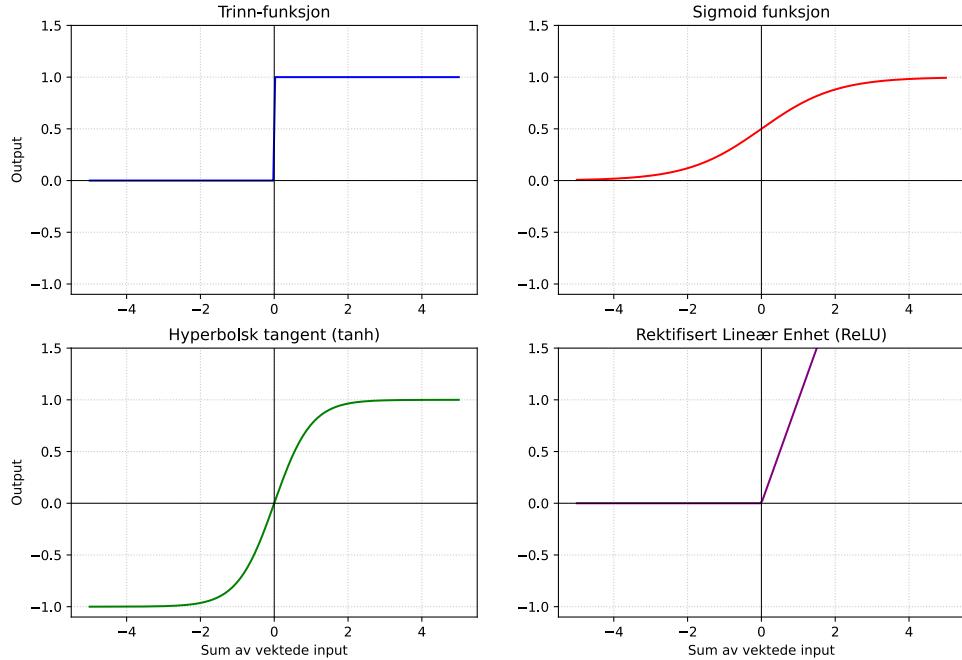
## 2.2 Perceptron

Et *Perceptron* 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 aktiveringen 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, se figur 2a.



Figur 2: Perceptron

Formler for perceptronet er samlet i figur 2b. Aktivering skjer ved at prikkproduktet til input- og vektvektor (1), pluss bias, sendes til aktiveringsfunksjonen (4). Flere funksjoner kan brukes som aktiveringsfunksjon for perceptronet, se figur 3. Her skal vi bruke den enkle trinn-funksjonen, som gir binær output  $\sigma$  (5).

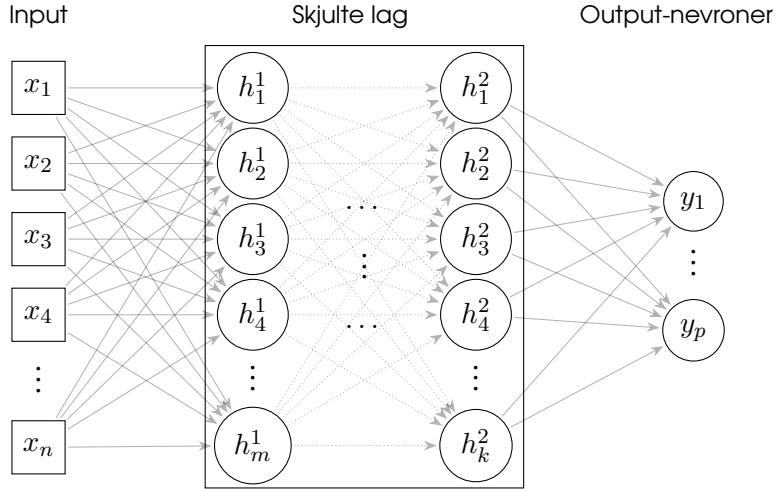


Figur 3: Aktiveringsfunksjoner

### 2.2.1 Nevrale nettverk (ANN)

Et kunstig nevralt nettverk er kort sagt flere kunstige neuroner, for eksempel *Perceptroner*, som kobles sammen for å løse en oppgave [1, kap. 9]. De kan kobles sammen i flere ulike konfigurasjoner, men de følger gjerne strukturen i figur 4.

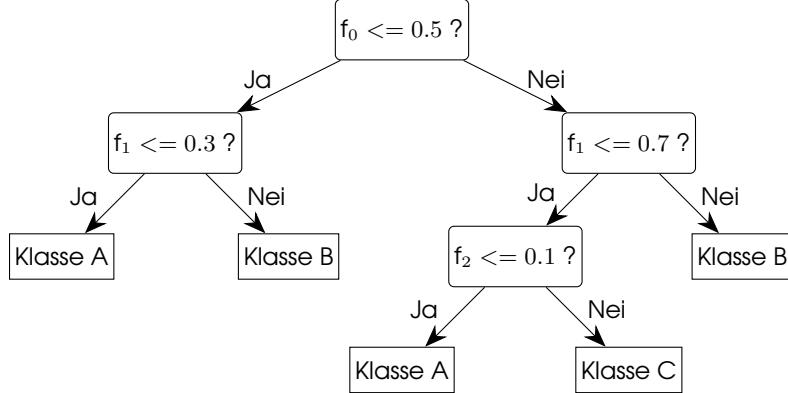
Styrken til nevrale nettverk ligger i deres evne til å lære komplekse, ikke-lineære sammenhenger i data, men de krever ofte store datamengder og betydelige beregningsressurser for trening.



Figur 4: Kunstig nevralt nettverk (ANN)

### 2.3 Beslutningstrær (DT)

Beslutningstrær er en struktur av binære kriterier eller spørsmål som rekursivt deler opp et egenskapsrom i mindre regioner. Hver node i treet representerer en test på form  $f_j \leq t$ , der  $f_j$  er verdien av en egenskap (feature) og  $t$  er en terskelverdi. Utfallet av testen bestemmer videre retning nedover i treet, til en løvnode nås som gir en predikasjon, se figur 5.



Figur 5: Beslutningstre

#### 2.3.1 Gini impurity (GI)

Beslutningstreet genereres iterativt gjennom å søke etter det spørsmålet som best egner seg til å redusere entropien i datasettet. Dette kan gjøres ved å måle *Gini impurity* før og etter en potensiell deling av datasettet. Denne verdien uttrykker

«urenheten» av data, der verdien representerer sannsynligheten for å ikke velge ønsket verdi dersom du velger en tilfeldig verdi fra datasettet. Et uniformt sett vil derfor ha Gini Impurity  $I_G = 0$ , da det er 0% sannsynlig å velge «feil» verdi fra denne noden. Et sett med  $inf$  elementer, der bare ett element har verdien man ønsker å trekke, vil ha en  $I_G$  som går mot 0.

I et sett med  $J$  klasser og sannsynligheten for å tilfeldig trekke en klasse  $i$  er  $p_i$ , er  $I_G$  defineres som:

$$I_G = 1 - \sum_{i=1}^J p_i (1 - p_i)$$

For å finne de beste spørsmålene for å dele datasettet, beregner algoritmen  $I_G$  for alle mulige delinger ved en node, og ser hvilket spørsmål som reduserer samlet entropi i størst mulig grad.

### 3 Metode

---

Målet vårt er som nevnt å bli bedre kjent med maskinlæringsmodellene *perceptron* og *beslutningstrær*. Vi skal gjøre dette gjennom en rekke eksperimenter på ett datasett. Datasettet vil tilpasses for noen eksperimenter, i henhold til modellens begrensninger og mulighet for å illustrere perceptronets beslutningsgrense i et 2D-plot. Se tabell 1 som gir en oversikt over eksperimentene som skal gjennomføres, 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

---

Datasettet vi bruker, *palmerpenguins*, har blitt et populært datasett for opplæring, illustrasjon og testing innen maskinlæring [3]. Det inneholder observasjonsdata av tre ulike pingvinarter, samlet av en forskningsstasjon på Antarktis. Datasettet har 344 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 datasettet *palmerpenguins*

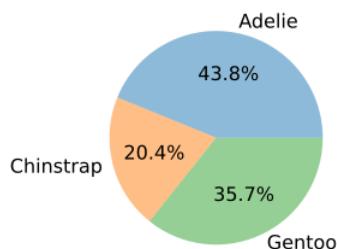
Datasettet inneholder tre kategoriske egenskaper, i tillegg til fire numeriske, kontinuerlige egenskaper. Art (*species*) er målverdien vi er ute etter. Av de øvrige egenskapene skal vi kun bruke de numeriske, og velger bort kjønn (*sex*) og øy (*island*).

I tabell 3 vises en enkel analyse av verdiene i datasettet 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
<i>N</i> 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.

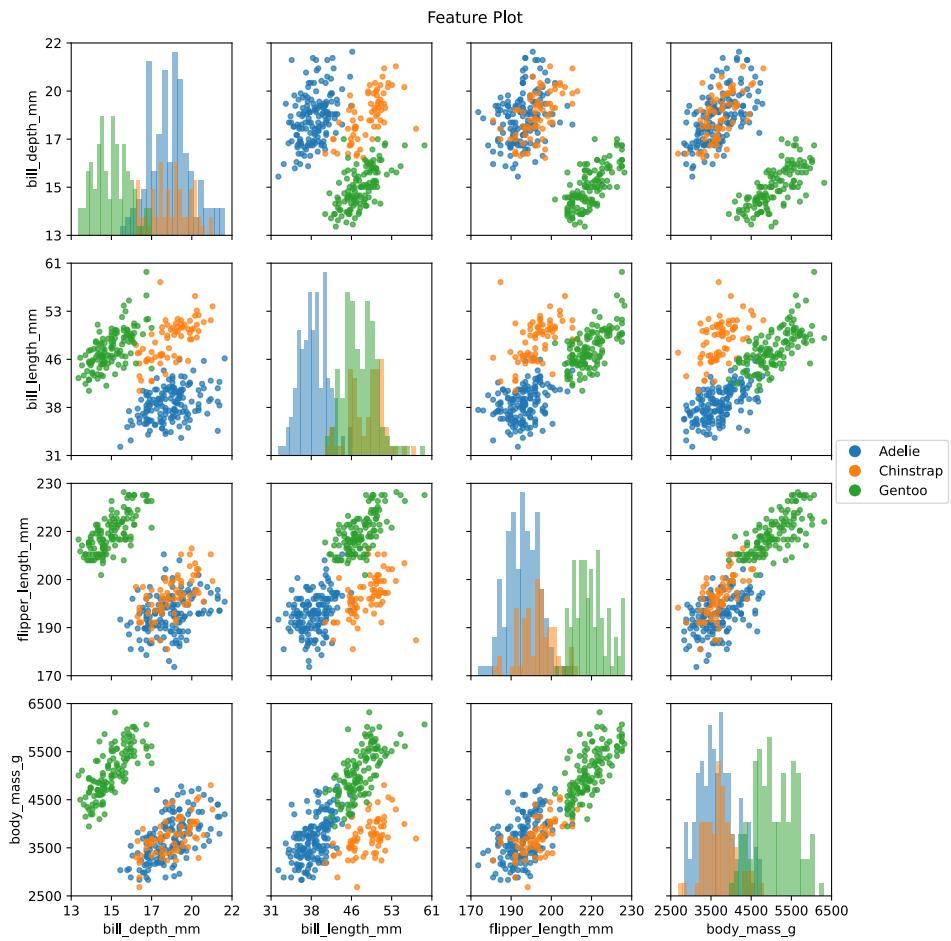
For å få oversikt over mulige sammenhenger mellom ulike egenskaper og klassen de tilhører, har vi laget en matrise av grafer der hver egenskap plottes mot hver av de andre egenskapene i figur 6. Langs diagonalen vises histogrammet for den aktuelle egenskapen. Spredningsdiagrammene gir gode indikasjoner på hvilke egenskaper som egner seg godt til å skille de ulike artene fra hverandre.



Figur 7: Representasjon.

Figur 7 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.

Figur 6 viser tydelig klustering av de ulike artene. Gode egenskaper for å skille **Gentoo** fra de to andre artene kan være egenskapene



Figur 6: Sammenhenger mellom egenskaper i datasettet.

*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 egenskaper 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 trenete modellen skulle operere i.

### 3.2 Steg 2: Forbehandling av data

Før datasettet kan brukes til trening av en modell, må vi sørge for at datasettet har et konsekvent og lesbart format. Vi har noen utfordringer i datasettet vårt: ulike dataformater, manglende data og variabler med ulik skala. Det mangler data for

to av observasjonene. Disse har vi valgt å fjerne. *Z-score* er brukt for å normalisere verdiene, slik at ulik skala ikke skal gi skjeheter i vektingen 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
	- Fleksibel		
DT	- Ikke-lineære sammenhenger	- Overtilpasning	- Dybde og beskjæring
		- Ustabil struktur	- Håndtere egenskapsskjevhet
	- Velger beste egenskaper	- Egenskapsskjevhet	

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 skjehet mot 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 målklasse. 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 målklasser øker, da vi trenger ett perceptron for hver målklasse.

### 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 treet 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 som testdata og resten ble brukt til treningsdata. Denne prosessen ble gjentatt til hver del ble brukt som testdata én gang.

### 3.5 Steg 5 - 7: Trening, evaluering og justeringer

---

For systematisk evaluering og optimalisering av modellene ble det implementert en iterativ prosess for hyperparameterjustering. Prosessen er implementert i RandomSearch klassen som utfører følgende steg for hver modell:

- **Hyperparametersøk:**

- Tilfeldig sampling av hyperparametre fra definerte intervaller
- 5-fold kryssvalidering for hver parameterkombinasjon
- Beregning av gjennomsnittlig b-skår over alle folds

- **Evaluering:**

- Plotting av modellytelse mot hver hyperparameter
- Generering av forvirringsmatriser for beste parametersett
- Beregning av presisjon, recall og b-skår per klasse

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

For å finne de beste hyperparametrerne for modellene ble det implementert en algoritme for å trenere og skåre modellene med ulike hyperparametre, for flere tilfeldig valgte delinger av trenings- og testdata. Algoritmen velger aktuelle hyperparametere tilfeldig fra gitte intervaller. Trening og testing skjer så med de valgte hyperparametrerne, etter valgt metode for splitting av datasettet, et gitt antall ganger. Snittet av skårene etter testingen for de valgte hyperparametrerne 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	Parameter	Søkerom	Beste verdi
Perceptron 1	Læringsrate	[0.01, 0.15]	0.10
	Maks epoker	{100, 200, ..., 600}	300
	Accuracy goal	{0.90, 0.91, ..., 1.0}	1.0
Perceptron 2	Læringsrate	[0.001, 0.2]	0.10
	Maks epoker	{100, 200, ..., 600}	300
	Accuracy goal	{1.0}	1.0
Perceptron OVA	Læringsrate	[0.01, 0.15]	0.095
	Maks epoker	{100, 200, ..., 600}	300
	Accuracy goal	{0.90, 0.91, ..., 1.0}	1.0
DT 1	Max dybde	[3, 20]	5
	Min samples split	[2, 100]	10
	Min samples leaf	[1, 50]	5
DT 2	Max dybde	[3, 20]	5
	Min samples split	[2, 100]	100
	Min samples leaf	[1, 50]	50
DT 3	Max dybde	[3, 20]	5
	Min samples split	[2, 100]	16
	Min samples leaf	[1, 50]	8

Tabell 5: Hyperparametre, søkerom og beste verdier funnet for hver modell

### 3.6 Måling av modellens prestasjon

For å evaluere modellenes ytelse ble det implementert flere evalueringsmetrikk i `measure.py`. Hovedfokus var på følgende mål:

- **Forvirringsmatrise (confusion matrix):** En  $K \times K$  matrise for  $K$  klasser der element  $(i, j)$  viser antall observasjoner av klasse  $i$  som ble predikert som klasse  $j$ . Dette gir grunnlag for å beregne:

- True Positives (TP): Korrekt klassifiserte observasjoner for hver klasse
- False Positives (FP): Observasjoner feilaktig klassifisert som klassen
- False Negatives (FN): Observasjoner fra klassen klassifisert som andre klasser

- **Presisjon og recall:** For hver klasse  $k$  beregnes:

$$\text{Presisjon}_k = \frac{\text{TP}_k}{\text{TP}_k + \text{FP}_k}$$

$$\text{Recall}_k = \frac{\text{TP}_k}{\text{TP}_k + \text{FN}_k}$$

der presisjon måler andelen korrekte positive prediksjoner, og recall måler andelen av

faktiske positive tilfeller som ble funnet.

- **B-skåre:** Et balansert mål som kombinerer presisjon og recall:

$$\text{B-score}_k = \frac{\text{Presisjon}_k + \text{Recall}_k}{2}$$

*B-skåre* ble valgt som hovedmetrikk for hyperparametersøk da det gir lik vekt til både presisjon og recall, noe som er viktig for et ubalansert datasett.

## 3.7 Implementasjon

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

Koden er delt inn i to pakker. `data_tools` tar for seg verktøy til forbehandling av data, måling og justering av maskinlæringsmodeller og visualisering. Den andre pakken, `models` inneholder implementeringen av maskinlæringsmodellene.

### 3.7.1 Pakken `models`

Den abstrakte baseklassen `MLModel` definerer et felles grensesnitt for maskinlæringsmodellene i pakken. Slik kan verktøyene fra `data_tools` brukes til å måle, visualisere og sammenligne de ulike maskinlæringsmodellene. Alle modeller må implementere følgende funksjoner:

- `fit(X, y)`: Trening av modell på datasett
- `predict(X)`: Prediksjon av klasser for nye observasjoner
- `set_params(**kwargs)`: Justering av hyperparametre
- `get_model_props()`: Henting av modellens egenskaper

### Perceptron

Pakken inneholder to perceptron-klasser:

- `Perceptron`: Implementerer binær klassifisering med ett enkelt perceptron
- `PerceptronOVAClassifier`: Implementerer multiclass klassifisering ved hjelp av One-vs-All strategi, der ett perceptron trenes for hver klasse. et enkelt nevralt nett

Perceptronet bruker trinn-funksjonen som aktiveringsfunksjon. Trening gjøres ved å iterativt oppdatere vekter og bias basert på prediksionsfeilen:

$$w_j = w_j + \text{learning\_rate} \cdot (y - \hat{y}) \cdot x_j$$
$$b = b + \text{learning\_rate} \cdot (y - \hat{y})$$

Modulen for perceptron-klassen inneholder også en funksjon for å plotte beslutningslinjen for perceptron med to egenskaper som input.

## Beslutningstre

Beslutningstreimplementasjonen består av tre hovedklasser:

- `DecisionTree`: Hovedklassen som implementerer selve beslutningstreet
- `DTBranchNode`: Implementerer forgreiningsnoder med beslutningsregler
- `DTLeafNode`: Implementerer løvnoder som inneholder klassetilhørighet

Treet bygges rekursivt ved å finne den beste splitten i hver node, basert på Gini impurity reduksjon. For å unngå overtilpasning brukes både pre- og post-pruning:

- **Pre-pruning**: Styres av hyperparametrene `max_depth`,  
`min_samples_split` og `min_samples_leaf`
- **Post-pruning**: Fjerner deltrær der alle løvnoder predikerer samme klasse

### 3.7.2 Støtteverktøy

Et omfattende sett med støtteverktøy er implementert i `data_tools` pakken:

#### Datasplitting og hyperparametersøk

- `RandomSplit`: Tilfeldig oppdeling i trenings- og testsett
- `KFold`: K-fold cross-validation
- `RandomSearch`: Tilfeldig søk i hyperparameterrom

#### Evaluering og visualisering

- `confusion_matrix`: Beregning av forvirringsmatrise
- `precision_recall`: Beregning av presisjon og recall
- `plot_confusion_matrix`: Visualisering av forvirringsmatrise med matplotlib
- `feature_plot`: Visualisering av egenskapsdistribusjoner og sammenhenger

#### Databehandling

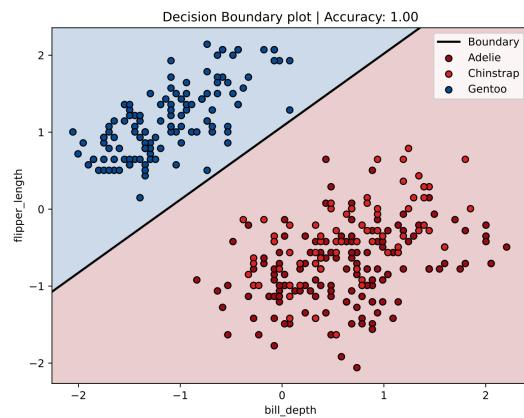
- `normalize_data`: Z-score normalisering av numeriske egenskaper
- `remove_na_rows`: Fjerning av rader med manglende verdier
- `binary_remap_vector`: Konvertering til binære klassevariabler

## 4 Resultat

### 4.1 Perceptron 1

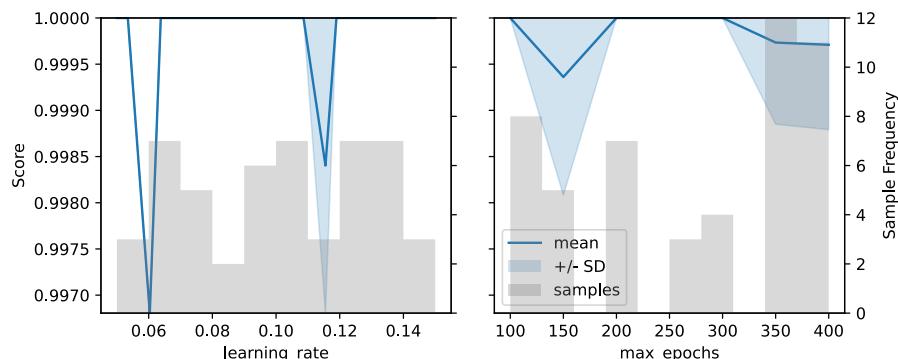
I første eksperiment skulle et perceptron skille **Gentoo** fra de andre artene ved hjelp av egenskapene *flipper\_length* og *bill\_depth*. Siden **Gentoo** var tydelig adskilt fra de andre artene i dette feature-rommet (figur 8), var dette en enkel oppgave for perceptronet.

Med utgangspunkt i læringsrate 0.1 og maksimalt 300 treningsiterasjoner oppnådde modellen svært gode resultater. Hyperparametersøket (figur 9) bekrefet dette med b-skårer nær 1.0 for alle testede parameterkombinasjoner. I figurene viser blå linje gjennomsnittlig skår med standardavvik, mens grå søyler indikerer samplingfrekvens for hver parameterverdi.

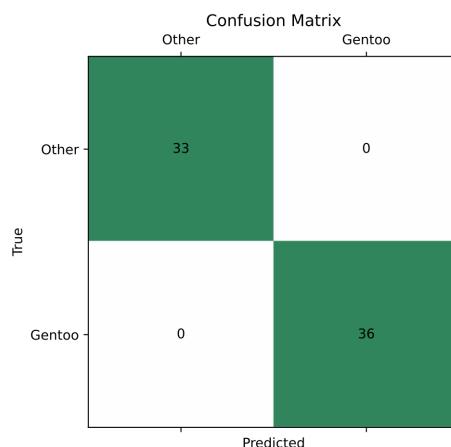


Figur 8: perceptron 1 - beslutningsgrense

Hyperparametersøket (figur 9) bekrefet dette med b-skårer nær 1.0 for alle testede parameterkombinasjoner. I figurene viser blå linje gjennomsnittlig skår med standardavvik, mens grå søyler indikerer samplingfrekvens for hver parameterverdi.

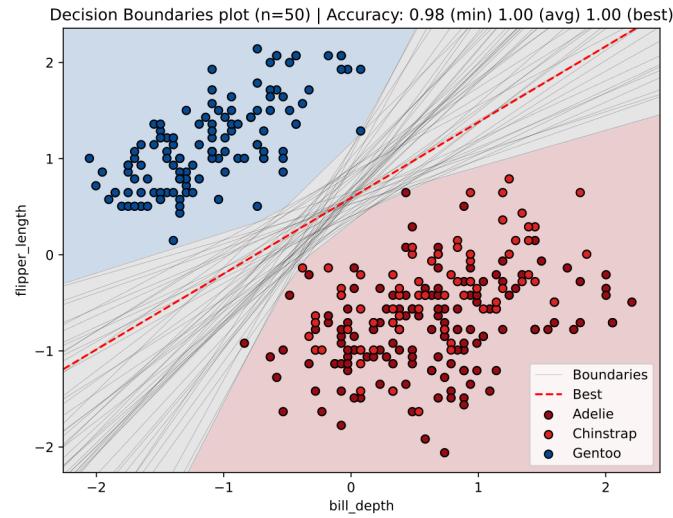


Figur 9: Perceptron 1 - Hyperparametersøk



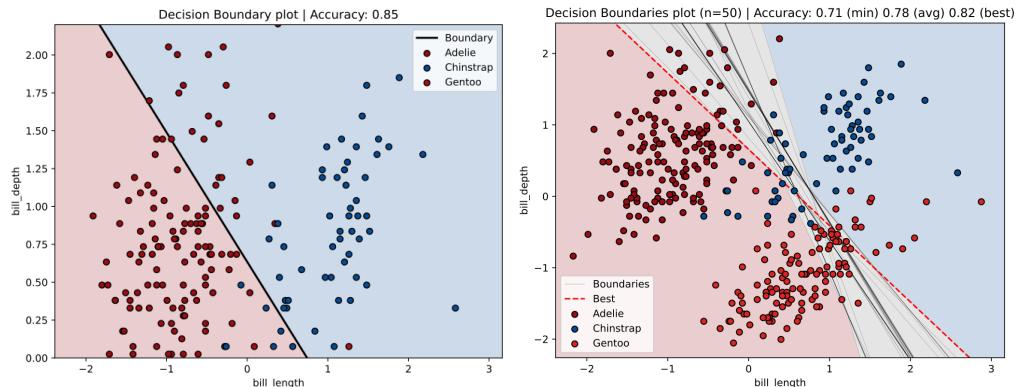
Modellen oppnår perfekt klassifisering på testsettet med b-skår 1.0 (figur 10). En interessant svakhet ved perceptronet vises imidlertid i figur 11, der 50 treninger med identiske hyperparametre gir svært ulike beslutningsgrenser. Selv om alle grensene oppnår høy presisjon, mangler perceptronet evnen til å velge den mest robuste løsningen. Dette kan være problematisk da en beslutningsgrense som ligger tett mot en klasse kan gi dårligere generalisering til nye data enn en som ligger midt mellom klassene.

Figur 10: Perceptron 1,  
Forvirringsmatrise



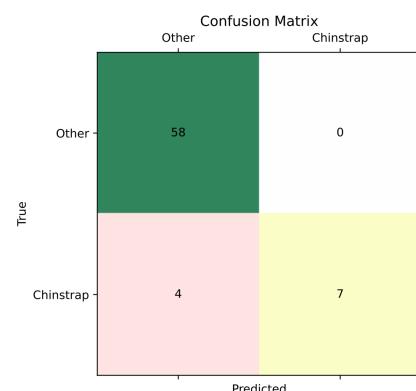
Figur 11: Perceptron 1 - Beslutningslinjer under identiske treningsforhold

## 4.2 Perceptron 2



Figur 12: Perceptron 2 - Beslutningslinjer

I andre eksperiment skulle perceptronet skille **Chinstrap** fra øvrige arter ved hjelp av *bill\_depth* og *bill\_length*. Figur 12 viser at dette er et vanskeligere problem uten klar lineær separasjon. Forvirringsmatrisen i figur 13 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. Klassi-

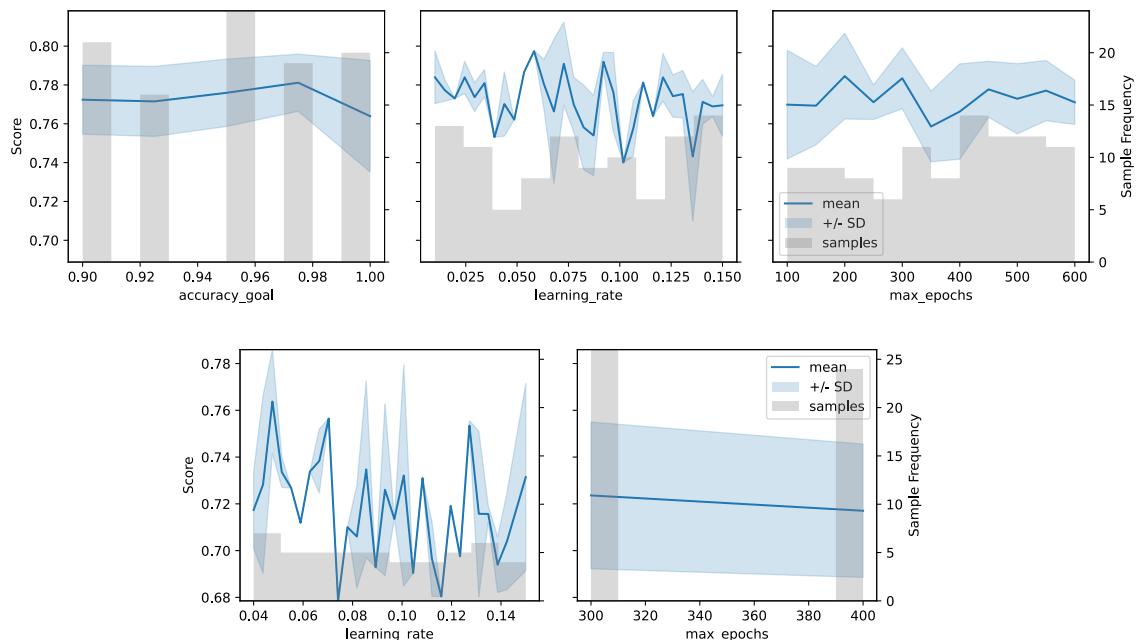


Figur 13: Perceptron 2  
- Forvirringsmatrise

fisering av **Chinstrap** er svakere, med en *b-score* på 0.69. Samlet har de en skåre på 0.81. *Presisjonen* 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.

Hyperparametersøket (figur 14) viser at b-skåren er særlig sensitiv for læringsraten, men svingningene er moderate ( $\pm 0.1$ ). Fornuftige parametervalg viste seg å være læringsrate rundt 0.1 og *max\_epochs* mellom 200 og 300.

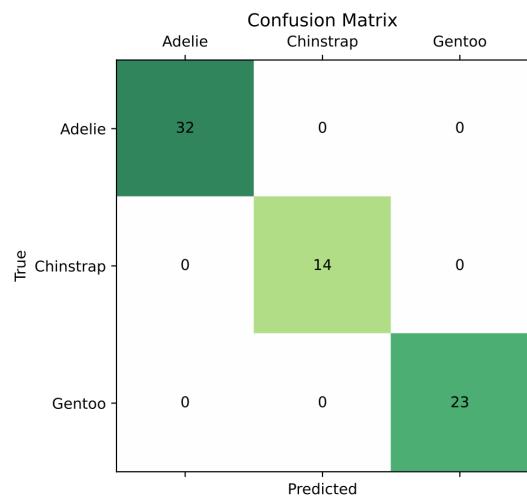


Figur 14: Perceptron 2 - Hyperparametersøk

### 4.3 Perceptron One-vs-All

For det siste perceptron-eksperimentet har vi implementasjonen av et svært enkelt kunstig nevralt 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.

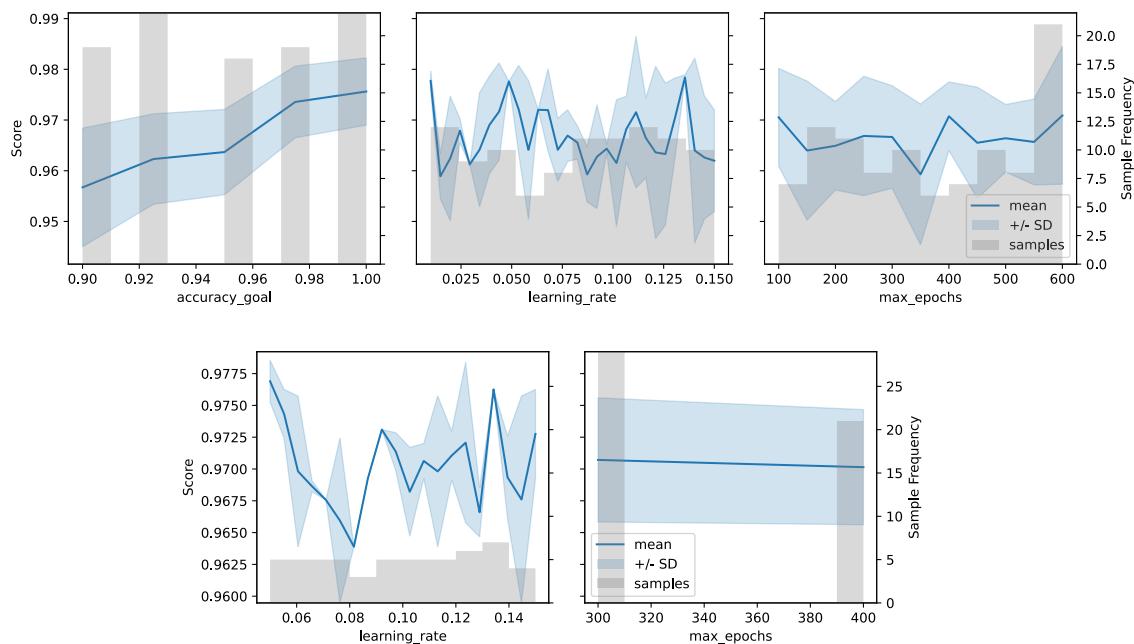
Forvirringsmatrisen i figur 15 viser



Figur 15: Perceptron OVA  
- Forvirringsmatrise

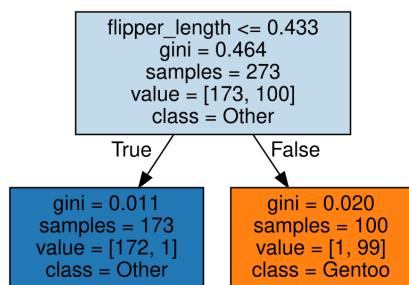
gode resultater for klassifisering av testdata, der alle individene har blitt klassifisert riktig. Dette gir altså høyeste skåre, 1.0, for alle mål på treffsikkerhet og sensitivitet.

Resultater av hyperparametersøkene for denne modellen er i figur 16. Ikke overraskende ser vi noen av de samme tendensene som for eksperimentene 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.



Figur 16: Perceptron OVA - Hyperparametersøk

#### 4.4 Beslutningstre 1 (DT 1)

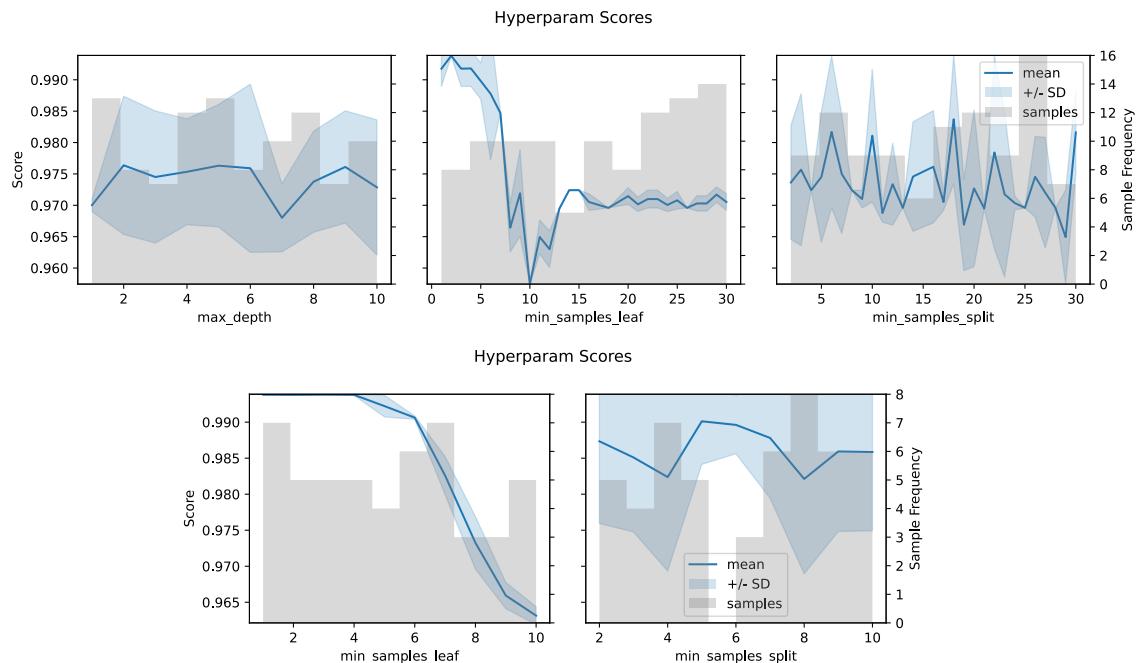


Det første eksperimentet med beslutningstrær brukte samme input og mål som for perceptron: å skille **Gentoo** fra andre klasser basert på *flipper\_length* og *bill\_depth*. Uten beskjæring klassifiserer treet datasettet perfekt, men for å unngå ressursbruk, kompleksitet og overtilpasning har vi beskjært treet både under generering (*pre-pruning*) og etter (*post-pruning*). Etter

Figur 17: Beslutningstre 1 (DT 1)

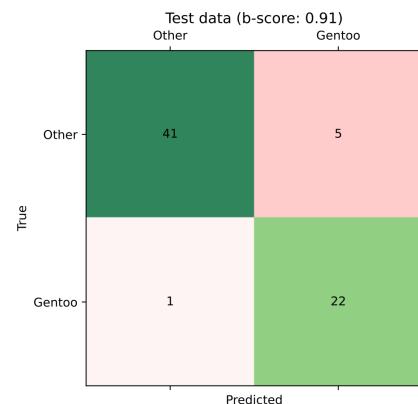
beskjæring fikk vi treet vist i figur 17, hvor hver node viser statistikk som *gini-impurity*, antall individer og splittespørsmål. Nodens farge og klasse bestemmes av den dominerende klassen, der fargen er sterkest ved  $gini = 0$  og svekkes mot hvitt ved  $gini = 0.5$ .

Beskjæring ble styrt av hyperparametre fra søket i 18, som hindrer videre forgrening ved få individer, stor dybde eller små løvnoder. B-skårer på over 0.96 ble oppnådd for alle undersøkte parametere. For å balansere presisjon og kompleksitet valgte vi  $min\_samples\_split = 10$  og  $min\_samples\_leaf = 5$ . Selv om dette ikke ga høyest skårer, unngår vi overtilpasning ved å begrense treets dybde i tråd med datasettets størrelse.



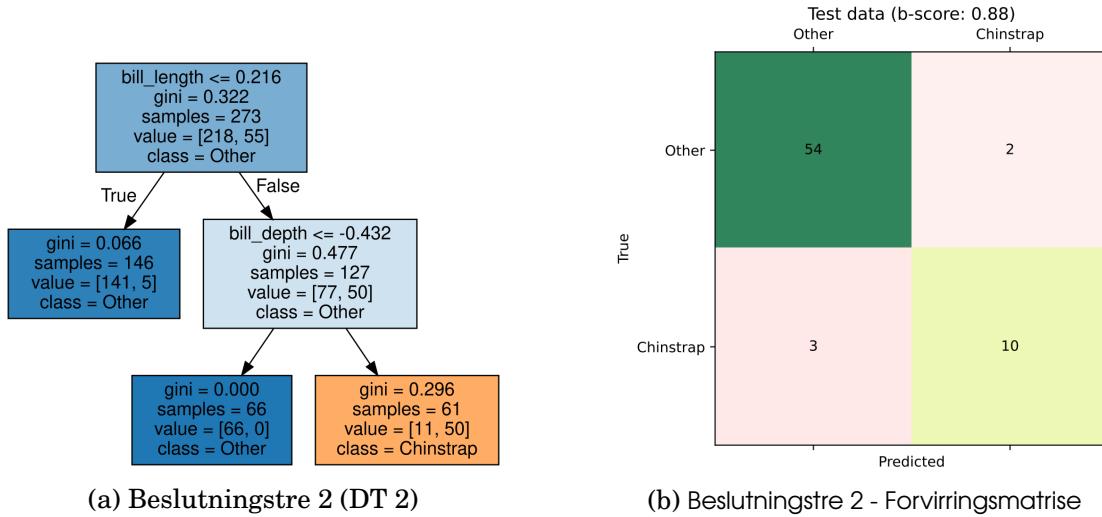
Figur 18: DT 1 - Hyperparametersøk

I figur 19 ser vi forvirringsmatrisen til det ferdigrente beslutnignstreet, målt opp mot prestatasjon 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 19: DT 1  
- Forvirringsmatrise

## 4.5 Beslutningstre 2 (DT 2)



Figur 20: 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 egenskapene *bill\_depth* og *bill\_length*. Resultatet etter beskjæring er i figur 20a.

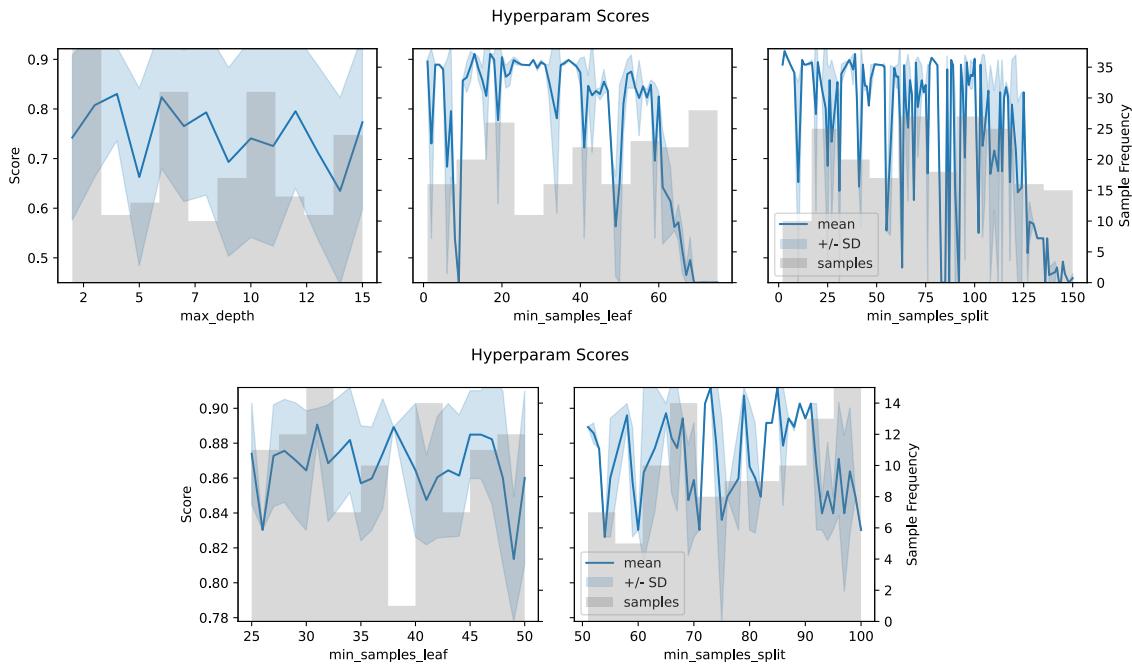
Figur 20b viser at nesten alle individer har blitt klassifisert korrekt. Dette gir *b-skårene* 0.80 for **Chinstrap**, 0.96 for den andre gruppen og 0.88 totalt for alle predikasjonene.

Hyperparametersøket i figur 21 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 søker og tester ble *min\_samples\_split* satt til 100 og *min\_samples\_leaf* til 50.

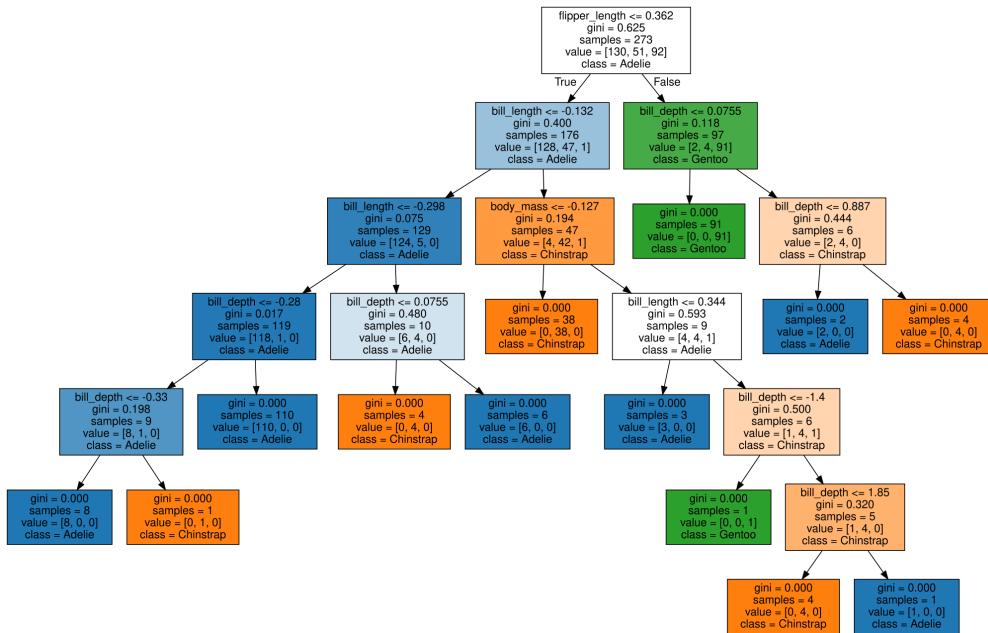
## 4.6 Beslutningstre 3 (DT 3)

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 22.

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. Hyperparametre kan brukes til å stoppe genereringen av treet, før det blir unødig komplekst. Etter beskjæring med hyperparametre ser treet ut som i figur 23.

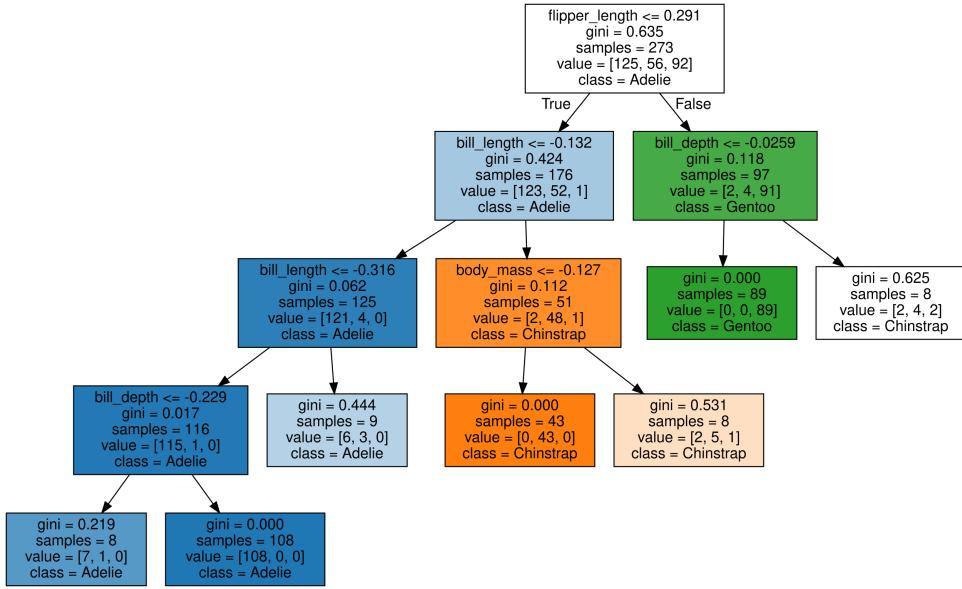


Figur 21: Beslutningstre 2 - Hyperparametersøk



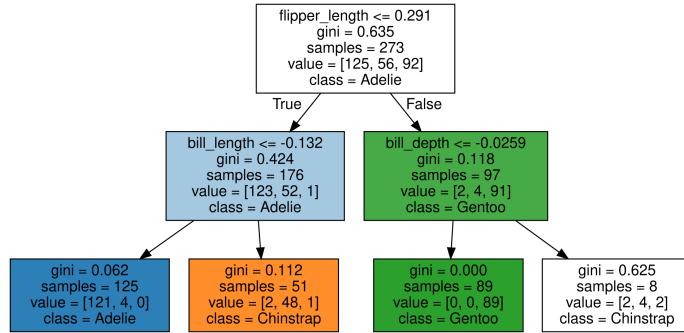
Figur 22: DT 3 uten beskjæring

Dette ser vedre 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*



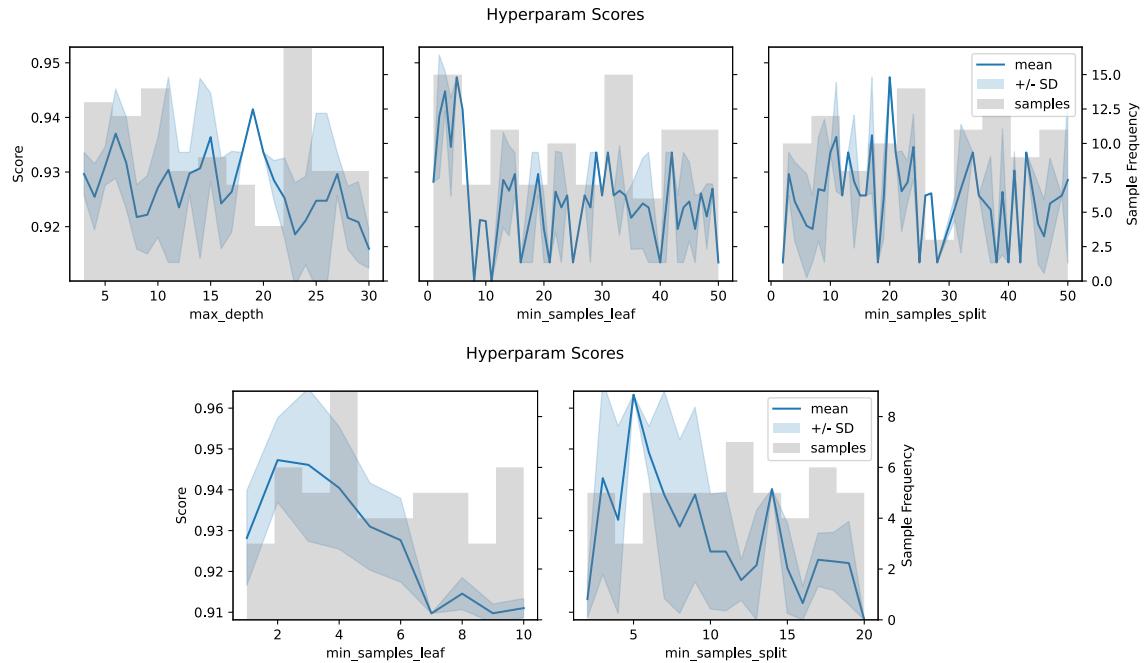
Figur 23: DT 3 før post-pruning

etter at treet er generert. Da står vi igjen med det ferdig beskjærte treet i figur 24.



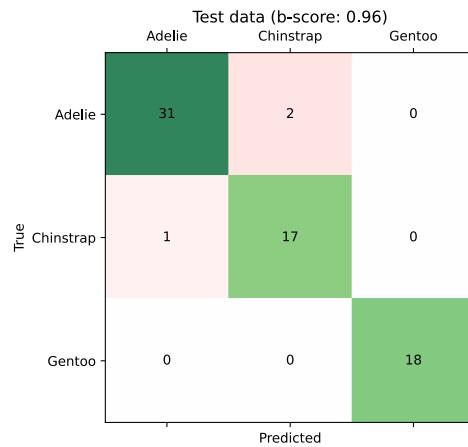
Figur 24: DT 3 etter post-pruning

Resultater av hyperparametersøket for Beslutningstre 3 er i figur 25. Vi har valgt 16 for *min\_samples\_split* og 8 for *min\_samples\_leaf*.



Figur 25: DT 3 - Hyperparametersøk

Modellens ytelse illustreres av forvirringsmatrisen i figur 26. Overordnet *b-score* er 0.96, skåre for **Chinstrap** er 0.94, *Adelie* har *b-skåre* 0.94 og **Gentoo** har skåre 1.0.



Figur 26: DT 3 - Forvirringsmatrise

## 5 Diskusjon og konklusjon

---

### 5.1 Sammenligning av modellene

---

#### 5.1.1 Perceptron 1 og Beslutningstre 1

Begge modellene mestret godt å skille **Gentoo** fra de andre artene ved hjelp av egenskapene *bill\_depth* og *flipper\_length*. Det kom tydelig frem av plot av egenskaper i figur 6 at **Gentoo** var tydelig separert fra de andre artene i dette egenskapsrommet, så det er ikke overraskende at modellene mestret denne oppgaven.

Perceptronet oppnådde perfekt klassifisering av testdata, men avslørte en interessante egenskaper der ulike treninger under like omstendigheter kunne gi helt forskjellige beslutningslinjer, som vist i figur 11. Dette illustrerer en svakhet ved perceptronet, at den ikke kan skille flere løsninger med samme score fra hverandre. Den tilfeldig valgte «beste» løsningen trenger ikke være den som er mest robust i møte med ny data.

Beslutningstreet oppnådde også perfekt klassifisering. Vi så samtidig at modellen naturlig ønsker å overtilpasser seg til testdata, ved å lage mange splitter og regler for å skille alle datapunktene fra hverandre. Etter implementering av beskjæringsalgoritmer, viste det seg at treet kunne beskjæres betydelig uten særlig tap av presisjon.

#### 5.1.2 Perceptron 2 og Beslutningstre 2

Å skille **Chinstrap** fra de andre artene med egenskapene *bill\_depth* og *bill\_length* var en mye vanskeligere oppgave. Vi kan se av figur 6 at disse de to gruppene ikke er lineært separerbare. Det skapte særige utfordringer for perceptron-modellen, som bare kan gjennomføre lineære oppdelinger av egenskapsrommet.

Perceptronet oppnådde en samlet *b-skår* på 0.81, som for så vidt er ganske høyt. Samtidig var det stor forskjell på scorene mellom klassene, 0.69 for **Chinstrap** og 0.93 for den andre gruppen.

Beslutningstreet presterede bedre med en samlet *b-skår* på 0.88. Her så vi også en bedre fordeling mellom klassene, der **Chinstrap** hadde *b-skår* 0.80.

Som forventet mestret beslutningstreet bedre denne klassifiseringsoppgaven, da den ikke er avhengig av lineært separerbare data slik som perceptronet er.

#### 5.1.3 Perceptron OVA og Beslutningstre 3

I den siste gruppen av eksperimenter ble alle tilgjengelige numeriske egenskaper tatt i bruk for å multiklasse-klassifisering.

Neuralnettet i Perceptron OVA oppnådde overraskende god ytelse med perfekt klassifisering av testsettet. Det kan tenkes at de ekstra dimensjonene fra flere egenskaper gjorde egenskapsrommet lineært separerbart.

I forsøkene med beslutningstrær demonstrerte hvordan beskjæring effektivt kan redusere treets kompleksitet, uten å ofre særlig presisjon. Det ubeskjærte treet hadde 25 noder, som ble kuttet ned til 13 noder gjennom *pre-pruning*. Ved ytterligere beskjæring etter treningen var ferdig, ble antall noder redusert til 7. Til tross for kompleksitetsreduksjonen ble god klassifiseringsnøyaktighet opprettholdt, med *b-skår* på 1.0 for **Gentoo**, 0.95 for **Adelie** og 0.92 for **Chinstrap**.

Ytelsesskårene for det beskjærte beslutningstreet ligger noe under den «perfekte» skåren for *Perceptron OVA*. Dette kan bety at perceptron-nettverket er overtilpasset. Siden vi ikke hadde tilgang til andre testdata, er det vanskelig å si hvor mye vi mistet eller vant på beskjæringen av beslutningstreet.

## 5.2 Begrensninger og videre arbeid

---

### Databegrensninger

Datasettet var ubalansert i forhold til antall observasjoner av hver klasse, og vi hadde lite tilgjengelig data til å teste og utfordre modellene våre på andre datasett. Dermed er det vanskelig å vurdere generaliserbarheten til modellene, eller om de kun er presise for den data som var tilgjengelig under eksperimentet.

### Modeller

I implementasjonen av *Perceptron OVA* er det ikke tatt høyde for «kollisjoner» der flere perceptroner aktiveres og gir signal om at individet tilhører den klassen perceptronet er trent til å gjenkjenne. Det vil si at det perceptronet som er gitt den minste indeksen i nettverket vil bli prioritert først. En mulig forbedring er å bytte ut aktiveringsfunksjonen med en sigmoid-funksjon, da denne gir en gradert aktivering som er relatert til hvor «sikker» perceptronet er på klassifiseringen.

Den nevnte svakheten ved perceptronet, at den ikke den evner å sortere mellom flere løsninger som gir samme score, kan løses ved å ta i bruk andre relaterte modeller som *Support Vector Machine (SVM)* [2].

For beslutningstreet kan det være interessant å se på flere beskjæringsstrategier, som mer avanserte *post-pruning* metoder.

### Målinger

Selv om *KFold* ble tatt i bruk under hyperparametersøk, har skårene for de endelige modellene kun blitt beregnet ut fra én enkelt randomisert deling av datasettet i test og treningsdata. Bruk av flere delinger av datasettet, for eksempel med *KFold*, for så å måle og se på snittet av prestasjoner for modeller trent på ulike delsett av datasettet, ville vært interessant for å få bedre innsikt i modellenes prestasjon.

## 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/>.
- [4] K. E. Tranøy, *induksjon – filosofi*, i *Store Norske Leksikon*, 2024. [Online]. Hentet fra: [https://snl.no/induksjon\\_-\\_filosofi](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.