Palmer penguins: Adelie, Chinstrap, Gentoo

Karaktersatt oppgave 2 | DTE-2602 | H23

Almaz Ermilov
almaz.ermilov@uit.no

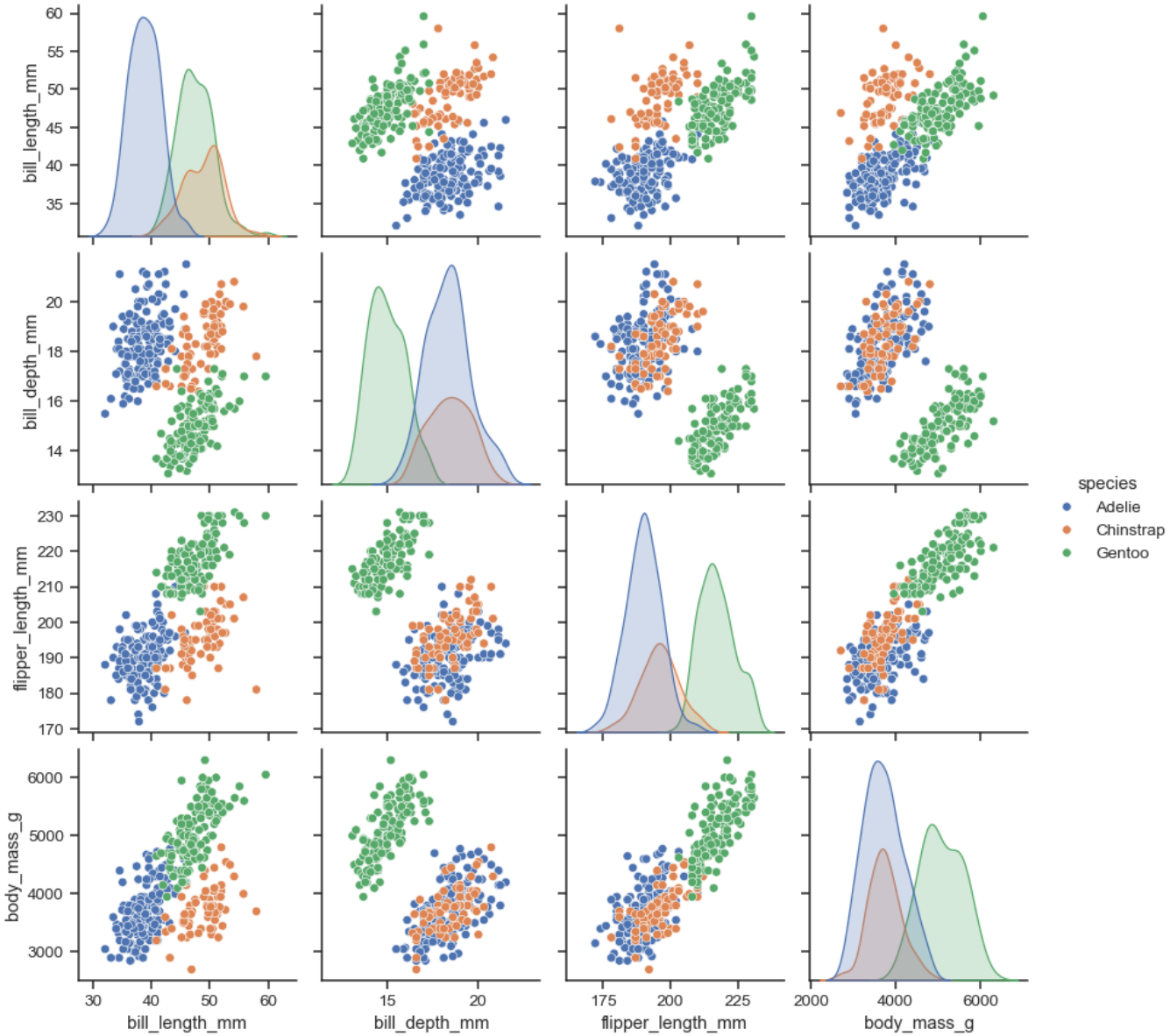
Table of Contents

Introduksjon

I denne oppgaven skal jeg bruke 2 ulike varianter av "supervised learning" for å prøve å artsbestemme pingviner basert på ulike målinger. Datasettet som skal brukes er "Palmer penguins" (datasettet palmer_penguins.csv).

Figuren under viser ulike varianter av "scatterplott" for de fire numeriske størrelsene i datasettet: bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g

Plottene viser alle parvise kombinasjoner av disse fire. Diagonalen viser histogram for enkelt-størrelser. Legg merke til hvordan noen kombinasjoner gir punktskyer der de tre klassene (pingvinartene) overlapper mye, mens andre kombinasjoner gir bedre separasjon.



Figur 1 – Scatterplots for Adelie, Chinstrap, Gentoo

Oppgavebeskrivelse

Forberede data og vurdere resultater

1. Les inn datasettet fra palmer_penguins.csv. Fjern alle rader som mangler data (rader som inneholder "NA"). Legg data for de fire numeriske størrelsene (bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g) i en matrise X. Normaliser verdiene i hver kolonne i X gjennom å trekke fra gjennomsnittsverdi for kolonnen og dele på standardavviket ("Z-score"). Konverter informasjon om art ("species") til heltall 0, 1 og 2 for henholdsvis Adelie, Chinstrap og Gentoo, og legg dette i en vektor y. Implementeres som funksjon read_data(). **5 poeng**
2. Del opp datasettet ditt i et trenings-datasett (X_train,y_train) og et test-datasett (X_test,y_test). Gi en begrunnelse for hvor mange observasjoner (samples) du legger i hvert av disse. Implementeres som funksjon train_test_split(). **5 poeng**
3. Lag en funksjon accuracy() som tar en vektor y_true med "sanne" verdier og en vektor y_pred med estimerte verdier som input. Funksjonen skal beregne hvor stor andel av elementene som er *like* i de to vektorene. Vi skal bruke dette senere for å vurdere samsvar mellom "fasit" og output fra våre maskinlæringsmodeller. **5 poeng**

Perceptron

4. Implementer en klasse Perceptron. Klassen skal ha metodene .train() og .predict(), som indikert i startkoden. Metodene brukes til henholdsvis trening av modellen og anvendelse av modellen på nye data. **15 poeng**
5. Bruk datasettet (X_train,y_train) til å trene en Perceptron-modell til å gjenkjenne pingvinarten **Gentoo**. Du skal kun bruke kolonnene *bill_depth_mm* og *flipper_length_mm* fra X-matrisa. Output fra modellen (y_pred) skal være 1 hvis arten er Gentoo, og 0 hvis den ikke er det (dvs. en av de andre to artene). Du kan bruke funksjonen convert_y_to_binary() for å konvertere til binære verdier. Bruk datasettet (X_test,y_test) sammen med funksjonen accuracy() fra oppgave 3 for å måle nøyaktigheten til modellen. **10 poeng**
6. Visualiser "decision boundary" for modellen i oppgave 5. Lag et plott som viser hver pingvin som et punkt (bruk matplotlib.pyplot.scatter()). Plott så decision boundary for modellen som en rett linje (bruk hjelpemetoden get_line_x_y(), og matplotlib.pyplot.plot()). Hvis alt er riktig bør linja ligge mellom Gentoo-punktene og punktene for de to andre artene. **5 poeng**
7. Lag et nytt perceptron som skal skille arten **Chinstrap** fra de to andre. Du står fritt til å bruke hele X-matrisa eller et utvalg av kolonnene fra X. Gi en begrunnelse for hvilke features du velger. Merk at det ikke er sikkert at modellen konvergerer. Forklar i så fall hvorfor. Mål nøyaktigheten til modellen med (X_test,y_test) og accuracy(). **10 poeng**

Decision tree

8. Implementer funksjonene gini_impurity(), gini_impurity_reduction() og best_split_feature_value() som vist i startkoden. **15 poeng**
9. Implementer en klasse DecisionTree. Deler av implementasjonen er allerede ferdig og er oppgitt i startkoden. Du skal implementere den rekursive metoden ._predict(), som bruker et ferdig trent beslutningstre for å klassifisere nye data. **10 poeng**
10. Bruk datasettet (X_train,y_train) til å trene en modell for å gjenkjenne arten **Gentoo**, på samme måte som i oppgave 5. Mål nøyaktigheten med (X_test,y_test) og accuracy(). **5 poeng**
11. Bruk datasettet (X_train,y_train) til å trene en modell for å gjenkjenne arten **Chinstrap**, på samme måte som i oppgave 7. Mål nøyaktigheten med (X_test,y_test) og accuracy(). **5 poeng**
12. Bruk datasettet (X_train,y_train) til å trene en modell som skal skille mellom **alle tre pingvinartene**. Kommentér i rapporten: Hvorfor er klassifisering av mer enn to kategorier mulig med et decision tree, og ikke med et perceptron? Mål nøyaktigheten med (X_test,y_test) og accuracy(). **5 poeng**.

Oppsummering

13. Sammenlign resultatene for de ulike modellene du har trent og testet. Er det forskjeller mellom resultatene? Hva er sannsynlige grunner til det, i så fall? **5 poeng**

Solution

Forberede data og vurdere resultater

1. read_data(path) funksjonen behandler pingvin-data fra en fil. Den åpner en csv-fil som inneholder pingvin-data og leser den, fjerner den alle rader som mangler informasjon, tar fire typer målinger (som "bill length") fra dataene og endrer dem til en felles skala (normalisering), slik at de blir enklere å sammenligne og analysere. Deretter endres navnene på pingvinartene til tall (f.eks. 0 for "Adelie") for å gjøre det enklere å behandle dataene. Så funksjonen returnerer to datasett: ett med de normaliserte målingene og ett med artsnumrene.

   Jeg bruker funksjonen i main_perceptron() og main_decision_tree() i «pre-processing»
   (linje **476 og 545** i supervised_learning.py).

2. train_test_split(X,y,train_frac) funksjonen deler et datasett i to deler: en for trening og en for testing. Den mottar et datasett (X), verdiene som skal predikeres (y) og andelen data som skal brukes til opplæring (train_frac), og omorganiserer deretter dataene tilfeldig slik at delingen blir tilfeldig. Basert på den angitte opplæringsfraksjonen deler den dataene i to deler: opplæringsdatasettet (delen definert av train_frac) og testdatasettet (den resterende delen). Så funksjonen gir altså tilbake fire grupper med data - opplæringsdata (X_train, y_train) og testdata (X_test, y_test).

   Funksjonen brukes i main_perceptron() og main_decision_tree() i «pre-processing»
   (linje **476 og 545** i supervised_learning.py).

3. accuracy(y_pred, y_true) funksjonen beregner hvor ofte prediksjonene stemmer overens med de faktiske verdiene. Den tar to lister: én med predikerte verdier og én med sanne verdier, og returnerer deretter et tall mellom 0 og 1 som viser andelen korrekte prediksjoner.

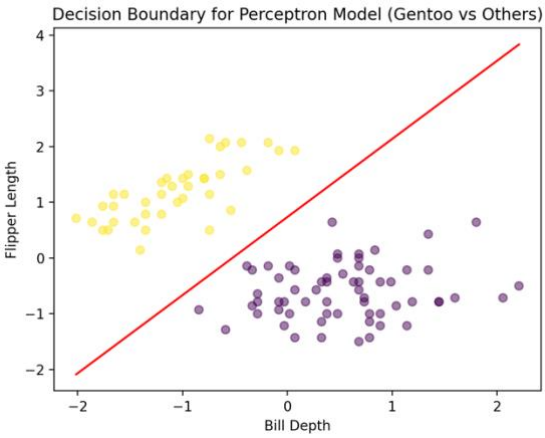
   Funksjonen brukes i main_perceptron() og main_decision_tree() på slutten for å teste perceptron-er og decision trær.

Perceptron

4. train-metoden i Perceptron-klass justerer perceptronets vekter og bias basert på treningsdata (X) og deres sanne labels (y). Dette gjøres gjentatte ganger i et bestemt antall ganger (epoker) eller til vektene stabiliserer seg (konvergerer). Hvis en prediksjon er feil, oppdaterer den vektene og skjevheten for å forbedre fremtidige prediksjoner. Denne prosessen styres av læringshastigheten. Metoden sjekker også om treningen har konvergert og stopper tidlig hvis den har det.
   Vær så snill sjekk **linje 148** i supervised_learning.py.

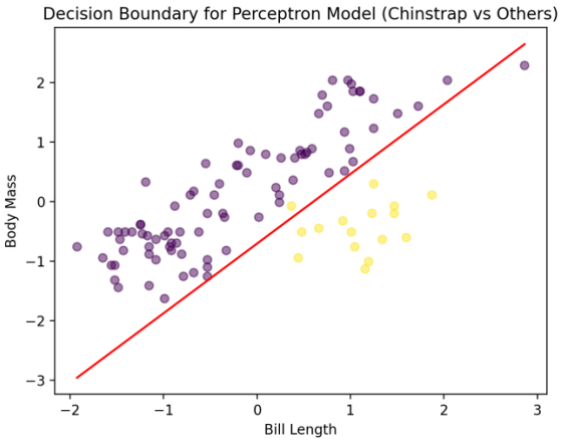
   predict-metoden i Perceptron-klass tar en input (x) og bruker perceptronens nåværende vekter og bias til å lage en prediksjon. Hvis den beregnede verdien er større enn 0, predikerer den 1, ellers predikerer den 0 (linje **133** i supervised_learning.py).

5. Først gjør jeg en pre-processing, starter med å dele et datasett med pingvinegenskaper inn i et opplærings- og et testsett (train_frac = 0.7, så de fleste data går til training). Dataene behandles med tanke på å identifisere Gentoo spesielt. Deretter brukes treningssettet til å trene opp en Perceptron-modell for å skille **Gentoo**-pingviner fra andre pingviner, basert på bill_depth og flipper_length (bestemt i oppgaven). Etter opplæringen testes modellen på testsettet. Nøyaktigheten til klassifiseringen av eselpingviner beregnes, og en beslutningsgrense plottes inn (figuren til høyre) for å visualisere modellens perfomance.



6. Så alt ser bra ut, vi har lineær boundary division som vi kan se visuelt og accuracy_gentoo blir 1.0 / 100 % på testdata. Vi kan si at bill_depth og flipper_length fungerer godt og har nok «sensitivitet» til å gi gode resultater for å identifisere Gentoo.

   Perceptron-læringsprosessen i Gentoo konvergerer raskt fordi vi har en klar grense mellom Gentoo og resten, og det er ingen overlapping av inndataparameterverdier (alt er entydig svart-hvitt på en måte). (Mer informasjon finner du på linje **476-508** i supervised_learning.py)



7. Vi lager det samme prosess konseptuelt for å skille Chinstrap fra andre pingviner. Nå må vi velge veldig forsiktig «features» som kan hjelpe oss, det er veldig viktig. Fra figur 1 «Scatterplots for Adelie, Chinstrap, Gentoo» kan vi si at body_mass + bill_length kan fungere ganske bra. Vi må ha inndataparameterverdier som kan løse oppgaven, så vi må ha «*sensitivitet*» for vår predicted labels. Derfor vår accuracy blir 0.96 eller 96 %.

   Et svært viktig spørsmål om konvergens. Sannsynligvis vil modellen vår aldri konvergere på grunn av overlappingen i dataene for inngangsparametrene våre. Jeg prøvde å leke med hyperparametere (læringshastighet og epoker), men det påvirket ikke resultatet så mye pga overlapping av parameterverdier, og Perceptron gir bare 0 eller 1 i utdataene. **Perceptron er en enkel, lineær modell, noe som betyr at den forsøker å skille data ved hjelp av en rett linje (i 2 dimensjoner) eller et flatt plan (i 3 dimensjoner).**
   Mer informasjon finner du på linje **510-543** i supervised_learning.py.

Decision tree

8. gini_impurity() beregner Gini impurity til en liste med klasse labels. Gini impurity måler hvor ofte et tilfeldig valgt element vil bli feilaktig identifisert. Det er et tall mellom 0 og 1, der 0 betyr at alle elementene er like.
   gini_impurity_reduction() måler reduksjonen i Gini impurity etter å ha delt et datasett i to deler ("venstre" og "høyre"). Den beregner hvor mye urenheten reduseres ved denne delingen, noe som bidrar til å vurdere hvor effektiv delingen er.
   best_split_feature_value() finner den beste måten å dele opp et datasett på for å redusere Gini impurity. Den ser på alle funksjonene og verdiene deres for å finne den delingen som gir størst reduksjon i urenhet. Returnerer reduksjonen i urenhet, hvilken funksjon som skal splittes og verdien som skal splittes. Mer informasjon finner du på linje **216-299** i supervised_learning.py.

9. Metoden ._predict() klassifiserer dataene basert på Decision Tree struktur. Hvis den støter på en bladnode, returnerer den klassen som er knyttet til dette bladet for alle datapunkter. Hvis det er en forgreningsnode, deler den opp dataene i venstre og høyre delmengde og kaller seg selv rekursivt for å klassifisere hver delmengde. Resultatene fra venstre og høyre delmengde kombineres for å danne den endelige prediksjonen. Vær så snill sjekk **linje 427** i supervised_learning.py.

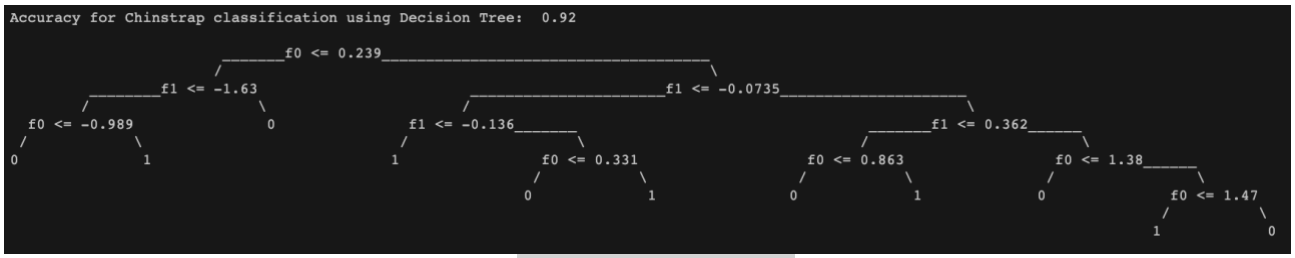
10. Decision tree for Gentoo ble implementert (basert på bill_depth + flipper_length) og jeg fikk accuracy 100%, og treet ser sånt ut:

```
Accuracy for Gentoo classification using Decision Tree:  1.0

        _______f1 <= 0.356_______
       /                         \
   f0 <= -1.4                  f0 <= 0.071
   /         \                 /          \
  1           0               1            0
```

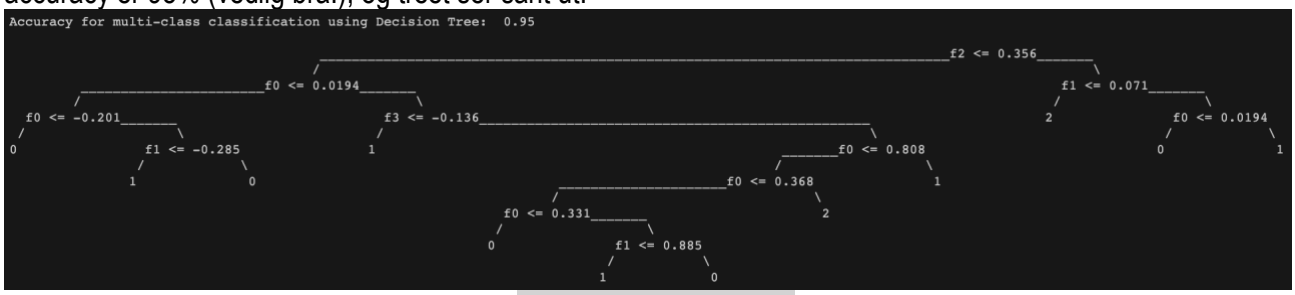
   Mer informasjon finner du på linje **545-566** i supervised_learning.py.

11. Decision Tree for Chinstrap ble implementert (basert på body_mass + bill_length), fikk accuracy 92%, og treet ser sånt ut:



Mer informasjon finner du på linje **568-582** i supervised_learning.py.

12. Decision Tree for alle typer pingviner ble implementert (basert på alle features), accuracy er 95% (vedlig bra!), og treet ser sånt ut:



Mer informasjon finner du på linje **584-593** i supervised_learning.py.

Veldig viktig spørsmål her om hvordan det er mulig å forutsi 3 forskjellige typer pingviner med Decision Tree, men umulig med Perceptron? Som jeg skrev tidligere, er Perceptron en enkel lineær modell, noe som betyr at den prøver å skille data ved hjelp av en rett linje og gir 0 eller 1 i resultatet. Det er umulig for Perceptron hvis vi har 3 eller flere forskjellige typer pingviner som vi skal predikere. **Decision Tree kan ta en rekke ulike beslutninger, slik at det kan skille ut mange typer ting i egne grupper, og nøkkelen til denne evnen til å håndtere flere klasser ligger i deres hierarkiske, multi-split struktur.**

Oppsummering og selvrefleksjon

Alt fungerte bra, og jeg oppdaget Decision Tree for meg selv, som er et veldig raskt og enkelt, ganske universelt verktøy å bruke.

Det viktigste jeg har lært eller innsett igjen:

- For å få gode resultater må vi ha et godt datagrunnlag (pre-process dem om trenges, filtrere osv, «garbage in, garbage out»), forstå dataene og målene godt, slik at vi kan definere hvilke funksjoner som er best å bruke, og hvilke funksjoner som kan ødelegge det forventede resultatet. Det er også viktig å kjenne egenskapene til datasettet for å velge riktig modell og å prøve ulike modeller og hyperparametere for å finne den beste tilnærmingen for et gitt datasett.
- Forskjellen mellom Perceptron og Decision Tree:
  Perceptron er en enkel, lineær modell som justerer vekter basert på treningsdata, konvergerer med klare grenser mellom klassene.
  Decision Tree er en mer kompleks modell som kan håndtere flere klasser og ikke-lineære sammenhenger. Den bruker Gini impurity og trestrukturer for beslutningstaking.

  Perceptroner kan være raskere til å trene og predikere med enkle, lineære datasett, mens Decision Tree kan være mer effektive med komplekse, ikke-lineære data.
- Resultater:
  For Gentoo oppnådde både Perceptron og Decision Tree en nøyaktighet på 100%. Dette skyldes sannsynligvis tydelige, distinkte egenskaper som skiller Gentoo fra andre.
  For Chinstrap var Perceptrons nøyaktighet ca. 96%, Decision Tree varierte mellom 90-95% avhengig av de valgte egenskapene.
  Decision Tree for alle 3 typer pingviner (alle features) sin accuracy er ca. 95%.

Bibliography

Jansson, A. D., & Skjelvareid, M. (2023). *Modul 7: Supervised learning.* Hentet fra Canvas: https://uit.instructure.com/courses/30442/modules/items/912466