Palmer penguins: Adelie, Chinstrap, Gentoo
Character set task 2 | DTE-2602 | H23

Almaz Ermilov
almaz.ermilov@uit.no

Table of Contents

Introduction

In this thesis I will use 2 different variants of supervised learning to try to determine the species of penguins based on different measurements. The dataset to be used is "Palmer penguins" (dataset palmer_penguins.csv).

The figure below shows different variants of scatter plots for the four numerical variables in the data set: bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g

The plots show all pairwise combinations of these four. The diagonal shows histograms for individual sizes. Notice how some combinations give point clouds where the three classes (penguin species) overlap a lot, while other combinations give better separation.

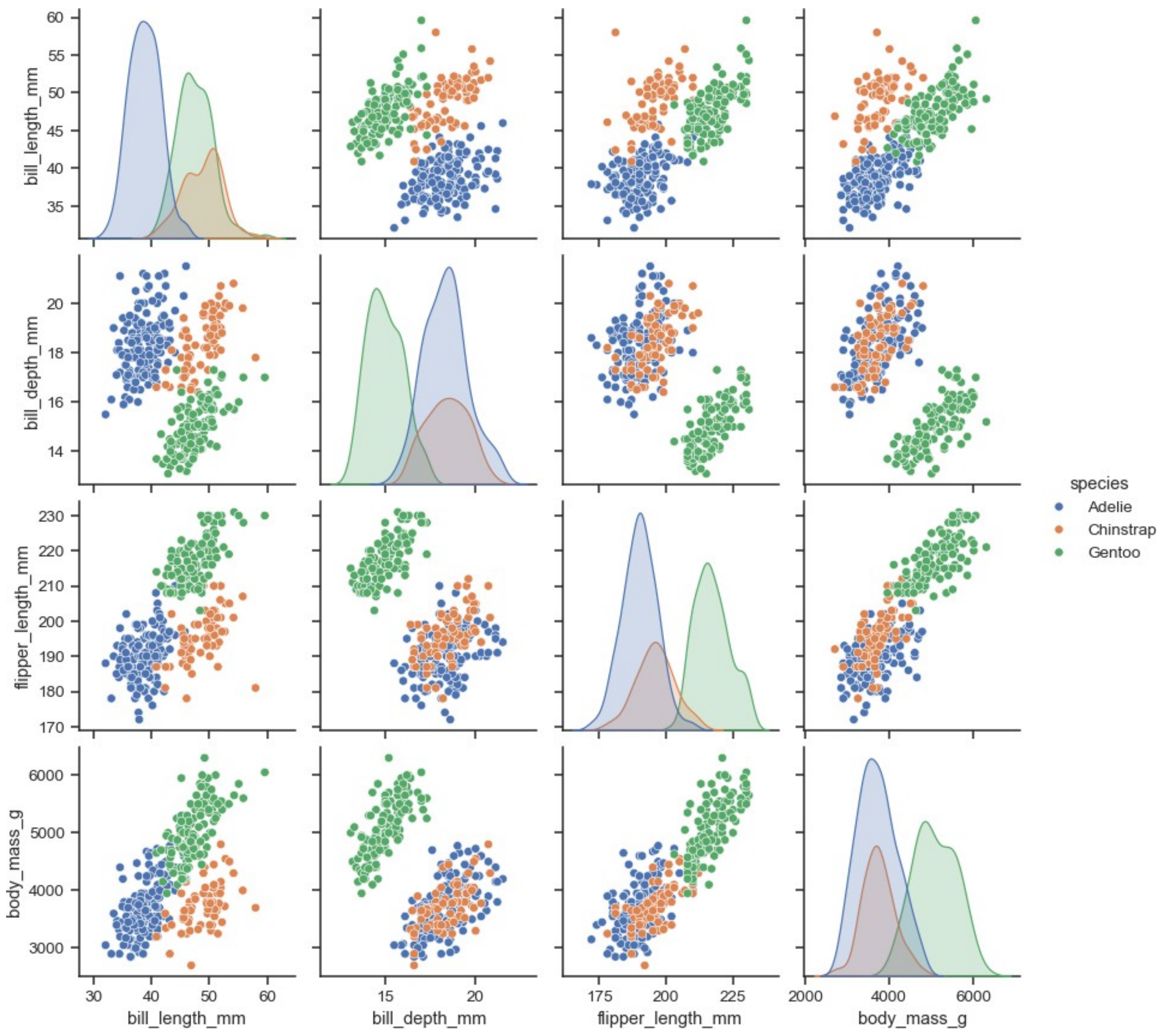


Figure 1 - Scatterplots for Adelie, Chinstrap, Gentoo

Task description

Prepare data and assess results

1. Load the dataset from palmer_penguins.csv. Remove all rows with missing data (rows containing "NA"). Place the data for the four numerical quantities (bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g) in a matrix X. Normalise the values in each column in X by subtracting the mean value for the column and dividing by the standard deviation ("Z-score"). Convert species information to integers 0, 1 and 2 for Adelie, Chinstrap and Gentoo respectively, and put this in a vector y. Implemented as function read_data(). **5 points**
2. Divide your dataset into a training dataset (X_train,y_train) and a test dataset (X_test,y_test). Provide a justification for how many observations (samples) you put in each of these. Implement as function train_test_split(). **5 points**
3. Create an accuracy() function that takes a vector y_true with "true" values and a vector y_pred with estimated values as input. The function will calculate the proportion of elements that are *similar* in the two vectors. We will use this later to assess the correspondence between the "facet" and the output from our machine learning models. **5 points**

Perceptron

4. Implement a class Perceptron. The class should have the methods .train() and .predict(), as indicated in the start code. The methods are used for training the model and applying the model to new data, respectively. **15 points**
5. Use the dataset (X_train,y_train) to train a Perceptron model to recognise the penguin species **Gentoo**. You should only use the columns *bill_depth_mm* and *flipper_length_mm* from the X-matrix. The output from the model (y_pred) should be 1 if the species is Gentoo, and 0 if it is not (i.e. one of the other two species). You can use the function convert_y_to_binary() to convert to binary values. Use the dataset (X_test,y_test) together with the accuracy() function from task 3 to measure the accuracy of the model. **10 points**
6. Visualise the decision boundary for the model in task 5. Create a plot showing each penguin as a point (use matplotlib.pyplot.scatter()). Then plot the decision boundary for the model as a straight line (use the helper method get_line_x_y(), and matplotlib.pyplot.plot()). If everything is correct, the line should lie between the Gentoo points and the points for the other two species. **5 points**
7. Create a new perceptron to distinguish the species **Chinstrap** from the other two. You are free to use the entire X matrix or a selection of the columns from X. Provide a justification for the features you choose. Note that the model may not converge. If so, explain why. Measure the accuracy of the model with (X_test,y_test) and accuracy(). **10 points**

Decision tree

8. Implement the functions gini_impurity(), gini_impurity_reduction() and best_split_feature_value() as shown in the starter code. **15 points**
9. Implement a DecisionTree class. Parts of the implementation are already completed and are provided in the starter code. You will implement the recursive method ._predict(), which uses a pre-trained decision tree to classify new data. **10 points**
10. Use the dataset (X_train,y_train) to train a model to recognise the species **Gentoo, in the** same way as in task 5. Measure the accuracy with (X_test,y_test) and accuracy(). **5 points**
11. Use the dataset (X_train,y_train) to train a model to recognise the species **Chinstrap, in the** same way as in task 7. Measure the accuracy with (X_test,y_test) and accuracy(). **5 points**
12. Use the dataset (X_train,y_train) to train a model that will distinguish between **all three penguin species**. Comment in the report: Why is classification of more than two categories possible with a decision tree, and not with a perceptron? Measure accuracy with (X_test,y_test) and accuracy(). **5 points**.

Summarising

13. Compare the results for the different models you have trained and tested. Are there differences between the results? If so, what are the likely reasons? **5 points**

Solution

Prepare data and assess results

1. The read_data(path) function processes penguin data from a file. It opens a csv file containing penguin data and reads it, removes any rows that are missing information, takes four types of measurements (like "bill length") from the data and changes them to a common scale (normalisation), making them easier to compare and analyse. It then changes the names of the penguin species to numbers (e.g. 0 for "Adelie") to make it easier to process the data. So the function returns two datasets: one with the normalised measurements and one with the species numbers.

   I use the function in main_perceptron() and main_decision_tree() in "pre-processing" (lines **476 and 545** in supervised_learning.py).

2. The train_test_split(X,y,train_frac) function splits a dataset into two parts: one for training and one for testing. It receives a dataset (X), the values to be predicted (y) and the proportion of data to be used for training (train_frac), and then randomly reorganises the data so that the split is random. Based on the specified training fraction, it splits the data into two parts: the training dataset (the part defined by train_frac) and the test dataset (the remaining part). So the function returns four groups of data - training data (X_train, y_train) and test data (X_test, y_test).

   The function is used in main_perceptron() and main_decision_tree() in pre-processing (lines **476 and 545** in supervised_learning.py).

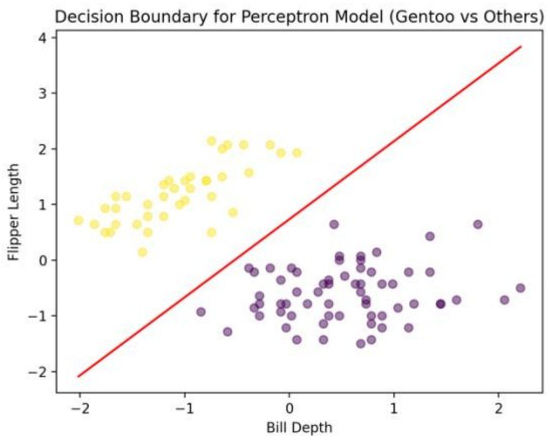
3. The accuracy(y_pred, y_true) function calculates how often the predictions match the actual values. It takes two lists: one with predicted values and one with true values, and then returns a number between 0 and 1 that shows the proportion of correct predictions.

   The function is used in main_perceptron() and main_decision_tree() at the end to test perceptrons and decision trees.

Perceptron

4. The Perceptron-class train method adjusts the perceptron's weights and bias based on training data (X) and their true labels (y). This is done repeatedly for a set number of times (epochs) or until the weights stabilise (converge). If a prediction is wrong, it updates the weights and bias to improve future predictions. This process is governed by the learning rate. The method also checks if the training has converged and stops early if it has.
   Please check **line 148** in supervised_learning.py.

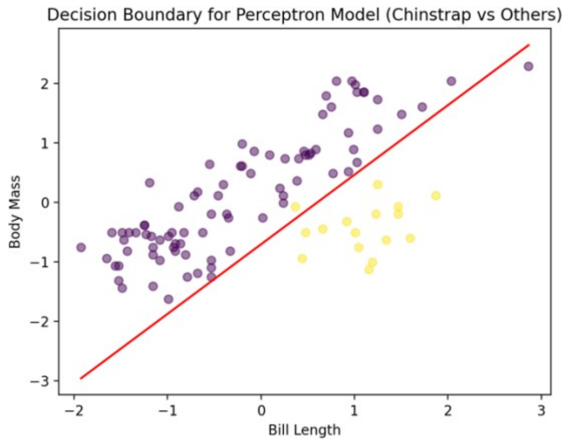
   The predict method in Perceptron class takes an input (x) and uses the perceptron's current weights and bias to make a prediction. If the calculated value is greater than 0, it predicts 1, otherwise it predicts 0 (line **133** in supervised_learning.py).



5. First, I do a pre-processing, starting by splitting a dataset with penguin characteristics into a training and a test set (train_frac = 0.7, so most data goes to training). The data is processed with a view to identifying Gentoo specifically. The training set is then used to train a Perceptron model to distinguish Gentoo penguins from other penguins, based on <u>bill_depth and flipper_length (determined in the task)</u>. After training, the model is tested on the test set. The accuracy of t h e classification of gentoo penguins is calculated and a decision boundary is plotted (figure on the right) to visualise the model's performance.

6. So everything looks good, we have linear boundary division that we can see visually and accuracy_gentoo becomes 1.0/100 % on test data. We can say that bill_depth and flipper_length work well and have enough "sensitivity" to give good results for identifying Gentoo.

   The Perceptron learning process in Gentoo converges quickly because we have a clear boundary between Gentoo and the rest, and there is no overlap of input parameter values (everything is unambiguously black and white in a way). (More information can be found on lines **476-508** in supervised_learning.py)



7. We create the same process conceptually to distinguish Chinstrap from other penguins. Now we have to choose very carefully "features" that can help us, it is very important. From Figure 1 "Scatterplots for Adelie, Chinstrap, Gentoo" we can say that body_mass + bill_length can work quite well. We need to have input parameter values that can solve the task, so we need to have "*sensitivity*" for our predicted labels. Therefore our accuracy will be 0.96 or 96%.

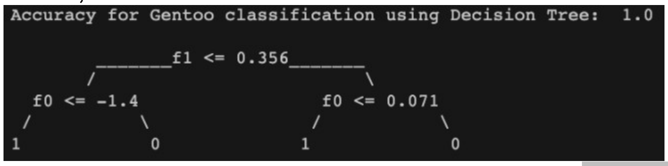
   A very important question about convergence. Probably our model will never converge because of the overlap in the data for our input parameters. I tried to play around with hyperparameters (learning rate and epochs), but it affected the result is not much due to the overlap of parameter values, and Perceptron only gives 0 or 1 in the output. **Perceptron is a simple linear model, meaning that it attempts to separate data using a straight line (in 2 dimensions) or a flat plane (in 3 dimensions).**
   More information can be found on lines **510-543** in supervised_learning.py.

Decision tree

8. gini_impurity() calculates the Gini impurity of a list of class labels. Gini impurity measures how often a randomly selected item will be incorrectly identified. It is a number between 0 and 1, where 0 means that all items are equal.
   gini_impurity_reduction() measures the reduction in Gini impurity after splitting a dataset into two parts ("left" and "right"). It calculates how much impurity is reduced by this split, which helps to assess how effective the split is.
   best_split_feature_value() finds the best way to split a dataset to reduce Gini impurity. It looks at all features and their values to find the split that gives the greatest reduction in impurity. Returns the reduction in impurity, which feature to split and the value to split. More information can be found on lines **216-299** in supervised_learning.py.

9. The ._predict() method classifies the data based on the Decision Tree structure. If it encounters a leaf node, it returns the class associated with that leaf for all data points. If it is a branch node, it splits the data into left and right subsets and calls itself recursively to classify each subset. The results from the left and right subsets are combined to form the final prediction. Please check **line 427** in supervised_learning.py.

10. Decision tree for Gentoo was implemented (based on bill_depth + flipper_length) and I got accuracy 100%, and the tree looks like this:

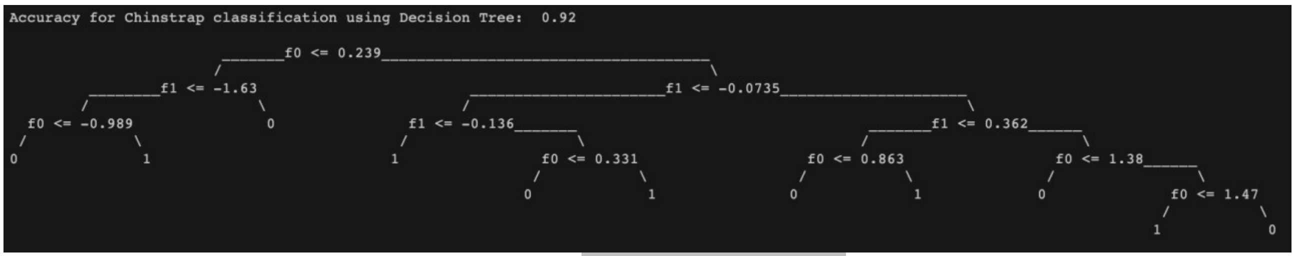


```
Accuracy for Gentoo classification using Decision Tree:  1.0

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

   More information can be found on lines **545-566** in supervised_learning.py.

11. Decision Tree for Chinstrap was implemented (based on body_mass + bill_length), got accuracy 92%, and the tree looks like this:

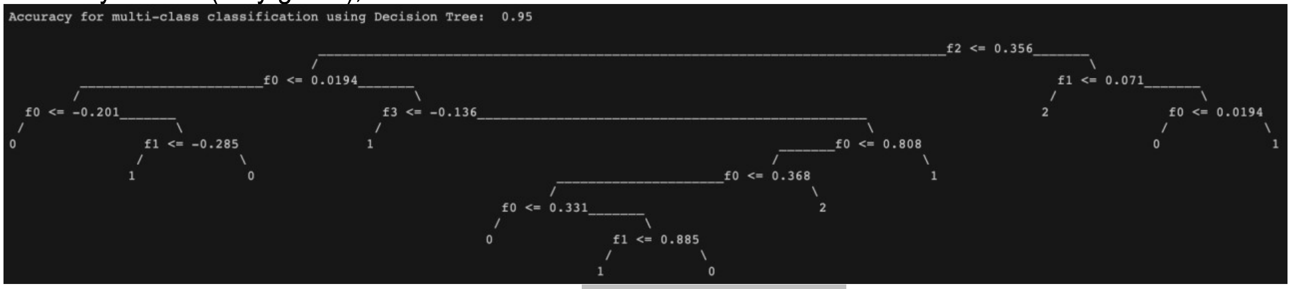


```
Accuracy for Chinstrap classification using Decision Tree:  0.92

                        _______f0 <= 0.239_____________________________________
                       /                                                       \
         ________f1 <= -1.63                        ______________________f1 <= -0.0735_____________________
        /                   \                      /                                                        \
  f0 <= -0.989               0               f1 <= -0.136______                                 _______f1 <= 0.362______
 /            \                             /                  \                               /                        \
0              1                           1                f0 <= 0.331                   f0 <= 0.863                 f0 <= 1.38______
                                                           /           \                 /           \               /                \
                                                          0             1               0             1             0              f0 <= 1.47
                                                                                                                                  /          \
                                                                                                                                 1            0
```

More information can be found on lines **568-582** in supervised_learning.py.

12. Decision Tree for all types of penguins was implemented (based on all features), accuracy is 95% (very good!), and the tree looks like this:



```
Accuracy for multi-class classification using Decision Tree:  0.95

                                       _______________________________________________________________________________f2 <= 0.356_______
                                      /                                                                                                 \
         _______________________f0 <= 0.0194_______                                                                                 f1 <= 0.071_______
        /                                          \                                                                               /                  \
  f0 <= -0.201_______                         f3 <= -0.136________________________________________________                       2              f0 <= 0.0194
 /                   \                        /                                                             \                                   /            \
0                f1 <= -0.285                1                                                    _______f0 <= 0.808                            0               1
                /            \                                                                   /                  \
               1              0                                      _____________________f0 <= 0.368               1
                                                                    /                                \
                                                              f0 <= 0.331_______                      2
                                                             /                  \
                                                            0               f1 <= 0.885
                                                                           /           \
                                                                          1             0
```

More information can be found on lines **584-593** in supervised_learning.py.

Very important question here about how it is possible to predict 3 different types of penguins with Decision Tree but impossible with Perceptron? As I wrote earlier, Perceptron is a simple linear model, which means it tries to separate data using a straight line and gives 0 or 1 in the result. That's impossible for Perceptron if we have 3 or more different types of penguins to predict. **Decision Tree can make a variety of different decisions, allowing it to separate out many types of things into their own groups, and the key to this ability to handle multiple classes lies in their hierarchical, multi-split structure.**

Summarising and self-reflection

Everything worked well and I discovered Decision Tree for myself, which is a very quick and easy, fairly universal tool to use.

The most important thing I've learnt or realised again:

- To get good results, we must have a good data basis (pre-process them if needed, filter, etc, "garbage in, garbage out"), understand the data and goals well, so that we can define which features are best to use and which features may spoil the expected result. It is also important to know the characteristics of the dataset to choose the right model and to try different models and hyperparameters to find the best approach for a given dataset.
- The difference between Perceptron and Decision Tree:
  Perceptron is a simple, linear model that adjusts weights based on training data, converging with clear boundaries between classes.
  Decision Tree is a more complex model that can handle multiple classes and non-linear relationships. It uses Gini impurity and tree structures for decision making.

  Perceptrons can be faster to train and predict with simple, linear data sets, while Decision Tree can be more effective with complex, non-linear data.
- Results:
  For Gentoo, both Perceptron and Decision Tree achieved 100% accuracy. This is likely due to clear, distinct characteristics that set Gentoo apart from others.
  For Chinstrap, Perceptron's accuracy was approximately 96%, Decision Tree ranged between 90-95% depending on the selected properties.
  The Decision Tree for all 3 types of penguins (all features) is approximately 95% accurate.

Bibliography

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