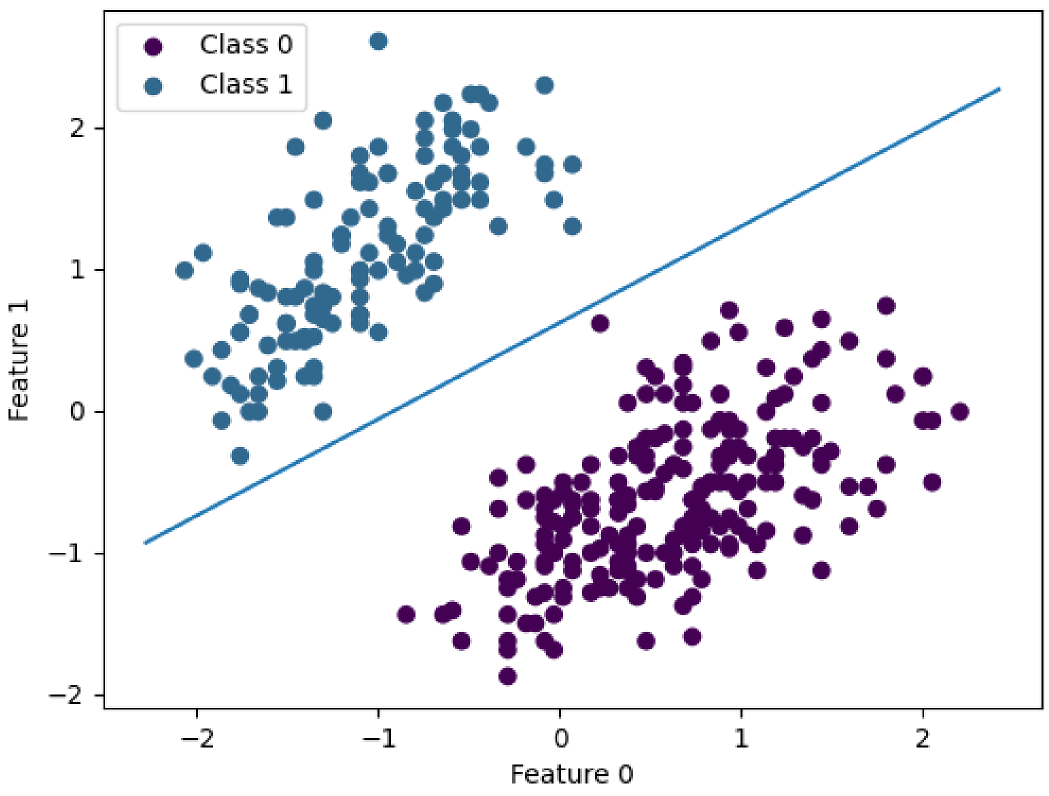
Supervised learning

Mathias Eira Karlsen

DET-2602 24H Introduksjon maskinlæring og AI

24 November 2024



1 Introduction

Supervised learning is a kind of machine learning where you use labeled datasets (Input and desired output) to train a model. In this report I'm going to implement and compare two supervised learning algorithms. I am going to do various experiments to learn the differences, weaknesses and strengths of these two algorithms.

The two algorithms are Perceptron, and Decision tree. These algorithms are used for classification, for example finding what animal a list of traits might belong to. A perceptron is a binary classifier, meaning it can only classify into two categories, for example an animal can be classified into mammal or not mammal. A decision tree is not binary, it can be used to classify 3 or more animal species.

To test these algorithms, the dataset palmer_penguins.csv is used. This dataset has 344 penguins (10 penguins are missing data and will not be used). For each penguin we know its species and four features that will be used for training. The features are bill_length_mm, bill_depth_mm, flipper_length_mm and body_mass_g. There are 3 penguin species, Adelie, Chinstrap and Gentoo. The algorithms will be trained to classify what species a penguin is based on the input features.

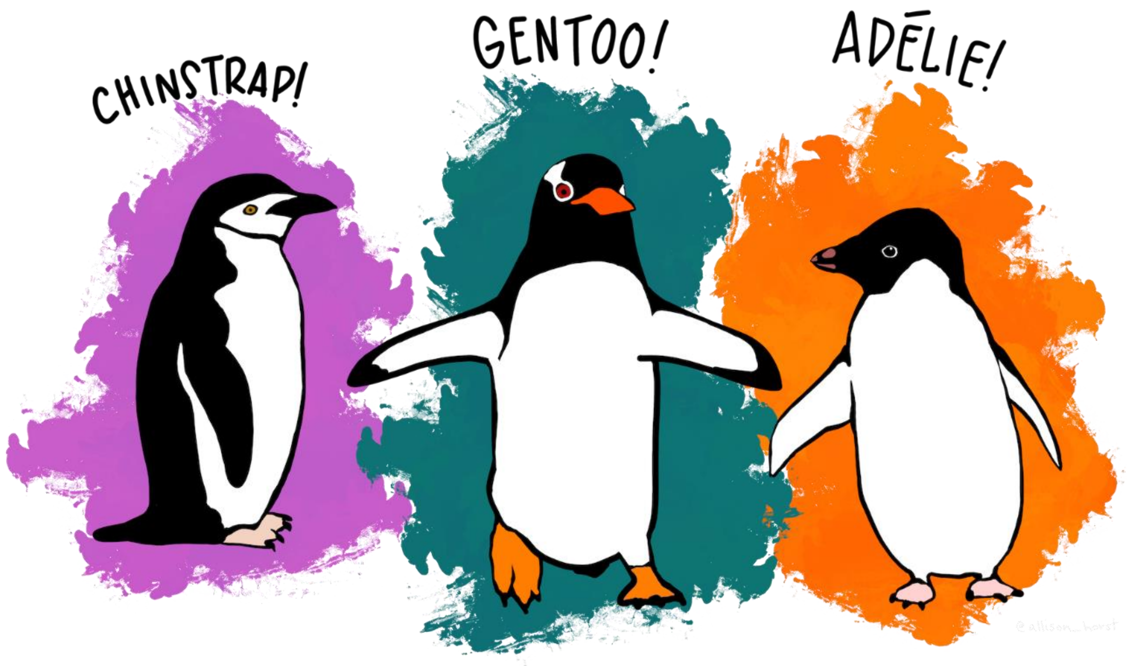


Figure 1 Palmer penguins. Artwork by @allison_horst [1]

2 THEORY

Perceptron and Decision tree are both algorithms for classifying data, but how they learn to classify is very different. The end result for a perceptron is a few weights that are multiplied with the input to get an output, while a decision tree asks multiple questions about the input data to move in a tree, eventually arriving at leaf node, which says what class the data belongs to.

To make the data easier to work with, it is normalized using z-score. This means it is centered around the average and divided by the standard deviation. This makes it so all the values are similarly sized, instead of one feature being in the thousands while another feature only goes up to about 20.

When building a decision tree, you need to find what the right question is to best split the data. This is done using what is called 'Gini impurity'. [2] It measures how impure a dataset is, ranging from 0 when all elements in the set are the same, and increasing the more mixed the data is, up to a maximum of 1 when every class is equally likely and there are unlimited classes. The formula for Gini impurity is $1 - \sum_{i=0}^{J-1} p_i^2$ where J is the number of unique classes and P_i is the probability to randomly pick class i. For example, if there are three classes that are 40%, 40% and 20% of the dataset, we get $p_0 = 0.4$, $p_1 = 0.4$, and $p_2 = 0,2$. Then Gini impurity is $1 – 0.4^2 – 0.4^2 – 0.2^2 = 0.64$.

When making a decision tree, you want to find the best way to split the data. You split the data by asking questions about the data. A question has a feature and a value, for example which penguins have a flipper length less than or equal to 200 mm? Then the data is split into penguins with flippers bigger than 200 mm and penguins with smaller flippers. The way to find the best question is to find the question that reduces the Gini impurity the most. This reduction is calculated by taking the weighted average impurity of the split dataset and finding the difference with the impurity of the data before the split.

Once the best question is found, and the data is split in two, you repeat finding questions recursively with the split data until you reach a Gini impurity of zero. At that point you have a finished decision tree and can use it to predict the class of a penguin using its features.

To use the decision tree you start at the root, and ask the question stored in the node, depending on the answer you move either left or right through the tree. Repeat until you reach a leaf node. The leaf node contains the predicted class.

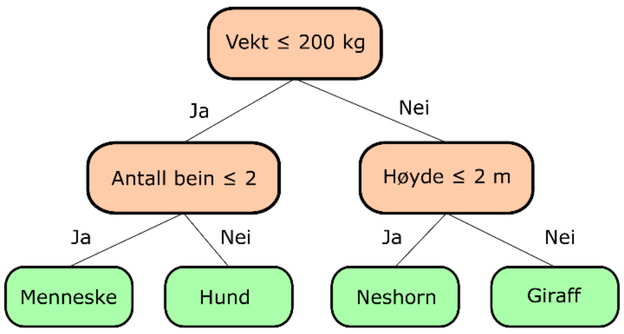


Figure 2 Example simple decision tree [3]

A perceptron works quite differently from a decision tree, instead of repeatedly cutting the space of possibilities in half by asking questions about single features, a perceptron linearly cuts space in half once using all the features. The perceptron is a binary classifier, meaning it can only determine if an input belongs or does not belong to a specific class. It does this by multiplying the input with weights, and then applies a bias to get a sum I that is then fed to an activation function to get the output.

The perceptron takes in an input-vector X = [x_0, x_1, x_2, x_3] and has some weights W = [w_0, w_1, w_2, w_3]. The perceptron also has a bias B. When the perceptron gets an input, it gets multiplied with the weights using the formula $\mathrm{I} = \sum_{\mathrm{i}=0}^{\mathrm{J}-1} \mathrm{x_i w_i} + \mathrm{B}$ where J is the number of features/weights.

The result I is then fed to the activation function. The activation function that was used is called Threshold. It equals 0 if I is less than 0 and 1 if I is greater than or equal to 0.
1 Is a positive classification meaning the perceptron thinks the input X does belong the class the perceptron is trained to detect. 0 means the input does not belong to the class.

To train the perceptron you use training data where you know the correct class to adjust the weights in the perceptron so that it better fits the training data. The formula to adjust the weights is $\mathrm{W[i]} = \mathrm{W[i]} + \mathrm{r*(d - V)*X[i]}$ where W is weight-vector, X is the input-vector, r is the learning rate, V is the perceptron output for input X using the current weights, and d is desired output. If the perceptron correctly classifies the data, (d-V) becomes 0, so the weights don't change. If the perceptron is wrong the weights are adjusted proportionally according to the learning rate and the input X.

The perceptron learns using the training data until it either converges when it correctly classifies all the input data, or until a certain number of epochs have been done. The perceptron works best if the data it is trained on is linearly separable.

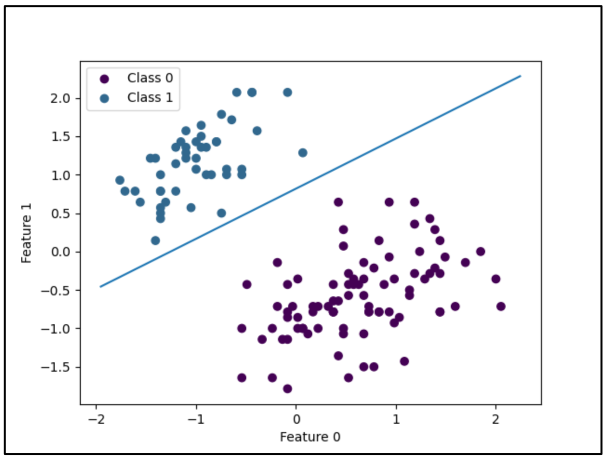


Figure 3 Dataset that is linearly separable

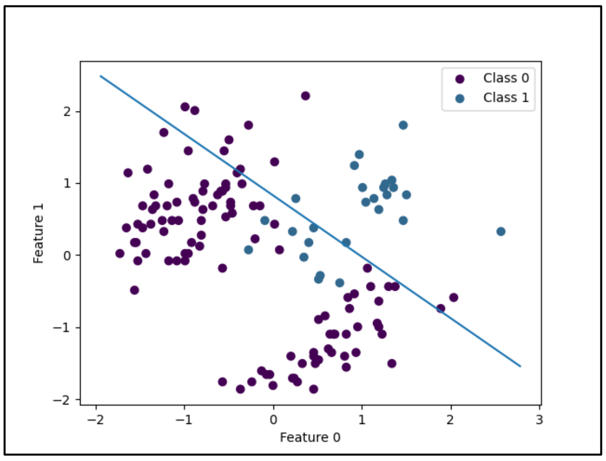


Figure 4 Dataset that is not linearly separable

On figure 3 you can see that the line perfectly cuts the space in half so that class 0 is on one side of the line, while class 1 is on the other side. Figure 4 has a dataset that is not linearly separable, there is no linear line that can perfectly split the data. This idea generalizes to higher dimensions, if the line is replaced by a hyperplane. [4]

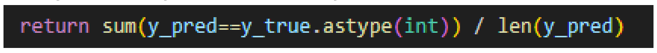
3 METHOD

To work with the dataset and to evaluate the algorithms, the functions read_data(), convert_y_to_binary(), train_test_split() and accuracy() are implemented.

read_data() reads the palmer penguins data file, and converts it into two NumPy arrays, X and y. X is an array with shape (n, 4) where n is the number of penguins, and the 4 is the 4 features that will be used for training and testing. X is then normalized like described in section 2. Y is the species array, it contains integer values 0, 1 or 2 representing the penguin species.

To train and test the algorithms, the data is split into two using train_test_split(). It takes in X and y, and floating point from 0 to 1 that represents the fraction of the data that will be used for training. Since the data is sorted, the function randomizes the order of the data before splitting it, so that the training data is more representative of the full dataset.

To evaluate the accuracy of the trained models, an accuracy function is used. It takes in two arrays, the predicted values of y and the true values. It counts up how many values in the two arrays are equal, and compares it to the total amount of values.



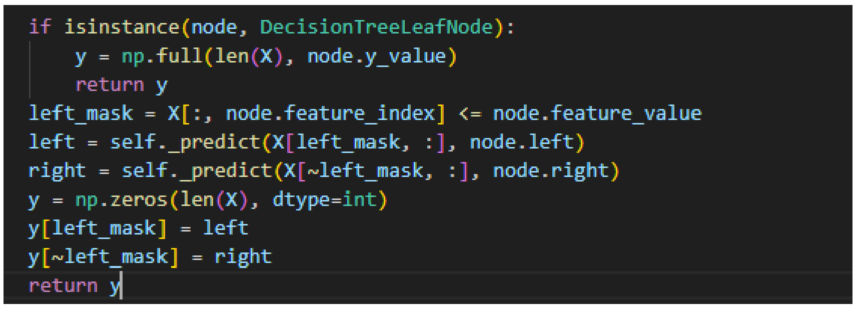
```
return sum(y_pred==y_true.astype(int)) / len(y_pred)
```

Figure 5 Accuracy function

Since the data has 3 species of penguin, but a perceptron can only work with binary data, a function convert_y_to_binary is used to convert y into a binary dataset. It takes in y a value y_value_true. Values in y that are equal to y_value_true will be turned to 1, all other values become 0.

Because Gini impurity is so important for decision tree, there are three functions for working with Gini impurity. The first is simply gini_impurity(), it calculates the Gini impurity of a NumPy array y, using the formula described in section 2. The next function is gini_impurity_reduction(). It takes in y and a left_mask that is used to split y in two. The function then calculates the reduction in impurity resulting from that split. The last function is best_split_feature_value(). It takes in both X and y. It loops through all the values of all the features to find the feature and value that will reduce the Gini impurity the most.

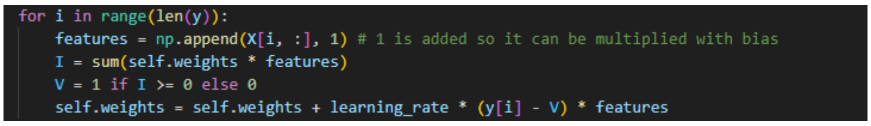
Most of the code for DecisionTree is included in the g2_supervised_learning GitHub page. [3] But the way the tree is built is by using the training set data and finding the best feature and value using best_split_feature_value() and then creating a node that contains those value. It recursively repeat splitting like that until the Gini impurity is 0. The prediction function works similarly, taking in a dataset and recursively splitting it to travel through the tree until it has found a value for each line in the data.



```
if isinstance(node, DecisionTreeLeafNode):
    y = np.full(len(X), node.y_value)
    return y
left_mask = X[:, node.feature_index] <= node.feature_value
left = self._predict(X[left_mask, :], node.left)
right = self._predict(X[~left_mask, :], node.right)
y = np.zeros(len(X), dtype=int)
y[left_mask] = left
y[~left_mask] = right
return y
```

Figure 6 DecisionTree recursive prediction function

The perceptron class has two attributes, the weights array and a Boolean value that says if the perceptron converged during training. I decided to include the bias inside the weights array because this made some of the calculations much simpler. A 1 is added to the end of the features array from data, that way the weights are feature arrays can simply be multiplied together and summed.

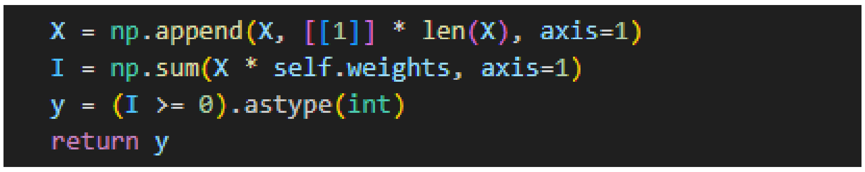


```
for i in range(len(y)):
    features = np.append(X[i, :], 1) # 1 is added so it can be multiplied with bias
    I = sum(self.weights * features)
    V = 1 if I >= 0 else 0
    self.weights = self.weights + learning_rate * (y[i] - V) * features
```

Figure 7 One epoch of perceptron training

When training the perceptron, before every epoch, a copy of the weights is stored, and then after the epoch, the new weights are compared to the old. If the weights are unchanged, then the perceptron has converged. If it has not converged, it continues training until it converges or until the max number of epochs is reached.

The perceptron prediction function uses the NumPy broadcasting function to multiply the weights with the entire test set in a single operation.



```
X = np.append(X, [[1]] * len(X), axis=1)
I = np.sum(X * self.weights, axis=1)
y = (I >= 0).astype(int)
return y
```

Figure 8 Perceptron predict function

To test the algorithms, a number of experiments were done. The first one I did was testing the accuracy of both algorithms for different train set fractions. I would do multiple epochs for every possible fraction of train_set and calculate the average. I then made a graph showing the accuracy for each fraction.

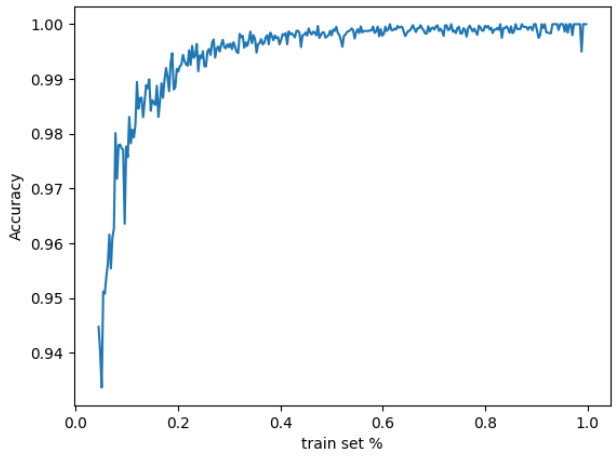


Figure 9 Graph of accuracy for different percentages of training data

Figure 9 shows a graph for the accuracy of a perceptron trained to look for Gentoo. It shows that increasing the fraction of data used by training has a big effect at first but past about 0 .4 the accuracy increase is very small. I made a similar graph for the other penguins, and for decision trees and I got a similar result. So, 0.4 is the fraction I will be using for the other experiments. Using 0.4 also means the test data has exactly 200 penguins, which is nice.

To keep the main file clean and organized i put all the experiments in a different file, experiments.py

4 RESULTS

The perceptron and decision tree are both good at classification, but there are some differences in how they perform. The first experiment is classifying Gentoo using only the features bill_depth and flipper_length. For this experiment the perceptron got an average accuracy of 99.7%, classifying 199.41 out of 200 penguins correctly with a very small standard deviation of 1.1. The perceptron converged meaning this data is linearly separable. This worst result from the perceptron was 192. The decision tree got an average accuracy of 99% with 197.93 of 200 penguins correctly classified. The standard deviation was 2.1 and the worst result was 188

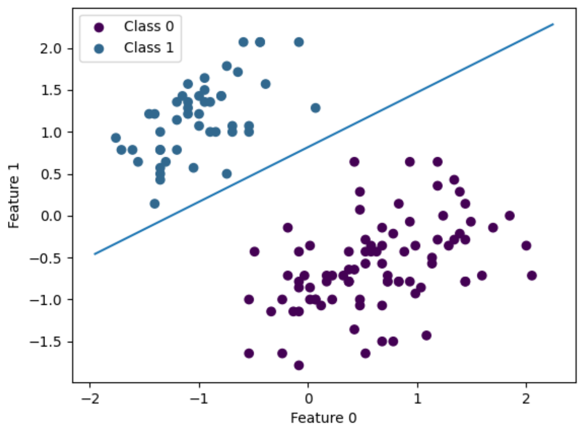


Figure 10 Experiment 1 data with perceptron decision boundary

For the second experiment the perceptron and decision tree had to classify Chinstrap, using bill_length and bill_depth. For this experiment the perceptron never converged meaning the data is not linearly separable. The perceptron got on average 163 of 200 penguins correct, with a standard deviation of 13.2, while the decision tree got an average of 186.7 of 200 with a standard deviation of 4.46. The worst results for the perceptron and decision tree were 103 and 172, while the best results were 180 and 195

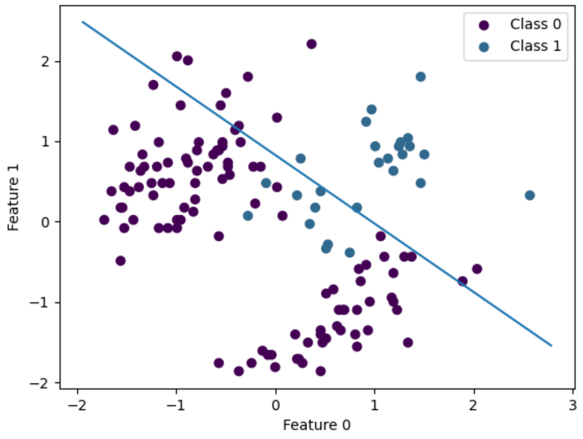
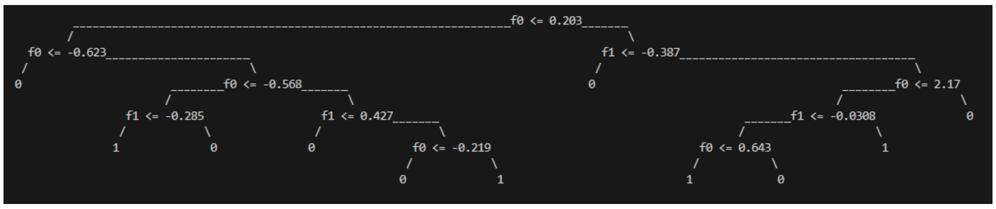


Figure 11 Experiment 2 data with perceptron decision boundary

For experiment 1 the decision tree was very simple with only about 3 questions on average, but for experiment 2 the tree has a lot more nodes. The shape of the tree also varies a lot based on which parts of the data are used to make the tree.

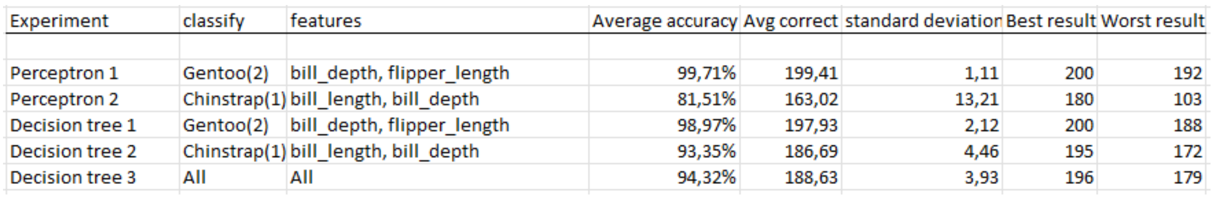


```
         __________________________________________________________________f0 <= 0.203_______
        /                                                                                     \
  f0 <= -0.623______________________                                                      f1 <= -0.387___________________________________
 /                                  \                                                    /                                                \
0                       ________f0 <= -0.568_______                                     0                                      _______f0 <= 2.17
                       /                           \                                                                          /                  \
                 f1 <= -0.285                  f1 <= 0.427_______                                              _______f1 <= -0.0308               0
                /            \                /                  \                                             /                    \
               1              0              0               f0 <= -0.219                               f0 <= 0.643                  1
                                                            /            \                              /           \
                                                           0              1                            1             0
```

Figure 12 Experiment 2 Decision tree

The last experiment is making a decision tree to classify all 3 penguin species using all 3 features. For this experiment the average correct classifications was 188.6 of 200 (94.32%) with a standard deviation of 3.93. The best result is 196 and worst is 179. Increasing the fraction of data used to 70% raises the accuracy to 95.5%.

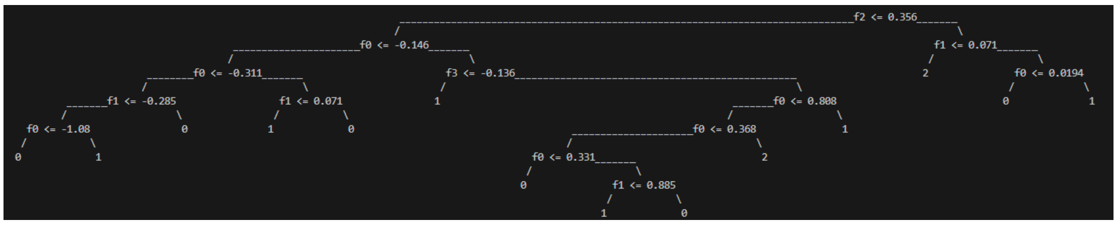
The results for all experiments are in the table below. When writing the results I grouped Perceptron 1 and Decision tree 1 together, and I did the same for Perceptron 2 and Decision tree 2



Experiment	classify	features	Average accuracy	Avg correct	standard deviatio	Best result	Worst result
Perceptron 1	Gentoo(2)	bill_depth, flipper_length	99,71%	199,41	1,11	200	192
Perceptron 2	Chinstrap(1)	bill_length, bill_depth	81,51%	163,02	13,21	180	103
Decision tree 1	Gentoo(2)	bill_depth, flipper_length	98,97%	197,93	2,12	200	188
Decision tree 2	Chinstrap(1)	bill_length, bill_depth	93,35%	186,69	4,46	195	172
Decision tree 3	All	All	94,32%	188,63	3,93	196	179

Figure 13 Data from experiments

I also made a decision tree using the full dataset just to see what it would look like.



```
                                                                   _____________________________________________________________________________f2 <= 0.356_______
                                                                  /                                                                                               \
                                      ______________________f0 <= -0.146_______                                                                               f1 <= 0.071_______
                                      /                                        \                                                                             /                  \
                        ________f0 <= -0.311______                         f3 <= -0.136_______________________________________________                      2               f0 <= 0.0194
                       /                          \                       /                                                            \                                 /              \
          _______f1 <= -0.285                 f1 <= 0.071                1                                                 _______f0 <= 0.808                            0              1
         /                   \               /           \                                                                /                   \
   f0 <= -1.08                0              1            0                                     _____________________f0 <= 0.368              1
  /            \                                                                                /                               \
 0             1                                                                        f0 <= 0.331_______                       2
                                                                                        /                  \
                                                                                       0               f1 <= 0.885
                                                                                                      /           \
                                                                                                     1             0
```

Figure 14 Decision tree for full dataset

5 DISCUSSION

Both algorithms worked mostly as expected, with the perceptron working really well for data that is linearly separable, and not working that great when its not. What I did find interesting was the decision tree also working much better for data that is linearly separable.

The decision tree got an average accuracy of 99% when only trying to classify Gentoo. The accuracy went down to about 94% for experiment 2 and 3. I think this is because the tree was much more complicated than it was in experiment 1. Since I only used 40% of the data to make the tree, it is susceptible to outliers in the data that don't fit the tree, and a more complicated tree is more susceptible to outliers than a less complicated tree. Raising the fraction of data used to make the tree increases accuracy slightly since the chance of there being outliers in the test set is smaller, but the average accuracy doesn't go much higher than 95%.

The perceptron did very badly on the second experiment, with the worst score being equal to just picking randomly. This was expected since the data was not linearly separable. It got a very high standard deviation so it could randomly get a decent accuracy, but it's impossible for it to get a perfect score and it's very likely to do very badly.

6 CONCLUSION

In this report I implemented and compared two supervised learning algorithms, Decision tree and Perceptron. Both algorithms are used to classify data. Perceptron is a linear classifier, while a decision tree can be used for more complex classifications.

Perceptron worked well for data that is linearly separable, but it did badly for data that was not linearly separable. Decision tree worked quite well, even if the data is not linearly separable, but it works even better for data that is linearly separable.

For data that is linearly separable, a perceptron works slightly better than a decision tree. But for data that is not linearly separable, you should avoid using a perceptron. A decision tree with a lot of nodes is a lot more accurate for that kind of data.

TABLE OF FIGURES

REFERENCES

[1] Meet the Palmer penguins. Artwork by @allison_horst https://allisonhorst.github.io/palmerpenguins/articles/art.html#meet-the-palmer-penguins

[2] Wikipedia Contributors, “Decision tree learning” *Wikipedia*, Jul. 16, 2024. https://en.wikipedia.org/w/index.php?title=Decision_tree_learning&oldid=1234846759#Gini_impurity

[3] g2_supervised_learning/DecisionTree_intro.ipynb, GitHub. https://github.com/uit-dte-2602/g2_supervised_learning/blob/main/DecisionTree_Intro.ipynb

[4] Wikipedia Contributors, “Linear separability” *Wikipedia*, Jul. 16, 2024. https://en.wikipedia.org/w/index.php?title=Linear_separability&oldid=1189026543