Portfolio Assignment 2

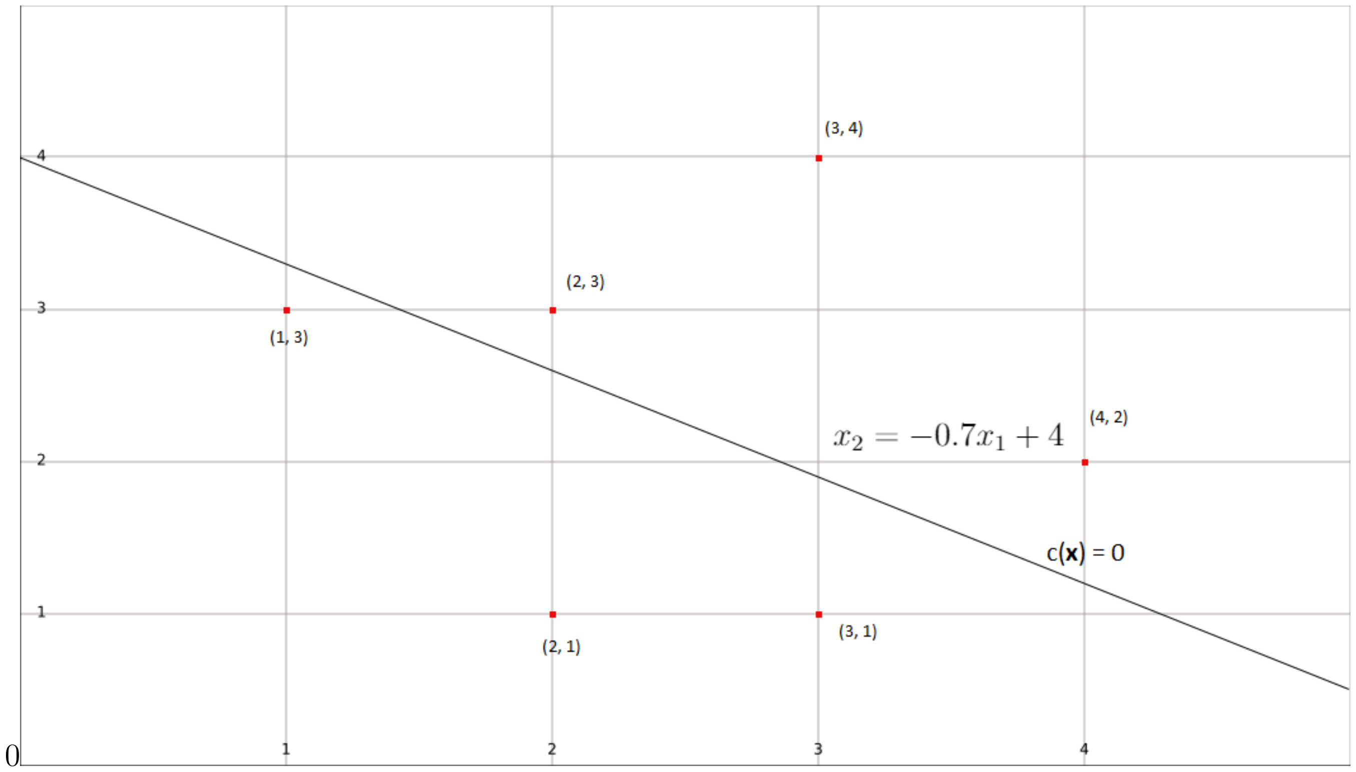
Candidate 25

Problem 1

(1a)

Figure 1: Decision boudary of a logistic descrimination classifier

-



In the plot above we see the points from the training set: $\{(\mathbf{x}^i, y^i)\}_{i=1}^{6}$ as red points. $\mathbf{x}^i$ is the trainig data, and y^i is the ground truth. The line going somewhat diagonally across the plot is the decision boundary. This is used to classify the points above the line as $y^i = 0$, and points underneath the decision boundary as $y^i = 1$. This decision boundary is given by the equation:

$$C(\mathbf{x}) = \boldsymbol{w}^T \cdot \mathbf{x} + w_0 \tag{1}$$

Which put simply, assigns a class to an vector $\mathbf{x}^i$. If the vector after being put into $C(\mathbf{x}) > 0 \implies \mathbf{x} \to C^1$ or the opposite case where $C(\mathbf{x}) < 0 \implies \mathbf{x} \to C^2$. Which means at $C(\mathbf{x}) = 0$ we are on the decision boundary. From this we can derive the equation of the decision boundary, $x_2 = -\frac{w_1}{w_2} \cdot x_1 - \frac{w_0}{w_2}$. In my guess for the decision boundary i used $w_1 = -0.7, w_2 = 1$ and $w_0 = 4$ to give the decision boundary of $x_2 = -0.7 \cdot x_1 + 4$. Here we see that the weights, $\boldsymbol{w}$ of the classifier determines the slope of the decision boundary, whereas the bias, w_0 determines the intercept with the second axis, x_2.

To figure out the distance between the origin, $\boldsymbol{O}$, and the decision boundary we can calulate this with the formula,

$$d = \frac{w_0}{\|\boldsymbol{w}\|} = \frac{4}{\sqrt{(-0.7)^2 + 1^2}} \approx \underline{\underline{3,277}}$$

Now i shall demonstrate how this decision boundary could give us a decision rule, based on the estimated weights and biases. Consider a test point,

$$\mathbf{x}^t = \begin{bmatrix} x_1^t \\ x_2^t \end{bmatrix}$$

this point will from what I stated earlier follow the decision rule,

$$\begin{cases} y^t = 0, \text{for} \quad C(\mathbf{x^t}) > 0 \\ y^t = 1, \text{for} \quad C(\mathbf{x^t}) \leq 0 \end{cases}$$

Here i have also made the assumption that if $C(\mathbf{x^t}) = 0$ then $\mathbf{x^t}$ is assigned to $y^t = 1$.

(1b)

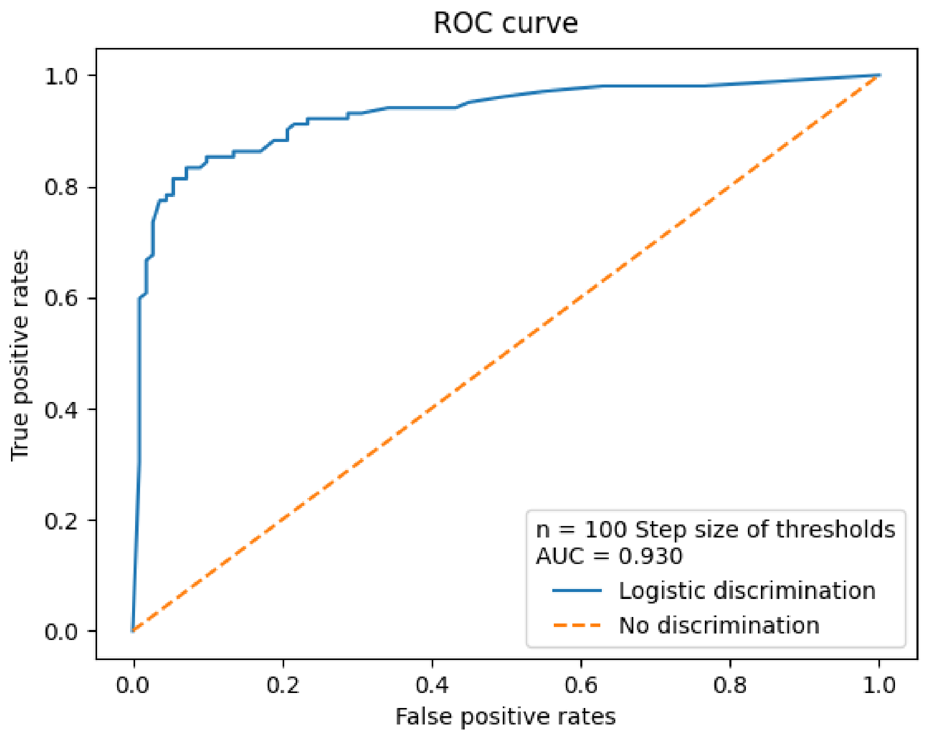
After having trained my logistic discrimination model on the *seals_train.csv* file, and the used the trained weights to classify the test data. I produce with the *seals_test.csv* file the confusion matrix:

$$\text{confusion matrix} = \begin{bmatrix} 87 & 15 \\ 11 & 100 \end{bmatrix}, \text{ and accuracy} \approx \underline{0.878}$$

Which means the model classified 87 class 0's correctly, 11 class 1's as class 0's, 15 class 0's as class 1's and 100 class 1's correctly.

(1c)

Figure 2: The Reciever Operating Characteristic



The ROC curve represents the models performance at different classification tresholds. This means that by varying the treshold of where a previously classified class one would be classified as a one to a threshold where it instead would be classified as a zero. In other words, we can use an iterative approach, using values between 0 and 1, to see how the classifications change. The ROC then simply is the true positive rate against the false positive rate over all thresholds between 0 and 1. The AUC on the other hand, is the area under the ROC curve, which also is a number between 0 and 1, where a AUC score of 1 means the model has all predictions correct, while AUC of 0 means all predictions are incorrect.

We can easily change the true-positive and false-positive rates, by varying the threshold value which determines if a given datapoint should be a class zero or class one. Usually this threshold value is set to 0.5, such that datapoints sent throught the sigmoid function larger than this threshold is class one, and lower, class zero. By varying this threshold all the way from 0 to one, we will get the ROC curve.

For the Area Under the Curve score i got AUC $\approx \underline{0.930}$ by using *sklearn.metrics* built-in function for calculating the AUC score.

(1d)

For this task I have created a function which classifies seal images corresponding to a given index, called *show_image*. Furthermore I use this function to show 5 random images which I know my classifier correctly classified in task (1b), and 5 random images I know are incorrectly classified.

Figure 3: Pictures of correctly classified seals

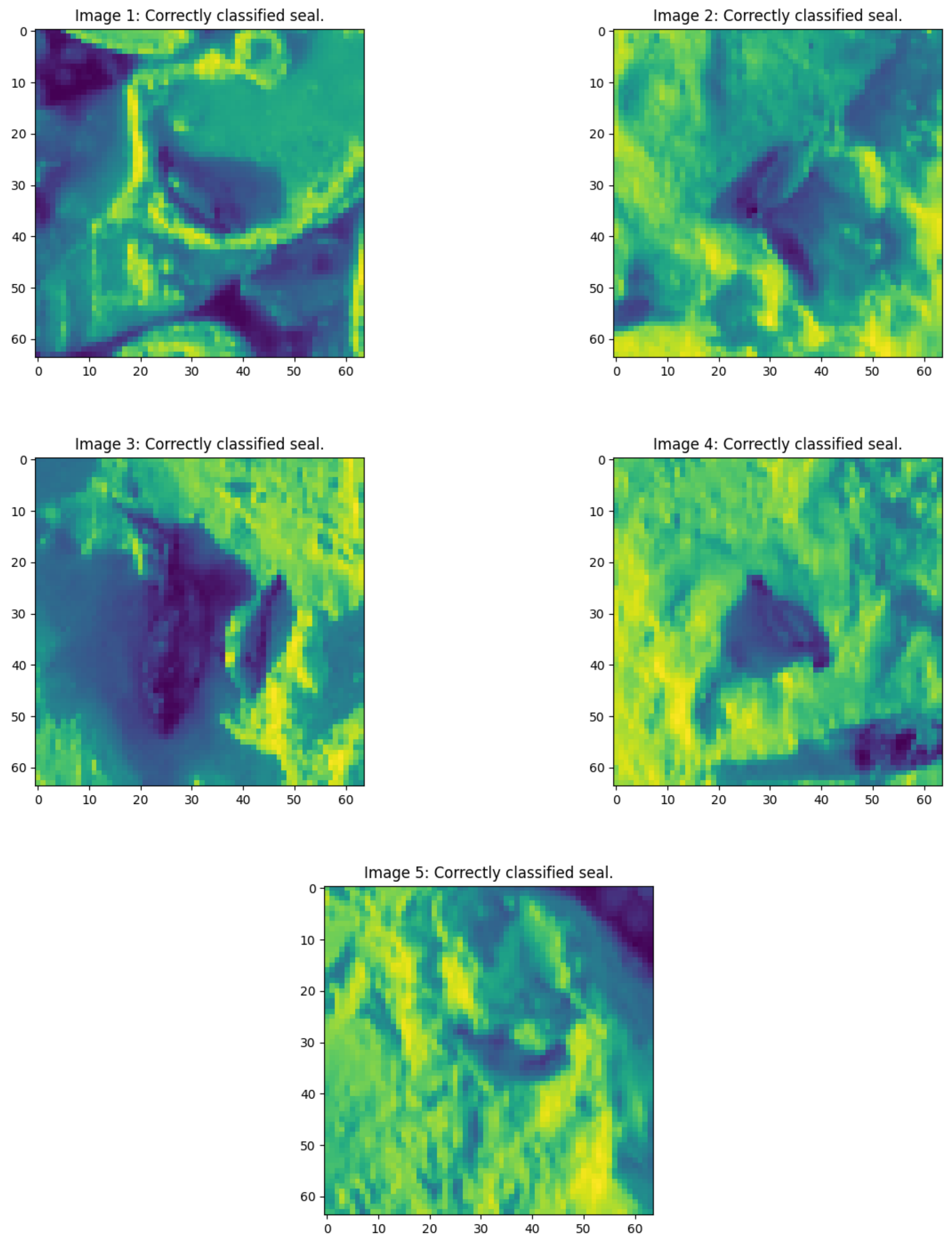


Image 1: Correctly classified seal.
Image 2: Correctly classified seal.
Image 3: Correctly classified seal.
Image 4: Correctly classified seal.
Image 5: Correctly classified seal.
0 10 20 30 40 50 60

Figure 4: Pictures of not correctly classified seals

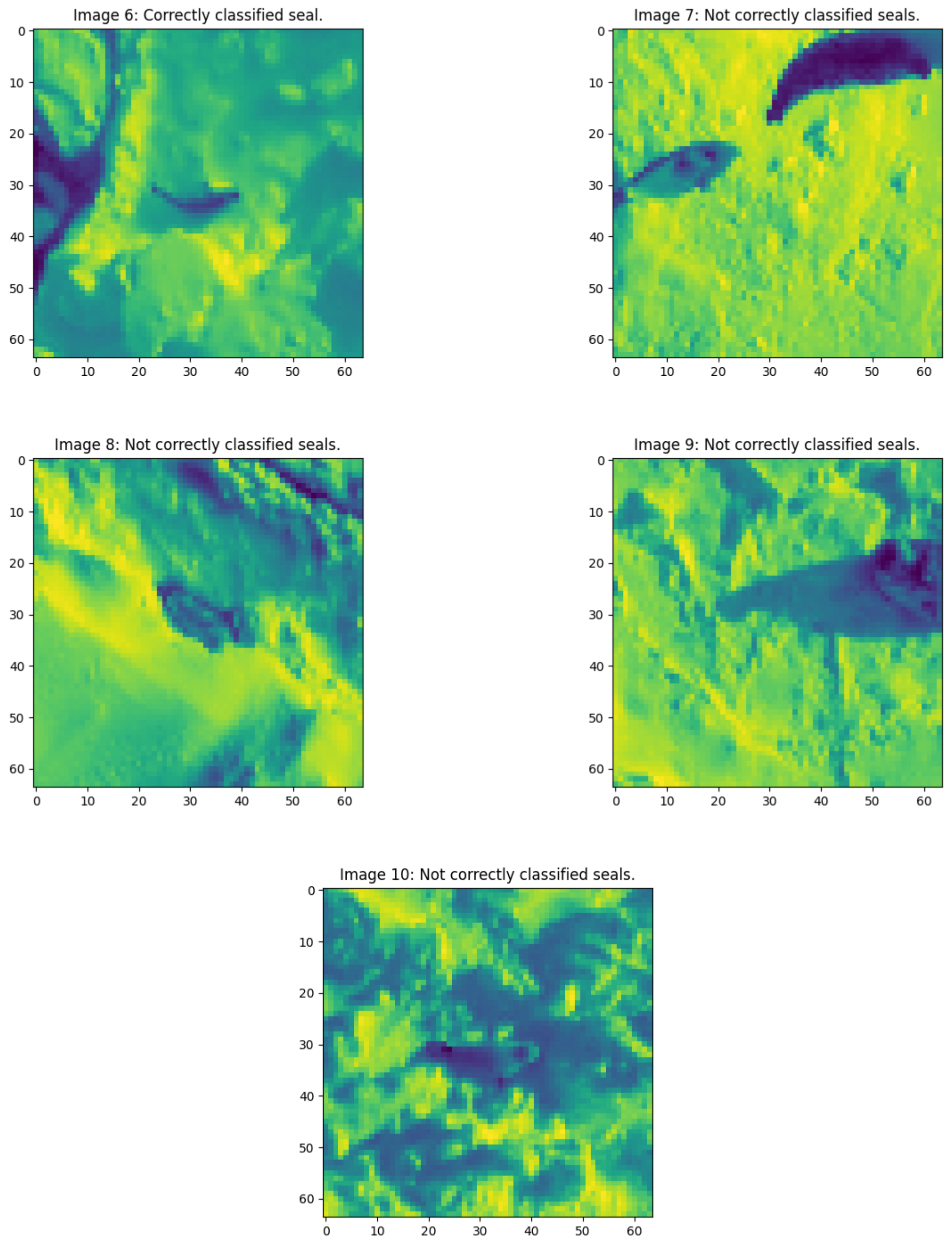


Image 6: Correctly classified seal.
Image 7: Not correctly classified seals.
Image 8: Not correctly classified seals.
Image 9: Not correctly classified seals.
Image 10: Not correctly classified seals.

In Figure 3. the pictures are of either *Harp seal pups* or *Hooded seal pups*, where the algorithm has classified theese seals correctly. Whereas in Figure 4. the pictures are of either *Harp seal pups* classified as *Hooded seal pups*, or *Hooded seal pups* classified as *Harp seal pups*.

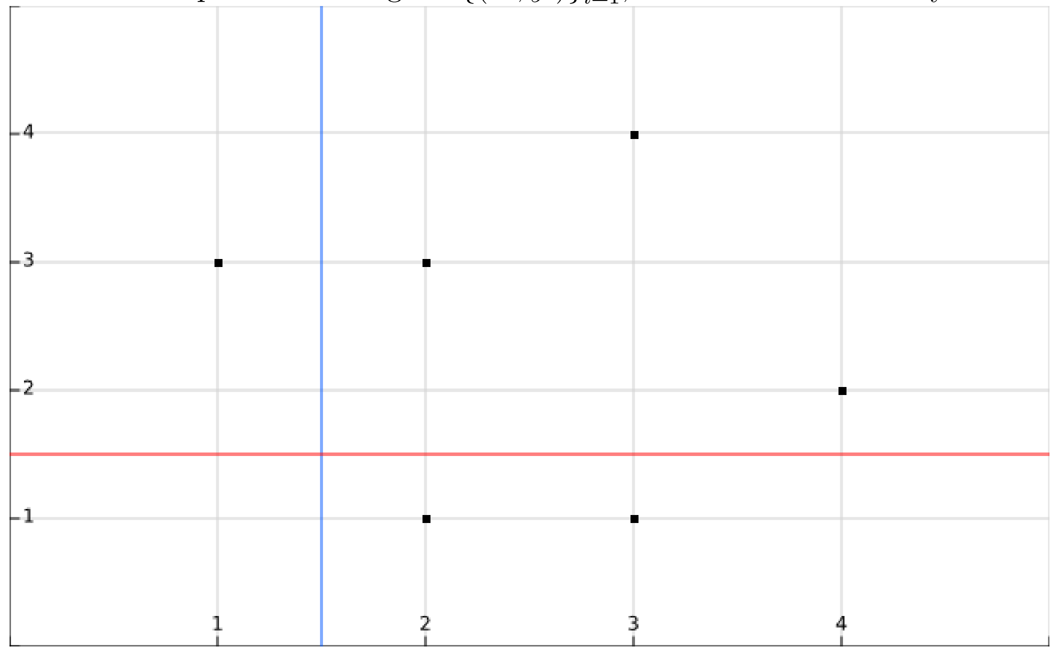
This leads to a question, what are the possible reasons as to why the pictures in Figure 4. are misclassified. By studying theese example photos I belive one reason for this comes from not the whole seal being present in the image. An exmple of this we see in Image 9. The seal is located between approximately pixel 50 to pixel 64 along the first axis, and pixels 10 to 30 along the second axis. Where not the whole seal is present, althoug we tell the classifier there is a seal in the image.

This leads to making a sort of edjucated guess, since not all of the features neccesary to make a correct classification is present.
In image 8. it seems as though the lack of a good image resolution may be hiding a lot of the features the classifier neeeds to make a correct decision, and hence it misclassifies the seal.
Image 10. demonstrates a noisy image where a lot of the features of the seal is hidden due to lots of darker areas, and very bright areas making it hard to pinpoint where the seal begins and ends.
In conclusion, the reason as to why the classifier fails to make a correct decision I belive arise when the images hides the distinct features of the seal.

Problem 2

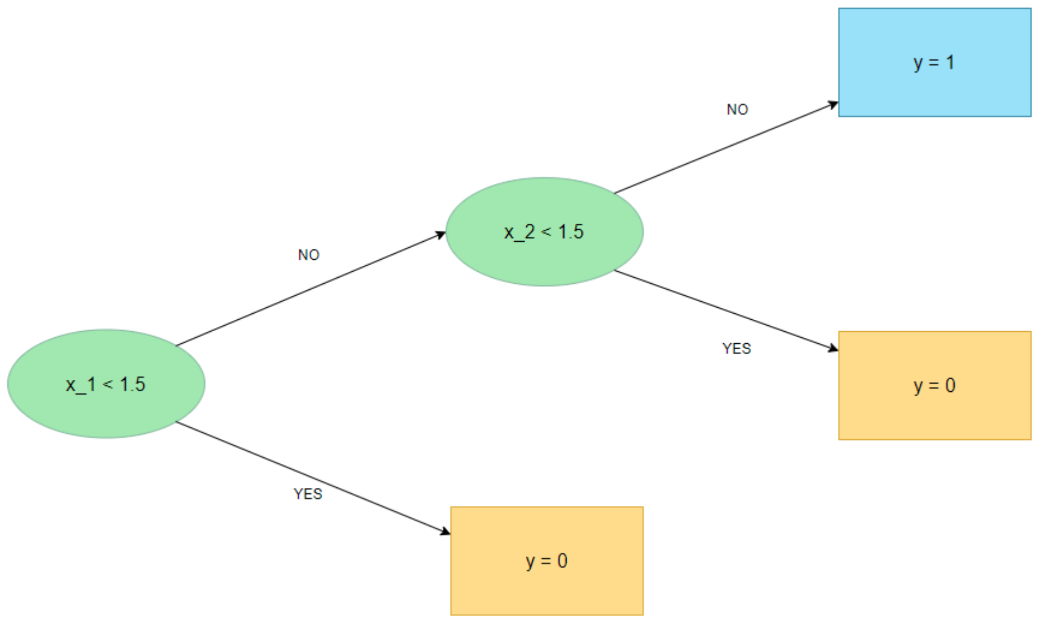
(2a)

Figure 5: 2-dimensional plot of training set $\{(\mathbf{x}^i, y^i)\}_{i=1}^6$, and decision boundary of a decision tree.



This plot illustrates the two decision boundaries needed to classify the training set with zero errors. Here I have used two boundaries. The first going from $x_1 = 1.5$ spanning the secound axis, and the second going from $x_2 = 1.5$ spanning the first axis.

Figure 6: Decision tree for training set.



From this decision tree we see that we can create a IF-THEN rule which describes how the tree would classify the datapoint:

$$\mathbf{x}^t = \begin{bmatrix} x_1^t \\ x_2^t \end{bmatrix}$$

This means that we can assume a decision rule,
IF: $x_1^t < 1.5$, THEN $\mathbf{x}^t \rightarrow y^t = 0$
IF: $x_1^t \geq$ AND $x_2^t < 1.5$ THEN $y^t = 0$
IF: $x_1^t \geq$ AND $x_2^t \geq 1.5$ THEN $y^t = 1$

(2b)

In this task I have implemented a object oriented decision tree. The way this decision tree works, is by splitting the data into smaller and smaller pieces. To decide where to split the data we introduce a impurity measure. In this case I used entropy. This simply is a measure of uncertainty within our splitted datasets.

Also we need to set a maximum depth of a decision tree. This is because the decision tree itself is prone to overfitting the data. In my implementation I have selected a maximum depth of ten. This means that the data can be splitted up to ten times, before it must declare a leaf node.

For determening a threshold when splitting a node, I have used a very inefficient way which loop throught all columns (features), of the dataset. Then it iterates through every value in each of the columns. Last step is to calculate the entropy of the potential new splitted dataset, as if one of the values of the columns were the threshold of the split, then we just select the threshold which has the lowest total entropy of each value of each column, to split the data. Because we do this recursively this means we end up doing this for every new splitted dataset, until we can declare a leaf node.

I have also implemented another way of declaring a leaf node. This just checks how many datapoints we have left on a branch. If this is smaller than a threshold, I have used 40 datapoints as this threshold, then we declare this node a leaf.

From the decision tree i got the confusion matrix,

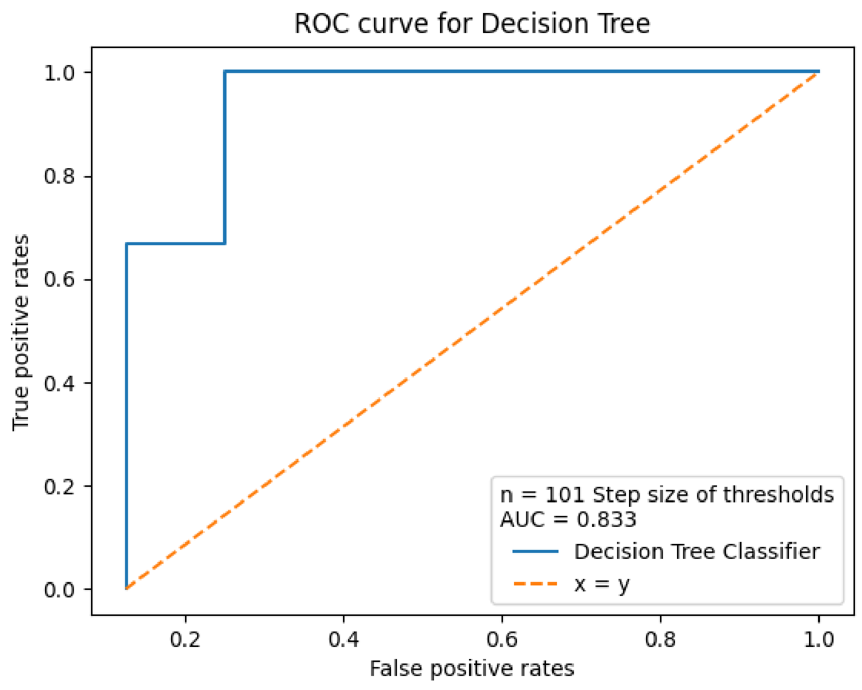
$$\text{confusion matrix} = \begin{bmatrix} 92 & 10 \\ 13 & 98 \end{bmatrix}, \text{ and accuracy} \approx \underline{0.892}$$

I have also created another object called a node. The node object keeps track of important information on each node, things like threshold values, the depth of the node and if the node is a leaf node or not to name a few.

(2c)

From our predictions we can find the amount of zeros, relative to the amount of elements at a leaf node, by dividing the amount of zeros over the total amount of elements. We know theese have been classified as either ones or zeros. If we take this relation, we can iterate over small values between zero and one, calling this a new threshold, classifying a proportion of the elements at this node if they are either larger or smaller than the threshold value. From this we can construct a ROC curve, since we varied the threshold between 0 and 1.

Figure 7:



Here i also included the AUC score, which clocked in at AUC ≈ 0.833.
If we compare this to the logistic discrimination classifier we see that the decision tree had a bit

better accuracy, at 0.892, over the other method which had an accuracy of 0.878. While the logisitc discrimination classifier has a better AUC score, of 0.930, while the decision tree had 0.833.

Problem 3

(3a)

In this task I am handling missing data. I have looked at data which contained *NaN* values, which is a acronym for *Not a Number*. The task looks into how to replace theese missing values with values which wont affect the total outcome of the data, if we wanted to use it for something.
First off I were given a dataset containig three columns, and a bunch of rows. The middle column contained theese missing values. The first task is to estimate what theese could be using three different methods. Doing this by finding the mean, median and maximum value of the column.

$$\begin{aligned}
&\text{Mean} \approx -0.0394 \\
&\text{Median} \approx -0.085 \\
&\text{max} = 9.34
\end{aligned}$$

I then inserted theese values where the dataset contained missing values.
Then I compared this with the values given in the uncensored data file, which contained the actual values of the dataset. Then to check the accuracy of the estimators mean, median and maximum value, I calculated the MSE, short for mean squared error, of each of the imputated values.

$$\begin{aligned}
&\text{MSE for mean estimator} \approx 14.056 \\
&\text{MSE for median estimator} \approx 13.988 \\
&\text{MSE for maximum estimator} \approx 117.18
\end{aligned}$$

This means the median estimator had the lowest error, and is the best approximation of the missing values.

(3b)

For this task I shall do the same as in the last task, but where instead of using mean, median and maximum as our estimator, I instead use linear regression to estimate the *NaN* values.
First off I need a independent and dependent variable, which should have some correlation between them. This can be thought of as if they have a strong correlation, they are somewhat proportional with each other.
Clearly the dependent variable will be the column, which is missing values. Because it is theese we are trying to estimate. For the independent variable however using a correlation matrix will give us an idea of which column would fit our description. By using a built-in Python library called Pandas we can find the correlation matrix.

$$\begin{bmatrix} 1 & 0.707 & 0.004 \\ 0.707 & 1 & -0.011 \\ 0.004 & -0.011 & 1 \end{bmatrix}$$

From this we see that in column number 2 we have the dependent variable. The first element of this column is the largest value. This is the correlation between column 1 and column 2. So we choose column 1 as our independent variable.

Now using this result we can train a linear regression model to predict the missing values. I chose to use the regression model we created in Portfolio Assignment 1, as my model.
From trainig the model of course predicted different values for each independent variable value. Testing this against the actual values we had from earlier i got the mean squared error to be:
MSE for regression estimator ≈ 6.842. Which is a large improvement over even the best estimator I previously used.

(3c)

In this task we will again predict using a tree model. This tree is called a regression tree, because of the impurity measure used, which is the mean in a splitted dataset. After creating this tree algorithm i got a mean squared error of:
MSE for regression tree estimator ≈ 12.410. This means the linear regression method gave the best approximation, then the regression tree, the median estimator, the mean estimator and the maximum estimator gave the worst prediction of the missing values.