1 Theory

1. Concept learning is the problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples.

2. Function approximation is the task of selecting a function that approximates the target function. This is needed because finding a the target function could be very difficult, or the target function could be unknown.

3. Inductive bias is the assumptions we make about the target concept, and allows us to generalize beyond the training examples the agent is given. The candidate elimination algorithm has the inductive bias of assuming the target concept is in the given hypothesis space, whereas decision tree learning has an inductive bias of preferring shorter trees over large trees and a preference for placing attributes with higher information gain closer to the top of the tree.

4. Overfitting refers to a hypothesis or model that models the training data best, but performs worse than some other hypothesis against the entire distribution of instances. This can happen when there is noise in the data or there are too few training examples. Underfitting refers to a hypothesis that performs poorly on both training data and new data.

   A validation set is one part of the available training data. Some parts of the training data is kept for validation and not used for learning. The remaining training set is used to form the learned hypothesis and then the validation set is used to evaluate the accuracy of the hypothesis, and to evaluate the effect of pruning the tree.

   When splitting the validation set from the training set the hypothesis tends to be reflected by the way the data was divided. Cross validation is therefore used to make sure there is an equal chance for each example to appear in either set, so that the learning hypothesis does not completely match an example from the validation set.

5. The candidate elimination algorithm begins with the version space of all hypotheses. As the algorithm learns it will contain all hypotheses that are consistent with an observed sequence of training examples. The version space is compactly stored as all possible hypotheses between two boundaries: the general boundary(G) and the specific boundary(S). At the beginning G is initialized to $\{<?, ?, ?, ?>\}$ and S is initialized to $\{<\emptyset, \emptyset, \emptyset, \emptyset>\}$.

Sex	**Problem Area**	**Activity Level**	**Sleep Quality**	**Treatment Successful**
Female	Back	Medium	Medium	Yes
Female	Neck	Medium	High	Yes
Female	Shoulder	Low	Low	No
Male	Neck	High	Medium	Yes
Male	Back	Medium	Low	Yes

Table 1: Training data

   The first training example is a positive one, and the algorithm must therefore check if the specific boundary covers the example. The updated boundaries can be seen in the first row of table 2. The second example further specializes S, making *problem area* and *sleep quality* more general. The third example is negative, and we must therefore make G less general so that it does not contain this example. There are several hypotheses that would make G classify the new example as negative. However, only one is consistent with the S boundary.

   When the fourth example is given, the specific boundary becomes so general that it contains every hypothesis. Since our general boundary hypothesis is inconsistent with this training example, we have to remove it, leaving us with no general boundary. This could mean there is an error in the dataset, or we do not have all the variables for correctly determining if the treatment was successful. After this example the algorithm should stop.

#	S	G
0	$\{<\emptyset,\emptyset,\emptyset,\emptyset>\}$	$\{<?,?,?,?>\}$
1	$\{<Female, Back, Medium, Medium>\}$	$\{<?,?,?,?>\}$
2	$\{<Female, ?, Medium, ?>\}$	$\{<?,?,?,?>\}$
3	$\{<Female, ?, Medium, ?>\}$	$\{<?,?,Medium,?>\}$
4	$\{<?,?,?,?>\}$	$\{\}$

Table 2: Results of candidate elimination on the physiotherapy questionnaire

2 Programming

Task 1: Linear Regression

1. See the function *ordinary_least_squares()* in *linear_regression.py*
2. After running the *linear_regression.py* file the following output is produced:

```
$ python3 linear_regression.py
Weights: [[0.24079271 0.48155686 0.0586439]]
Training data error: [0.01038685]
Test data error: [0.00952976]
```

   Based on the relatively small error, we can say that the model generalizes well.

3. After running the *linear_regression.py* file on the second dataset this output is produced:

```
$ python3 linear_regression.py
Weights: [[0.1955866  0.61288795]]
Training data error: [0.01375879]
Test data error: [0.01244246]
```

   There is more error in this dataset than the previous, but it is still a good fit as can be seen in figure 1.

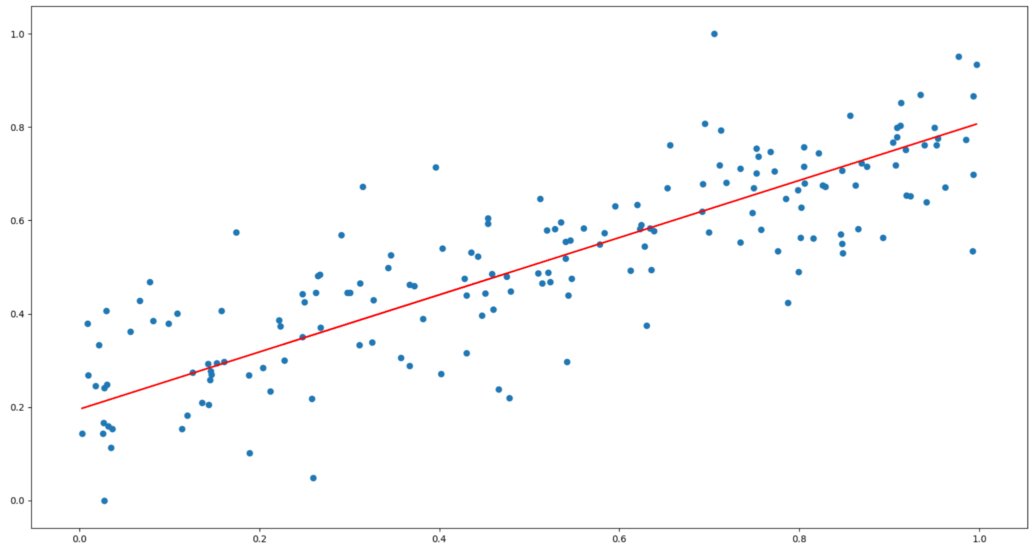

Figure 1: Linear regression of the test_1d_reg_data.

Task 2: Logistic Regression

1. See *logistical_ regression.py* for the implementations.

   The data is linearly separable because we can draw a straight line that separates the positive (1) examples from the negative examples (0).

   The cross entropy for both data sets, as they develop over 1000 iterations, can be seen in figure 2. The learning rate used was $\eta = 0.01$, and the initial parameters $\mathbf{w} = \{0, 0, 0\}$. After training was done the weights for the training set were $\mathbf{w} = \{4.22448447, -12.27806252, 6.36330889\}$ and for the test set: $\mathbf{w} = \{5.32779934, -10.79201844, 3.10085983\}$.

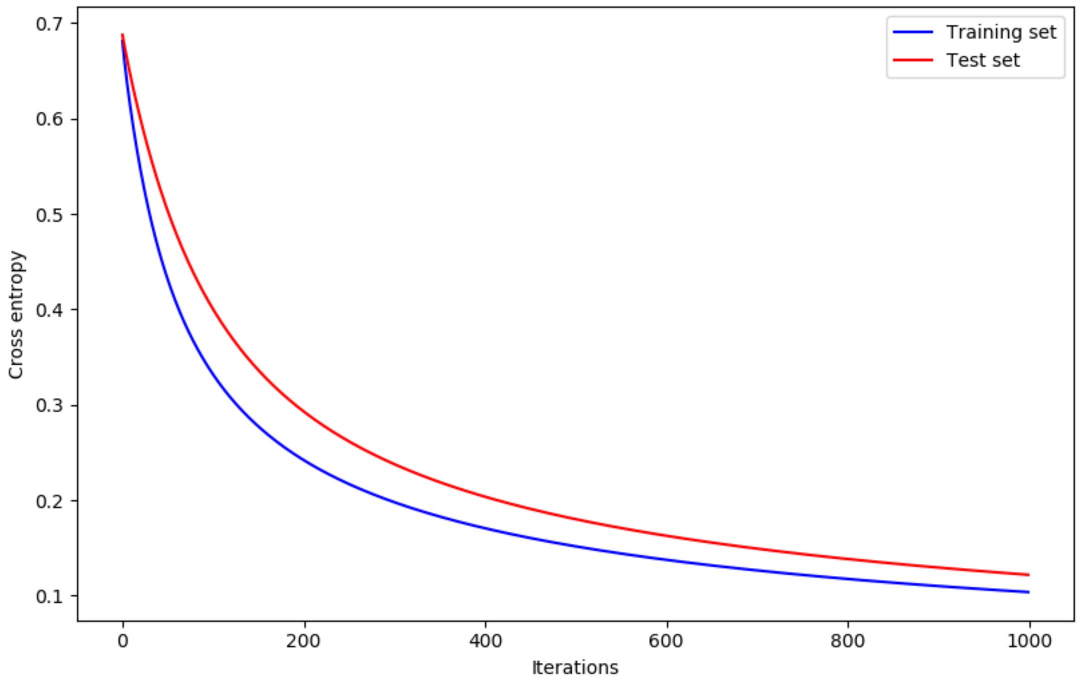

Figure 2: Cross entropy error development for both the training and test data set

   Figure 3 shows the decision boundary after running through the learning set. The blue dots represents inputs whose output should be 1, and red represents inputs whose output is 0. As we can see, the fit is very good with only one point that is on the wrong side of the decision line. This tells us it is generalizing well.

2. From figure 4 we can see that there is no straight line that would be able to partition these points, so this data is not linearly separable. We also see the decision boundary calculated is not very good.

   In order to correctly classify the dataset we have to create more features by multiplying together the features we already have so we go from $\mathbf{w} = \{w_0, w_1, w_2\}$ to $\mathbf{w} = \{w_0, w_1, w_2, w_1^2 w_2^2\}$. This will allow us to create circular decision boundaries. The updated result was generated by *regression_ 1()* in *logistical_ regression.py*, and the result can be seen in figure 5.

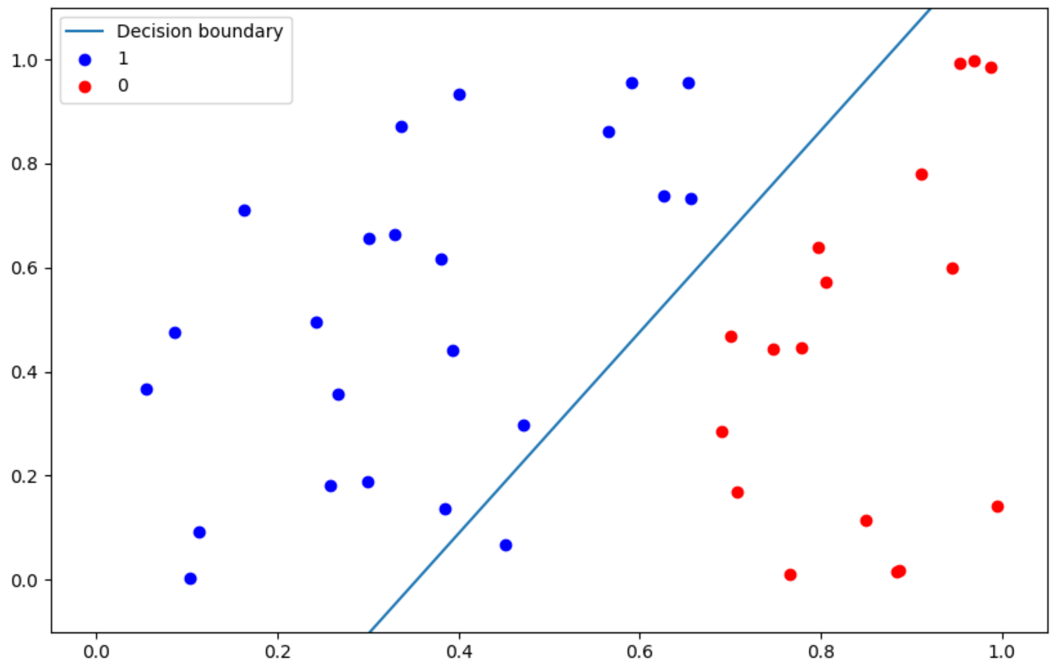

Figure 3: Results of the test data after training.

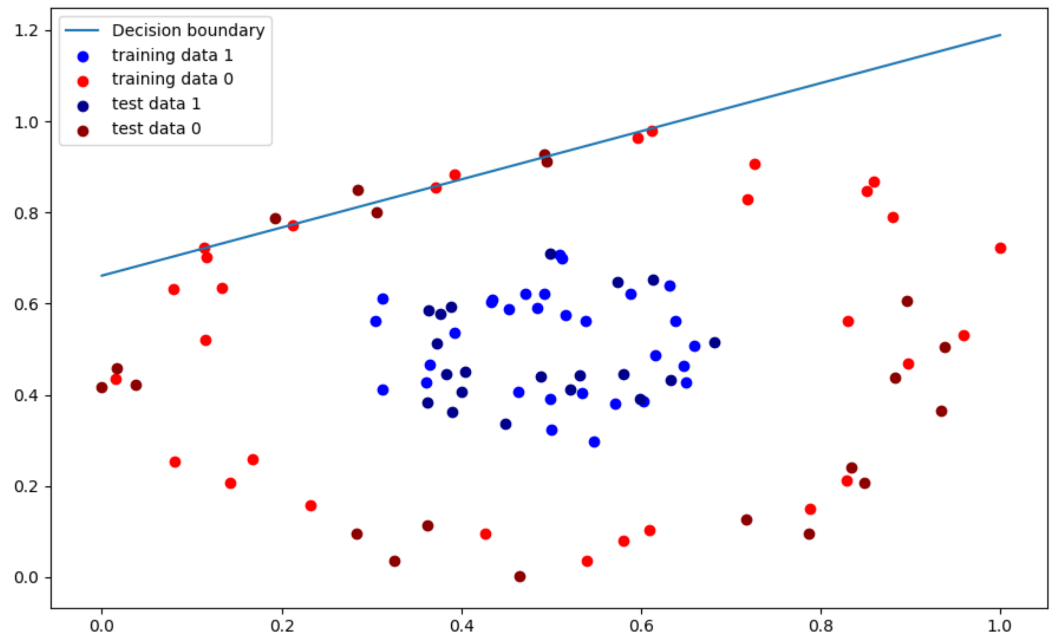

Figure 4: Plot of second data sets with decision boundary.

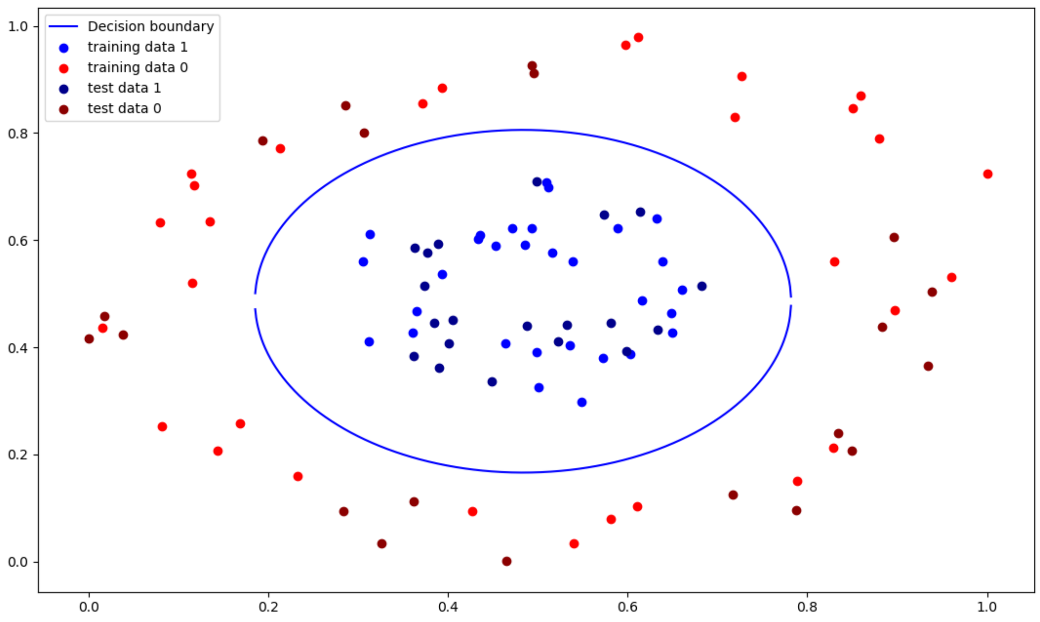

Figure 5: An updated plot with the new decision boundary.