TDT4173: Machine Learning and Case-Based Reasoning
Assignment 1
Stian Ismar

1 Theory:

1.1 *Concept learning* can be defined as searching through a space of already defined hypothesis for training examples, and finding the most suitable hypothesis. This can be done using the Find-S algorithm. When doing concept learning on a data set, one can for example try to find a hypotheses for a data set with the following features and target:

f1	**f2**	**f3**	**target**
Temp (high,medium,low)	snowing(Yes/No)	wind(Strong/weak)	Good day for skiing (Y/N)

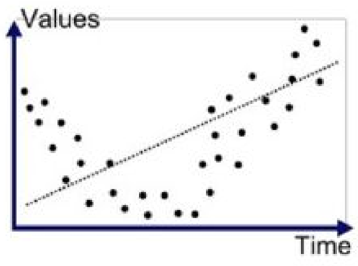
The goal of the concept learning task is to find the "best" concept-description from the training data.

1.2 *Function approximation* is a search through a space of hypotheses for one that best fits our training data. The search and learning methods vary. They are important because they allow us to create models of the input data.

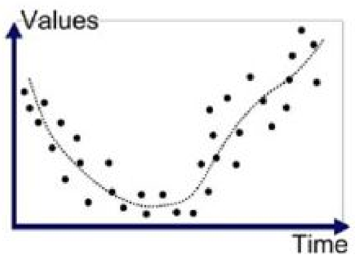
1.3 *Inductive bias* is used by every machine learning algorithm to generalize beyond the seen data. It is a set of assumptions that the learning uses to predict output given inputs that it has not yet seen. This is essential for our learning algorithm to have any use for unseen data.

There are two types of bias- search bias and representational bias. In decision tree learning, we have the type *search bias*. The search bias chooses which part of the space to search, since all hypothesis are represented. The candidate elimination algorithm has a representational bias. The bias searches the whole hypothesis space. It has this bias because it cannot represent all hypotheses.

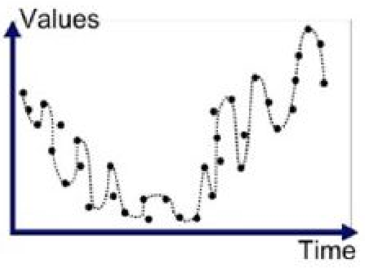
1.4 A hypothesis *overfits* training data if there is another hypothesis that generalizes better. This can happen if a model is trained with too much data, or if there is noise in the data. There are many ways of avoiding overfitting the data. The opposite, *underfitting,* happens when the model lacks complexity to capture the relationships between the features and targets. One should strive to find a good fit for the training data in order to have a useful system.



Underfitted



Good Fit/Robust



Overfitted

In machine learning problems we often have a training set, a validation set and a test set. The training set is used for learning on the model. The validation set is used to tune the parameters in the system, in order to find the optimal algorithm to use on the training set. The error of the algorithms on the validation set are studied, and the best one is selected.

Cross validation is used to estimate the prediction skill for a ML model on unseen data. Cross-validation is when you randomly split the training data into more than one subset. For each subset:
- Hold off the subset while training the model on the rest
- Evaluate the model on the test set
- Retain the evaluation score for each iteration
- After doing this for every subset, average the resulting test score for each iteration.

Cross-validation lets the designer know that the model is overfitting if the test error is high compared to the training error.

1.5 Candidate elimination

For this task, I used the Candidate Elimination Algorithm on the data given to describe the version space, specific hypothesis and general hypothesis. Every member of the version space lies between the General boundary G and the specific boundary S.

For the third fourth iteration, when looking at d = Male, Neck, High, Medium with target "Yes". In this iteration, we need to remove the only hypothesis in G, because it is inconsistent with d. "s" is also inconsistent with d, so it is removed. That leaves ut with an empty set for both the General boundary G and the specific boundary S.

The algorithm ran like this:

Iterations number	**G**	**S**
0	{<?,?,?,?>}	{<∅,∅,∅,∅>}
1	{<?,?,?,?>}	{<F,B,M,M>}
2	{<?,?,?,?>}	{<F,?,M,?>}
3	{<?,?,M,?>}	{<F,?,M,?>}
4	{}	{}

Discussion:

There could be multiple reasons as to why the General hypothesis and the Specific hypothesis become the empty set. One reason could be that the data is incorrect. Another reason could be that some relevant features were left out. This would lead to the wrong approximated model.

2 Programming

Task 1: Linear regression

2.1.1:

Implementing linear regression using OLS:

Linear regression with ordinary least square was implemented in Python with NumPy. These operations could have been done in one line, but I chose to split them up so that the code would be easier to read.

The data was loaded into **Y** and **X** matrices.

```
def loadCSV(filename):
    data = np.loadtxt(filename, delimiter=',', skiprows=1,
unpack=False)
    numOfColumns = data.shape[1]

    X = data[:,range(data.shape[1]-1)] # Kolonne 1 og 2 kommer ut for
2D
    Y = data[...,(numOfColumns-1)]
    Y = np.matrix(Y).T
    xarray = data[...,0] # Need to get this on a matrix-form!
    return X, Y, xarray, data, numOfColumns
```

```
# Getting the correct X-matrix:
ones = np.ones(X.shape[0])
X = np.c_[ones,X] # Merge two matrices
```

Calculating the weights:

```
# Calculating the weights:
Xtrans = np.transpose(X)

XtransDotX = np.dot(Xtrans, X)
```

```
# inverse of this result:
matrix1 = np.matrix(XtransDotX)

# Inverse it
matrix1 = matrix1.I

# Xtrans dotted with Y:
XtransDotY = np.dot(Xtrans, Y)

# Finally, to get w:
w = np.dot(matrix1, XtransDotY)
```

2.1.2:

train_2d_reg_data and test_2d_reg_data:

```
X,Y, xarray, data, numOfColumns = loadCSV('train_2d_reg_data.csv')
```

The weights calculated for the training data:

w0	w1	w2
0.24079271	0.48155686	0.0586439

Table 1: The weights for the 2D training data using least squares.

The model error error implementation:

```
def model(w,x):
   return x.dot(w.T)

def mse(w,x,y):
    error = np.mean(np.square(model(w,x)-y))
    return error
error = mse(w,X,Y)
print(error)
```

The errors for the 2D training and test set:

Training set error	Test set error
0.010387	0.0095298

Table 2: The errors for the 2D data set using mse(w, x, y).

Generalising model:

The error for the test data is *better* than for the training data. Because of this, I thought there might have been an error in the implementation. I checked that the weights were not calculated for the test data, and they were not. It would make sense that the test data would have a little larger mean error than the training error. A model is said to be generalising well if the error is small on unseen data, which it is here.

2.1.3:

train_1d_reg_data and test_1d_reg_data:

```
X,Y, xarray, data, numOfColumns = loadCSV('train_1d_reg_data.csv')
```

The resulting weights and errors are in the tables underneath. The weights were only calculated on the training data.

w0	w1
0.19559	0.61289

Table 3: The weights from the training data using least squares.

Training set error	Test set error
0.013759	0.012442

Table 4: The weights from the training data using least squares.

Plot of data and model

The training data was plotted along with the linear model.

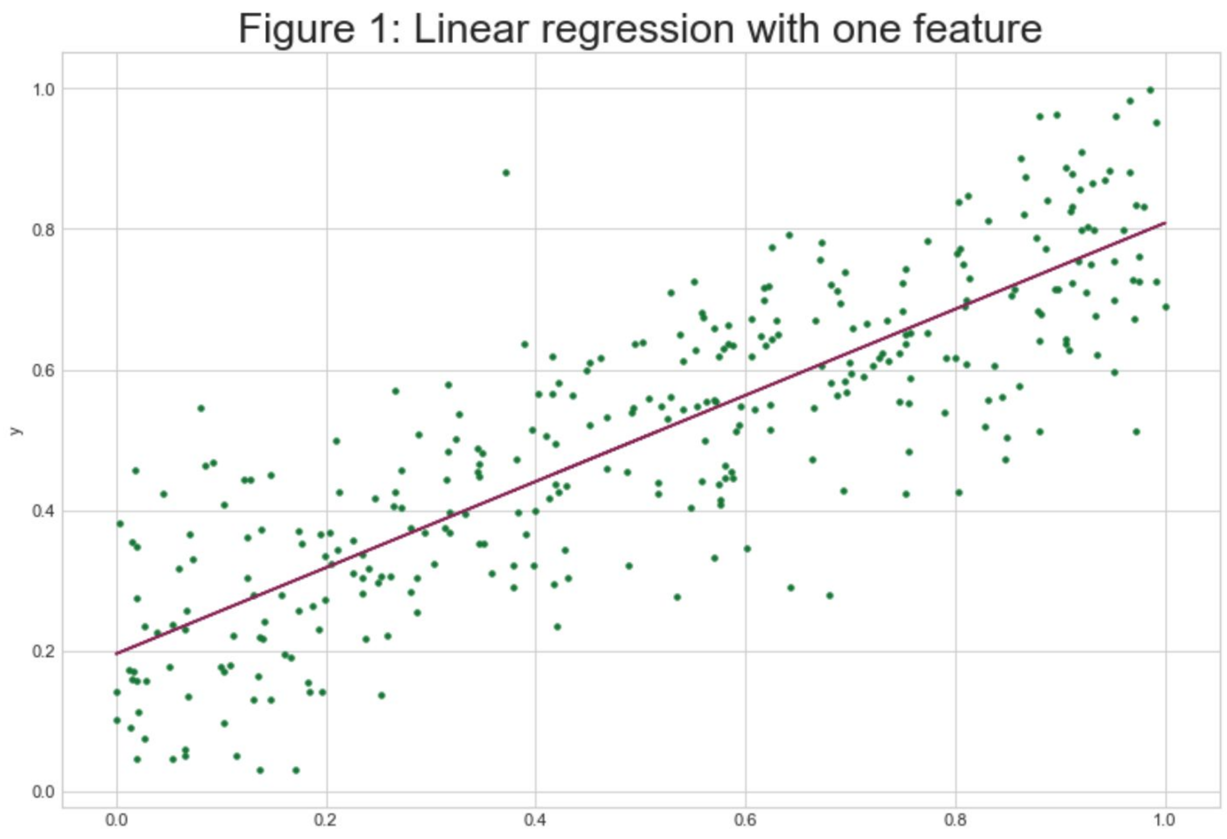


Figure 1: Training data with model

The test data was plotted with the model trained from the weights:

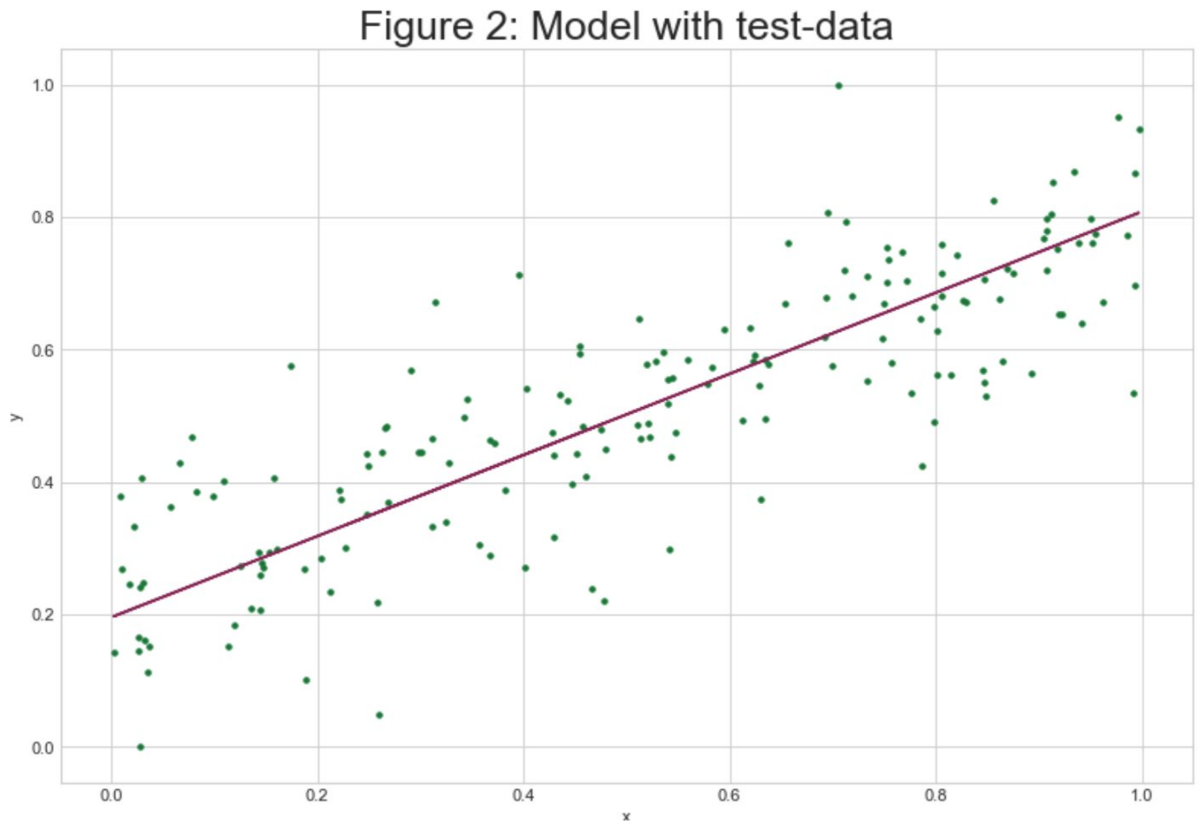


Figure 2: Training data with model

Task 2: Logistic regression

2.2.1:

c1_train_1.csv and c1_test_1.csv:

Error calculation

The data is loaded with the *loadCSV*-function in the file “Task 2_LogisticReasoning”. Logistic regression is implemented with using the functions *linearSignal*, *logistic* and *prob*. The cross-entropy error is calculated with the function *CEE*(X,Y,trainedWeights). This error was plotted against iterations.

The final error after 1000 iterations shows that the training data-error is a little higher than for the test data. This result makes sense, since the weights were calculated on the training data.

Initial Weights	Learning rate	Error_1000 iterations (Training data)	Error_1000 iterations (Test data)	Calculated weights
[0.5, 0.5, 0.5]	0.2	0.01471	0.07293	[12.20510561, -34.23846311, 17.66821299]

Table 1: Error results for task 2.2.1.

Figure 1: Cross Entropy error for each iteration

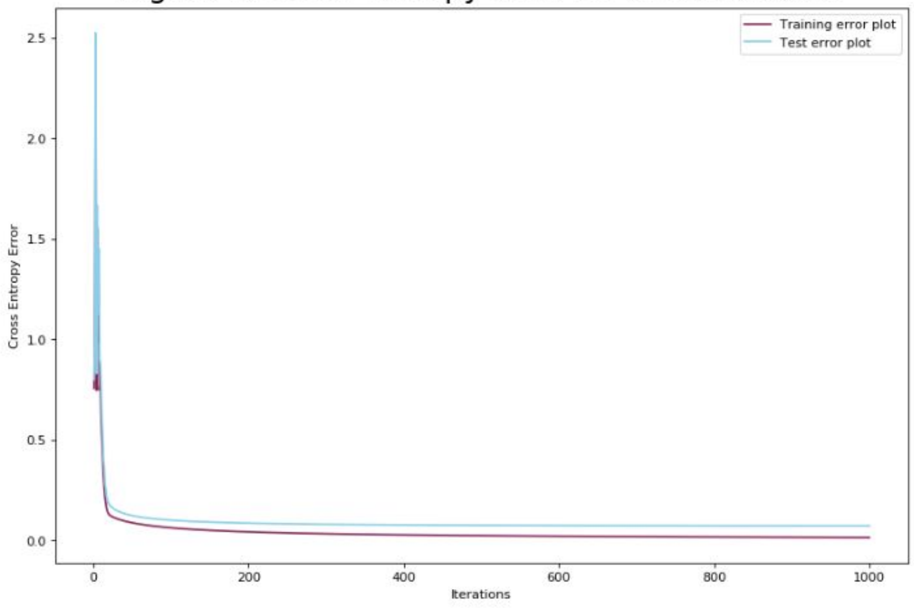


Figure 3 : Cross entropy error for the training and test data.

Separability- Decision boundary plot

The weights are calculated on the training set using the *gradientDescent*-function. With the weights calculated over 1000 iterations, the points were classified and plotted with two different colours. If the z-value was equal or larger than 0, the point would be plotted as blue, and red for the other values.

The data was plotted using linspace, meshgrid and contour from the numpy library in Python:

```
# Plotting the model z = h(x,w) as well:
x_axis = np.linspace(0,1,1000)
y_axis = np.linspace(0,1,1000)

xx, yy = np.meshgrid(x_axis,y_axis)
z = weights[0] + weights[1]*xx + weights[2]*yy

plt.contour(xx, yy, z, levels=1, colors='green')
```

The training and test-data is linearly separable, simply because it is possible to separate the classified data points using a linear decision boundary. This can be seen in the figure under, where the two features are plotted against each other.

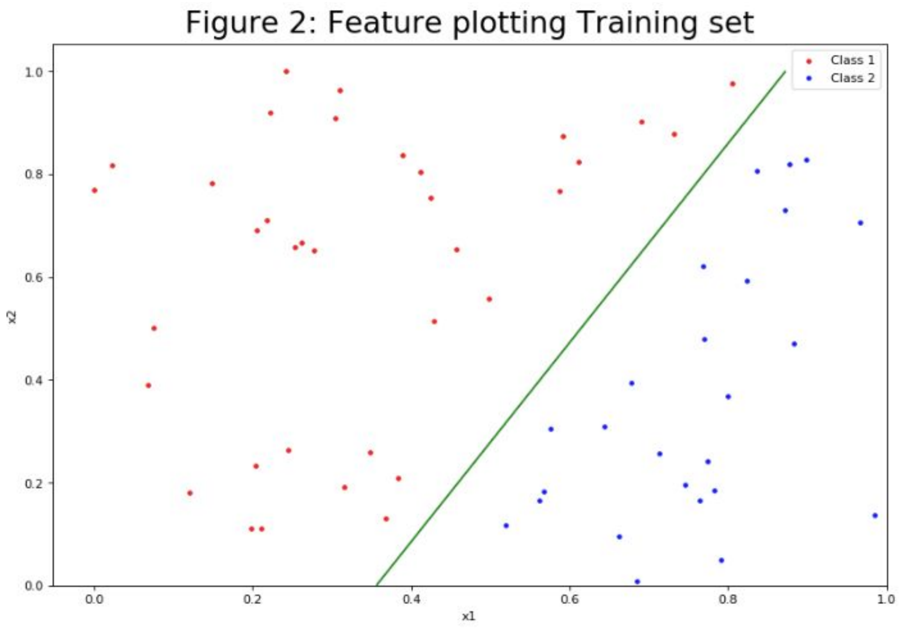


Figure 4: Model with the feature plot from the training set.

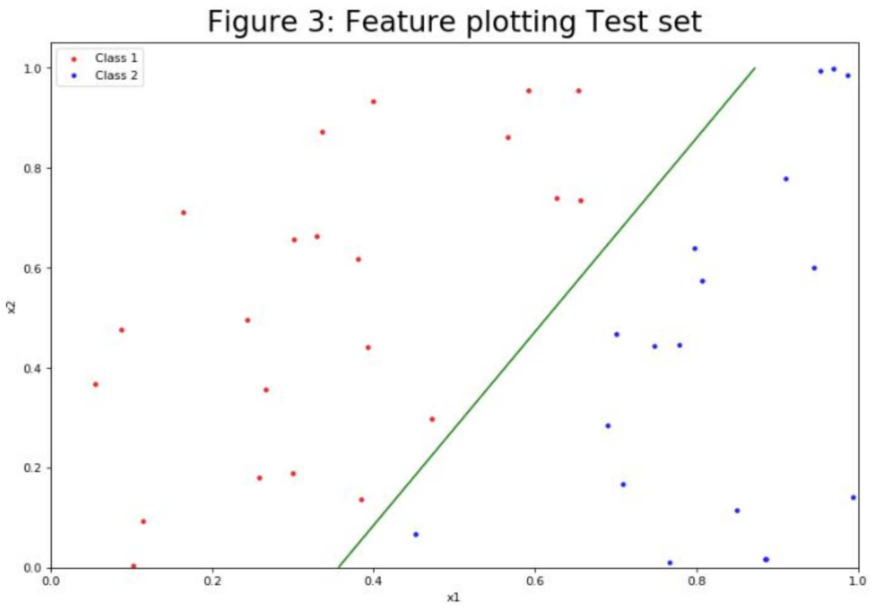


Figure 5: Model with the feature plot from the test set.

Model generalising

The percentage increase in cross-entropy error from the training data to the test data was very small, so the model is generalising well. The error-increase was only 0.0396 % from the training set to the test set. Ideally, the error from the training set is equal to the test set. However, this is rarely the case, since the test data has not been "seen" by the algorithm before.

2.2.2:

c1_train_s2.csv and c1_test_2.csv:

For this circular boundary problem, som changes were made to **X**. Two columns were appended to it using column_stack in numpy. Every vector in **X** was made to the form x_n = [1, x1, x2, $x1^2$, x^2]. **w** also had to be extended to [0.5, 0.5, 0.5, 0.5, 0.5]. This was done in NumPy like this:

```
XTrain_2 = np.column_stack((XTrain_2,(XTrain_2[:,1])**2))
XTrain_2 = np.column_stack((XTrain_2,(XTrain_2[:,2])**2))

XTest_2 = np.column_stack((XTest_2,(XTest_2[:,1])**2))
XTest_2 = np.column_stack((XTest_2,(XTest_2[:,2])**2))
```

Error calculation

The error results are shown underneath from training the weights using gradient descent on the training data.

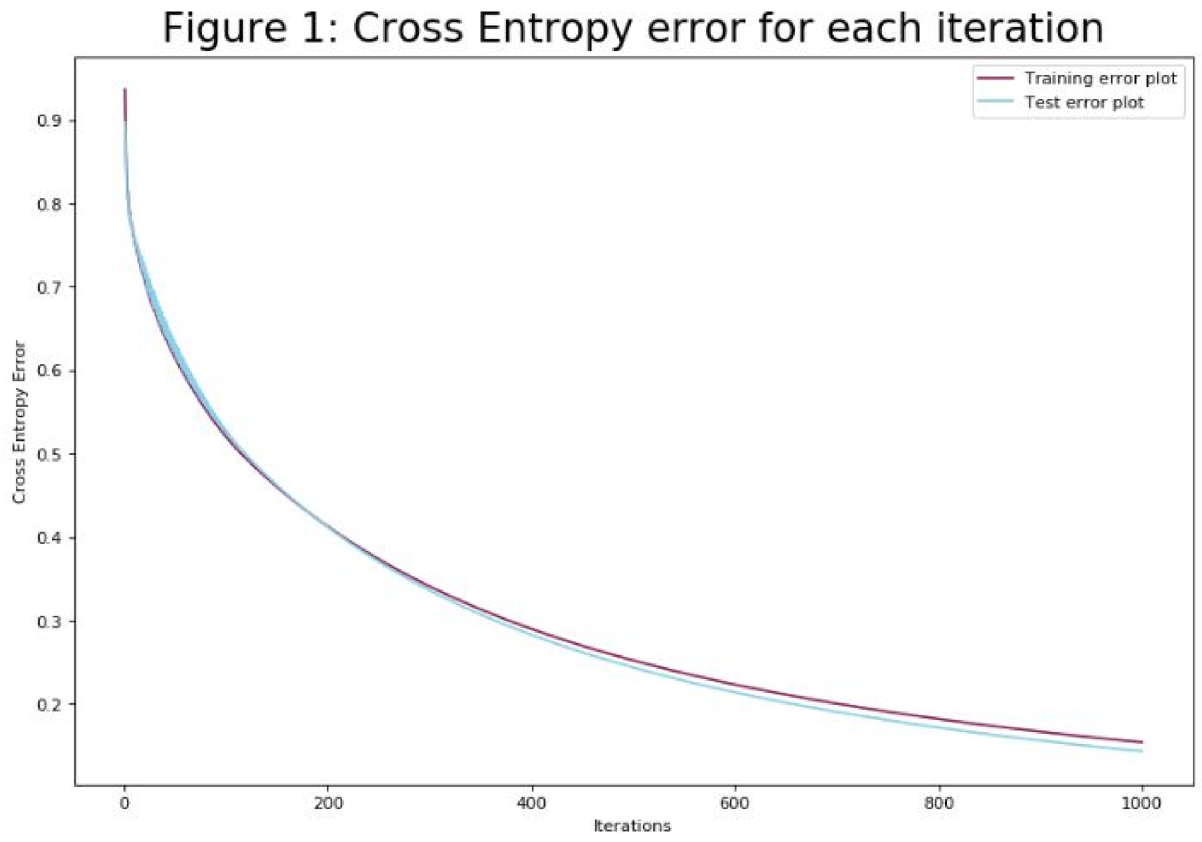


Figure 6: The error for the two datasets.

The same algorithms were used to train the weights here as in task 2.2.1.

Initial Weights	**Learning rate**	**Error_1000 iterations (Training data)**	**Error_1000 iterations (Test data)**	**Calculated weights**
[0.5, 0.5, 0.5, 0.5, 0.5]	0.2	0.1539954	0.143256	[-8.57652259, 23.15317832, 21.66377195, -23.37833469, -22.37205613]

Table 1: Error results for task 2.2.2.

Separability- Decision boundary plot

After classifying the points using the calculated weights from the training set, the decision boundary was plotted. To do this, linspace, meshgrid and contour was used to print this. The equation plotted with the contour function looked like this:

```
    # Plotting the model z = h(x,w) as well:
    x_axis = np.linspace(0,1,1000)
    y_axis = np.linspace(0,1,1000)
    xx, yy = np.meshgrid(x_axis,y_axis)
    z = weights[0] + weights[1]*xx + weights[2]*yy + weights[3]*xx**2 + 
weights[4]*yy**2

    # Plotting the contour with levels = 0:
    plt.contour(xx, yy, z, levels=0,  colors='green')
```

From the plot with the circular boundary, I can see that it is possible to separate the points in the two classes by a circular boundary. If the plot was changed from cartesian coordinates to polar coordinates, one would be able to see that the data was linearly separable. See the figures under for the training set and test set.

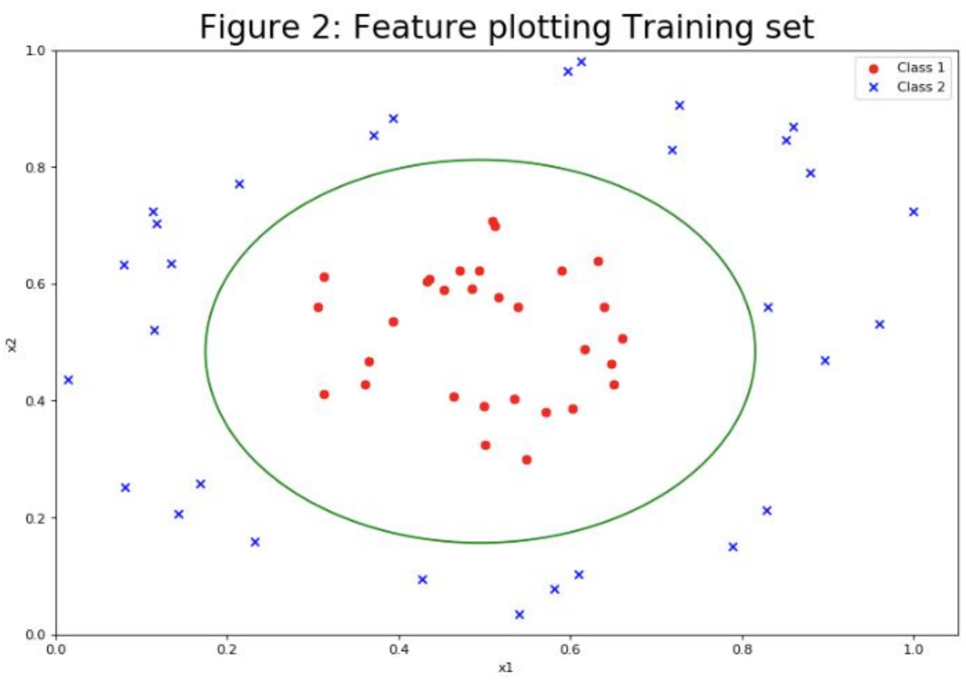


Figure 7: Feature plot with the non-linear decision boundary.

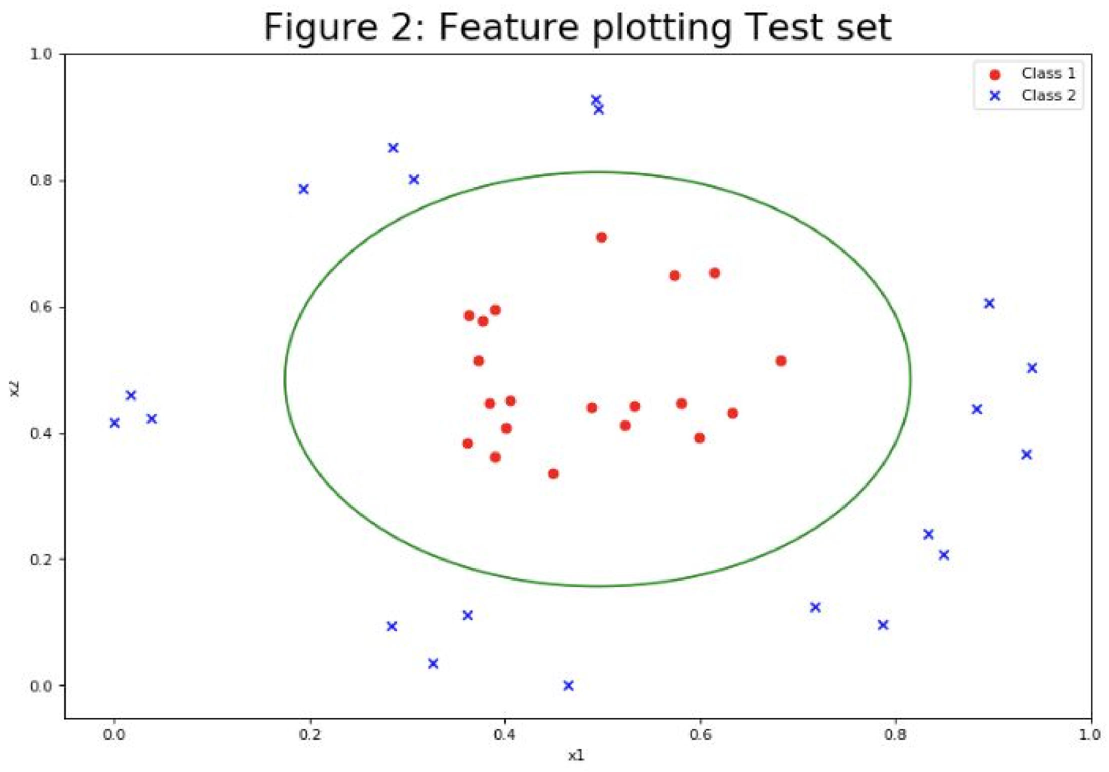


Figure 8: Feature plot with the non-linear decision boundary.

Model generalising

The model generalises well, with little error. However, the cross-entropy error for the test set is actually smaller than the error for the training set. This seems like an error, but I could not find where the possible bug was. I made sure the weights are being trained on the training data, and not the test, so this was not the problem. One source for the error might be that the model is overfit to the training data.