TDT4173 Machine learning and Case-Based Reasoning - Assignment 1

Trude Jostad

January 2018

1 Theory

1.1

Concept learning is the process where we search through the set of hypothises for the best hypotesis to distinguish if set of attributes is member of a group or not.

Example Check if a movie review that is part of a movie review set is a good movie review or not.

1.2

The function approximation are the function that updates the weights so that they best fit the training examples. Without the function approximation it would not be possible for the machine to learn (approximate the target function).

1.3

The inductive bias is a set of assumptions of the dataset. This is used to predict output of a given input.

In the candiate elimination algorithm is find in the version space. It can some sometime be to speific if it has too many training examples and overfitting can occurs.

In decistion tree inductive bias is in the book characteried that it is preferred to have kort trees with information gain attributes close to the root.

1.4

Overfitting is when the are a hypothesis h that has less error over the entire distribution of instances, but the chosen hypothesis h' has less error over the training set than h. Underfitting is when the training dataset is to small and makes the difficult to chose the correct hypothesis.

A validation set is a set of test data that is not in the training set. It is used to see if the system works after the training. Cross-validation is when the programmer runs to iterations through the training set and tests it with the validation set. If the error if very different a overfitting may have occured.

1.5 Candidat elimination

In the table 1 is the data that I will apply the candiate elimination algorithm. A version set is a subset of the hypothesis that is consistent with the training examples.In the candiate elimination algorithm is the version space represented by is most general and most specific members. With only storing these it is possible to enumerate all members of the version space.

ID	**SEX**	**Problem Area**	**Activity Level**	**Sleep Quality**	**Treatment Successful**
t0	Female	Back	Medium	Medium	Yes
t1	Female	Neck	Medium	High	Yes
t2	Female	Shoulder	Low	Low	No
t3	Male	Neck	High	High	Yes
t4	Male	Back	Medium	Low	Yes

Table 1: Training Set

To begin I have to initialize the most general hypotheses and most specific in the version space. They are shown i 2. The ? mean that it accepts all attributes and ø means it accepts none attributes.

Initilizes the most general and the most specific hypothesis
$S0 : \{\emptyset, \emptyset, \emptyset, \emptyset\}$
$G0 : \{?, ?, ?, ?\}$

Table 2: initilization

From the t0 we see that it is not consistent with S0, because all attributes are set to ø that means that it do not accept any attributes. Then updates the most specific hypothesis to be S1 that is equal to the t0. The most general hypothesis has not change, because it is consistent with t0.

Adds training example 1
$S1 : \{Female, Back, Medium, Medium\}$
$G0, G1 : \{?, ?, ?, ?\}$

Table 3: Iteration 1

After adding t1 the spefic hypothesis is not consistent with this example. To make it fit S needs to generalizing the Activity level and sleep quality attributes. t1 is consistent with G1 that makes G2 = G1.

Adds training example 2
$S2 : \{Female, ?, Medium, ?\}$
$G0, G1, G2 : \{?, ?, ?, ?\}$

Table 4: Iteration 2

The next training example t2 is negative example. Here is not Activity Level Medium, but it is Low. So the algorithm says that to add to G the minial specializations such that it is consistent with the training examples. That makes the Activity Level attribute medium at the general hypothesis. There is no inconsisten in S2.

Adds training example 3
$S2, S3 : \{Female, ?, Medium, ?\}$
$G0, G1, G2, G3 : \{?, ?, Medium, ?\}$

Table 5: Caption

The next training example t3 does not fit S, because the SEX is Male and Activity Level is High. The S hypothesis is now updated the SEX and Activity Level so it is more general and fits all the training examples. The G hypothesis is now removed, because it does not fit the traning examples and is updated back to the more general before the negative training example.

Adds training example 4
$S4 : \{?, ?, ?, ?\}$
$G0, G1, G2, G3, G4 : \{?, ?, ?, ?\}$

Table 6: Caption

The last training example does not change S and G, because it is consistent with the traing example.

Adds training example 5
$S4 : \{?, ?, ?, ?\}$
$G0, G1, G2, G3, G4 : \{?, ?, ?, ?\}$

Table 7: Caption

2 Linear Regression

First I made a modular python file called vector.py to excrated data from the csv file to create vectores called createCsv that returns X and Y np.array.

2.1

To implement the linear regression with ordinary least squares I used the equeation: $w = (X^T X)^{-1} X^T y$ In my code it is called OSL(x,y). x and y is numpy arrays and used numpy methods lik np.dot(vector multiplication), np.transpose and p.linalg.pinv(invers) to calculate the equation.

2.2

First I call the method train(filepath) with the path to train_2d_reg_data_csv. Then I call the OLS with the X and Y vectors to calculate the weights w.Then I call the calc_h with w and the X vector to calculate the estimated value for each xi. For these I use the eqation $h = w^T x$ and append the result to the H vector. The train method returns X,Y,w and H vectors that are used in print_info as arguments. In this method the calc_error method is called that uses the X,Y,w,H vectors as arguments.This method uses the mean square equation $e = \frac{1}{N} \sum_{i=1}^{n} (w^T xi - yi)^2$

I think it generalized well, because shown in figure 1 has not very large difference in error in the training set verous the test set.

```
Weights  [[ 0.24079271]
 [ 0.48155686]
 [ 0.0586439 ]]
Error [ 0.01038685]
Error of testdata  [ 0.0511435]
```

Figure 1: Weights, error training data and error test data

2.3

In this task I only changed to the new training set, test set and changed the methods calling. Except that I did the same as in 2.2.

```
Weights  [[ 0.1955866 ]
 [ 0.61288795]]
Error [ 0.01375879]
Error of testdata  [ 0.07023063]
```

Figure 2: Weights, error training data and error test data for data in task 2.2

To plot I created a method called plot_result that takes in the X and Y from the training set and test, the weights and H from the training and test set. For the ploting I used a python libiry called matplotlib. For ploting the data I used

X second element for the x-axes values and Y for the y-axes values. For ploting the line I used X second element for the x-axes values and H for the y-axes values. Se the code for more detailed information.

I think the line fits well, because it is the same line I would have drawn in both the traing and test case.

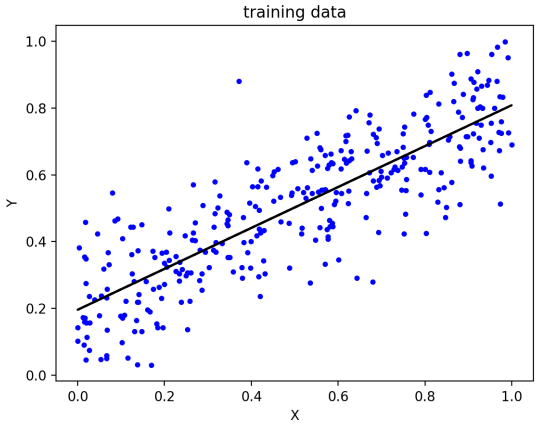

Figure 3: Blue dotes is the training data plot and the line is the x values with h

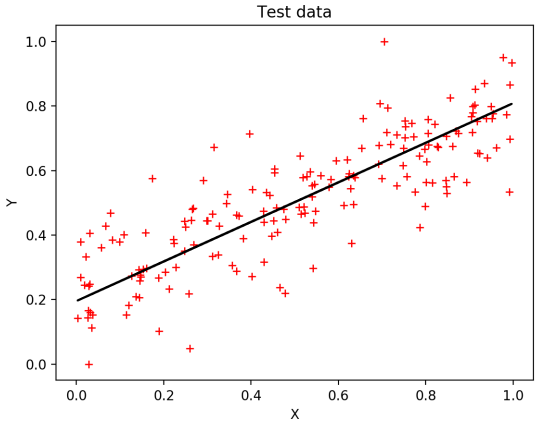

Figure 4: Red pluses is the test data plot and the line is the x values with h

3 Logistic Regression

3.1

After trying out different weights and learning rate I figured out that the best weights was $W = [0.1, 0.1, 0.1]$ and $LearningRate = 0.1$. I will say that my model is linear separable, because it separate them into the two groups in figure 6. In figure 7 it has only one dot that is on the wrong side of the line for the test data.

First I created the train method to estimate the weights. I did this in the method by calling the calc_weight 1000 times. The calc_weight method used the equation $w(k+1) = w(k) - n\sum_{i=1}^{N}(\sigma(w(k)^T xi) - yi)xi$ And for calculating the error I used the method calc_error that used

$Ece = -\frac{1}{N}\sum_{i=1}^{N} yi \ln \sigma(z) + (1 - yi)\ln(1 - \sigma(z))$

$z = w^T x$

$\sigma(z) = \frac{1}{1+e^{-z}}$

For calculation the plot I used

$0 = w0 + w1X1 + w2X2 \rightarrow X2(z) = -\frac{(w0w1X1)}{w}$

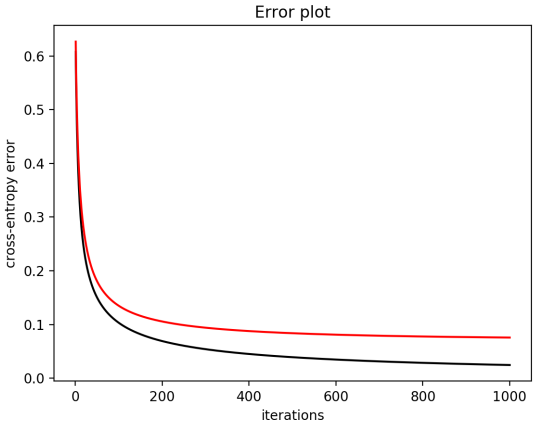

Figure 5: Result of error after each iteration where the black line is the training set and the red is the test set

3.2

In this case the decistion boundary is not linear separable shown in figure 8. The red dots that is the positive X is clustered together in a more circle form. This means that it is not possible to seperate the negative and positive X with a linear line. Instead it is possible to use the hypothesis from a unlinear boundary

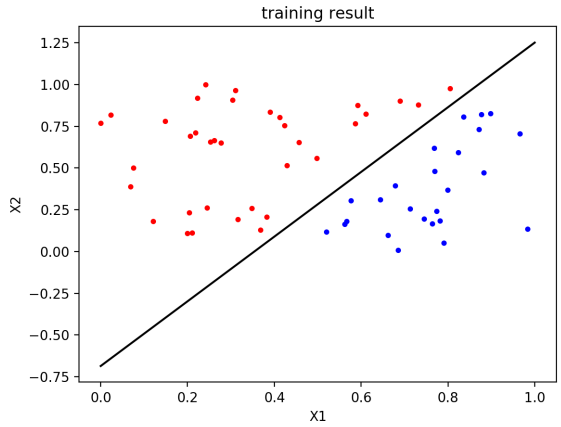

Figure 6: Blue dotes Y= 0 and red dotes Y = 1

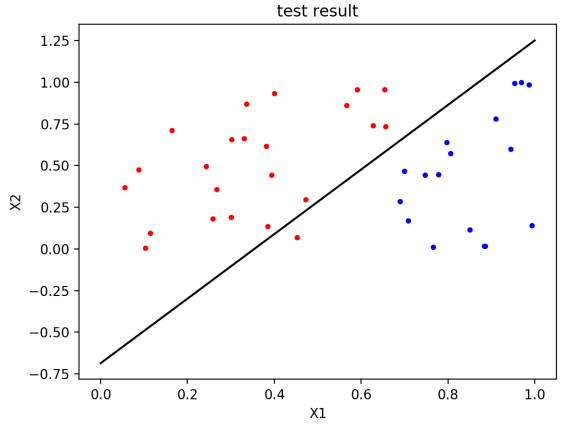

Figure 7: Result testing of c1

decistion $H = w0 + w1X0 + w2X2 + w3X1^2 + w4X2^2$. To make this happend X need to append $X1^2$ and $X2^2$. I solved this by calling the create_x_padd to do this. I happended a new parameter in train and get_value_from_file so that I knew when to pad the X. The new decistion boundary is function to call when it is not linear is called circle_h from the new equation. It uses a contour to plot the non linear decition boundary. See the code for more detailed implementation.

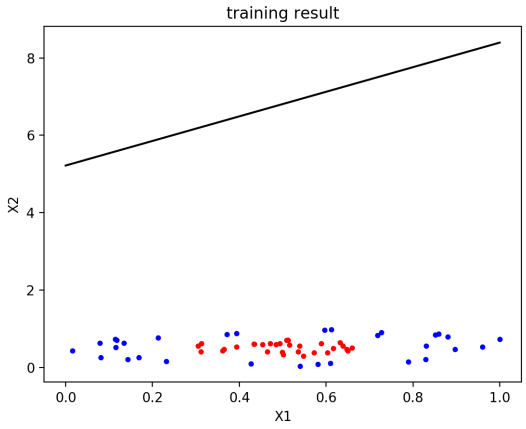

Figure 8: Linear decition boundary on the c2 dataset

.

```
Weight:  [ -9.74103026  26.16888498  24.52636526 -26.43587329 -25.26835216]
```

Figure 9: The weights after training with extended X

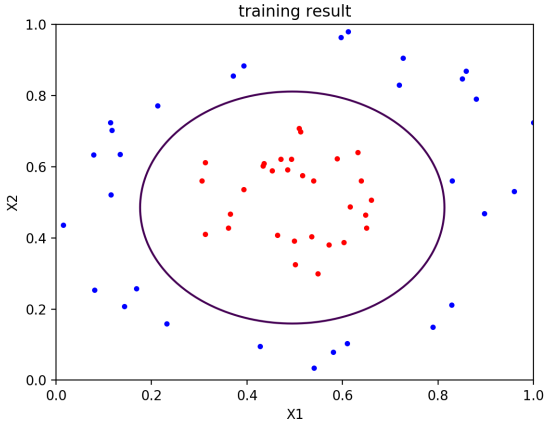

Figure 10: Result of the decition boundard on the training set

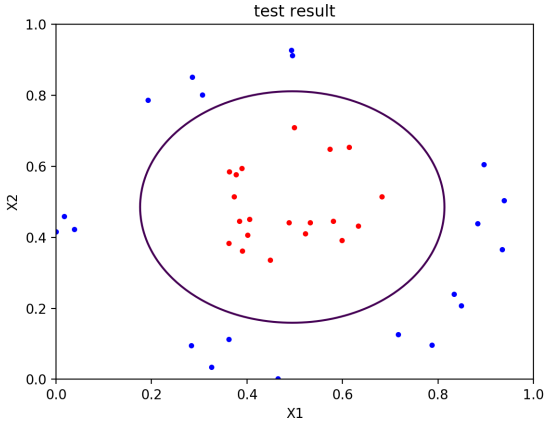

Figure 11: Result of the decition boundard on the training set