

Theory

1. Concept learning

A concept can be viewed as a mental category to be used for classification; these classifications can be viewed as a binary decision of either inside or outside the concept category. Concept learning is learning a strategy for classifying these concepts, usually based on a training set.

An example can be some domain containing a certain set of attributes and an output result, e.g: Answering if it will rain on a day based on the days before humidity, rain, temperature, weekday and month (Some of these are obviously more relevant than others)

2. Function approximation

Function approximation is to find the best fitting function to a certain data set, often in machine learning viewed as a search for the best fitting function through a space of hypothesis functions. This is essential for machine learning as it enables us to find the best hypothesis based on the training data.

3. Inductive bias

Inductive bias is the nature of assuming an output given an input, based on training data where the given input does not occur. This is important in machine learning as it enables us to make a conclusion of "unknown" input.

The inductive bias for Candidate elimination is such that the hypothesis space contains the target concept, whilst for decision trees is a preference for short trees over long trees.

4. Over and underfitting

Overfitting is modeling the training data too well, thus also modelling excessive details and noise.

Underfitting on the other hand is modeling the training data too simply, thus missing out essential tendencies within the data. Both models will perform poorly.

Validation Set is a set of data to estimate how well the model has been trained and also to estimate model properties.

Cross-validation runs the model multiple times using subsets of the prepared data as trainingsets and validation sets. This enforces the model to not be specialized on the given training data, in turn making it more generalized, thus mitigating the damage done when overfitting. Since only doing this once may become a problem as the split itself could be an issue multiple iterations of splitting and training/testing is used in cross-validation runs.

5. Candidate model

Version space is the validated hypothesis, a subset of all the hypothesis, that is consistent with the training data. The resulting version space, specific and general boundary is computed below.

We see here that all of them resulted in the general boundary, as there are no consistent attributes throughout the training data regarding the successful treatment.

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

$S_0 = \{\emptyset, \emptyset, \emptyset, \emptyset\}$

$G_0 = \{?, ?, ?, ?\}$

Case 1 = Female, Back, Medium, Medium → Yes (Specific is made more general)

$S_1 = \{\text{Female, Back, Medium, Medium}\}$

$G_1 = \{?, ?, ?, ?\}$

Case 2 = Female, Neck, Medium, High → Yes (Specific is made more general)

$S_2 = \{\text{Female, ?, Medium, ?}\}$

$G_2 = \{?, ?, ?, ?\}$

Case 3 = Female, Shoulder, Low, Low → No (General is made more specific)

$S_3 = \{\text{Female, ?, Medium, ?}\}$

$G_3 = \{?, ?, \text{Medium, ?}\}$

Case 4 = Male, Neck, High, Medium → Yes (Specific boundary becomes too general to hold any value)

$S_4 = \{?, ?, ?, ?\}$

$G_4 = \{?, ?, ?, ?\}$

Case 5 = Male, Back, Medium, Low → Yes (No change as we are at the most general boundary)

$S_5 = \{?, ?, ?, ?\}$

$G_5 = \{?, ?, ?, ?\}$

Result = $\{?, ?, ?, ?\}$

Task 2.1

```
def OLS(x,y): #Calculate Ordinary Least Square
    xt = x.transpose()
    r1 = np.dot(xt,x)
    r2 = np.linalg.inv(r1)
    r3 = np.dot(r2,xt)
    w = np.dot(r3,y)
    return w

def Emse(x,y,w): #Calculates Model Error
    Xw = np.dot(x,w)
    e = np.square(Xw - y).mean()
    return e
```

Main Function:

```
def main():
    folder_path = "datasets/datasets/regression/"

    print("----- 2.1.1 -----")
    train_path = "train_2d_reg_data.csv"
    x,y = get_xy(folder_path + train_path)

    w = OLS(x,y)
    print("Weights: {}".format(w))

    e = Emse(x,y,w)
    print("Emse Train: {}".format(e))

    test_path = "test_2d_reg_data.csv"
    xt,yt = get_xy(folder_path + test_path)
    e = Emse(xt,yt,w)
    print("Emse Test: {}".format(e))

    print("----- 2.1.2 -----")
    train_path = "train_1d_reg_data.csv"
    x,y = get_xy(folder_path + train_path)
    w = OLS(x,y)

    e = Emse(x,y,w)
    print("Emse Train: {}".format(e))
    plot(x,y,w,e,"train")

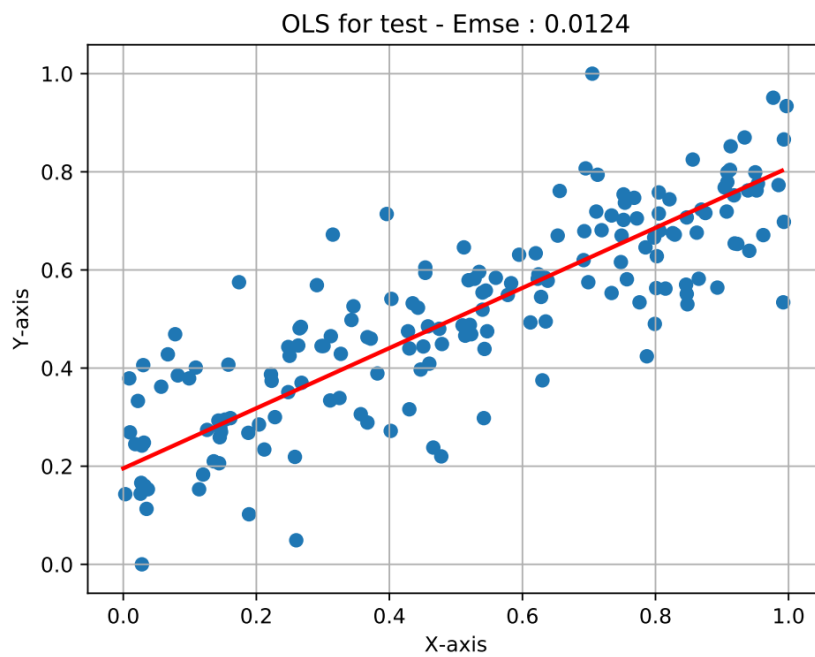
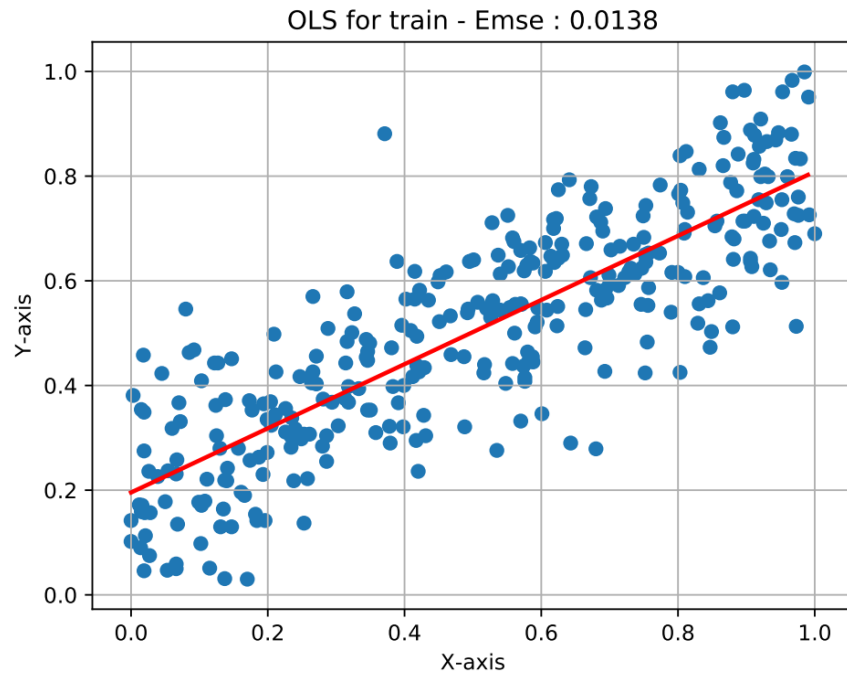
    test_path = "test_1d_reg_data.csv"
    xt,yt = get_xy(folder_path + test_path)

    e = Emse(xt,yt,w)
    print("Emse Test: {}".format(e))
    plot(xt,yt,w,e,"test")
```

Corresponding output with plots from 2.1.2 on next page:

```
C:\Users\Mikl1\Documents\Work\TDT41753\assignment1>1.1_practical.py
----- 2.1.1 -----
Weights: [0.24079271 0.48155686 0.0586439 ]
Emse Train: 0.010386850731462317
Emse Test: 0.009529764450618972
----- 2.1.2 -----
Weights: [0.1955866 0.61288795]
Emse Train: 0.013758791126537114
Emse Test: 0.012442457462048941
```

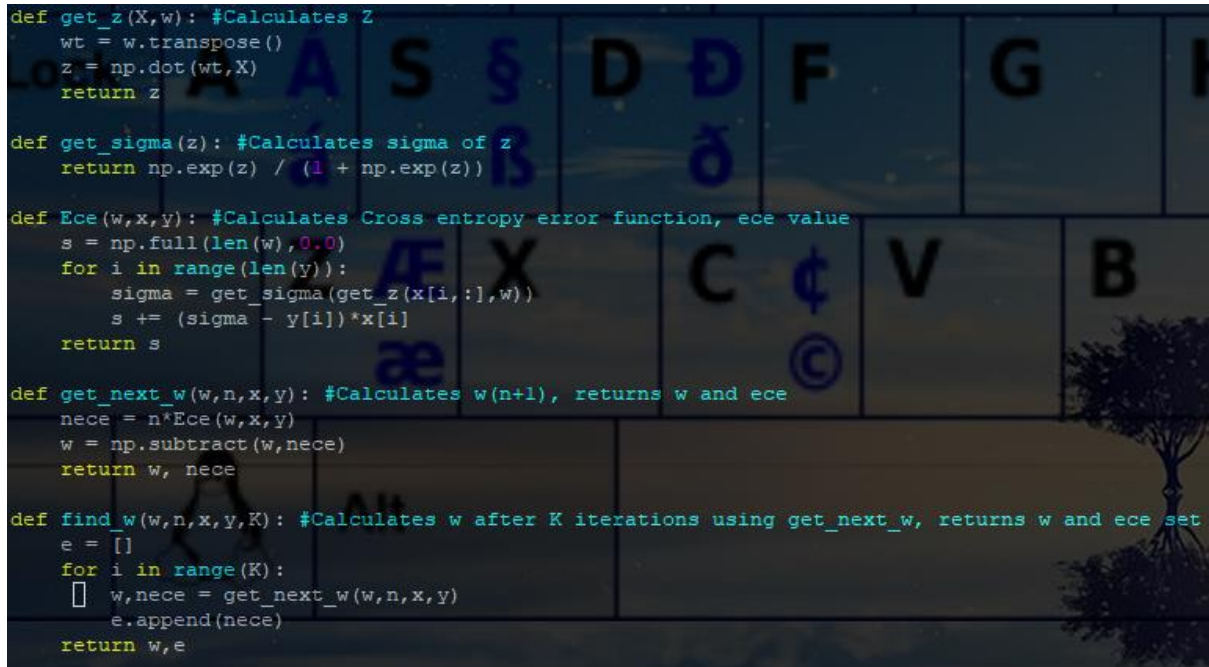
I believe the model is well generalized as we see the model error is low for both train and test data sets (actually lower in test)



Task 2.2

I chose learning rate of 0.1 and all starting weights of 1.0, as this was proven a well balance between generalized and specialized attributes.

The main functionality performing the calculations is shown in the following screenshot:



```
def get_z(X,w): #Calculates Z
    wt = w.transpose()
    z = np.dot(wt,X)
    return z

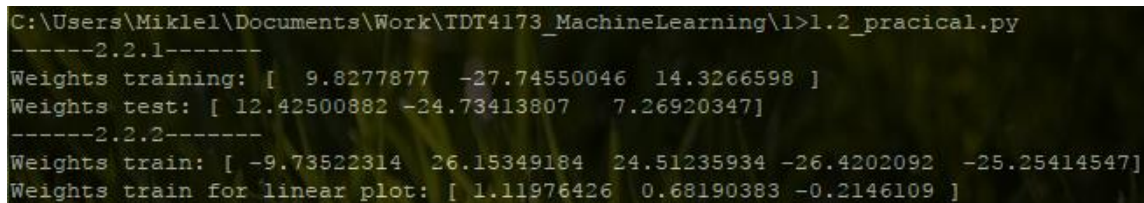
def get_sigma(z): #Calculates sigma of z
    return np.exp(z) / (1 + np.exp(z))

def Ece(w,x,y): #Calculates Cross entropy error function, ece value
    s = np.full(len(w),0.0)
    for i in range(len(y)):
        sigma = get_sigma(get_z(x[i,:],w))
        s += (sigma - y[i])*x[i]
    return s

def get_next_w(w,n,x,y): #Calculates w(n+1), returns w and ece
    nece = n*Ece(w,x,y)
    w = np.subtract(w,nece)
    return w, nece

def find_w(w,n,x,y,K): #Calculates w after K iterations using get_next_w, returns w and ece set
    e = []
    for i in range(K):
        w,nece = get_next_w(w,n,x,y)
        e.append(nece)
    return w,e
```

The main function running is shown below, which produced the following output in addition to the plots.



```
C:\Users\Mikl\Documents\Work\TDT4173_MachineLearning\1>1.2_pracical.py
-----2.2.1-----
Weights training: [ 9.8277877 -27.74550046 14.3266598 ]
Weights test: [ 12.42500882 -24.73413807 7.26920347]
-----2.2.2-----
Weights train: [ -9.73522314 26.15349184 24.51235934 -26.4202092 -25.25414547]
Weights train for linear plot: [ 1.11976426 0.68190383 -0.2146109 ]
```



```
def main():
    n = 0.1 #Learning rate
    K = 1000 #Iterations

    #Data paths
    folder_path = "datasets/datasets/classification/"

    print("-----2.2.1-----")
    #-----2.1 Train data-----
    train_path = "cl_train_1.csv"
    x,y = get_xy(folder_path + train_path)
    w = np.full(3,1.0)
    w,e = find_w(w,n,x,y,K)

    print("Weights training: {}".format(w))
    plot_e(e,"2_1_train")
    plot(x,y,w,True,"2_1_train")

    #-----2.1 Test data-----
    test_path = "cl_test_1.csv"
    xt,yt = get_xy(folder_path + test_path)
    wt = np.full(3,1.0)
    wt,et = find_w(wt,n,xt,yt,K)

    print("Weights test: {}".format(wt))
    plot_e(et,"2_1_test")
    plot(xt,yt,w,True,"2_1_test")

    #-----2.2 Train data-----
    print("-----2.2.2-----")
    train_path = "cl_train_2.csv"
    x,y = get_xy(folder_path + train_path)
    x = alter_x(x)
    w = np.full(5,1.0)

    w,e = find_w(w,n,x,y,K)

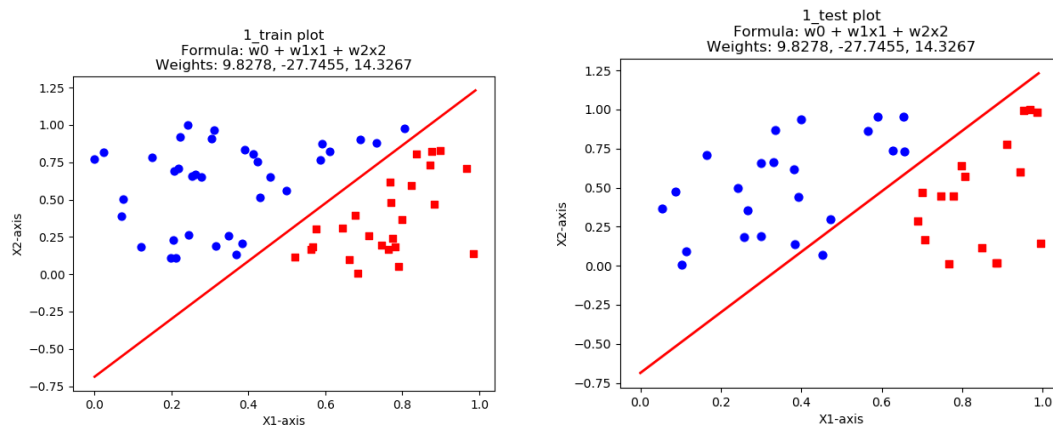
    print("Weights train: {}".format(w))
    plot(x,y,w,False,"2_2_train")
    plot_e(e,"2_2_train")

    #-----2.2 Test data-----
    xt,yt = get_xy(folder_path + train_path)
    xt = alter_x(xt)
    plot(xt,yt,w,True,"2_2_test")

    #-----2.2 Linear Plot-----
    xl,yl = get_xy(folder_path + train_path)
    wl = np.full(3,1.0)
    wl,e1 = find_w(wl,n,xl,yl,K)
    print("Weights train for linear plot: {}".format(wl))
    plot(xl,yl,wl,True,"2_2_linear")
```

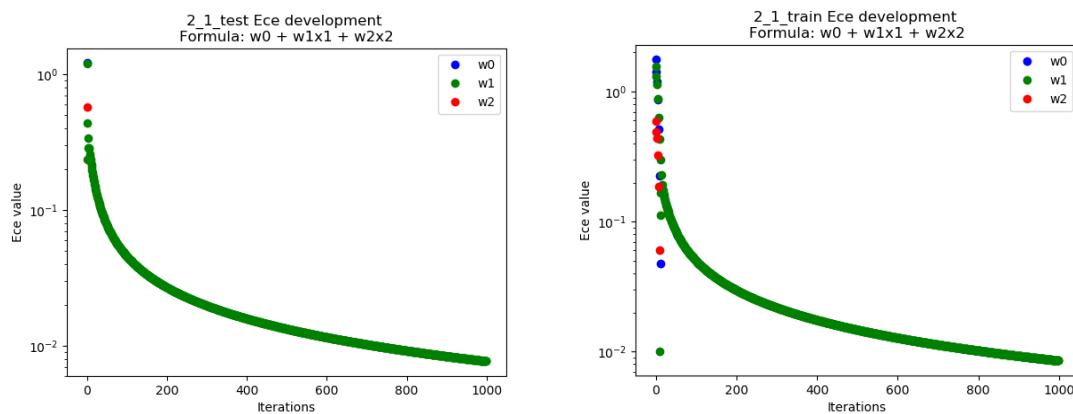
2.2.1

In assignment 2.1 the program produced the following plots with all starting weights equal 1.0 and learning rate 0.1, when training on Train and testing on Test.



This data is linearly separable, as we can clearly see that a single decision surface can separate the classes, here marked red and blue. The model is well generalized as it almost perfectly separates the test data as well.

In addition the assignment wanted us to train on both Train and Test in order to plot the error development. This was clarified on the discussion forum, as shown on the bottom of the report. These data produced the following plots:



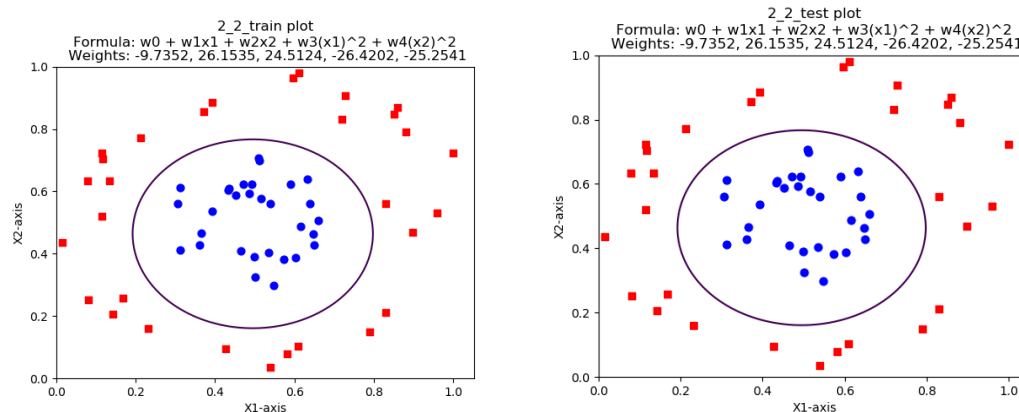
2.2.2

The dataset of 2.2 was shown to require a circular non-linear boundary, instead of a linear boundary as our model currently was.

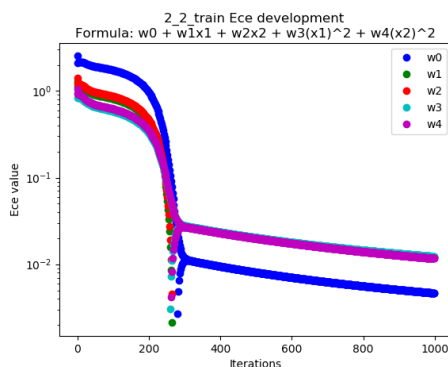
I therefor rewrote the data from the form x_1, x_2 to $1.0, x_1, x_2, (x_1)^2, (x_2)^2$.

This allowed for a model on the form $w_0 + w_1x_1 + w_2x_2 + w_3(x_1)^2 + w_4(x_2)^2 \geq 0$

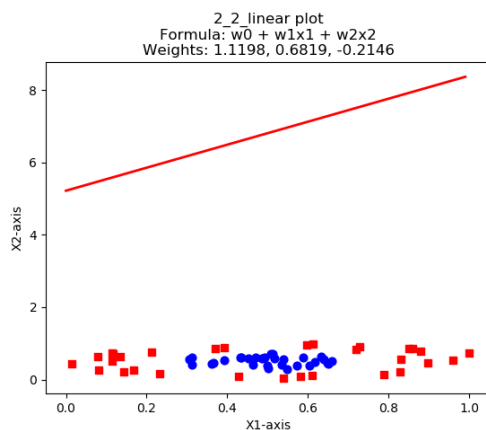
This resulted in the following plots when training on Train and Testing on Test:




Here we once again see a well generalized model, as it perfectly separates the Test data. The error development when training is shown below:



The plot of the linear model on this circular data is shown below, showing a clear offset. I did not tune the data to match it to the best of my ability, as it was clear it would not be a correct boundary



Assignment 1 – TDT4173
Spring 2018
Mikkel Svagård



Anonymous

1 day ago

2.2.1 Plot both training and test error

The assignment description of 2.2.1 says the following:


"Additionally, show a plot of the cross-entropy error for both the training and test set over 1000 iterations of training"




Are we here supposed to run both sets as if they were training sets, and thus plot the development of cross-entropy error in each?

Reply

Quote

Edit





Amar Deep Jaiswal 🌟

1 day ago

RE: 2.2.1 Plot both training and test error

Yes, you are right.

Regards,

Amar Jaiswal