

Linear Modeling - Least Square Approach

TF4063

Fadjar Fathurrahman

The material in this note is based on [1]. Most of the code in this note is written in Julia programming language [2, 3]. We only show only some portion of the code to illustrate the idea described in the note.

1 Linear model and its loss function

Given pair of data (x, t) where x are inputs dan t are targets, a linear model with parameter (w_0, w_1) can be written as:

$$t = f(x; w_0, w_1) = w_0 + w_1 x \quad (1)$$

We are now left with the task of choosing the best parameters (w_0, w_1) fir this model. We need to quantify how good the model is. One metric that we can use to quantify this is the squared difference (or error) between target and model prediction. For n -th data we can write

$$\mathcal{L}_n \equiv (t_n - f(x_n; w_0, w_1))^2 \quad (2)$$

By averaging contributions from all data:

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^N \mathcal{L}_n = \frac{1}{N} \sum_{n=1}^N (t_n - f(x_n; w_0, w_1))^2 \quad (3)$$

We will call this quantity as **loss function** and we want to this quantity to be as small as possible. Finding model parameters by minimizing loss functions such as in Eq. (3) is known as least square approach to linear regression.

2 Minimizing loss function

We can find the parameters (w_0, w_1) by using minimization procedures:

$$\arg \min_{w_0, w_1} \frac{1}{N} \sum_{n=1}^N \mathcal{L}_n \quad (4)$$

For our particular case of Eq. (3), we can found this analytically, i.e. calculating the first derivatives of \mathcal{L} with respect to w_0 and w_1 , equating them to zero, and solve the resulting equations for w_0 and w_1 . For more general cases, we can use various numerical optimization procedures such as gradient descent methods.

We begin by writing our loss function as:

$$\begin{aligned}\mathcal{L} &= \frac{1}{N} \sum_{n=1}^N (t_n - (w_0 + w_1 x_n))^2 \\ &= \frac{1}{N} \sum_{n=1}^N (w_1^2 x_n^2 + 2w_1 x_n (w_0 - t_n) + w_0^2 - 2w_0 t_n + t_n^2)\end{aligned}$$

Now we find the first derivatives of \mathcal{L} with respect to w_0 , w_1 and equating them to zero.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_1} &= 2w_1 \frac{1}{N} \left(\sum_{n=1}^N x_n^2 \right) + \frac{2}{N} \left(\sum_{n=1}^N x_n (w_0 - t_n) \right) = 0 \\ \frac{\partial \mathcal{L}}{\partial w_0} &= 2w_0 + 2w_1 \frac{1}{N} \left(\sum_{n=1}^N x_n \right) - \frac{2}{N} \left(\sum_{n=1}^N t_n \right) = 0\end{aligned}$$

We obtain

$$\begin{aligned}w_1 &= \frac{\overline{xt} - \bar{x}\bar{t}}{\overline{x^2} - \bar{x}^2} \\ w_0 &= \bar{t} - w_1 \bar{x}\end{aligned}\tag{5}$$

where symbols with overline denotes their average value, for examples

$$\begin{aligned}\bar{x} &= \frac{1}{N} \sum_{n=1}^N x_n \\ \bar{t} &= \frac{1}{N} \sum_{n=1}^N t_n\end{aligned}$$

Example

Now we want to implement least square approach to linear regression based on Eq. 5. For this purpose, we need a simple data to work on. We choose to work with olympic100m data which describes winning time of men's 100m sprint Olympic Games. You can choose to work with other data or synthetic data.

Year	Seconds
1896	12.00
1900	11.00
1904	11.00
...	...
2008	9.69

```
# Load data
data = readtble("../DATA/olympic100m.txt", ',')
x = data[:,1]
t = data[:,2]
xbar = mean(x)
tbar = mean(t)
w_1 = (mean(x.*t) - xbar*tbar)/(mean(x.^2) - xbar^2)
w_0 = tbar - w_1*xbar
```

We have the following result for the weights (parameters of the model)

```
w_0 = 36.41645590250286
w_1 = -0.01333088571096
```

We can visualize the result by plotting the data and the a line described linear equation Eq. (1). The result is shown in Fig. 1.

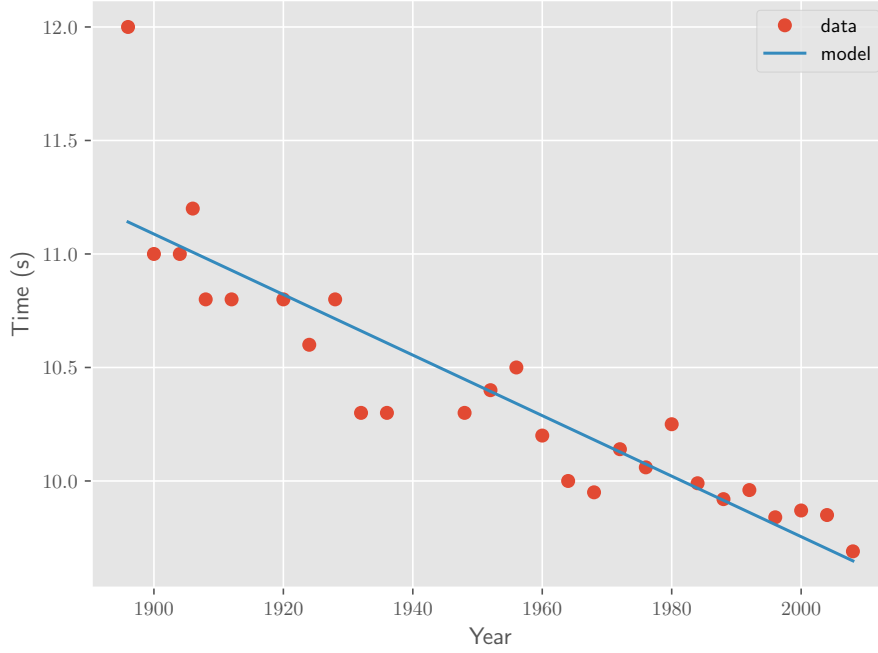


Figure 1: Result of linear regression for olympic100m.

3 Using matrix and vector notation

We will rewrite our previous problem in matrix and vector notation. This will give us more flexibility and enable us to generalize to more complex situations. We start by defining inputs and model parameters as vectors.

$$\mathbf{x}_n = \begin{bmatrix} 1 \\ x_n \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

Using this definition, we can write our previous linear model in Eq. (1) as:

$$f(x_n; w_0, w_1) = \mathbf{w}^T \mathbf{x}_n \quad (6)$$

The expression for loss function, Eq. (3), becomes

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^N \left(t_n - \mathbf{w}^T \mathbf{x}_n \right)^2 \quad (7)$$

We now arrange several input vector into a matrix:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}$$

As with inputs, we now define target vectors as

$$\mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix} \quad (8)$$

With this definition we can write the loss function as

$$\mathcal{L} = \frac{1}{N} (\mathbf{t} - \mathbf{X}\mathbf{w})^\top (\mathbf{t} - \mathbf{X}\mathbf{w}) \quad (9)$$

To find the best value of \mathbf{w} we can follow similar procedure that we have used in the previous part. We need to find the solution of $\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0$

$$\mathcal{L} = \frac{1}{N} (\mathbf{t}^\top \mathbf{t} + (\mathbf{X}\mathbf{w})^\top \mathbf{X}\mathbf{w} - \mathbf{t}\mathbf{X}\mathbf{w} - (\mathbf{X}\mathbf{w})^\top \mathbf{t}) \quad (10)$$

$$= \frac{1}{N} (\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} - 2\mathbf{w}^\top \mathbf{X}^\top \mathbf{t} + \mathbf{t}^\top \mathbf{t}) \quad (11)$$

Equating these to zeros we have

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{2}{N} (\mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{t}) = 0 \quad (12)$$

So we have

$$\mathbf{X}^\top \mathbf{X}\mathbf{w} = \mathbf{X}^\top \mathbf{t} \quad (13)$$

or

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t} \quad (14)$$

The following snippet describes how Eq. (14) is implemented.

```
data = readlm("../DATA/olympic100m.txt", ',')
x = data[:,1]
t = data[:,2]
Ndata = size(data,1)
# Build X matrix
X = zeros(Ndata,2)
for i in 1:Ndata
    X[i,1] = 1.0
    X[i,2] = x[i]
end
# Calculate w
w = inv(X' * X) * X' * t
println("w = ", w)
```

Result:

```
w = [36.416455902505334, -0.013330885710962845]
```

4 Generalization to more complex models

We can use more "complex" models than (6). For example the quadratic equation:

$$f(x; w_0, w_1, w_2) = w_0 + w_1 x + w_2 x^2 \quad (15)$$

Note that the model is still linear in parameter, so this model is also a linear model. Using matrix and vector notation, we can fit the data to this equation simply by adding one column to the matrix X :

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 \end{bmatrix} \quad (16)$$

This scheme also applies to higher order polynomials. The following snippet shows how to build the matrix X , find the parameters w and also how to make prediction given model parameter and input.

```
function do_fit(x, t, Norder)
    @assert Norder >= 1
    Ndata = size(x,1)
    # Build X matrix
    X = zeros(Ndata,Norder+1)
    for i in 1:Ndata
        X[i,1] = 1.0
        for n in 1:Norder
            X[i,n+1] = x[i]^n
        end
    end
    # Calculate w
    w = inv(X'*X) * X' * t
    return w
end

function do_predict(w, x)
    Ndata = size(x,1)
    Norder = size(w,1) - 1
    t = ones(Ndata)*w[1]
    for n in 1:Norder
        t[:] = t[:] + w[n+1]*x.^n
    end
    return t
end
```

In the following figure we show the result of fitting olympic100m data to 3rd order polynomial.

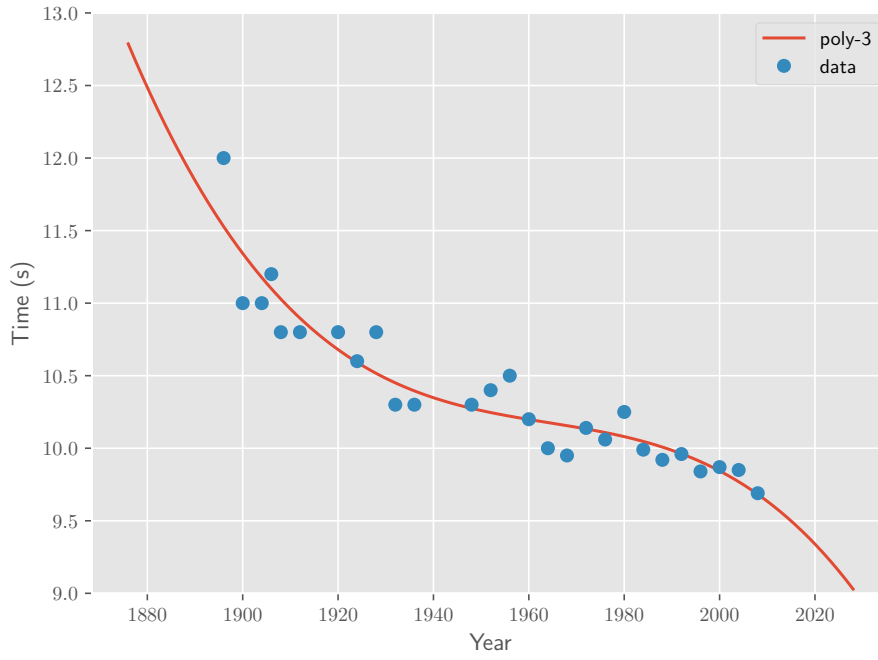


Figure 2: 3rd order polynomial fit to olympic100m data.

5 Model selection

Evaluating the value of loss according to Eq. (7) on the observation data for different polynomial order we have the following result.

Order:	1	Loss:	0.05031
Order:	2	Loss:	0.03796
Order:	3	Loss:	0.02961
Order:	4	Loss:	0.02706
Order:	5	Loss:	0.02350
Order:	6	Loss:	0.02202
Order:	7	Loss:	0.01970
Order:	8	Loss:	0.01698

We note that the loss is decreasing as we increase the polynomial order. Using this result, we might decide to choose 8-th order polynomial as the best model. However, this is not a good idea. The result of 8-th order fit is shown in Figure 3.

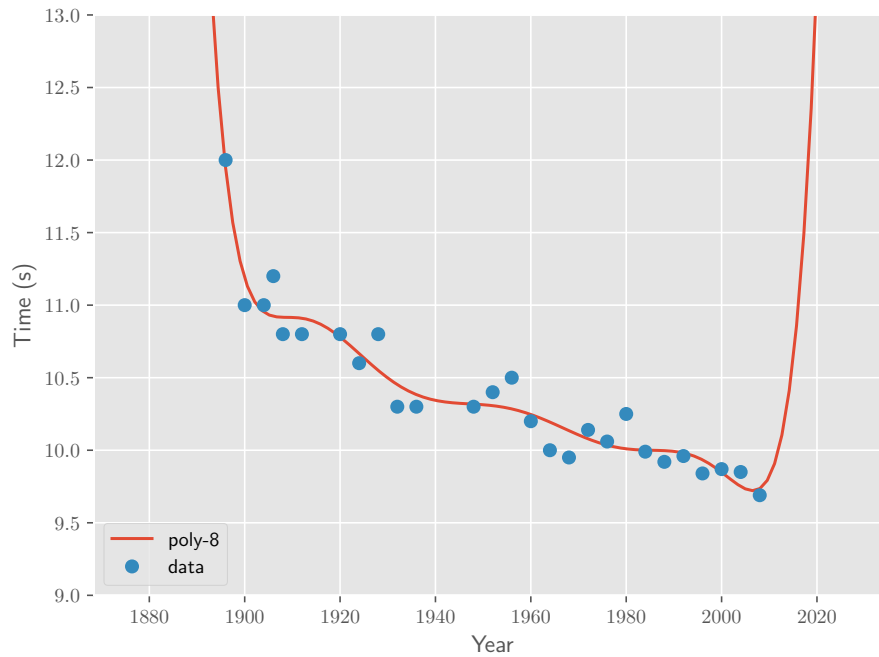


Figure 3: 8-th order polynomial fit to olympic100m data.

It can be seen that the model is not behaved very well if we try to predict t for other years other than the ones in our original data.

One solution to mitigate this problem is to split the data that we have into two subsets: the *training* and *validation* data. The model parameters w is calculated (or trained) using training data while the loss is calculated using validation data.

As an example, let's say that we want to use the data for year larger than 1979 as validation data and otherwise as training data. The following snippet shows how to do this.

```
# Load data
data = readlm("../DATA/olympic100m.txt", ',')
x = data[:,1]
t = data[:,2]
Ndata = size(data,1)
# Data indices for validation and training data
idx_val = x .> 1979
idx_train = x .<= 1979
# Scale x, to prevent numerical problems with large numbers
x_min = minimum(x)
x = x .- x_min
x = x/4
# Validation data
x_val = x[idx_val]
t_val = t[idx_val]
# Training data
x_train = x[idx_train]
t_train = t[idx_train]
# Train using training data
w = do_fit(x_train, t_train, Norder)
# Predict for validation data
t_pred = do_predict(w, x_val)
```

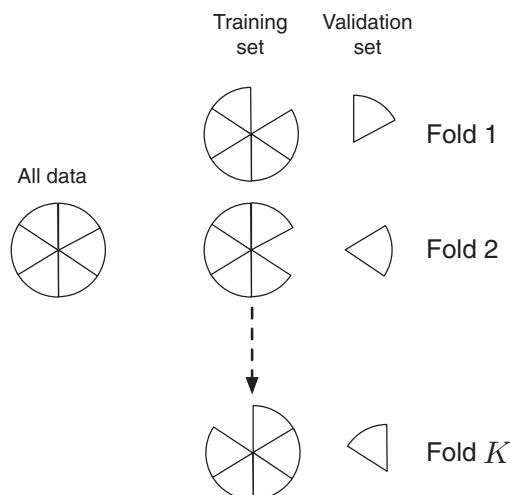
```
# Evaluate loss for validation data
NdataVal = size(t_val,1)
loss = sum( (t_val - t_pred).^2 )/NdataVal
@printf("Order: %3d Loss: %10.5f\n", Norder, loss)
```

The result is shown below.

```
Order:  1 Loss:    0.10130
Order:  2 Loss:    0.16763
Order:  3 Loss:    1.06188
Order:  4 Loss:    4.45706
Order:  5 Loss:    5.51420
Order:  6 Loss: 1533.43686
Order:  7 Loss:   61.91361
Order:  8 Loss: 6023.27708
```

From this result, we can say that first order polynomial is the best model.

However, note that the above result is very sensitive on the choice of validation data. This is particularly problematic if our data is small. A technique called *cross-validation* can be used to mitigate this problem. *K*-fold cross-validation splits the data into *K* equally (or as close to equal as possible) sized blocks.



Each block takes its turn as a validation set for a training set comprised of the other $K-1$ blocks. Averaging over the resulting k loss values gives us our final loss value.

An extreme case of *K*-fold cross-validation is where $K = N$, the number of observations in our dataset: each data observation is held out in turn and used to test a model trained on the other. This particular form of cross-validation is also known as Leave-One-Out Cross-Validation (LOOCV).

Task 1

Implement LOOCV on olympic100m dataset. Which polynomial gives the minimum loss?

One drawback of illustrating model selection on a real dataset is that we don't know what the "true" model is and therefore don't know if our selection techniques are working. We can overcome this by generating a synthetic dataset.

Task 2

Create a synthetic dataset from, for example generated from 3rd order polynomial plus random noise term. Fit

polynomial from first to, say 7th order, to this data. Plot order vs LOOCV loss. Which polynomial order give the minimum loss?

6 Regularization

We can define a measure of complexity of our linear model by

$$\sum_i w_i^2 \text{ or } \mathbf{w}^T \mathbf{w} \quad (17)$$

As opposed to minimizing loss function (7), we can minimize a regularized loss function by adding penalty for overcomplexity:

$$\mathcal{L}' = \mathcal{L} + \lambda \mathbf{w}^T \mathbf{w}$$

where the parameter λ control the trade off between model accuracy and model complexity. To find the best parameter, we can proceed by using similar procedure as before. The regularized loss function is written as

$$\mathcal{L}' = \frac{1}{N} \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - \frac{2}{N} \mathbf{w}^T \mathbf{X}^T \mathbf{t} + \frac{1}{N} \mathbf{t}^T \mathbf{t} + \lambda \mathbf{w}^T \mathbf{w} \quad (18)$$

First derivative of the loss function with respect to model parameter \mathbf{w} is

$$\frac{\partial \mathcal{L}'}{\partial \mathbf{w}} = \frac{2}{N} \mathbf{X}^T \mathbf{X} \mathbf{w} - \frac{2}{N} \mathbf{X}^T \mathbf{t} + 2\lambda \mathbf{w}$$

Equating this to zero:

$$(\mathbf{X}^T \mathbf{X} + N\lambda \mathbf{I}) \mathbf{w} = \mathbf{X}^T \mathbf{t}$$

we obtain

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + N\lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{t} \quad (19)$$

We can implement this by simple modification to our code.

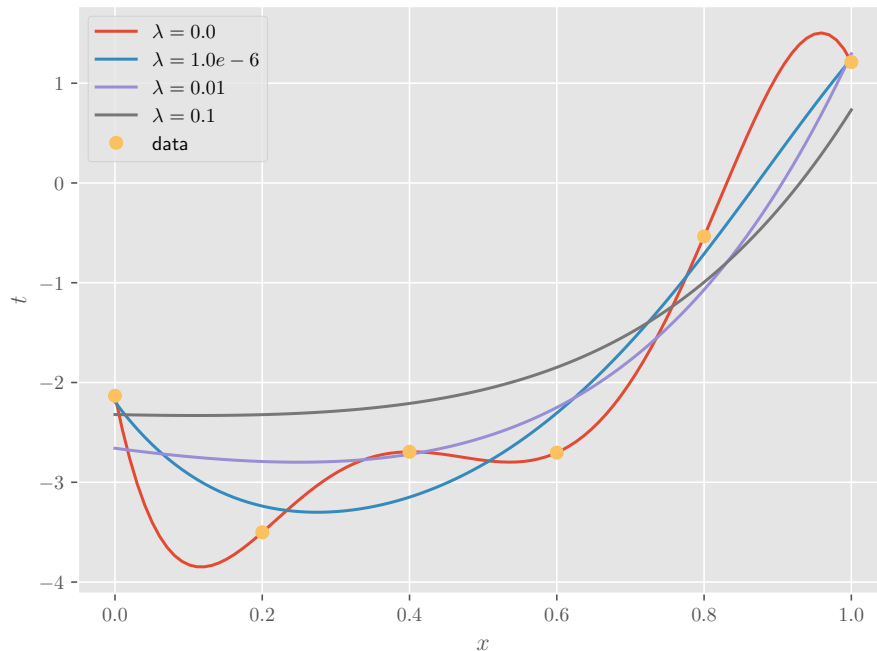
```
function do_fit(x, t, Norder; λ=0.0)
    @assert Norder >= 1
    Ndata = size(x,1)
    # Build X matrix
    X = zeros(Ndata,Norder+1)
    for i in 1:Ndata
        X[i,1] = 1.0
        for n in 1:Norder
            X[i,n+1] = x[i]^n
        end
    end
    # Calculate w
    w = inv( X'*X + Ndata*diag(λ*ones(Ndata)) ) * X' * t
    return w
end
```

As an example we will implement this for synthetic data generated from linear function plus noise term.

```
x_min = 0.0; x_max = 1.0
x = range(x_min, x_max, step=0.2) # input data
y = 2*x .- 3 # linear function
A_noise = 1.0
```

```
t = y + A_noise*randn(size(x,1)) # add noise
```

Fitting 5-th order polynomial to this data and using various values for regularization parameter λ , we obtain the following result.



Best value for λ can be obtained by using validation data as we have done before.

Task 3

Apply regularized linear regression to olympic100m data or synthetic data or other simple dataset of your choice. Determine the best model parameters (along with regularization parameter λ) and describe the procedure that you have used to determine them.

7 Linear regression in Scikit Learn Python package

Linear regression in Scikit Learn [4] can be done using the `sklearn.linear_model.LinearRegression` module. Several regularization methods are also applied such as:

- ridge regression
- lasso regression

References

- [1] Simon Rogers and Mark Girolami. *A First Course in Machine Learning*. CRC Press, Boca Raton, 2nd edition, 2017.
- [2] J. Bezanson, A. Edelman, S. Karpinski, and V. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.
- [3] Julia programming language. <https://julialang.org/>.

- [4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.