FYS-STK Exercises week 35

Håkon Sjøvik Olsen, Even Sletten Garvang og Ellen-Beate Tysvær

August 2024

```
import numpy as np
import matplotlib.pyplot as plt
```

Exercise 1

Part 1

Show that:

$$\frac{\partial(\mathbf{a}^T\mathbf{x})}{\partial\mathbf{x}} = \mathbf{x}^T \tag{1}$$

Answer: For a vector $\mathbf{a}$ and $\mathbf{x} \in \mathbb{R}^n$ we have:

$$\mathbf{a}^T\mathbf{x} = \sum_{i=0}^{n-1} a_i x_i \tag{2}$$

and we know that the partial derivative can be written as

$$\frac{\partial \mathbf{a}^T\mathbf{x}}{\partial x_i} = a_i \tag{3}$$

This means that we can write the vector as

$$\begin{aligned} &\frac{\partial(\mathbf{a}^T\mathbf{x})}{\partial\mathbf{x}} \\ &\quad= \begin{bmatrix} \frac{\partial \mathbf{a}^T\mathbf{x}}{\partial x_0} & \frac{\partial \mathbf{a}^T\mathbf{x}}{\partial x_1} & \dots & \frac{\partial \mathbf{a}^T\mathbf{x}}{\partial x_{n-1}} \end{bmatrix} \\ &\quad= \begin{bmatrix} a_o & a_1 & \dots & a_{n-1} \end{bmatrix} \\ &\quad= \mathbf{a}^T \end{aligned} \tag{4}$$

Part 2

Show that:

$$\frac{\partial(\boldsymbol{a}^T \boldsymbol{A}\boldsymbol{a})}{\partial \boldsymbol{a}} = \boldsymbol{a}^T(\boldsymbol{A} + \boldsymbol{A}^T)$$

Answer: For a vector $\mathbf{a} \in \mathbb{R}^n$ and a matrix $\mathbf{A} \in \mathbb{R}^{nxn}$ which consists of a selection of column vectors $\mathbf{c_0}, \mathbf{c_1} \ldots \mathbf{c_{n-1}}$ and row vectors $\mathbf{r_0}, \mathbf{r_1} \ldots \mathbf{r_{n-1}}$ we can write the expression of the scalar function y as:

$$\begin{aligned} y &= \boldsymbol{a}^T \boldsymbol{A}\boldsymbol{a} \\ &= \begin{bmatrix} \mathbf{a^T} * \mathbf{c_0} & \mathbf{a^T} * \mathbf{c_1} & \ldots & \mathbf{a^T} * \mathbf{c_{n-1}} \end{bmatrix} * \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} \\ &= a_0 * \mathbf{a^T} * \mathbf{c_0} + a_1 * \mathbf{a^T} * \mathbf{c_1} + \ldots + a_{n-1} * \mathbf{a^T} * \mathbf{c_{n-1}} \end{aligned} \tag{5}$$

If we replace each column vector $\mathbf{c_j}$ from $\mathbf{A}$ with the matrix element $A_{i,j}$ the above expression can be written as

$$\sum_{j=0}^{n-1} a_j \sum_{i=0}^{n-1} a_i * A_{i,j} \tag{6}$$

When we differentiate this expression with respect to $\boldsymbol{a}$ we get:

$$\frac{\partial(\boldsymbol{a}^T \boldsymbol{A}\boldsymbol{a})}{\partial \boldsymbol{a}} = \frac{\partial}{\partial \boldsymbol{a}} \sum_{j=0}^{n-1} a_j \sum_{i=0}^{n-1} a_i * A_{i,j}. \tag{7}$$

Now for $j = k$ we get

$$a_k \sum_{i=0}^{n-1} a_i * A_{i,k}, \tag{8}$$

and similarly for $i = k$:

$$a_k \sum_{j=0}^{n-1} a_j * A_{k,j}. \tag{9}$$

Thus, since the summations are independent, the differential component of the k-th element will give us

$$\frac{\partial(\boldsymbol{a}^T\boldsymbol{A}\boldsymbol{a})}{\partial a_k} = \sum_{i=0}^{n-1} a_i * A_{i,k} + \sum_{j=0}^{n-1} a_j * A_{k,j} = \sum_{i=j=0}^{n-1} a_i(A_{i,k} + A_{k,j}) \tag{10}$$

for all $k = 0, 1, ..., n-1$. Then the collection of k derivatives will be in the form of a k-dimensional row vector. We notice that $A_{i,k}$ and $A_{k,j}$ are entries of transposed matrices, as the indexes are interchanged. From here we easily recognize that we have arrived at the summation expression for the k entries of a row vector $\mathbf{a}^T$ multiplied by the sum of a matrix $\mathbf{A}$ and its transpose $\mathbf{A}^T$, that is: $\boldsymbol{a}^T(\boldsymbol{A} + \boldsymbol{A}^T)$. $\square$

For completeness, we may write it in vector form as

$$\frac{\partial(\boldsymbol{a}^T\boldsymbol{A}\boldsymbol{a})}{\partial \boldsymbol{a}} = \sum_{i=j=0}^{n-1} (\boldsymbol{a}^T * \boldsymbol{c_j} + \boldsymbol{a}^T * \boldsymbol{r_i^T}), \tag{11}$$

where a transposed row $\boldsymbol{r_i^T}$ equals a column $\boldsymbol{c_j^T}$ of the transposed matrix $\boldsymbol{A^T}$, and thus we´re left with the expression

$$\begin{aligned} \sum_{j=0}^{n-1} (\boldsymbol{a}^T * \boldsymbol{c_j} + \boldsymbol{a}^T * \boldsymbol{c_j^T}) &= \boldsymbol{a^T} * \sum_{j=0}^{n-1} (\boldsymbol{c_j} + \boldsymbol{c_j^T}) \\ &= \boldsymbol{a}^T(\boldsymbol{A} + \boldsymbol{A}^T). \end{aligned} \tag{12}$$

Part 3

Show that:

$$\frac{\partial \left(\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s}\right)^T \left(\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s}\right)}{\partial \boldsymbol{s}} = -2 \left(\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s}\right)^T \boldsymbol{A} \tag{13}$$

In terms of a summation, the i-th element of $\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s}$ may be expressed as

$$x_i - \sum_{j=0}^{n-1} A_{i,j} s_j, \tag{14}$$

and so the product with the transpose will be the sum of the square of the i elements, that is:

$$(\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s})^T (\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s}) = \sum_{i=0}^{n-1} \left(x_i - \sum_{j=0}^{n-1} A_{i,j} s_j \right)^2 \equiv y \tag{15}$$

By differentiating with respect to s_k (where k is a given value of j, that is, $j = 0, 1, 2, ..., k, ..., n-1$) using the chain rule, we then get

$$\frac{\partial y}{\partial s_k} = -2 \sum_{i=0}^{n-1} \left(x_i - \sum_{j=0}^{n-1} A_{i,j} s_j \right) A_{i,k}. \tag{16}$$

For all j derivatives, we will then end up with the row vector

$$\frac{\partial y}{\partial \boldsymbol{s}} = -2 (\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s})^T \boldsymbol{A}. \tag{17}$$

Now we proceed by doing the second derivative on the k-th element, and get

$$\frac{\partial^2 y}{\partial s_k^2} = 2 \sum_{i=0}^{n-1} A_{i,k} A_{i,k} = 2 \sum_{i=0}^{n-1} A_{i,k}^2 \tag{18}$$

As this was for the k-th element, in total we will have for all j entries:

$$\frac{\partial^2 y}{\partial \boldsymbol{s}^2} = 2 \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} A_{i,j}^2 \tag{19}$$

The square of the i, j-th element constitutes the entries of the product of the matrix $\boldsymbol{A}$ and its transpose, that is, $\boldsymbol{A}^T \boldsymbol{A}$. Thus we have found

$$\frac{\partial^2 (\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s})^T (\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s})}{\partial \boldsymbol{s}^2} = 2\boldsymbol{A}^T \boldsymbol{A} \tag{20}$$

which is proportional to the Hessian matrix $\boldsymbol{H} = \boldsymbol{A}^T \boldsymbol{A}$. □

Exercise 2

First we create our random x, and a dependent y with noise drawn from a normal distribution with standard deviation 0.1 (code modified from the exercise text).

```
np.random.seed(10)
x = np.random.rand(100,1)
noise = 0.1 * np.random.randn(100,1)
y = 2.0+5*x*x + noise
```

Then we create our design matrix, with intercept, and x up to the second power.

```
X = np.zeros((100,3))
X[:,0] = 1
X[:,1] = x.T
X[:,2] = (x**2).T
```

Then we optimize the cost function, here mean squared error (MSE), to get our betas with the analytical expression

$$\boldsymbol{\beta} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

```
# optimize MSE
xtx_inv = np.linalg.inv(X.T.dot(X))

Best = xtx_inv.dot(X.T).dot(y)
print(f"Beta vector: {Best.T}")
```

```
Beta vector: [[2.0146372  0.02474757 4.93762474]]
```

We then get our predicted $\tilde{\boldsymbol{y}}$ using

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta},$$

```
ytilde = X.dot(Best)
```

and calculate MSE

$$MSE(\boldsymbol{y},\tilde{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2.$$

```
MSE = sum((y - ytilde)**2) / len(y)
print(f"MSE: {round(MSE.item(), 5)}")
```

```
MSE: 0.00971
```

The MSE is 0.00971, which is very close to the value 0.1 that we used for our simulation.

Finally, we plot the data and regression

```
sorted_index = np.argsort(x, axis=None)
#print(sorted_index)
xsort = x[sorted_index]
ysort = y[sorted_index]
ytsort = ytilde[sorted_index]

plt.plot(xsort, ysort, ".")
plt.plot(xsort, ytsort, "-")
plt.show()
```

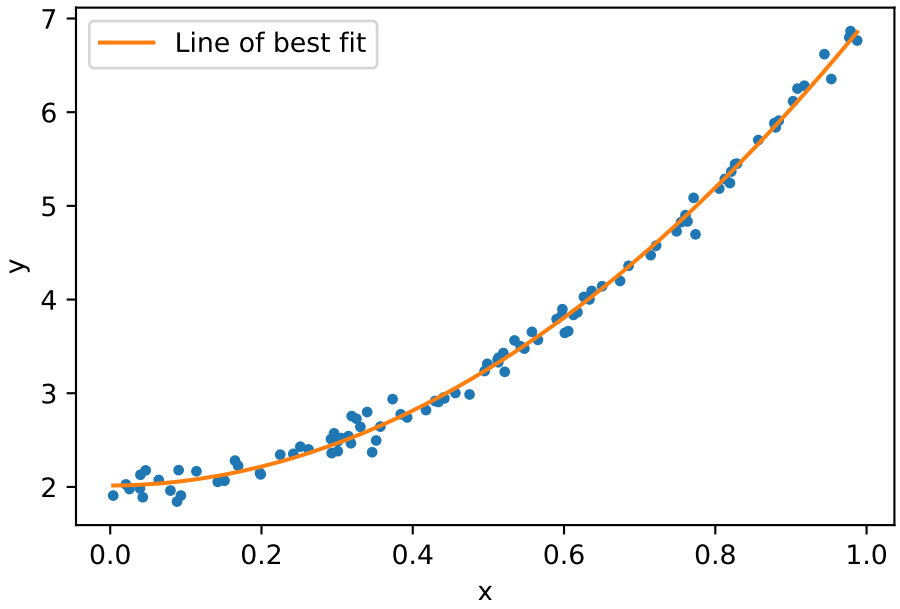

Figure 1: Linear regression for a model fit with polynomials up to the second degree.

Exercise 3

a)

For this exercise we also import `train_test_split` from `sklearn`.

```
from sklearn.model_selection import train_test_split
```

Create data (code from the exercise text)

```
np.random.seed()
n = 100
# Make data set.
x = np.linspace(-3, 3, n).reshape(-1, 1)
y = np.exp(-x**2) + 1.5 * np.exp(-(x-2)**2)+ np.random.normal(0, 0.1, x.shape)
```

Create a function to make a design matrix with an nth order polynomial

```
def design_poly_n(x, n):
    X = np.zeros((len(x), n))
    X[:,0] = 1
    for i in range(1, n):
        X[:,i] = (x**i).T

    return X
```

Then create design matrix up to a 5th order polynomial, and split in training and test data

```
X = design_poly_n(x, 5)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

b)

Calculate optimal beta and MSE

```
xtx_inv = np.linalg.inv(X_train.T.dot(X_train))
Best = xtx_inv.dot(X_train.T).dot(y_train)
print("Beta:", Best.T)
```

```
Beta: [[ 0.82356159  0.45530857 -0.00424467 -0.03723883 -0.00640265]]
```

```
ytilde_train = X_train.dot(Best)
MSE = sum((y_train - ytilde_train)**2) / len(y_train)
print("MSE", MSE)
```

```
MSE [0.02819904]
```

Compare to MSE when applying betas from training data to test data

```
ytilde_test = X_test.dot(Best)
MSE = sum((y_test - ytilde_test)**2) / len(y_test)
print("MSE", MSE)
```

```
MSE [0.03948381]
```

MSE is a bit higher for the test data.

c)

```
np.random.seed(3489)
n = 100
# Make data set.
x = np.linspace(-3, 3, n).reshape(-1, 1)
y = np.exp(-x**2) + 1.5 * np.exp(-(x-2)**2)+ np.random.normal(0, 0.1, x.shape)

maxpoly = 15
poly = np.arange(maxpoly) + 1
MSE_train_out = np.zeros(maxpoly)
MSE_test_out = np.zeros(maxpoly)

for i in poly:
    X = design_poly_n(x, i)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    xtx_inv = np.linalg.inv(X_train.T.dot(X_train))
    Best = xtx_inv.dot(X_train.T).dot(y_train)

    ytilde_train = X_train.dot(Best)
    MSE_train_out[i-1] = sum((y_train - ytilde_train)**2) / len(y_train)

    ytilde_test = X_test.dot(Best)
```

```
    MSE_test_out[i-1] = sum((y_test - ytilde_test)**2) / len(y_test)
```

Then plot the MSEs to compare test and train set. Passing on to R for plotting.

```
# R code
# passing onto R for plotting
library(reticulate)
library(tidyverse)

polydeg <- py$poly
MSEtest <- py$MSE_test_out
MSEtrain <- py$MSE_train_out

df <- data.frame(
    poly = polydeg,
    test = MSEtest,
    train = MSEtrain
) %>%
    pivot_longer(c(test,train), names_to = "data_type", values_to = "MSE")

ggplot(df, aes(poly, MSE, col = data_type)) +
    geom_line() +
    theme_bw() +
    labs(
        x = "Polynomial degree",
        color = "Data type"
    )
```

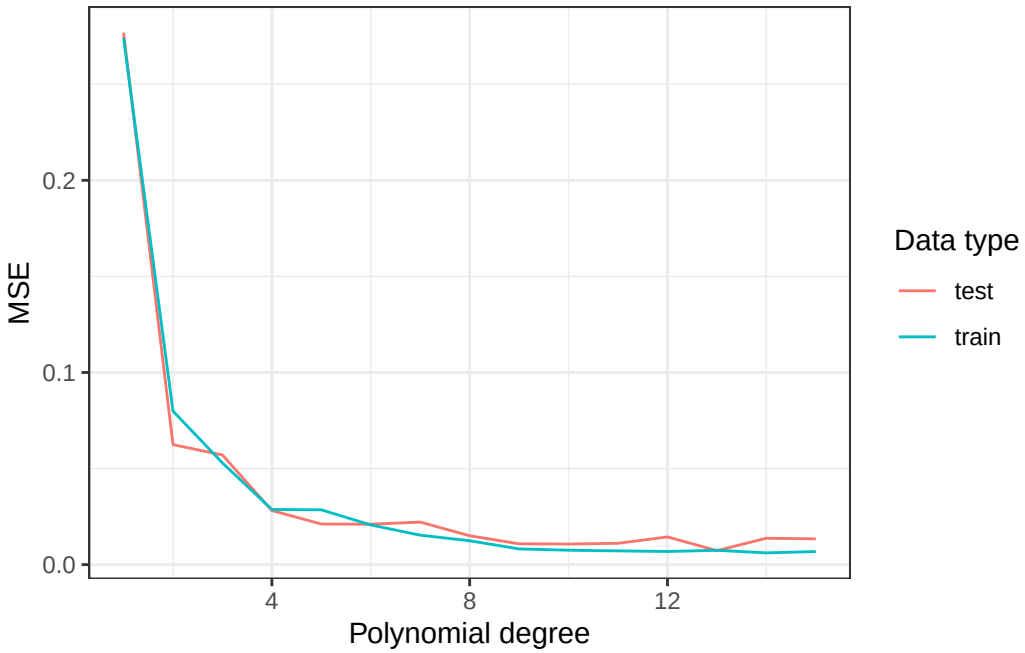

Figure 2: MSE as function of highest polynomial degree included in the model for training (80%) and test (20%) data. n = 100

For 15 degree polynomials the the test and training are still pretty similar. MSE for the training data is decreasing for each polynomial degree, while for the test data it fluctuates a bit more, the minimum seemingly at 13 degrees.

Try a higher-degree polynomial

```
np.random.seed(3489)
n = 100
# Make data set.
x = np.linspace(-3, 3, n).reshape(-1, 1)
y = np.exp(-x**2) + 1.5 * np.exp(-(x-2)**2)+ np.random.normal(0, 0.1, x.shape)

maxpoly = 20
poly = np.arange(maxpoly) + 1
MSE_train_out = np.zeros(maxpoly)
MSE_test_out = np.zeros(maxpoly)

for i in poly:
    X = design_poly_n(x, i)
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    xtx_inv = np.linalg.inv(X_train.T.dot(X_train))
    Best = xtx_inv.dot(X_train.T).dot(y_train)

    ytilde_train = X_train.dot(Best)
    MSE_train_out[i-1] = sum((y_train - ytilde_train)**2) / len(y_train)

    ytilde_test = X_test.dot(Best)
    MSE_test_out[i-1] = sum((y_test - ytilde_test)**2) / len(y_test)
```

```
# R code
# passing onto R for plotting

polydeg <- py$poly
MSEtest <- py$MSE_test_out
MSEtrain <- py$MSE_train_out

df <- data.frame(
    poly = polydeg,
    test = MSEtest,
    train = MSEtrain
) %>%
    pivot_longer(c(test,train), names_to = "data_type", values_to = "MSE")

ggplot(df, aes(poly, MSE, col = data_type)) +
    geom_line() +
    theme_bw() +
    labs(
        x = "Polynomial degree",
        color = "Data type"
    )
```

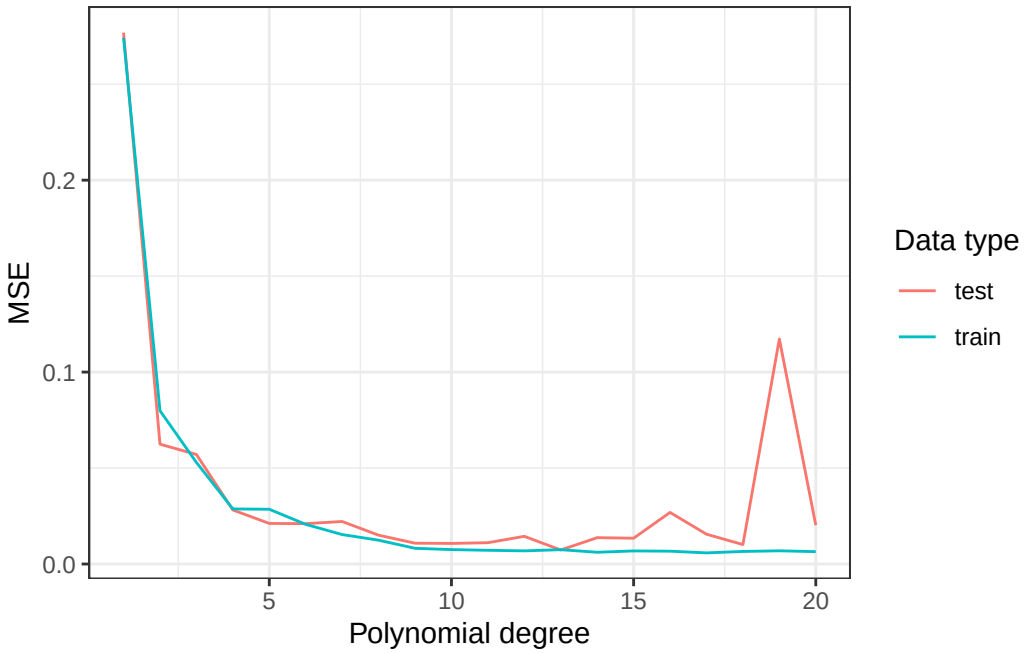

Figure 3: MSE as function of highest polynomial degree included in the model for training (80%) and test (20%) data. Same data as the previous figure, but for a higher polynomial degree. n = 100

The MSE starts to really deviate when polynomial degree goes above around 15. The best trade-off to avoid overfitting is probably around 10-15 degrees.

References

[1] Morten Hjort-Jensen. *Applied Data Analysis and Machine Learning*. Jupyter Notebook, 2021.