m1-l1-p1

January 27, 2024

1 M1-L1 Problem 1 (10 points)

You are given 14 temperature measurements from 14 thermocouples in a factory. A model has produced 14 temperature predictions, one for each thermocouple. You must compute the the error vector and MSE between the predicted and measured temperatures via a few methods.

Run the next cell to load the data; then proceed through the notebook.

- `y_data` is y, a 14x1 array of temperature measurements (in deg C)
- `y_pred` is $\hat{y}$, a 14x1 array of temperature predictions

[68]:
```
import numpy as np
np.set_printoptions(precision=4)

y_data = np.array([[20,21,30,30,21,25,38,37,30,22,22,38,20,35]],dtype=np.
 ↪double).T
y_pred = np.array([[21,21,31,30,20,28,36,32,31,20,21,39,21,34]],dtype=np.
 ↪double).T

print("y_data = \n", y_data)
print("y_pred = \n", y_pred)
```

```
y_data =
 [[20.]
 [21.]
 [30.]
 [30.]
 [21.]
 [25.]
 [38.]
 [37.]
 [30.]
 [22.]
 [22.]
 [38.]
 [20.]
 [35.]]
y_pred =
 [[21.]
```

```
 [21.]
 [31.]
 [30.]
 [20.]
 [28.]
 [36.]
 [32.]
 [31.]
 [20.]
 [21.]
 [39.]
 [21.]
 [34.]]
```

1.1 Error vector

First, compute the error vector $y_{err} = y - \hat{y}$. Call the result `y_err`. It should be 14x1.

You may do this with a loop, or – better yet – by simply subtracting the two arrays.

[69]:
```python
# YOUR CODE GOES HERE
# Compute y_err
y_err = y_data - y_pred

print("Size of y_err:", np.shape(y_err))
```

```
Size of y_err: (14, 1)
```

1.2 Mean squared error (MSE)

Now compute the MSE,

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{N}\sum_{i=1}^{N} y_{err}^2$$

1.2.1 MSE with Loop

First, compute this quantity by using a for loop to loop through y_err, performing the necessary operations to compute MSE.
Call the result `MSE_loop`.
Your result should be ≈ 3.5714.

[70]:
```python
# YOUR CODE GOES HERE
# Compute MSE_loop
MSE_loop = 0

for i in range(len(y_err)):
    MSE_loop += y_err[i]**2
```

```python
MSE_loop = MSE_loop / len(y_err)

print("MSE (loop) = ", MSE_loop)
```

```
MSE (loop) =  [3.5714]
```

1.2.2 MSE by matrix multiplication

Another way to compute the MSE is by recognizing that the sum \$ *{i=1}^N y*{err}^2 \$ equals the matrix product \$ y_{err}' y_{err} \$.
Therefore:

$$MSE = \frac{1}{N} y'_{err} \cdot y_{err}$$

Compute the MSE this way. Call it `MSE_mm`, and make sure the result is the same. This is a much more efficient way of computing the MSE in Python.

Note that you can compute the transpose of a 2D array `A` with `A.T`, and you can multiply matrices `A` and `B` with `A @ B`.

[71]:
```python
# YOUR CODE GOES HERE
# Compute MSE_mm
MSE_mm = y_err.T @ y_err / len(y_err)

print("MSE (matrix multiplication) = ", MSE_mm)
```

```
MSE (matrix multiplication) =  [[3.5714]]
```

1.2.3 MSE by numpy mean

Now you will compute the MSE once more, but using numpy operations. Use `np.mean()` to take an average. Compute the square of `y_err` with either `np.square()` or `y_err * y_err`.

Call your `MSE_np`, and make sure the result is the same. This is also much more efficient than a Python for loop.

[72]:
```python
# YOUR CODE GOES HERE
# Compute MSE_np
MSE_np = np.mean(np.square(y_err))

print("MSE (Numpy) = ", MSE_np)
```

```
MSE (Numpy) =  3.5714285714285716
```