Workshop 11: Introduction to scikit-learn

FIE463: Numerical Methods in Macroeconomics and Finance using Python

Richard Foltyn

NHH Norwegian School of Economics

March 27, 2025

See GitHub repository for notebooks and data:

https://github.com/richardfoltyn/FIE463-V25

Exercise 1: Polynomial under- and overfitting

Consider the following non-linear model,

$$y_i = f(x_i) + \epsilon_i\,, \qquad f(x) = \cos\left(\frac{3}{2}\pi x\right)$$

where y_i is a trigonometric function of x_i but is measured with an additive error ϵ_i. In this exercise, we are going to approximate y_i using polynomials in x_i of varying degrees:

1. Create a sample of size $N = 50$ where the x_i are randomly drawn from a uniform distribution on the interval $[0, 1]$ and $\epsilon_i \overset{\text{iid}}{\sim} N(0, 0.2^2)$. Initialize your RNG with a seed of 1234. Then generate y_i according to the equation given above.

   Create a scatter plot of the sample (x_i, y_i) and add a line depicting the true non-linear relationship (without measurement error).

   Hint: The cosine function and the constant π are implemented as `np.cos()` and `np.pi` in NumPy.

2. Use the `PolynomialFeatures` transformation and `LinearRegression` to approximate y as a polynomial in x. Fit this model using the polynomial degrees $d \in \{0, 1, 2, 3, 10, 15\}$. You should build a pipeline (e.g., using `make_pipeline()`) to create the polynomial and perform the fitting in one step.

   Create a figure with 6 panels, one for each polynomial degree. Each panel should show the sample scatter plot, the true function $y = f(x)$ and the polynomial approximation of a given degree.

   How does the quality of the approximation change as you increase d? Do higher-order polynomials always perform better?

   Hint: When creating polynomials with `PolynomialFeatures(..., include_bias=True)`, you need to fit the model *without* an additional intercept as the intercept is already included in the polynomial.

3. You want to find the optimal polynomial degree using cross-validation. For this purpose, program a function with the signature

```python
def compute_average_mse(d, x, y, n_splits):
    """
    Compute mean squared error averaged across splits in k-fold cross-validation.
    """
```

which takes as arguments the polynomial degree d, the sample observations (x, y) and the number of splits `n_splits`, and returns the mean squared error (MSE) for the test sample averaged across all splits.

Using the function you wrote, compute the average MSEs for polynomial degrees $d = 0, \dots, 15$ using 10 splits. Use these to create a plot of the average MSE on the y-axis against d on the x-axis. Which degree d results in the lowest average MSE?

Hint: You do not need to assign training/test samples manually. Use the `KFold` class and call its `split()` method to do the work for you!

Hint: To compute the MSE for each test sample, you can use `mean_squared_error()`.

4. Re-estimate the model using the optimal polynomial degree you just found and create a scatter plot with the original data, the true function $y = f(x)$ and the fitted polynomial.
5. You recall from the lecture that the steps in Part (3) can be implemented in an easier way using `cross_val_score()`. Re-implement the cross-validation using this function.

   Hint: Don't forget that you have to use the *negative* MSE as the relevant criterion, i.e., specify the argument `scoring='neg_mean_squared_error'` when calling `cross_val_score()`.