Linear Regression and Test-Driven Development

Applied Machine Learning in Engineering - Exercise 01

TU Berlin, Summer Term 2025

Prof. Dr.-Ing. Merten Stender – `merten.stender@tu-berlin.de`

This exercise will teach the `Python` implementation of basic linear regression applied to a small automotive engineering dataset. Furthermore, the test-driven software development paradigm is covered.

Linear Regression

Implement a function `lin_regressor()` in the file `my_lin_regressor.py` that solves a linear scalar regression problem and returns the model parameters θ_0 and θ_1.

(a) Use `numpy` to implement the normal form and solve for the model coefficients (use `np.linalg.solve()` for the solution of a linear system of equations).

(b) Make a new Python file, e.g. `exercise_01.py`, and import your linear regressor.

(c) Test your implementation by passing the vectors `x = np.arange(0, 100, 1)` and a synthetic data vector `y = 2 + x + np.random.randn(100)` to your function, receiving the coefficients, and plotting the results using a scatter plot, such as `plt.scatter(x, y)`. Are the coefficients calculated correctly?

(d) Plot the visual regression model evaluation plot, i.e. showing the actual values vs. the predicted ones in a scatter plot with a diagonal identity line.

Case study: Rolling resistance estimation

Estimate the effective rolling resistance factor of a car from measurements of vehicle speed v and engine power P_{engine}. The underlying equations for the wind's force F_{wind}, the rolling resistance force F_{roll}, and the resulting power P are given in the following

$$\begin{aligned} F_{\mathrm{wind}} &= c_{\mathrm{w}} A \frac{\rho_{\mathrm{air}} v_{\mathrm{rel}}^2}{2} \\ F_{\mathrm{roll}} &= c_{\mathrm{R}} M g \cos(\alpha) \\ P &= v \cdot F \end{aligned} \tag{1}$$

where the known parameters are $c_{\mathrm{w}} = 0.4$, face area $A = 1.5\,\mathrm{m}^2$, air density $\rho_{\mathrm{air}} = 1.2\,\mathrm{kg/m}^3$, gravity $g = 9.81\mathrm{m/s}^2$ and the vehicle's mass $M = 2400\,\mathrm{kg}$.

(a) Read the data file using `data = np.genfromtxt("driving_data.csv", delimiter=",")`, where the first column is the velocity (in m/s), and the second column carries the instantaneous engine power (in W).

(b) Derive the balance of powers that holds true for the given application case. Identify the known and unknown terms.

(c) Re-formulate the problem such that the rolling resistance can be read from a linear fit to the existing data

(d) Report your estimate of the rolling resistance value c_{R} and check if your result is plausible. How do you interpret the constant offset coefficient θ_0?

Test-Driven Development

Red phase

Implement the R^2 score for computing the distance between a ground truth vector $\mathbf{y} \in \mathbb{R}^N$ and a predicted vector $\hat{\mathbf{y}} \in \mathbb{R}^N$. Use test-driven development for the implementation, i.e. first write tests and then implement the actual distance metrics.

(a) Define empty Python function `r2_score(y_true, y_pred)` in a Python file `my_r2_score.py`. Use type-hints to ease readability of your code. Employ `numpy.ndarray` for representing true and predicted values.

(b) Create a unittesting script and customize the template code provided as a separate file in `unittest_template.py`.

(c) Define a case object `TestR2Score` and write methods `test_perfect_pred()`, `test_mean_pred()`, `test_input_dims()`, `test_data_type()` for each class using the methods `assertAlmostEqual()`, `assertEqual()`, `assertRaises()`. Refer to `https://docs.python.org/3/library/unittest.html` for more details. Think of reasonable test cases that you want to check.

(d) Ensure that all tests fail (Python however should not raise any syntax error!), e.g. by returning a negative dummy value.

Green phase

(a) Implement the actual code for the distance metric using trivial indexing and looping over entries of the arrays `y_true` and `y_pred`. Use only the operator `**2` for squaring a number and the Numpy methods `numpy.sum()` and `numpy.mean()`. Test that the numeric tests are passed.

(b) Implement a dimensionality check for the inputs `x` and `y`, and raise a `TypeError` if wrong types or wrong dimensions are supplied to the distance functions. Make use of `numpy.shape`, `numpy.ndim` and `type()` methods.

(c) Ensure that all tests are passed.

Refactoring phase

(a) Review your code and refactor. Discuss with your neighbor. Put in comments and docstrings.

(b) Check that all tests are still passed.

Evaluate

Compute the R^2 value of your fit. Validate your method using the scikit-learn library.