Supervised Learning: Regressions

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Machine Learning for Economics and Finance
Bachelor in Economics

Marcel Weschke

27.05.2025

Learning Goals

At the end of this lecture, you should be able to:

- Revisit the basics of linear regressions
- Understand the difference between in-sample and out-of sample prediction errors
- Understand what overfitting and flexibility of a method is
- Apply the concepts above in Python

Book Chapters: 2 and 3

Recall: Supervised Learning

Suppose you have a quantitative response Y and p different predictors $X = (X_1, X_2, \ldots, X_p)$.

In supervised learning, we try to establish a relation

$$Y = f(X) + \epsilon$$

f: unknown function that represents the systematic information that X provides about Y.

ϵ: random, independent error term with mean zero

Key task: find $\hat{f} \approx f$ that ‘fits the data well’

Recall: Supervised Learning

We differentiate between two types of problems:

- **Regression**: Y is quantitative (predict the stock return tomorrow)
- **Classification**: Y is a category (predict whether returns tomorrow are positive or negative)

Today: Introduction to regression problems.

Example: Linear Regression

Assume a linear relation between the outcomes Y and the predictors $X = (X_1, X_2, \ldots, X_p)$:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon$$

So the prediction $\hat{Y}$ is given by

$$\hat{Y} = \hat{f}(\hat{\beta}, X) = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2 + \ldots + \hat{\beta}_p X_p$$

where $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \ldots \hat{\beta}_p)$.

Question: How do we obtain the unknown coefficients $\hat{\beta}$?

Estimating the Regression Coefficients

Suppose we have data on the outcomes y_i, and predictors $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^T$, where $i = 1, 2, \ldots, n$.

Least squares approach: We choose $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \ldots \hat{\beta}_p)$ to minimize the sum of squared prediction errors:

$$\min_{\hat{\beta}} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\Rightarrow \min_{\hat{\beta}} \sum_{i=1}^{n} (y_i - \hat{f}(x_i, \hat{\beta}))^2$$

$$\Rightarrow \min_{\hat{\beta}} \sum_{i=1}^{n} (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \ldots + \hat{\beta}_p x_{ip}))^2$$

How to do the minimization? Standard software like Python does it for you

Example: Automobile Data

The ISLP dataset *Auto* contains data on cars like *horsepower*, *year*, *weight*, ...

The goal is to predict the fuel consumption of a car measured in miles per gallon (*mpg*).

Suppose we want to estimate the following univariate linear regression model:

$$mpg = \beta_0 + \beta_1 horsepower + \epsilon$$

```python
import statsmodels.formula.api as smf
from ISLP import load_data

# Load data
Auto = load_data('Auto')

# Fit univariate linear regression
fit_lm = smf.ols('mpg ~ horsepower', data = Auto).fit()
print(fit_lm.summary())
```

Example: Automobile Data (2)

Suppose we want to estimate the following multivariate linear regression model:

$$mpg = \beta_0 + \beta_1 horsepower + \beta_2 year + \beta_3 cylinders + \epsilon$$

```python
# Fit multivariate linear regression
fit_lm = smf.ols('mpg ~ horsepower + year + cylinders', data =
↪  Auto).fit()
print(fit_lm.summary())
```

Example: Automobile Data (3)

Suppose we want to estimate a multivariate linear regression model using **all** variables in the *Auto* dataset:

$$mpg = \beta_0 + \beta_1 horsepower + \beta_2 year + \beta_3 cylinders + \ldots + \epsilon$$

```python
# Concatenate all predictor variable names except 'mpg'
predictors = [col for col in Auto.columns if col != 'mpg']
formula = 'mpg ~ ' + ' + '.join(predictors)

# Fit linear regression using all predictors from Auto data
fit_lm = smf.ols(formula=formula, data=Auto).fit()
print(fit_lm.summary())
```

Task 1:

1. Run the Python-file `01_Auto_data_1.ipynb` that fits a linear regression model

$$\mathit{mpg} = \beta_0 + \beta_1 \mathit{horsepower} + \epsilon$$

2. Compute the R^2 and analyze the sign and significance of the estimated coefficients
3. Step by step add more variables to the regression. Analyze how the R^2 as well as the significance of the estimated coefficients changes when adding more predictors. Interpret your findings.

Note: The dataset contains a variable *name* which is a character variable. We will later learn how to work with qualitative variables. For now please simply leave the variable out of your analysis.

Model Accuracy

Throughout the course we will learn about a wide range of machine learning techniques

We need ways to measure the accuracy of a method, that is, how

well it predicts the outcomes Y.

This allow us to compare different methods and decide which

method is best for a specific data problem (one of the key learning goals of this course)

Measuring Model Accuracy

There are several ways to determine the accuracy of a machine learning technique

You have just computed one of these measures: the R^2 statistic

Another standard measure for the fit of a model is the mean squared error (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

The MSE is small when the predicted responses are close to the true responses.

The Mean Squared Error

You have used the MSE already when you fitted linear regressions That is, you minimized the mean squared error in the dataset to

determine the optimal coefficients β:

$$\min_{\hat{\beta}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{f}(x_i, \hat{\beta}))^2$$

Hence, you know that you have found the coefficients, that provide the smallest MSE in the **training data** which we used for fitting the coefficients

This is what we call the **in-sample** fit of our machine learning method

Computing the MSE in for the Auto Data

```python
# Compute predicted values ŷ for training data
y_head = fit_lm.predict(Auto)

# Function to compute the mean squared error (MSE)
# Takes realized values y and corresponding predictions ŷ
# as inputs and returns MSE as output
def MSE(y, y_head):
 return ((y - y_head) ** 2).mean()

# Compute the mean squared error
MSE_Auto = MSE(Auto['mpg'], y_head)
print(f"Mean Squared Error: {MSE_Auto:.3f}")
```

In-Sample vs Out-of-Sample Errors

But we do not really care about the accuracy of our method for the training data, but rather about how well the method performs when we apply it to previously unseen **test data**.

This is what we call the **out-of-sample** fit of our machine learning method

Examples:

- Train model for stock return predictions using the last year of data (training data). Use the model to predict returns for the next year (unseen test data).
- Kaggle competitions

Test Data and Model Evaluation

Suppose we have m test observations with outcomes $\tilde{y}_i$ and features $\tilde{x}_i = (\tilde{x}_{i1}, \tilde{x}_{i1}, \dots, \tilde{x}_{ip})^T$, where $i = 1, \dots m$.
The **test MSE** is then defined by

$$\frac{1}{m}\sum_{i=1}^{m}(\tilde{y}_i - \hat{f}(\tilde{x}_i))^2$$

We are fitting a method based on the **training MSE**, but we are eventually interested in a low **test MSE**
Problem: A low training MSE does not guarantee a low test MSE

Flexibility of a Method

The training and test MSE depend on the flexibility of the machine learning method

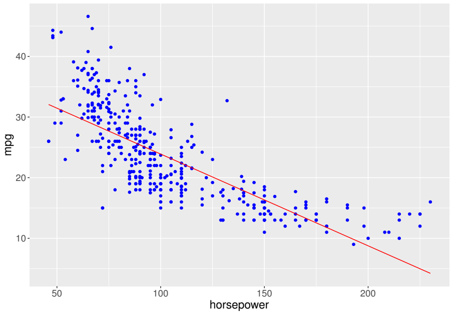
What does flexibility mean?

Consider again the example where we try to predict *mpg* with the variable *horsepower* using a linear regression:

$$\textit{mpg} = \beta_0 + \beta_1 \textit{horsepower} + \epsilon$$

Flexibility: Linear Model

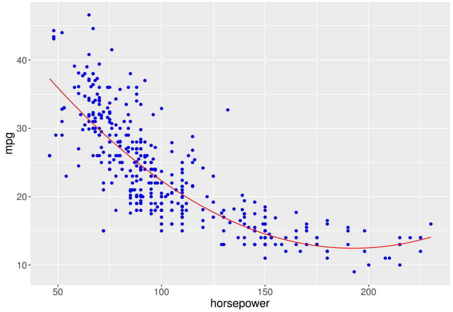
$$mpg = \beta_0 + \beta_1 \, horsepower + \epsilon$$



This model has only 2 parameters, β_0 and β_1, and is rather inflexible

Flexibility: Quadratic Model

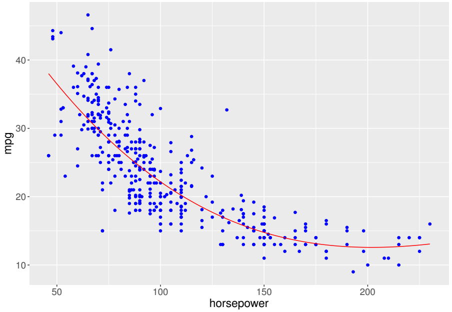
$$mpg = \beta_0 + \beta_1 horsepower + \beta_2 horsepower^2 + \epsilon$$



This model has 3 parameters, β_0, β_1 and β_2, and is a bit more flexible

Flexibility: Degree-3 Polynomial

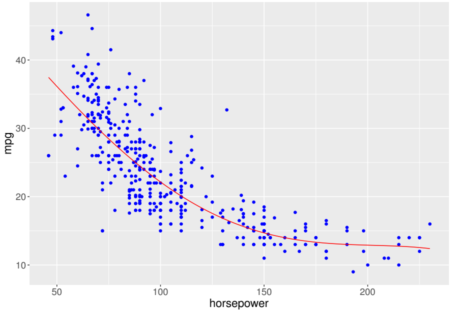
$$mpg = \beta_0 + \beta_1 hp + \beta_2 hp^2 + \beta_3 hp^3 + \epsilon$$



$\Rightarrow$: 4 parameters and more flexibility

Flexibility: Degree-4 Polynomial

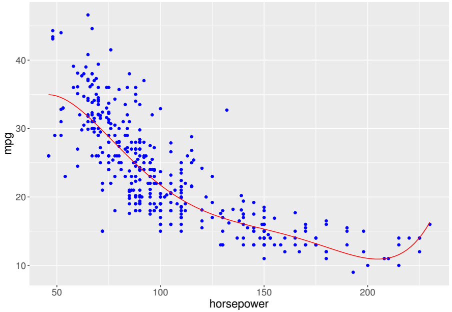
$$mpg = \beta_0 + \beta_1 hp + \beta_2 hp^2 + \beta_3 hp^3 + \beta_4 hp^4 + \epsilon$$



$\Rightarrow$: 5 parameters and even more flexibility

Flexibility: Degree-5 Polynomial

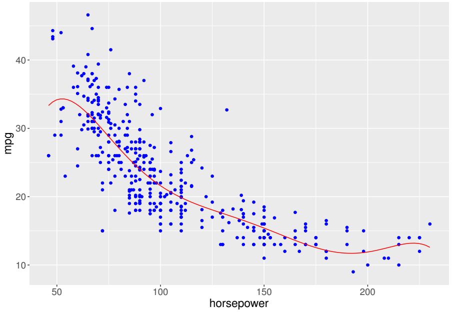
$$mpg = \beta_0 + \beta_1 hp + \beta_2 hp^2 + \ldots + \beta_5 hp^5 + \epsilon$$



$\Rightarrow$: 6 parameters and even more flexibility

Flexibility: Degree-6 Polynomial

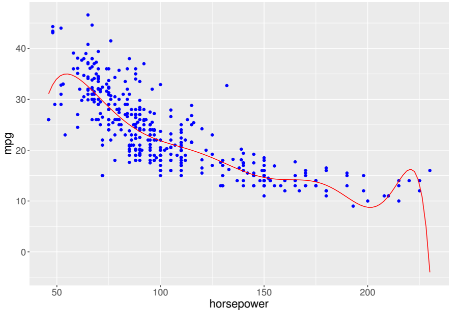
$$mpg = \beta_0 + \beta_1 hp + \beta_2 hp^2 + \ldots + \beta_5 hp^6 + \epsilon$$



$\Rightarrow$: 7 parameters and even more flexibility

Flexibility: Degree-10 Polynomial

$$mpg = \beta_0 + \beta_1 hp + \beta_2 hp^2 + \ldots + \beta_5 hp^{10} + \epsilon$$

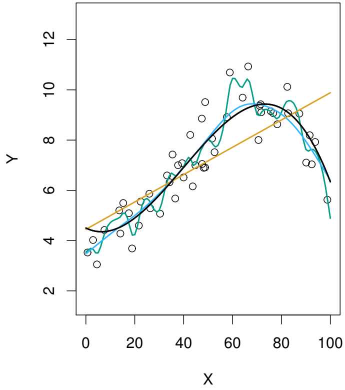


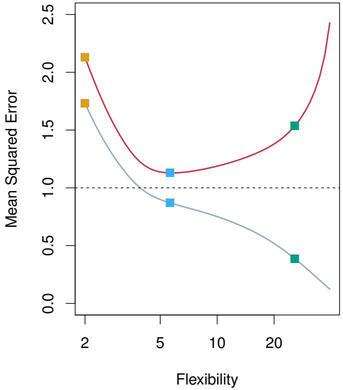
$\Rightarrow$: 11 parameters and even more flexibility

Flexibility and Prediction Errors

- As the flexibility of a method increases the training MSE will decrease
- However: Lower **training MSE**, does **NOT** guarantee lower **test MSE**
- Flexible methods: risk of 'overfitting' (finding patterns in the data that are not actually there)

Ex. 1: Training and Test MSE for Non-Linear Model

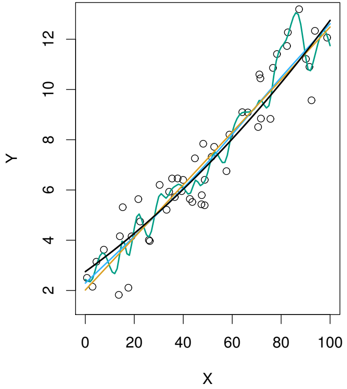


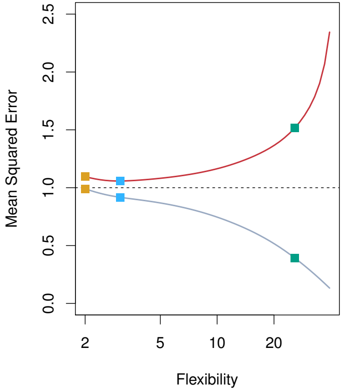


Black dots/line: data and true f
yellow: linear regression
blue: more flexible method
green: even more flexible method

grey: training MSE
red: test MSE

Ex. 2: Training and Test MSE for ≈ Linear Model





Black dots/line: data and true f
yellow: linear regression
blue: more flexible method
green: even more flexible method

grey: training MSE
red: test MSE

Flexibility and Overfitting

- As the flexibility of a method increases, training MSE will decrease
- **Lower training MSE**, does **NOT** guarantee **lower test MSE**
- Flexible methods: risk of 'overfitting' (finding patterns in the data that are not actually there)
- **Overfitting** training data yields **large test MSE**

Key challenge in machine learning: Find a method with the right degree of flexibility to **minimize test MSE**

Task 2:

- The Python-file `01_Auto_data_2.ipynb` splits the *Auto* data into a training (*train_data*) and a test (*test_data*) set.
- You are asked to run regressions with different flexibility and compute the training and test MSE.
- We start with the following univariate linear regression:

$$\mathit{mpg} = \beta_0 + \beta_1 \mathit{horsepower} + \epsilon$$

- The code to compute the regression coefficients using the training data as well as to compute the training MSE is already provided.

Task 2:

1. Compute the test MSE for the univariate linear model and note the training and test MSE.

   Hint: First copy line 28 and generate predictions $\hat{y}$ for the test data. Then copy line 37 and change it correspondingly to compute the test MSE.

2. Compute and note the training and test MSE for a quadratic model:

$$mpg = \beta_0 + \beta_1 horsepower + \beta_2 horsepower^2 + \epsilon$$

```python
# Fit a quadratic model
fit_lm = smf.ols(formula='mpg ~ horsepower + I(horsepower**2)', data =
↪  Auto).fit()

# Note that the function I() is needed here as the operator ** has a
↪  special meaning in formulas
```

Task 2:

3. Redo Step 2 but add the term *horsepower*^{3} to the regression. Then add *horsepower*^{4} and so on. For each model note the training and test MSE. How do the two MSE change with the flexibility of the method?