Introduction to Machine Learning and Data Science

4. Supervised learning framework

Pierre Vandenhove
Course material by Souhaib Ben Taieb

Université de Mons

Table of contents

Table of contents

Input and output variables

The **input variables**[1] are typically denoted using the symbol X. If we observe p different variables, we write $X = (X_1, X_2, \ldots, X_p)$. The inputs belong to an *input space* $\mathcal{X}$.

- Examples: $\mathcal{X} \subseteq \mathbb{R}^p$ or $\mathcal{X} = \{0, 1\}^p$.

The **output variable**[2] is typically denoted using the symbol Y. The output belongs to an *output space* $\mathcal{Y}$.

- Regression: $\mathcal{Y} \subseteq \mathbb{R}$.
- Classification (with K categories): $\mathcal{Y} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K\}$.
    - Binary classification ($K = 2$): $\mathcal{Y} = \{-1, 1\}$ or $\mathcal{Y} = \{0, 1\}$.

[1] also called *predictors*, *independent variables*, *features*, *variables*, or just *inputs*.
[2] also called the *response* or *dependent variable*.

Unknown joint distribution

We assume $(X, Y) \sim p_{X,Y}$ where $p_{X,Y}$ is a fixed *unknown* distribution which can be factorized as

$$p_{X,Y}(x, y) = p_X(x) p_{Y|X}(y \mid x),$$

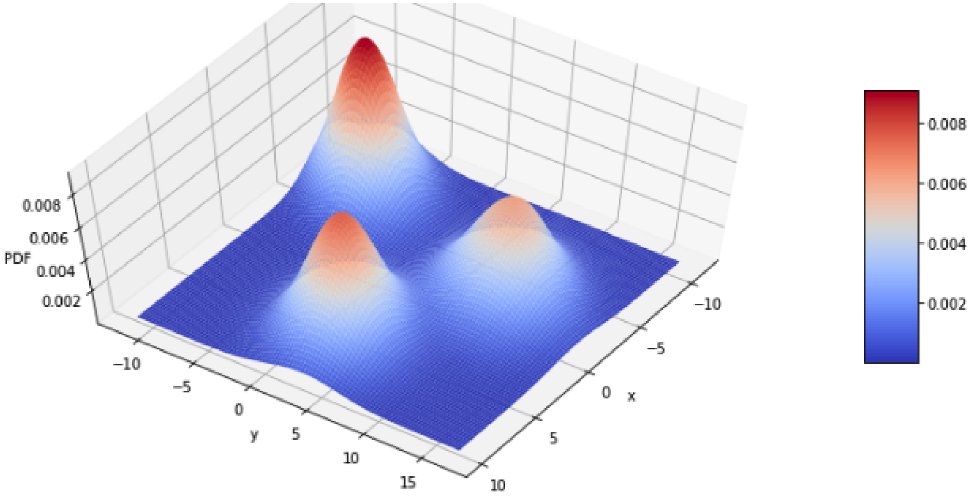
where

- the **marginal** distribution p_X models uncertainty in the sampling of the inputs.
- the **conditional** distribution $p_{Y|X}$ describes a stochastic (non-deterministic) relation between inputs and output.

Equivalently, we have

$$X \sim p_X \text{ and } Y \mid X = x \sim p_{Y|X}(\cdot \mid x).$$

Joint distribution



Optimal predictions

Define a **loss function** $L\colon \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$. Given a prediction $\hat{y} \in \mathcal{Y}$ and the true (observed) value $y \in \mathcal{Y}$, $L(y, \hat{y})$ measures how far $\hat{y}$ is from y.

Examples include the *squared error loss* $L(y, \hat{y}) = (y - \hat{y})^2$, the *absolute error loss* $L(y, \hat{y}) = |y - \hat{y}|$, and the *zero-one loss* $L(y, \hat{y}) = \mathbb{1}\{y \neq \hat{y}\}$, where $\mathbb{1}\{\cdot\}$ is the indicator function.

The **optimal prediction function** which minimizes the **expected error** (or **expected risk**) is given by

$$f = \operatorname*{argmin}_{h:\mathcal{X}\to\mathcal{Y}} \mathbb{E}_{x,y}[L(y, h(x))].$$

In practice, we cannot compute f since we **do not know** $p_{X,Y}$.

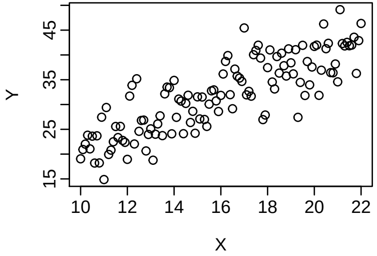
The dataset

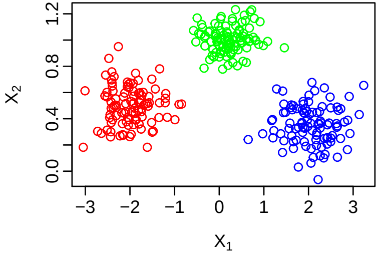
The **dataset**, also called *training set*, is a set of n input-output pairs

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} = \{(x_i, y_i)\}_{i=1}^{n},$$

where the data points (x_i, y_i) are assumed to be i.i.d. realizations of $p_{X,Y}$.

Each pair, also called an *example* or a *data point*, belongs to the *data space* $\mathcal{X} \times \mathcal{Y}$.





- Left figure: $\mathcal{X} \subseteq \mathbb{R}$ $(p = 1)$ and $\mathcal{Y} \subseteq \mathbb{R}$.
- Right figure: $\mathcal{X} \subseteq \mathbb{R}^2$ $(p = 2)$ and $\mathcal{Y} = \{R, G, B\}$.

The supervised learning problem

Let $\mathcal{H}$ be a **hypothesis set**, i.e., a set of prediction functions (hypotheses) under consideration. An example is the linear hypothesis set

$$\mathcal{H} = \{h(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \mid \beta_0, \beta_1, \ldots, \beta_p \in \mathbb{R}\}.$$

Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, the goal of supervised learning is to learn a **prediction function** $h\colon \mathcal{X} \to \mathcal{Y}$ to map new/unseen examples with minimal expected **prediction error**.

We can compute the **in-sample error** or **training error**[3]

$$E_{\text{in}}(h) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i)),$$

and solve the following optimization problem:

$$g_{\mathcal{D}} = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\; E_{\text{in}}(h).$$

[3]Also called the **empirical risk**.

The supervised learning problem

Ideally, we would like to select the hypothesis $h \in \mathcal{H}$ which minimizes the **out-of-sample error**

$$E_{\text{out}}(h) = \mathbb{E}_{x,y}[L(y, h(x))], \tag{1}$$

and compute

$$g^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\ E_{\text{out}}(h).$$

In summary, there are three different prediction functions:

$$f = \underset{h\colon \mathcal{X} \to \mathcal{Y}}{\operatorname{argmin}}\ E_{\text{out}}(h),$$

$$g^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\ E_{\text{out}}(h),$$

and

$$g_{\mathcal{D}} = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\ E_{\text{in}}(h).$$

Summary

The **dataset** $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ is composed of n input-output pairs (x_i, y_i), which are i.i.d. realizations from an **unknown joint distribution** $p_{X,Y}$ where $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$.

The **loss function** $L\colon \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$ allows us to measure the error we incur in predicting $\hat{y}$ in place of y.

The **hypothesis set** $\mathcal{H}$ is a set of prediction functions under consideration. Each hypothesis $h \in \mathcal{H}$ has an **in-sample error** $E_{\text{in}}(h)$, computed on $\mathcal{D}$, and an **out-of-sample error** $E_{\text{out}}(h)$ which depends on $p_{X,Y}$.

Given $\mathcal{H}$ and using $\mathcal{D}$, the **learning algorithm** $\mathcal{A}$ picks the best hypothesis g from $\mathcal{H}$ according to the loss function L.

Together, the hypothesis set and the learning algorithm are referred to as the **learning model**.

Table of contents

Linear models

Let us consider a regression problem where $x \in \mathbb{R}^p$.

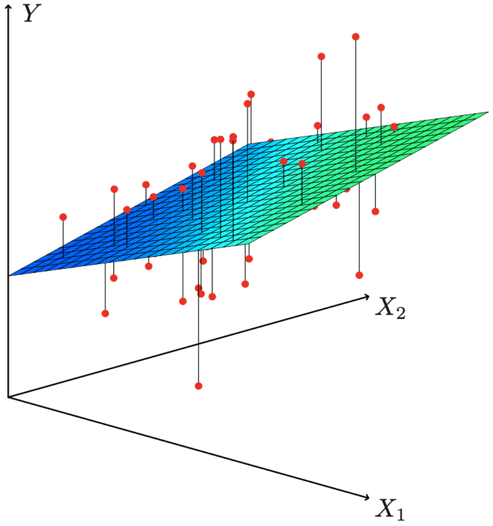
- The **hypothesis set** for linear models is given by

$$\mathcal{H} = \{h(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \mid \beta_0, \beta_1, \ldots, \beta_p \in \mathbb{R}\}.$$

- One **learning algorithm** is the *(ordinary) least squares* method.

Linear models

Example with $p = 2$.



K-Nearest Neighbors (KNN) model

- **Input**: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, number of neighbors $K \leq n$, and new input x_*.
- **Classification** with M categories ($y \in \{C_1, \ldots, C_M\}$):
  - Find the K nearest points to x_* in $\mathcal{D}$; this set of points is denoted $\mathcal{N}_*$.
  - The predicted probability for each class is given by

$$\hat{p}(C_m \mid x = x_*) \approx \frac{1}{K} \sum_{i \in \mathcal{N}_*} \mathbb{1}\{y_i = C_m\} \quad (m = 1, 2, \ldots, M).$$

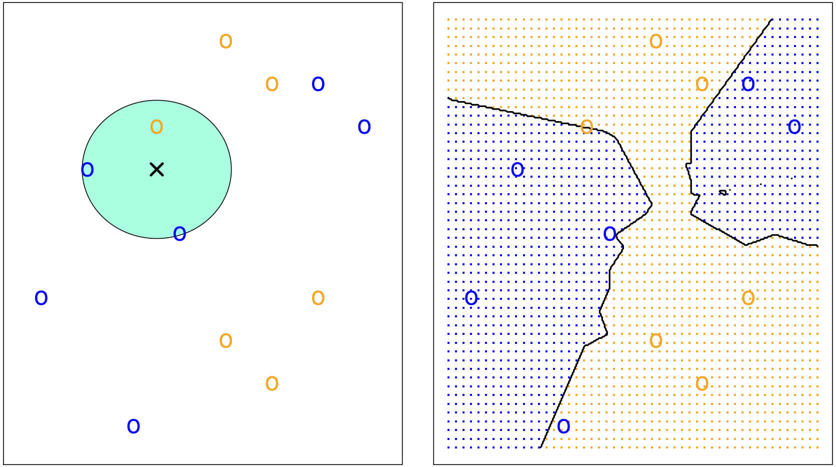
  - The predicted class is then

$$h(x_*) = C_{m^*} \text{ where } m^* = \operatorname{argmax}_m \hat{p}(C_m \mid x = x_*).$$

- **Regression**
  - Find the K nearest points to x_* in $\mathcal{D}$; this set of points is denoted $\mathcal{N}_*$.
  - Choose

$$h(x_*) = \frac{1}{K} \sum_{i \in \mathcal{N}_*} y_i.$$

K-Nearest Neighbors (KNN)



Picture: Classification problem, $K = 3$. Right: *decision boundary*.
K-Nearest Neighbors (KNN) is one of the simplest machine learning model for both classification and regression.

Parametric and non-parametric models

Two classes of models: **parametric** and **non-parametric**.

- In a **parametric model**, every hypothesis is uniquely defined by a **fixed number of parameters**. Example: $p + 1$ for linear regression with $\mathcal{X} = \mathbb{R}^p$.
- In a **non-parametric model**, we can not describe a hypothesis with a fixed number of parameters. Usually the number of "parameters" **grows with the size of the dataset**. Example: KNN, in which every prediction uses the whole dataset.
- Both parametric and non-parametric models have **hyper-parameters** (structural parameters), while parametric models also have **parameters**
  - For KNN, K is a **hyper-parameter**.
  - For linear models, p is a **hyper-parameter**, and the coefficients β_j are **parameters**.
- We also make a distinction between **linear** and **non-linear** models.

Table of contents

How does E_{out} relates to E_{in}?

Observe that

$$E_{\text{out}}(h) = E_{\text{in}}(h) + [E_{\text{out}}(h) - E_{\text{in}}(h)].$$

To obtain a small $E_{\text{out}}(h)$, we thus want

1. small $E_{\text{in}}(h)$, and

2. small $[E_{\text{out}}(h) - E_{\text{in}}(h)]$.

Selecting the best hypothesis by minimizing $E_{\text{in}}(h)$ only can be misleading.

For example, a (stupid) model that remembers all training data and returns the right value for a training example but a random value otherwise will have a $E_{\text{in}}(h) = 0$ but a very high $E_{\text{out}}(h)$!

This is not a reasonable choice for a model. But how to select the "best" hypothesis set? What we care about is the *accuracy of the predictions that we obtain when we apply our method to previously unseen data*.

Training and test errors in regression

Let us consider multiple hypothesis sets $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_M$, and define

$$g_m = \underset{h \in \mathcal{H}_m}{\operatorname{argmin}}\ E_{\text{in}}(h) = \underset{h \in \mathcal{H}_m}{\operatorname{argmin}}\ \frac{1}{n} \sum_{i=1}^{n} (y_i - h(x_i))^2, \tag{2}$$

for $m = 1, 2, \ldots, M$. Here, we consider the *mean squared error* (MSE) loss.

Let $\mathcal{D}_{\text{test}} = \{(x'_i, y'_i)\}_{i=1}^{n'}$ be **another** sample (**independent** of $\mathcal{D}$) where $(x'_i, y'_i) \overset{\text{i.i.d.}}{\sim} p_{X,Y}$. We define the **testing/test error** of a hypothesis h as

$$E_{\text{test}}(h) = \frac{1}{n'} \sum_{i=1}^{n'} (y'_i - h(x'_i))^2.$$

Note that the test set **is not involved** in the learning process, and is **only used to evaluate the hypothesis** h.

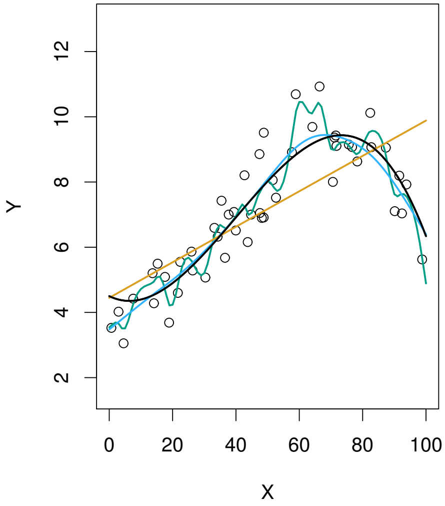
Training and test errors in regression (1/6)

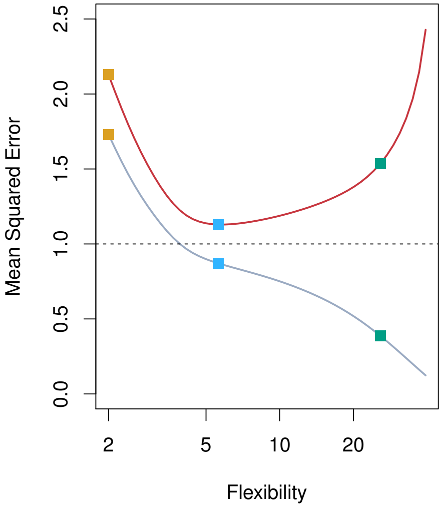
Let us compare $E_{\text{in}}(g_m)$ and $E_{\text{test}}(g_m)$ for $m = 1, 2, \ldots, M$ on an example (with $M = 3$).

We start from a function f which we know. We generate datasets $\mathcal{D}$ and $\mathcal{D}_{\text{test}}$ from f by sampling points and adding a small amount of noise (with mean 0). We then find the hypothesis g_m that minimizes the training error for each hypothesis set $\mathcal{H}_m$.

Obviously, in general, we do not know the "true" function f; this is just for illustration purposes.

Training and test errors in regression (2/6)

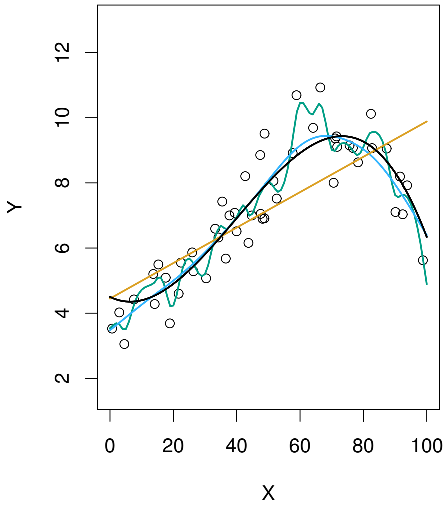


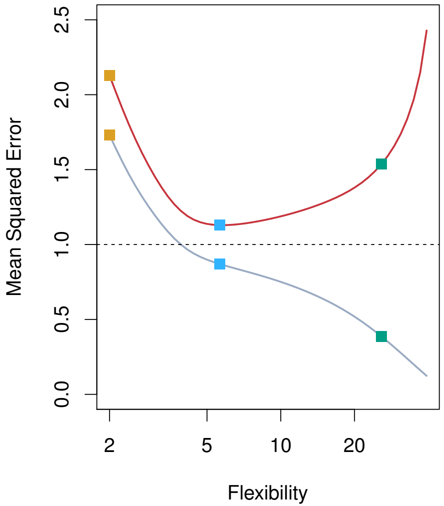


Black: true curve f
Orange: linear regression
Blue/green: nonlinear regressions

Grey: Training MSE
Red: Test MSE
Dashed: Minimum test MSE

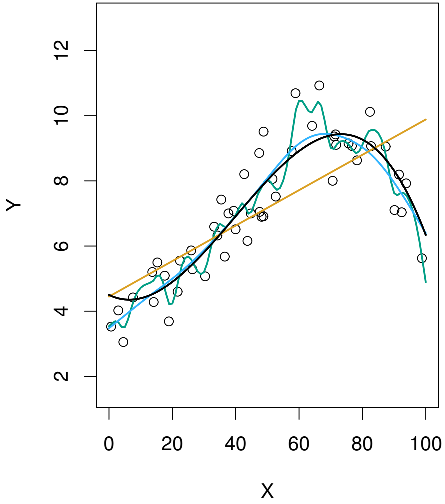
Training and test errors in regression (2/6)

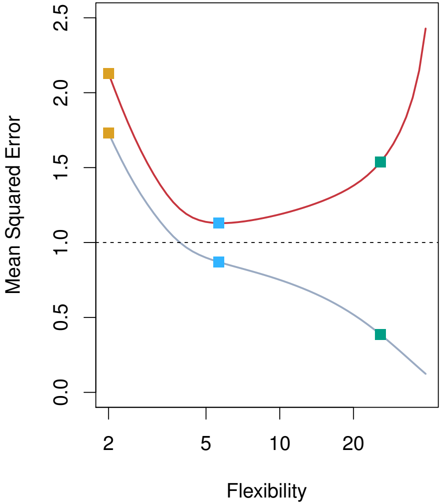




Here, **flexibility** refers to the "degrees of freedom" of the hypothesis set. For example, two parameters for linear regression, and more for nonlinear regression. The dashed line represents the **theoretical minimum test error** due to noise (called *irreducible error*).

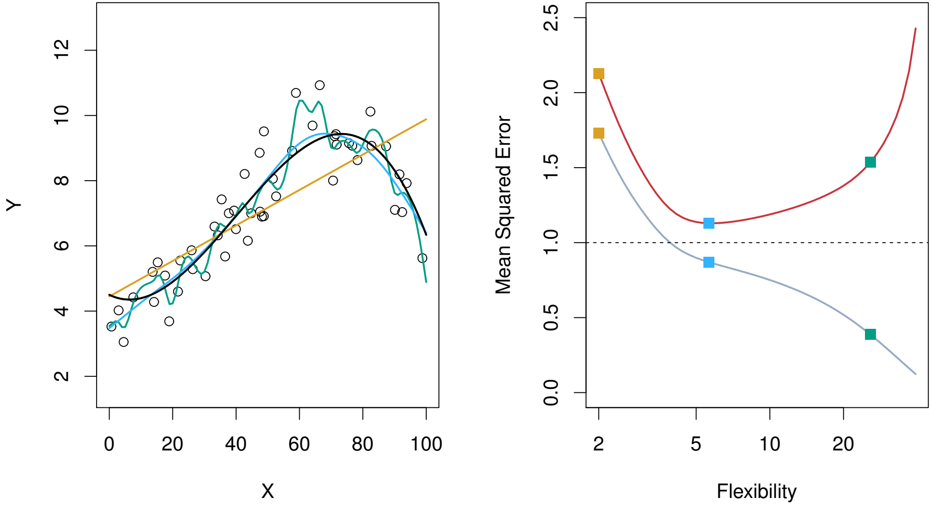
Training and test errors in regression (2/6)





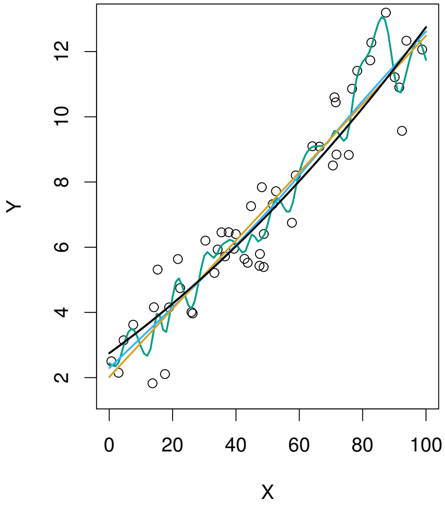
It is natural that the **training MSE decreases as the flexibility of the hypothesis set increases**, since we compute an argmin over a larger set. However, the test MSE may increase if the hypothesis set is too flexible.

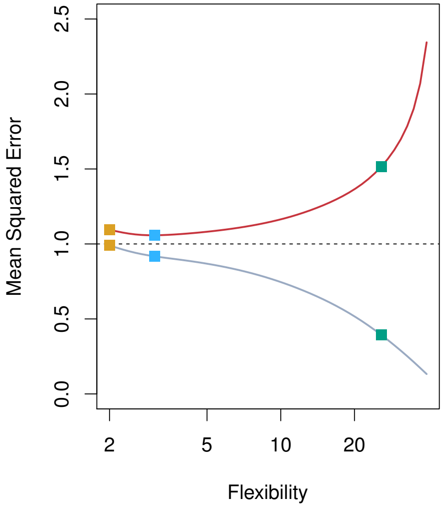
Training and test errors in regression (2/6)



Key takeaway: flexibility is not always better; *U-shaped* test error curve.

Training and test errors in regression: simpler f (3/6)

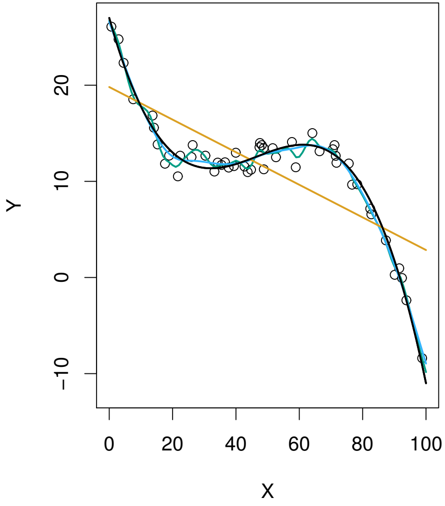


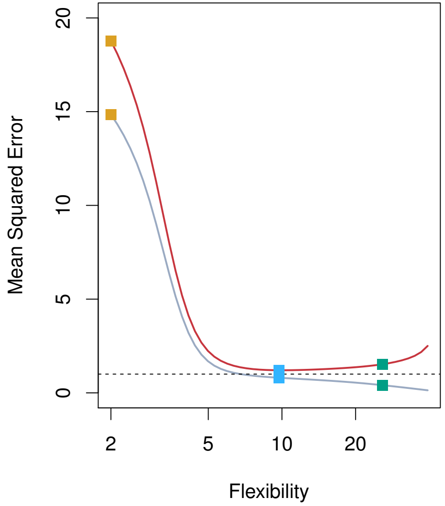


Black: true curve f
Orange: linear regression
Blue/green: nonlinear regression

Grey: Training MSE
Red: Test MSE
Dashed: Minimum test MSE

Training and test errors in regression: more complex f (4/6)

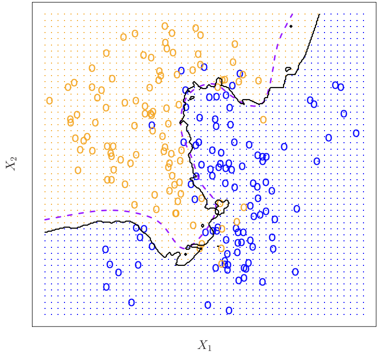




Black: true curve f
Orange: linear regression
Blue/green: nonlinear regression

Grey: Training MSE
Red: Test MSE
Dashed: Minimum test MSE

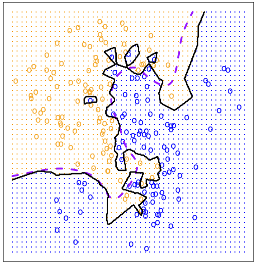
Training and test errors in classification (5/6)

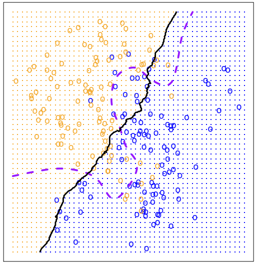


The same phenomenon occurs with classification: too much flexibility could lead to a increase in the test error.

Above: KNN with $K = 10$.
Bottom left: KNN with $K = 1$.
Bottom right: KNN with $K = 100$.

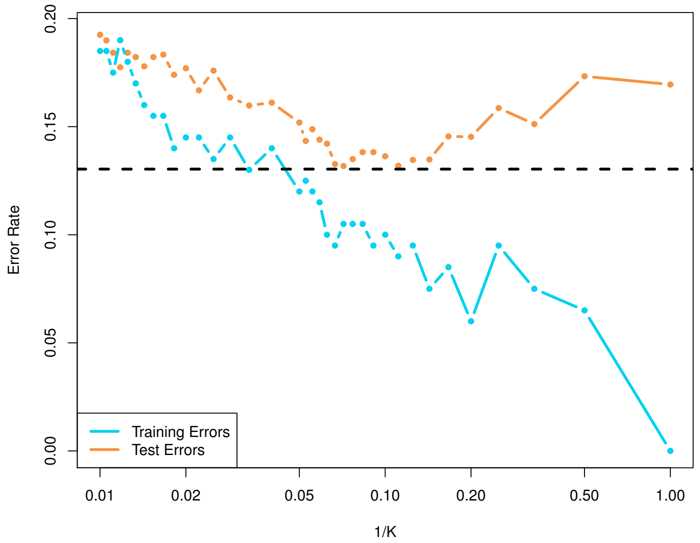
Observe that taking $K = 1$ leads to a training error of 0.



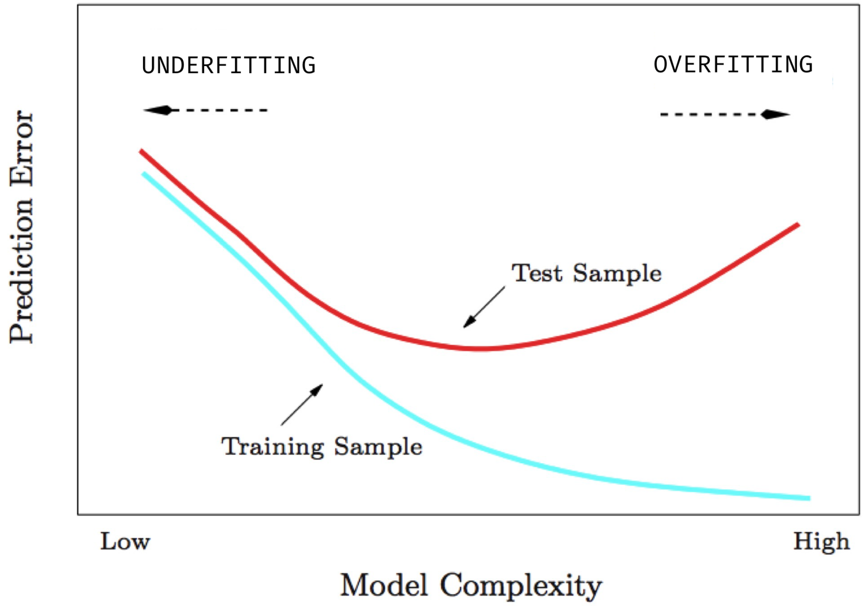


Training and test errors in classification (6/6)

The training and test errors of KNN as the flexibility increases (i.e., as K decreases). The dashed line represents the minimum test error.



A fundamental picture: overfitting vs. underfitting



Training and test errors

Consider

$$f = \operatorname*{argmin}_{h\colon \mathcal{X}\to\mathcal{Y}} E_{\text{out}}(h),$$

$$g^* = \operatorname*{argmin}_{h\in\mathcal{H}} E_{\text{out}}(h),$$

and

$$g = \operatorname*{argmin}_{h\in\mathcal{H}} E_{\text{in}}(h).$$

How different are $E_{\text{out}}(g)$ and $E_{\text{out}}(f)$?

The approximation-generalization tradeoff

The difference between the out-of-sample error of g and f can be decomposed as follows

$$E_{\text{out}}(g) - E_{\text{out}}(f) = \underbrace{\left[E_{\text{out}}(g^*) - E_{\text{out}}(f)\right]}_{\text{Approximation error}} + \underbrace{\left[E_{\text{out}}(g) - E_{\text{out}}(g^*)\right]}_{\text{Estimation error}}.$$

- **Approximation error** is how far the entire hypothesis set is from f. Larger hypothesis sets have lower approximation error.
- **Estimation error** is how good g is with respect to the best function in the hypothesis set. Larger hypothesis sets have higher estimation error because it is harder to find a good prediction function based on limited data.

This is called the **approximation-generalization** tradeoff.

Table of contents

E_{in} for MLE

Recall that, given a set of possible distributions

$$\mathcal{H} = \{p(y;\boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \boldsymbol{\Theta}\},$$

and a dataset $y_1, y_2, \ldots, y_n$, the maximum log-likelihood estimator is given by

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmax}} \sum_{i=1}^{n} \log\, p(y_i;\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} -\log\, p(y_i;\boldsymbol{\theta})}_{E_{\text{in}}(\boldsymbol{\theta})}. \tag{3}$$

Since each hypothesis h is completely characterized by $\boldsymbol{\theta}$, this is equivalent to

$$g = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\ E_{\text{in}}(h).$$

E_{out} for MLE

By the (strong) law of large numbers,

$$\frac{1}{n}\sum_{i=1}^{n} -\log p(y_i;\boldsymbol{\theta}) \overset{n\to\infty}{\longrightarrow} \mathbb{E}[-\log p(y;\boldsymbol{\theta})].$$

In other words, we can think of maximum likelihood estimation as trying to minimize

$$E_{\text{out}}(\boldsymbol{\theta}) = \mathbb{E}[-\log p(y;\boldsymbol{\theta})].$$

E_{out} for MLE

$$\begin{aligned}
\underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\ E_{\text{out}}(\boldsymbol{\theta}) &= \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\ \mathbb{E}[-\log p(y;\boldsymbol{\theta})] \\
&= \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\ \mathbb{E}\left[\log p(y) - \log p(y;\boldsymbol{\theta})\right] \\
&= \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\ \mathbb{E}\left[\log\left(\frac{p(y)}{p(y;\boldsymbol{\theta})}\right)\right] \\
&= \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\ \int \log\left(\frac{p(y)}{p(y;\boldsymbol{\theta})}\right) p(y)\ \mathrm{d}y \\
&= \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\ \mathrm{KL}(p_\theta, p),
\end{aligned}$$

where $KL(q, p)$ is the **Kullback-Leibler divergence** between two distributions q and p, which measures the discrepancy between the two distributions. It is also called *relative entropy*. Note that KL-divergence is not a distance measure (it is not symmetric).
You can compare this general formula with the expression we found for the MLE of the logistic regression.