Regression and Value Estimation 20

by **Patrick Boily**

In Chapter 19 (*Machine Learning 101*), we provided a (basically) math-free general overview of machine learning. In this chapter, we present an introductory mathematical treatment of the discipline, with a focus on regression and value estimation methods (in particular, on parametric methods).

Our approach borrows heavily from [2, 3]; explanations and examples are also available in [233].

We will continue the ML treatment in Chapters 21 and 22 (and 23, to a lesser extent).

20.1 Statistical Learning

Statistical learning is a series of procedures and approaches that allows analysts to tackle problems such as:

- identifying risk factors associated to breast/prostate cancer;
- predicting whether a patient will have a second, fatal heart attack within 30 days of the first on the basis of demographics, diet, clinical measurements, etc.;
- establishing the relationship between salary and demographic information in population survey data;
- predicting the yearly inflation rate using various indicators, etc.

Statistical learning tasks are typically divided into 2 main classes: **supervised learning** and **unsupervised learning**.[1]

1: There are other types, such as semi-supervised or reinforcement learning, but these are topics for future chapters.

20.1.1 Supervised Learning Framework

In the supervised learning environment, the **outcome** (response, target, dependent variable, etc.) is denoted by Y, and the vector of p **predictors** (features) by $\vec{X} = (X_1, \dots, X_p)$.

If Y is quantitative (price, height, etc.), then the problem of predicting Y in terms of $\vec{X}$ is a **regression** task; if Y takes on values in a finite unordered set (survived/died, colours, vegetation types, etc.), it is a **classification** task. This is typically achieved with the use of **training data**, which is to say historical observations or instances, which we often denote by $[\mathbf{X} \mid \mathbf{Y}]$ (the column denoting the observation IDs is dropped).

obs.	predictors	predictors	predictors	resp.
1	$x_{1,1}$	$\cdots$	$x_{1,p-1}$	y_1
$\vdots$	$\vdots$		$\vdots$	$\vdots$
n	$x_{n,1}$	$\cdots$	$x_{n,p-1}$	y_n

The objectives of supervised learning are usually to:

- accurately predict **unseen** test cases;
- understand which inputs affect the outcomes (if any), and how;
- assess the quality of predictions and/or inferences made on the basis of the training data, etc.

In unsupervised learning, on the contrary, there are no outcome variables, only features on a set of observations **X**.[2]

2: The response variable **Y** that was segregated away from **X** in the supervised learning case could now be one of the variables in **X**.

The objectives are much more **vague** – analysts could seek to:

- find sets of features that behave similarly across observations;
- find combinations of features with the most variation;
- find natural groups of similar observations, etc.

We will discuss such methods in detail in Chapter 22.

Statistical Learning vs. Machine Learning The term "statistical learning" is not used frequently in practice;[3] we speak instead of **machine learning**. If a distinction must be made, we could argue that:

3: Except by mathematicians and statisticians, perhaps.

- statistical learning arises from statistical-like models, and the emphasis is usually placed on **interpretability**, **precision**, and **uncertainty**, whereas
- machine learning arise from artificial intelligence studies, with emphasis on **large scale applications** and **prediction accuracy**.

The dividing line between the terms is blurry – the vocabulary used by practitioners mostly betrays their educational backgrounds (but see [166] for another take on this).

Motivating Example Throughout, we will illustrate the concepts and notions *via* the `gapminder.csv` dataset, which records socio-demographic information relating to the planet's nations, from 1960 to 2011 [61, 234].

We will be interested in determining if there is a link between life expectancy, at various moments in time, and the rest of the predictors.

The dataset contains 7139 observations of 9 columns:

- a country × year identifier (2 variables, i and X_1);
- a region and continent pair of categorical predictors (2 variables, X_2 and X_3);
- four numerical predictors: population X_4, infant mortality X_5, fertility X_6, gross domestic product in 1999 dollars X_7, and
- life expectancy Y, the numerical response.

Setting up the Gapminder dataset

```
library(dplyr)
gapminder.ML = read.csv("gapminder.csv",
                        stringsAsFactors=TRUE)
gapminder.ML <- gapminder.ML[complete.cases(gapminder.ML),]
gapminder.ML <- gapminder.ML[,c("country","year","region",
        "continent","population","infant_mortality",
        "fertility","gdp","life_expectancy")]
```

The structure is provided below:

```
str(gapminder.ML)
```

```
'data.frame':   7139 obs. of  9 variables:
 $ country         : Factor w/ 185 levels "Albania","Algeria",..: 2 5 8 9 11 13 14 16 18 20 ...
 $ year            : int  1960 1960 1960 1960 1960 1960 1960 1960 1960 1960 ...
 $ region          : Factor w/ 22 levels "Australia and New Zealand",..: 11 15 1 22 2 18 2 ...
 $ continent       : Factor w/ 5 levels "Africa","Americas",..: 1 2 5 4 2 3 2 4 1 2 ...
 $ population      : int  11124892 20619075 10292328 ...
 $ infant_mortality: num  148.2 59.9 20.3 37.3 51 ...
 $ fertility       : num  7.65 3.11 3.45 2.7 4.5 6.73 4.33 2.6 6.28 6.7 ...
 $ gdp             : num  1.38e+10 1.08e+11 9.67e+10 5.24e+10 1.31e+09 ...
 $ life_expectancy : num  47.5 65.4 70.9 68.8 62 ...
```

In other words, we will be looking for **models** of the form

$$Y = f(X_1, \ldots, X_7) + \varepsilon \equiv f(\vec{X}) + \varepsilon,$$

where f is the **systematic component of** Y **explained by** X, and ε is the **random error term**, which accounts for measurement errors and other deviations and discrepancies.[4]

4: Generally, we require $\mathrm{E}(\varepsilon) = 0$.

20.1.2 Systematic Component and Regression

It is the systematic component that is used for **predictions** and **inferences**. As long as f is "good", we can:

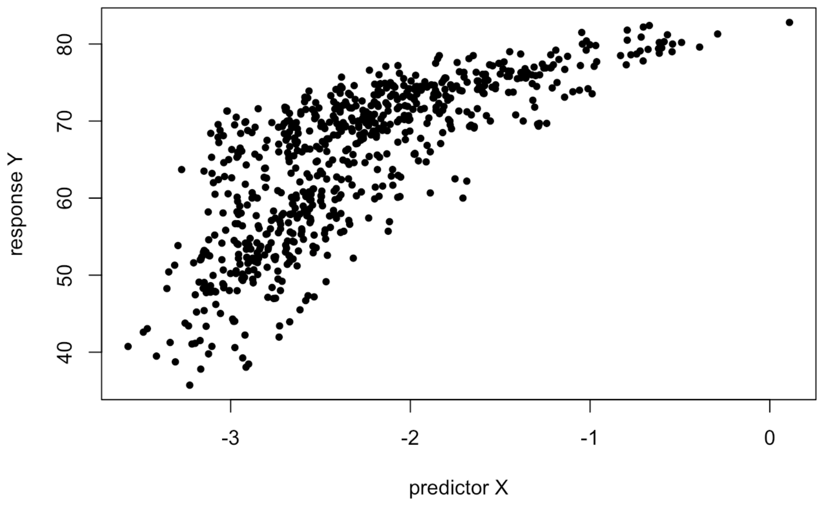
- make predictions for the response Y at new points $\vec{X} = \mathbf{x}$;
- understand which features of $\vec{X} = (X_1, \ldots, X_p)$ are important to explain the variation in Y, and
- depending on the complexity of f, understand the effect of each feature X_j on Y.

Imagine a model with one predictor X and a target Y, with systematic component f, so that

$$Y = f(X) + \varepsilon.$$

For instance, consider the following subset of the Gapminder dataset.

```
attach(gapminder.ML)
x=log(fertility[10*(1:730)]/infant_mortality[10*(1:730)])
y=life_expectancy[10*(1:730)]
x=x[!is.na(x)]
y=y[!is.na(y)]
plot(x,y,ylab="response Y", xlab="predictor X", pch=20)
```



What is the ideal f in this case? How can we find it?

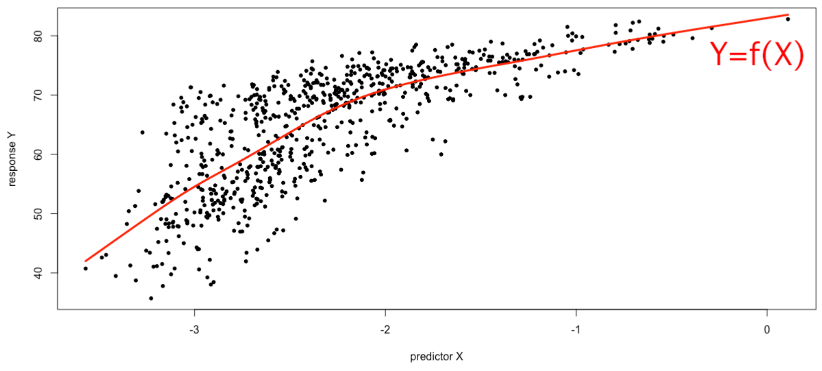


Figure 20.1: Regression model for a subset of the Gapminder data.

In that case, what would be a good value of $f(-2)$, say?

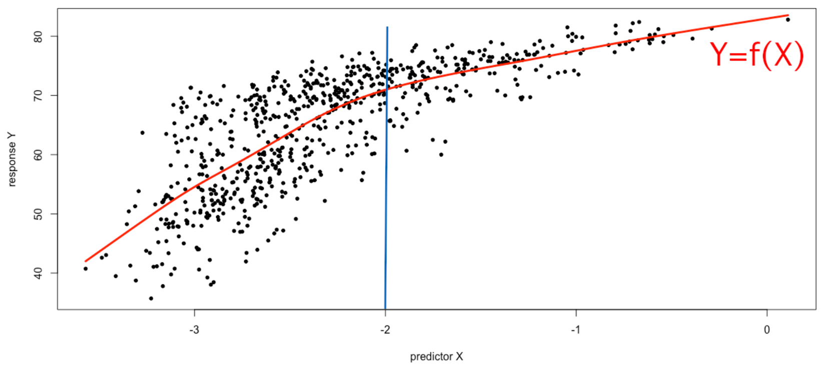


Figure 20.2: Regression model for a subset of the Gapminder data, with vertical line at $X = -2$.

Ideally, we would like to have $f(-2) = \mathrm{E}[Y \mid X = -2]$.[5] For any x in the range of X, the function

$$f(x) = \mathrm{E}[Y \mid X = x]$$

5: Why?

is the **regression function of Y on X**. In the general setting with p predictors, the regression function is

$$f(\mathbf{x}) = f(x_1, \dots, x_p) = \mathrm{E}[Y \mid X_1 = x_1, \dots, X_p = x_p] = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}].$$

It is **optimal** in the sense that this regression function minimizes the average square deviation from the response variable, that is to say,

$$f = \arg\min_g \left\{ \mathrm{E}\left[(Y - g(\vec{X}))^2 \mid \vec{X} = \mathbf{x}\right] \right\}.$$

The term

$$\varepsilon = \varepsilon_{\vec{X}} = Y - f(\vec{X})$$

is the **irreducible error** of the regression. Typically, $\varepsilon_{\vec{X}} \neq 0$ for all $\vec{X}$, since, even when f is known exactly, there will still be some uncertainty in the predictions due to some noise-generating mechanism in the "real world".

If $\hat{f}$ is any estimate of the regression function f,[6] then

6: In particular, $\hat{f}(\vec{X}) = \hat{Y} \approx Y = f(\vec{X}) + \varepsilon$.

$$\begin{aligned} \mathrm{E}[(Y - \hat{Y})^2 \mid \vec{X} = \mathbf{x}] &= \mathrm{E}[(f(\vec{X}) + \varepsilon - \hat{f}(\vec{X}))^2 \mid \vec{X} = \mathbf{x}] \\ &= \underbrace{[f(\mathbf{x}) - \hat{f}(\mathbf{x})]^2}_{\text{reducible}} + \underbrace{\mathrm{Var}(\varepsilon)}_{\text{irreducible}}. \end{aligned}$$

Since the **irreducible component** is not a property of the estimate $\hat{f}$, the objective of minimizing $\mathrm{E}[(Y - \hat{Y})^2]$ can only be achieved by working through the **reducible component**. When we speak of **learning** a model, we mean that we use the training data to find an estimate $\hat{f}$ of f that minimizes this reducible component, in some way.

Estimating the Regression Function In theory, we know that the regression function is

$$f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}];$$

in practice, however, there might be too few (or even no) observations at $\vec{X} = \mathbf{x}$ to trust the estimate provided by the sample mean. One solution is to approximate the expectation by a **nearest neighbour average**

$$\hat{f}(\mathbf{x}) = \mathrm{Avg}\{Y \mid \vec{X} \in N(\mathbf{x})\},$$

where $N(\mathbf{x})$ is a neighbourhood of $\mathbf{x}$.

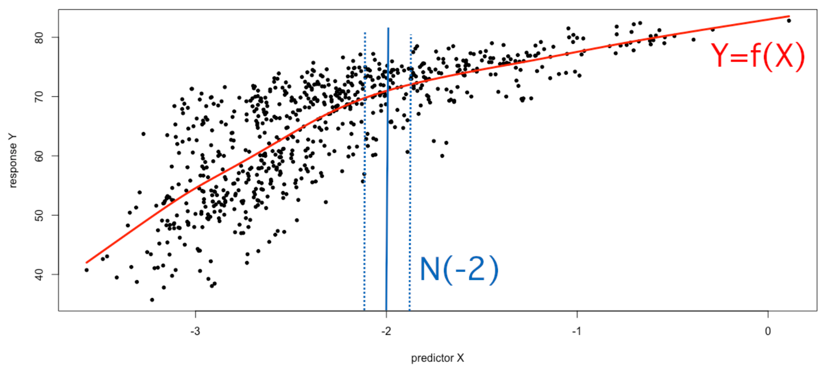


Figure 20.3: Regression model for a subset of the Gapminder data, with vertical line at $X = -2$ and neighbourhood $N(-2)$.

In general, this approach works reasonably well when p is "small" ($p \leq 4$?) and N is "large", but it fails when p is too large because of the **curse of dimensionality**. The problem is that nearest neighbours are usually far when p is large. Indeed, if $N(\mathbf{x})$ is defined as the nearest 5% of observations to $\mathbf{x}$, say,[7] then we need to leave the "local" neighbourhood of $\mathbf{x}$ to build $N(\mathbf{x})$, which could compromise the quality of $\hat{f}(\mathbf{x})$ as an approximation to $f(\mathbf{x})$.

7: The proportion must be large enough to bring the variance down.

We provide more details in Chapter 23, but this is a topic about which it is worth being well-read (see [2] for a formal treatment).

The various statistical learning methods attempt to provide estimates of the regression function by minimizing the reducible component through **parametric** or **non-parametric** approaches.[8] For instance, the **classical linear regression** approach is parametric: it assumes that the true regression function f is linear and suggests the estimate

8: In this context, "parametric" means that assumptions are made about the form of the regression function f; "non-parametric" means that no such assumptions are made.

$$f_L(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

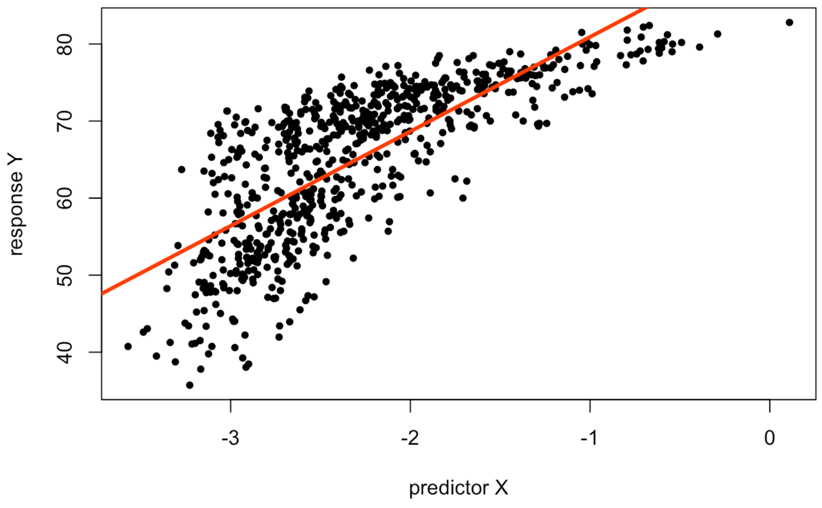
The objective, in this case, is to learn the $p+1$ parameters $\beta_0, \beta_1, \ldots, \beta_p$ with the help of the training data.

In practice, this assumption almost never holds, but it often provides an **interpretable**[9] approximation to the true regression function f (see below for an example).

9: We will revisit this concept at a later stage.

Gapminder subset and linear regression

```
lin.reg = lm(y~x)
plot(x,y,ylab="response Y", xlab="predictor X", pch=20)
abline(lin.reg, col="red", lwd=3)
```



As an example, if the true fit of the motivating example was

$$\text{life expectancy} = f(\text{fertility}, \text{infant mortality}, \text{gdp}) + \varepsilon,$$

say, then the linear regression approach would assume that

$$\begin{aligned} f(\text{fertility,infant mortality}, \text{gdp}) &\approx f_L(\text{fertility}, \text{infant mortality}, \text{gdp}) \\ &= \beta_0 + \beta_1 \cdot \text{fertility} + \beta_2 \cdot \text{infant mortality} + \beta_3 \cdot \text{gdp}. \end{aligned}$$

The main advantages of the linear model are that it is interpretable and that it is easier to learn $p + 1$ parameters than it is to learn a whole function f. On the flip side, the linear model does not usually match the true regression function f; if $f_L \not\approx f$, then predictions will suffer.

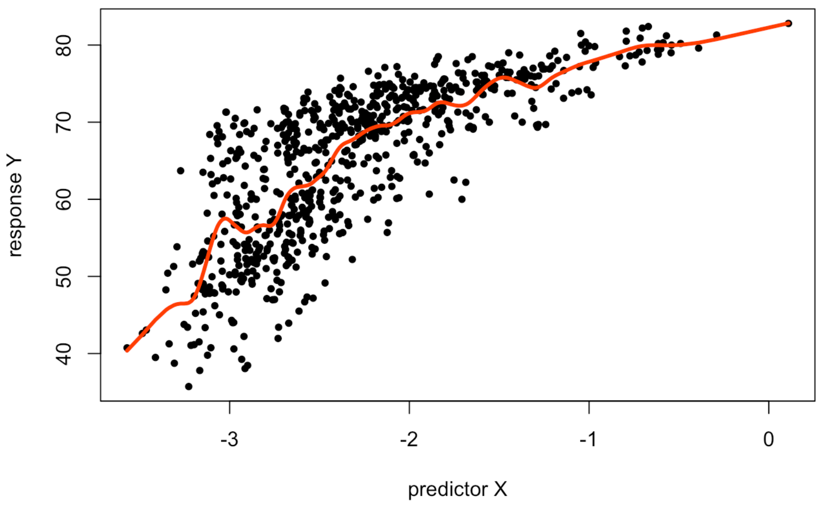
We could decide to consider more complex functions in order to get better estimates (and thus better prediction accuracy), but this comes at a **cost** – the resulting functions are usually more difficult to learn and they tend to **overfit the data**.[10]

10: Which is to say, they mistake noise in the data for a signal to model, see Section 20.1.4 for details.

Splines provide examples of non-parametric models (see Section 20.5.2): they make no assumption about the form of f – they simply seek to estimate f by getting close to the data points without being too **rough** or too **wiggly**, as below.

Gapminder subset and smoothing spline

```
smoothingSpline = smooth.spline(x, y, spar=0.7)
plot(x,y,ylab="response Y", xlab="predictor X", pch=20)
lines(smoothingSpline, col="red", lwd=3)
detach(gapminder.ML)
```



The main advantage of non-parametric approaches is that they have the potential to fit a **wider range** of regression shapes. But since estimating f is not reduced to learning a small number of parameters, substantially more data is required to obtain accurate estimates.[11]

11: And the whole situation is susceptible to overfitting.

Non-parametric methods are usually more **flexible** (they can produce a large range of shapes when estimating the true regression function f); parametric models are usually more **interpretable**.[12]

12: The set of parameters to learn is small and we can more easily make sense of them, which leads us to a better understanding of how the predictors interact to produce outputs.

Approaches that provide:

- high flexibility, but low interpretability include ensemble learning, support vector machines, neural networks, and splines;
- low flexibility, but high interpretability include the LASSO and OLS, and
- medium flexibility and medium interpretability include generalized additive models and regression trees.

There are no **high-flexibility/high-interpretability** approaches. The **trade-off** between two competing desirable properties is the calling card of machine learning; we will encounter such trade-offs time and time again; they dictate what the discipline can and cannot hope to achieve.

20.1.3 Model Evaluation

In an ideal world,[13] we would want to identify the modeling approach that performs "best", and use it for all problems.

13: From a model performance point of view.

The discussion on **trade-offs** shows that the concept of "best performance" is impossible to define in practice in a way that meets all desired requirements, and a balance must be struck. Another issue lurks around the corner, even when we settle on an "optimal" performance evaluation measure: no single method is optimal over all possible datasets.[14]

14: In reality, machine learning is simply applied optimization; the proof of this **No-Free Lunch Theorem** falls outside the scope of this document (but see [216, 235] for details).

Given a specific task and dataset, then, how do we select the approach that will yield the best results (for a given value of "best")? In practice, **this is the main machine learning challenge.**

In order to evaluate a model's performance at a specific task, we must be able to measure how well predictions match the **observed data**. In a **regression/value estimation setting**, various metrics are used:

- **mean squared error** (MSE):
$$\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{f}(\mathbf{x}_i))^2;$$
- **mean absolute error** (MAE):
$$\frac{1}{N}\sum_{i=1}^{N}|y_i-\hat{f}(\mathbf{x}_i)|;$$
- **normalized mean squared error** (NMSE):
$$\frac{\sum_{i=1}^{N}(y_i-\hat{f}(\mathbf{x}_i))^2}{\sum_{i=1}^{N}(y_i-\overline{y})^2};$$
- **normalized mean absolute error** (NMAE):
$$\frac{\sum_{i=1}^{N}|y_i-\hat{f}(\mathbf{x}_i)|}{\sum_{i=1}^{N}|y_i-\overline{y}|};$$
- **mean average percentage error** (MAPE):
$$\frac{1}{N}\sum_{i=1}^{N}\frac{|y_i-\hat{f}(\mathbf{x}_i)|}{y_i};$$
- **correlation** $\rho_{\hat{y},y}$, etc.

The MSE has convenient mathematical properties, and we will follow the lead of just about every reference in making it our go-to metric, but note that the conceptual notions we will discuss would be qualitatively similar for all performance evaluation tools.

Note that in order to evaluate the suitability of a model for **predictive** purposes, these metrics should be evaluated on **testing data** (or unseen data), not on the training data.[15]

15: Failure to do so means that the model can at best be used to describe the training dataset (which might still be a valuable contribution).

For instance, if we are trying to determine whether any clinical measurement in patients are likely to predict the onset of Alzheimer's disease, we do not particularly care if the algorithm does a good job of telling us that the patients we have already tested for the disease have it or not – it is new patients that are of interest.[16]

16: Although it would be surprising if the performance on the test data is any good if the performance on the training data is middling. We shall see at a later stage that the training/testing paradigm can also help with problems related to overfitting.

Let $\text{Tr} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, N\}$ be the **training set** and suppose that we use some statistical learning method to estimate the true relationship $Y = f(\vec{X}) + \varepsilon$ by $\hat{Y} = \hat{f}(\vec{X})$, i.e., **we fit $\hat{f}$ over Tr**.

Hopefully, we have $\hat{f}(\mathbf{x}_i) \approx y_i$ for all $i = 1, \ldots, N$, and

$$\text{MSE}_{\text{Tr}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{f}(\mathbf{x}_i))^2$$

is small.

If it is indeed small, then the model does a good job of **describing** Tr. But, as discussed above, this is largely irrelevant to (if not uncorrelated with) our ability to make **good predictions**; what we would really like to know is if

$$\hat{f}(\mathbf{x}^*) \approx f(\mathbf{x}^*) = y^*$$

for observations $(\mathbf{x}^*, y^*) \notin \text{Tr}$.

An **optimal** statistical learning method for a given combination of task and dataset is one that minimizes

$$\text{MSE}_{\text{Te}} = \frac{1}{M}\sum_{j=N+1}^{N+M=n}(y_j - \hat{f}(\mathbf{x}_j))^2$$

over the testing set $\text{Te} = \{(\mathbf{x}_j, y_j) \mid j = N+1, \ldots, N+M = n\}$, where, *a priori*, none of the test observations were in Tr.[17] The general situation is illustrated in Figures 20.4 and 20.5.

17: New test observations may end up assuming the same values as some of the training observations, but that is an accident of sampling and/or it is due to the reality of the scenario under consideration.

20.1.4 Bias-Variance Trade-Off

The "U" shape of the testing MSE in Figure 20.5 is **generic** – something of this nature occurs for nearly all datasets and choice of supervised learning family of methods (for regression and for classification): **underfitting** and **overfitting** is a fact of machine learning life.

The generic shape can be explained by two properties of SL methods: the **bias** and the **variance**. Consider a test observation $(\mathbf{x}^*, y^*)$, and a fitted model $\hat{f}$ (trained on Tr), which approximates the true model

$$Y = f(\vec{X}) + \varepsilon, \quad \text{where } f(\mathbf{x}) = \text{E}[Y \mid \vec{X} = \mathbf{x}].$$

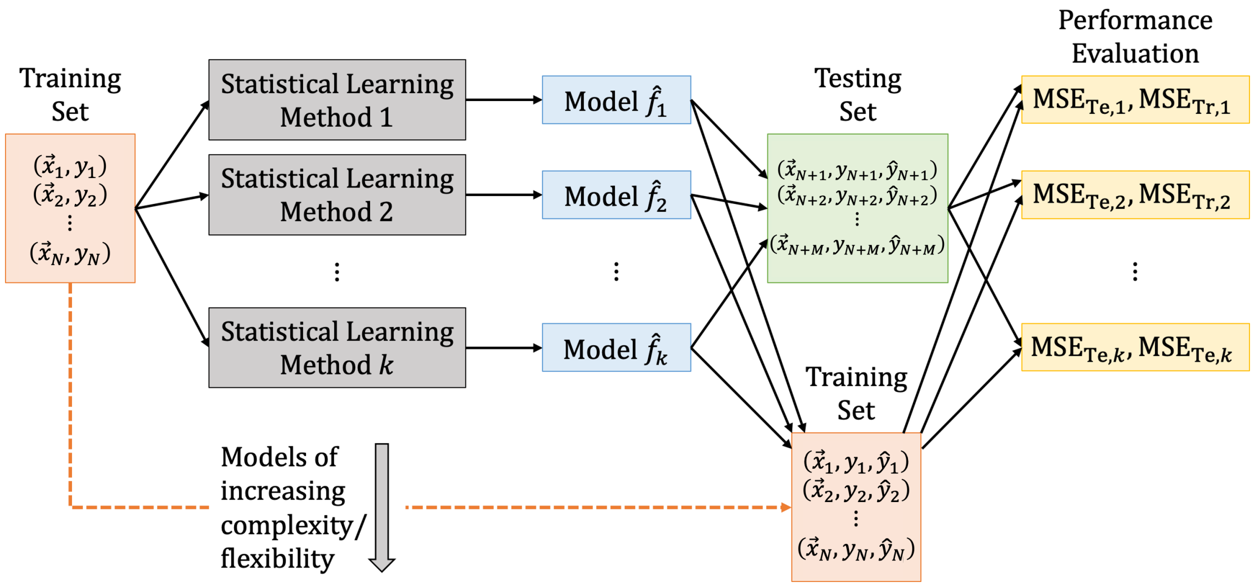

Training Set
$(\vec{x}_1, y_1)$
$(\vec{x}_2, y_2)$
$\vdots$
$(\vec{x}_N, y_N)$
Statistical Learning Method 1
Statistical Learning Method 2
Statistical Learning Method k
Model $\hat{f}_1$
Model $\hat{f}_2$
Model $\hat{f}_k$
Testing Set
$(\vec{x}_{N+1}, y_{N+1}, \hat{y}_{N+1})$
$(\vec{x}_{N+2}, y_{N+2}, \hat{y}_{N+2})$
$(\vec{x}_{N+M}, y_{N+M}, \hat{y}_{N+M})$
Training Set
$(\vec{x}_1, y_1, \hat{y}_1)$
$(\vec{x}_2, y_2, \hat{y}_2)$
$(\vec{x}_N, y_N, \hat{y}_N)$
Performance Evaluation
$\text{MSE}_{\text{Te},1}, \text{MSE}_{\text{Tr},1}$
$\text{MSE}_{\text{Te},2}, \text{MSE}_{\text{Tr},2}$
$\text{MSE}_{\text{Te},k}, \text{MSE}_{\text{Te},k}$
Models of increasing complexity/ flexibility

Figure 20.4: The training/testing paradigm. Training data is fed into a variety of statistical learning methods, possibly arranged in increasing order of complexity, yielding a sequence of models. These models are then used to make predictions on the testing set (using only the predictors variables); the predictions are then compared with the actual values to evaluate the performance of the models on the testing set. The performance of the models on the training set can also be evaluated.

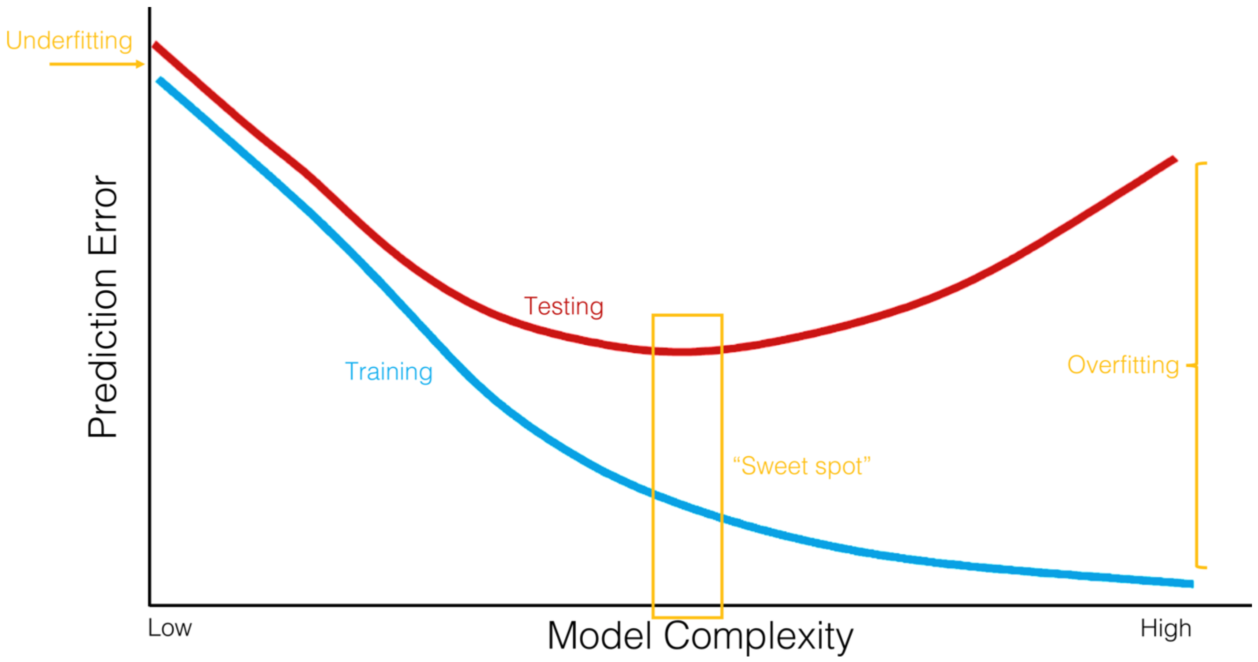


Figure 20.5: Generic illustration of the bias-variance trade-off; when the complexity of the model increases, the training error decreases, but the testing error eventually starts increasing. Generally, models that are too simple will have 'large' prediction errors on both the training and the testing sets (underfitting), whereas for models that are too complex, the training error tends to be "small" while the testing error tends to be "large" (overfitting). Based on [2, 3].

The **expected test MSE at** $\mathbf{x}^*$ can be decomposed into 3 fundamental quantities

$$\begin{aligned}\mathrm{E}\left[\mathrm{MSE}_{\mathrm{Te}}(\mathbf{x}^*)\right] &= \mathrm{E}\left[(y^* - \hat{f}(\mathbf{x}^*))^2\right] \\ &= \underbrace{\mathrm{Var}(\hat{f}(\mathbf{x}^*))}_{\text{variance}} + \underbrace{\left\{\mathrm{E}\left[\hat{f}(\mathbf{x}^*) - f(\mathbf{x}^*)\right]\right\}^2}_{\text{squared bias}} + \mathrm{Var}(\varepsilon).\end{aligned}$$

As before, $\mathrm{Var}(\varepsilon)$ is the **irreducible error** (due to the inherent noise in the data); the **variance component error** $\mathrm{Var}(\hat{f}(\mathbf{x}^*))$ arises since different training sets would yield different fitted models $\hat{f}$, and the **(squared) bias component error** arises, in part, due to the "difficult" problem being approximated by a "simple" model (see [2, 3] for details).

The **overall expected test MSE** $\mathrm{E}[\mathrm{MSE}_{\mathrm{Te}}]$ is the average of $\mathrm{E}[\mathrm{MSE}_{\mathrm{Te}}(\mathbf{x}^*)]$ over all allowable $\mathbf{x}^*$ in the testing space. Note that

$$\mathrm{E}[\mathrm{MSE}_{\mathrm{Te}}] \geq \mathrm{Var}(\varepsilon),$$

by construction.

In general, more flexible methods (i.e., more complex methods) tend to have higher variance and lower bias, and *vice-versa*: simpler methods have higher bias and lower variance. It is this interplay between bias and variance that causes models to underfit (high bias) or overfit (high variance) the data (see **bias-variance trade-off** diagram below).

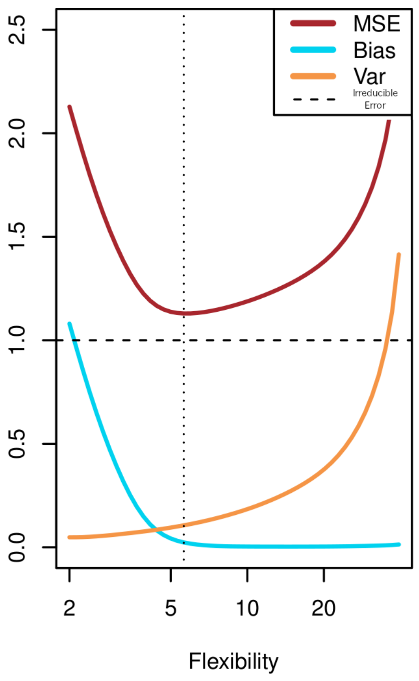


Figure 20.6: Expected test error decomposition, artificial dataset [3].

Let us summarize the main take-aways from the first section:

- the optimal regression function $Y = f(\vec{X}) + \varepsilon$ for **numerical responses** is
$$f(x) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}];$$
- models are **learned** on training data Tr;

- in practice, we learn the best model from a **restricted** group of model families;
- the best model $\hat{f}(\mathbf{x})$ minimizes the reducible part of the prediction error MSE_{Te}, **evaluated** on testing data Te;
- the **bias-variance trade-off** tells us that models that are too simple (or too rigid) **underfit** the data, and that models that are too complex (or too "loose") **overfit** the data;
- the total prediction error on Te is **bounded below** by the irreducible error.

Finally, remember that a **predictive** model's performance **can only be evaluated on unseen data** (i.e., on data not drawn from the training set Tr); if this requirement is not met, the model is at best **descriptive**.

20.2 Regression Modeling

In the regression setting, the goal is to estimate the regression function

$$f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}],$$

the solution to the regression problem

$$Y = f(\vec{X}) + \varepsilon.$$

The best estimate $\hat{f}$ is the model that minimizes

$$\text{MSE}_{\text{Te}}(\hat{f}) = \text{Avg}_{\mathbf{x}^* \in Te}\mathrm{E}\left[(y^* - \hat{f}(\mathbf{x}^*))^2\right].$$

In practice, this can be hard to achieve without restrictions on the functional form of $\hat{f}$, so we try to **learn** the best $\hat{f}$ from **specific families of models**. Remember, however, that no matter what the approximation function $\hat{f}$ is,

$$\text{MSE}_{\text{Te}}(\hat{f}) \geq \text{Var}(\varepsilon).$$

What else can we say about $\hat{f}$? In the **ordinary least square framework** (OLS), we assume that

$$\hat{f}_{\text{OLS}}(\mathbf{x}) \approx \mathbf{x}^{\top}\boldsymbol{\beta},$$

which is to say that we assume that $\hat{f}_{\text{OLS}}$ is nearly globally linear.[18]

18: We neglect the intercept term, in this interpretation.

In practice, the true regression function is almost never linear, but the linear assumption yields models $\hat{f}$ that are both **conceptually** and **practically** useful – the model $\hat{f}$ is easily interpretable, and the associated prediction error $\text{MSE}_{\text{Te}}(\hat{f})$ is often "small-ish".

The most common data modeling methods are **linear and logistic regression methods**. By some estimation, 90% of real-world data applications end up using these as their final model, typically after very carefully preparing the data (cleaning, encoding, creation of new variables, transformation of variables, etc.).

That is mostly due to the:

- regression models being straightforward to **interpret** and to **train**;
- MSE_{Te} having a closed-form linear expression, and

- OLS solution being computable using simple matrix manipulations.

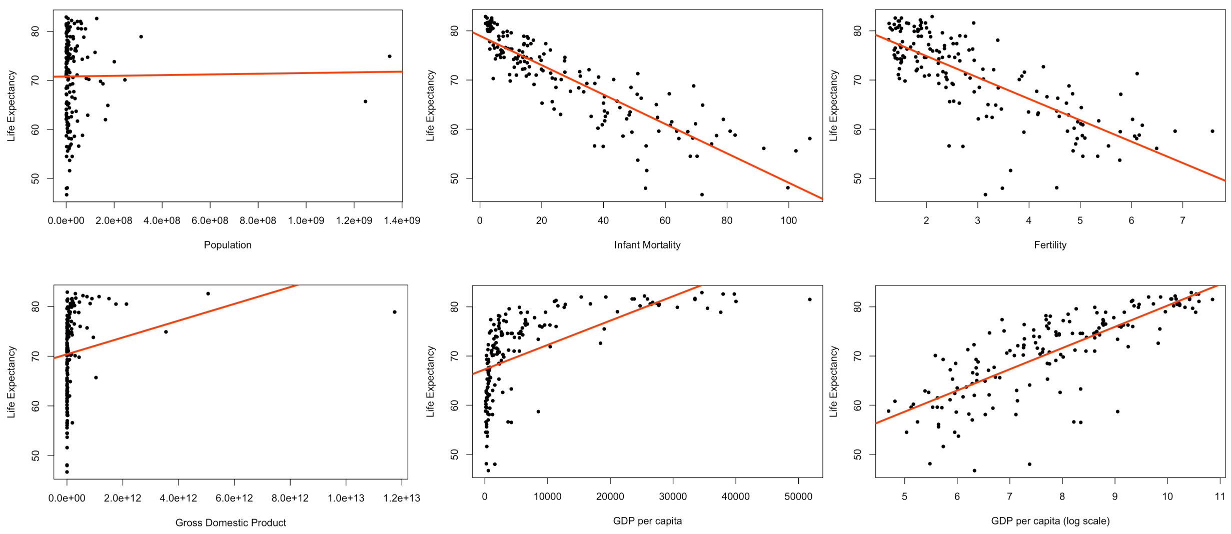
Gapminder Example Let us revisit the `Gapminder` dataset, focusing on observations from 2011.

- Is there a relationship between gross domestic product and life expectancy?
- How strong is the relationship?
- Which factors contribute to the life expectancy?
- How accurately could we predict life expectancy given a set of new observations?
- Is the relationship linear?
- Are there combinations of factors that are linked with life expectancy?

Can the scatterplots of various predictors against life expectancy for the 2011 Gapminder data, shown below with line of best fit, be used to answer these questions?

```r
gapminder.2011 <- gapminder.ML |> filter(year==2011)
attach(gapminder.2011)

x=population
y=life_expectancy
plot(x,y, xlab="Population", ylab="Life Expectancy")
abline(lm(y~x), col="red",lwd=3)

x=infant_mortality
y=life_expectancy
plot(x,y, xlab="Infant Mortality", ylab="Life Expectancy")
abline(lm(y~x), col="red",lwd=3)

x=fertility
y=life_expectancy
plot(x,y, xlab="Fertility", ylab="Life Expectancy")
abline(lm(y~x), col="red",lwd=3)

x=gdp
y=life_expectancy
plot(x,y, xlab="Gross Domestic Product",
     ylab="Life Expectancy")
abline(lm(y~x), col="red",lwd=3)

x=gdp/population
y=life_expectancy
plot(x,y, xlab="GDP per capita", ylab="Life Expectancy")
abline(lm(y~x), col="red",lwd=3)

x=log(gdp/population)
y=life_expectancy
plot(x,y, xlab="GDP per capita (log scale)",
     ylab="Life Expectancy")
```

```
abline(lm(y~x), col="red",lwd=3)

detach(gapminder.2011)
```



20.2.1 Formalism

Consider a dataset Tr $= \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ with N observations and $p-1$ features. The corresponding **design matrix**, **response vector**, and **coefficient vector** are, respectively,

$$\mathbf{X} = \begin{pmatrix} 1 & x_{1,1} & \cdots & x_{1,p-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_{N,1} & \cdots & x_{N,p-1} \end{pmatrix}, \quad \mathbf{Y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{p-1} \end{pmatrix}.$$

The objective is to find f such that $\mathbf{Y} = f(\mathbf{X}) + \boldsymbol{\varepsilon}$. The OLS solution assumes that $f(\mathbf{X}) = \mathbf{X}\boldsymbol{\beta}$; we must thus learn $\boldsymbol{\beta}$ using the training data Tr.

If $\hat{\boldsymbol{\beta}}$ is an estimate of the true coefficient vector $\boldsymbol{\beta}$, the **linear regression model** associated with Tr is

$$\hat{f}(\mathbf{x}) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_{p-1} x_{p-1}.$$

How do we find $\hat{\boldsymbol{\beta}}$? The OLS estimate minimizes the **loss function**

$$\begin{aligned} \mathscr{L}(\boldsymbol{\beta}) &= \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 = (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^\top (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) \\ &= \mathbf{Y}^\top \mathbf{Y} - ((\mathbf{X}\boldsymbol{\beta})^\top \mathbf{Y} + \mathbf{Y}^\top \mathbf{X}\boldsymbol{\beta}) + (\mathbf{X}\boldsymbol{\beta})^\top \mathbf{X}\boldsymbol{\beta} \\ &= \mathbf{Y}^\top \mathbf{Y} - (\boldsymbol{\beta}^\top \mathbf{X}^\top \mathbf{Y} + \mathbf{Y}^\top \mathbf{X}\boldsymbol{\beta}) + \boldsymbol{\beta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\beta}. \end{aligned}$$

The loss function is a non-negative symmetric quadratic form in $\boldsymbol{\beta}$, with no restriction on the coefficients, so any minimizer of $\mathscr{L}$ must also be one of its critical points (assuming certain regularity conditions on the data). We are thus looking for coefficients for which $\nabla \mathscr{L}(\boldsymbol{\beta}) = \mathbf{0}$. Since

$$\nabla \mathscr{L}(\boldsymbol{\beta}) = -2(\mathbf{X}^\top \mathbf{Y} - \mathbf{X}^\top \mathbf{X}\boldsymbol{\beta}),$$

any minimizer $\hat{\boldsymbol{\beta}}$ must satisfy the **canonical** (normal) **equations**:

$$\mathbf{X}^\top \mathbf{Y} = \mathbf{X}^\top \mathbf{X}\hat{\boldsymbol{\beta}}.$$

If $\mathbf{X}^\top\mathbf{X}$ is invertible, the minimizer $\hat{\boldsymbol{\beta}}$ is unique and is given by

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{Y}, \quad \text{with} \quad \text{Var}(\hat{\boldsymbol{\beta}}) = \hat{\sigma}^2(\mathbf{X}^\top\mathbf{X})^{-1},$$

where $\hat{\sigma}^2$ is the variance of the residuals.[19] We say that "**we have learned the coefficients $\hat{\boldsymbol{\beta}}$ on the training data Tr using linear regression**".

19: Note that $\mathbf{X}^\top\mathbf{X}$ is a $p \times p$ matrix, which makes the inversion relatively easy to compute even when N is large.

In what follows, we sometimes write $\mathbf{x}$ to represent the observation vector

$$(1, x_1, \ldots, x_{p-1})^\top;$$

it should be clear what is meant from the context.

The **fitted value of the model $\hat{f}$ at input $\mathbf{x}_i$** $\in$ Tr is

$$\hat{y}_i = \hat{f}(\mathbf{x}_i) = \mathbf{x}_i^\top\hat{\boldsymbol{\beta}},$$

and the **predicted value at an arbitrary $\mathbf{x}^*$** is

$$\hat{y}^* = \hat{f}(\mathbf{x}^*) = \mathbf{x}^{*\top}\hat{\boldsymbol{\beta}}.$$

The **fitted surface** is thus entirely described by the $p+1$ parameters $\hat{\boldsymbol{\beta}}$; the number of (effective) parameters is a measure of the **complexity** of the learner.

Motivating Example We study a subset of the Gapminder dataset: the observations for 2011, the predictor variables infant mortality X_1 and fertility X_2, and the response variable life expectancy Y. The training data Tr contains $N = 166$ observations and $p = 2$ predictor features.

The design matrix $\mathbf{X}$ is thus of dimension 166×3.

```
library(matlib)
gapminder.2011 <- gapminder.2011 |> dplyr::mutate(const=1)
design.X = gapminder.2011[,c("const","infant_mortality",
                "fertility")]
str(design.X)
```

```
'data.frame':   166 obs. of  3 variables:
 $ const           : num  1 1 1 1 1 1 1 1 1 1 ...
 $ infant_mortality: num  14.3 22.8 106.8 7.2 12.7 ...
 $ fertility       : num  1.75 2.83 6.1 2.12 2.2 1.5 1.88 1.44 1.96 1.9 ...
```

The response is a 166×1 vector.

```
resp.Y = gapminder.2011[,c("life_expectancy")]
```

The constituents of the canonical equations are:

```
(X.t.X = t(as.matrix(design.X)) %*% as.matrix(design.X))
(X.t.Y = t(as.matrix(design.X)) %*% as.matrix(resp.Y))
```

We thus see that

$$\mathbf{X}^{\top}\mathbf{X} = \begin{pmatrix} 166.0 & 4537.3 & 486.54 \\ 4537.3 & 225043.25 & 18445.28 \\ 486.54 & 18445.28 & 1790.238 \end{pmatrix}$$

and

$$\mathbf{X}^{\top}\mathbf{Y} = \begin{pmatrix} 11756.7 \\ 291153.33 \\ 32874.95 \end{pmatrix}.$$

We can now compute $\hat{\boldsymbol{\beta}}$:

```
(beta.hat = inv(X.t.X) %*% X.t.Y)
```

Thus,

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\top}\mathbf{X})^{-1}\mathbf{X}^{\top}\mathbf{Y} = \begin{pmatrix} 79.677 \\ -0.276 \\ -0.443 \end{pmatrix}.$$

We have seen that the fitted surface is

$$y^* = \hat{f}(\mathbf{x}^*) = 79.677 - 0.276x_1^* - 0.443x_2^*$$

for an observation $\mathbf{x}^* = (x_1^*, x_2^*)$.

Warning: predictions should not be made for observations outside the range (or the envelope) of the training predictors. In this example, the predictor envelope is shown in red in the figure below – one should resist the temptation to predict y^* for $\mathbf{x}^* = (100, 2)$, say.

Least Squares Assumptions Since the family of OLS learners is a subset of all possible learners, the best we can say about $\hat{f}_{\text{OLS}}$ is that

$$\text{MSE}_{\text{Te}}(\hat{f}_{\text{OLS}}) \geq \min_{\hat{f}} \left\{\text{MSE}_{\text{Te}}(\hat{f})\right\} \geq \text{Var}(\varepsilon).$$

In practice, we are free to approximate f with **any** learner $\hat{f}$. If we want $\hat{f}$ to be useful, however, we need to verify that it is a "decent" approximation.

There is another trade-off at play: when we restrict learners to specific families of functions,[20] we typically also introduce a series of assumptions on the data.

20: That is, when we impose structure on the learners.

The OLS assumptions are

- **linearity:** the response variable is a linear combination of the predictors;
- **homoscedasticity:** the error variance is constant for all predictor levels;

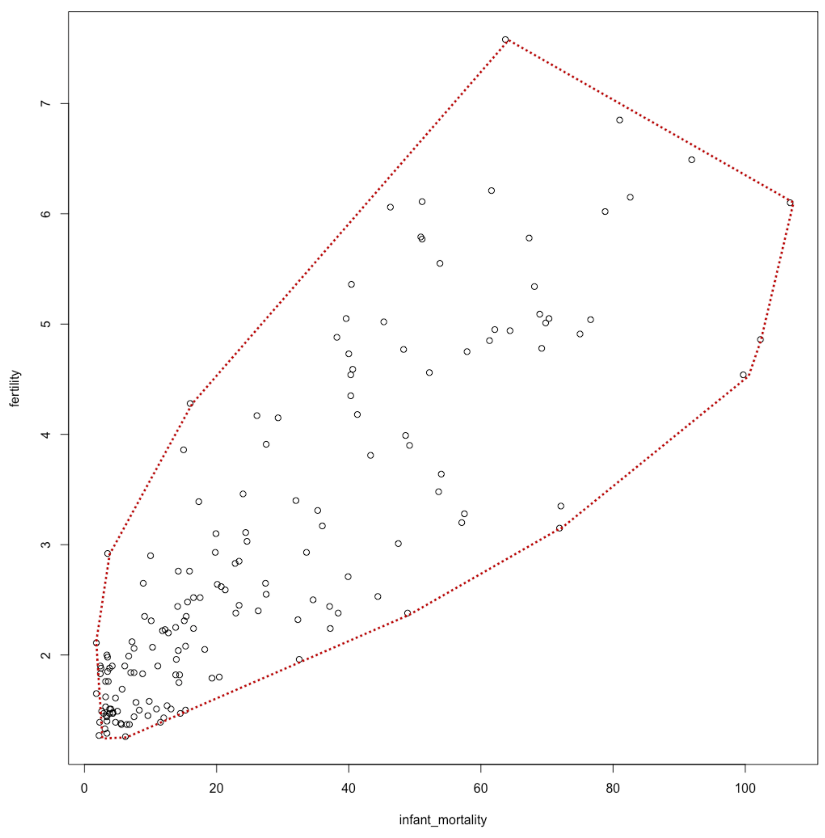


Figure 20.7: Predictor envelope for the Gapminder subset.

- **uncorrelated errors:** the error is uncorrelated from one observation to the next;
- **full column rank for design matrix X:** the predictors are not perfectly multi-collinear;
- **weak exogeneity:** predictor values are free of measurement error.

Mathematically, the assumptions translate to

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where $\boldsymbol{\beta} \in \mathbb{R}^{p+1}$ is determined on a training set Tr without measurement error, and for which

$$\mathrm{E}[\boldsymbol{\varepsilon} \mid \mathbf{X}] = \mathbf{0} \quad \text{and} \quad \mathrm{E}[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top \mid \mathbf{X}] = \sigma^2 I_n.$$

Although it is not a requirement, it is also often further assumed that

$$\boldsymbol{\varepsilon} \mid \mathbf{X} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_n).$$

We will discuss how these assumptions can be generalized at a later stage. In the meantime, however, how can we determine if the choice of model is valid? In the traditional statistical analysis context, there is a number of tests available to the analyst (we will discuss them shortly). In the machine learning context, there is only one real test:

does the model make good predictions?

20.2.2 Least Squares Properties

Let us assume that the OLS assumptions are satisfied. What can we say about the linear regression results? (see Chapter 8 and [43], say, for a refresher).

For the Gapminder example above, for instance, we could us R's `lm()`.

```
f.model = lm(life_expectancy~infant_mortality+fertility)
summary(f.model)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-15.3233  -2.0057   0.2003   2.9570  10.6370

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)       79.6759     0.7985  99.786   <2e-16 ***
infant_mortality  -0.2763     0.0248 -11.138   <2e-16 ***
fertility         -0.4440     0.4131  -1.075    0.284
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.172 on 163 degrees of freedom
Multiple R-squared:  0.7612,    Adjusted R-squared:  0.7583
F-statistic: 259.8 on 2 and 163 DF,  p-value: < 2.2e-16
```

Coefficient of Determination Let

$$\text{SSE} = \mathbf{Y}^\top[I_n - \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top]\mathbf{Y} = \mathbf{Y}^\top[I_n - \mathbf{H}]\mathbf{Y}$$

and

$$\text{SST} = \mathbf{Y}^\top\mathbf{Y} - n\overline{y}^2.$$

In the Gapminder example, we have:

```
(SSE=anova(f.model)[[2]][3])
```

```
[1] 2837.69
```

```
(SST=as.vector(t(as.matrix(resp.Y)) %*% as.matrix(resp.Y)
     - nrow(as.matrix(resp.Y))*(mean(resp.Y))^2))
```

```
[1] 11882.18
```

The coefficient of determination of the OLS regression is the quotient

$$R^2 = \frac{\text{SST} - \text{SSE}}{\text{SST}} = \frac{\text{Cov}^2(\mathbf{Y}, \mathbf{X}\hat{\boldsymbol{\beta}})}{\sigma_y^2\sigma_{\hat{y}}^2}.$$

In the Gapminder example, we have:

```
(R.2 = 1-SSE/SST)
```

```
[1] 0.761181
```

The coefficient of determination identifies the proportion of the variation of the data explained by the linear regression; as such, $0 \leq R^2 \leq 1$.

If $R^2 \approx 0$, then the predictor variables have little explanatory power on the response; if $R^2 \approx 1$, then the linear fit is deemed to be "good", as a lot of the variability in the response is explained by the predictors. In practice, the number of predictors also affects the goodness-of-fit (this is related to the curse of dimensionality discussed previously).

The quantity

$$R_a^2 = 1 - \frac{N-1}{N-p}(1-R^2) = 1 - \frac{\text{SSE}/(N-p)}{\text{SST}/(N-1)}$$

is the **adjusted coefficient of determination** of the linear regression. While R_a^2 can be negative, it is always smaller than R^2. It also plays a role in the **feature selection** process.

In the Gapminder example, we have:

```
(R.a.2 = 1-(nrow(as.matrix(resp.Y))-1)
     /(nrow(as.matrix(resp.Y))-nrow(X.t.X))*(1-R.2))
```

```
[1] 0.7584
```

This suggests that a fair proportion of the variability in the life expectancy (about 75.7%) is explained by infant mortality and fertility.

Significance of Regression We can determine if at least one of the predictors $X_1, \ldots, X_{p-1}$ is useful in predicting the response Y by pitting

$$H_0 : (\beta_1, \ldots, \beta_{p-1}) = \mathbf{0} \quad \text{against} \quad H_1 : (\beta_1, \ldots, \beta_{p-1}) \neq \mathbf{0}.$$

Under the null hypothesis H_0, the $F-$statistic

$$F^* = \frac{(\text{SST} - \text{SSE})/p}{\text{SSE}/(N-p)} \sim F_{p,N-p}.$$

At significance level α, if $F^* \geq F_{p,N-p;\alpha}$ (the $1-\alpha$ quantile of the F distribution with p and $N-p$ degrees of freedom), then we reject the null hypothesis in favour of the alternative.

In the Gapminder model

$$Y = 79.677 - 0.276X_1 - 0.443X_2 + \varepsilon, \quad N = 166, p = 2,$$

we have:

```
(F.star = ((SST-SSE)/(nrow(X.t.X)))
       /(SSE/(nrow(as.matrix(resp.Y))-nrow(X.t.X))))
```

```
[1] 258.169
```

At a significance level $\alpha = 0.05$, the critical value of the $F_{2,164}$ distribution is:

```
qf(0.05,nrow(X.t.X),nrow(as.matrix(resp.Y))
      -nrow(X.t.X),lower.tail=FALSE)
```

```
[1] 3.051127
```

Since $F^* \geq F_{2,164;0.05}$, at least one of $\beta_1, \beta_2 \neq 0$, with probability 95% (in the frequentist interpretation).

Interpretation of the Coefficients For $j = 1, \ldots, p$, the coefficient β_j is the **average** effect on Y of a 1-unit increase in X_j, **holding all other predictors fixed**. Ideally, the predictors are uncorrelated (such as would be the case in a **balanced design** [42]). Each coefficient can then be tested (and estimated) separately, and the above interpretation is at least reasonable in theory.

In practice, however, we can not always control the predictor variables, and it might be impossible to "hold all other predictors fixed." When the predictors are correlated, there are potential **variance inflation** issues for the estimated regression coefficients, and the interpretation is risky, since when X_j changes, so do the other predictors.[21] More importantly, the interpretation can also be read as a claim of causality, which **should be avoided** when dealing with observational data.

21: If Y represents the total monetary value in a piggy bank, X_1 the number of coins, and X_2 the number of pennies, what is likely to be the sign of β_2 in the model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$? Are X_1 and X_2 correlated? What would the interpretation look like, in this case?

> "The only way to find out what will happen when a complex system is disturbed is to disturb the system, not merely to observe it passively." (paraphrased from [52])

In the Gapminder example, the correlation between X_1 and X_2 is:

```
cor(infant_mortality,life_expectancy)
```

```
[1] -0.8714863
```

The predictors are thus strongly correlated, and the standard interpretation is not available to us.

Hypothesis Testing We can also determine if a specific predictor X_j is useful in predicting the response Y, by testing for

$$H_0 : \beta_j = 0 \quad \text{against} \quad H_1 : \beta_j \neq 0.$$

Under the null hypothesis H_0, the test statistic

$$t^* = \frac{\hat{\beta}_j}{\text{se}(\hat{\beta}_j)} \sim T_{N-2},$$

where $\text{se}(\hat{\beta}_j) = \sqrt{\hat{\sigma}^2(\mathbf{X}^\top\mathbf{X})^{-1}_{j+1,j+1}}$, and $\hat{\sigma}^2 = \frac{\text{SSE}}{N-p}$, and T_{n-2} is the Student T distribution with $N-2$ degrees of freedom.

At a significance level α, if $|t^*| \geq |t_{n-2;\alpha/2}|$ (the $1-\alpha/2$ quantile of the T distribution with $N-2$ degrees of freedom), then we reject the null hypothesis in favour of the alternative.

In the Gapminder model, we have: $N = 166$, $p = 2$, and $\hat{\beta}_1 = -0.276$ so that

```
(sigma.hat.2=SSE/(nrow(as.matrix(resp.Y))
         -nrow(X.t.X)))
```

```
[1] 17.51661
```

```
(se.beta.hat.1=sqrt(sigma.hat.2*inv(X.t.X)[2,2]))
```

```
[1] 0.02488045
```

Thus

```
(t.star=(inv(X.t.X) %*% X.t.Y)[2]/se.beta.hat.1)
```

```
[1] -11.08275
```

At a significance level $\alpha = 0.05$, the critical value of the T_{164} distribution is:

```
qt(0.025,nrow(as.matrix(resp.Y))-2)
```

```
[1] -1.974535
```

Since $|t^*| \geq |t_{164;0.025}|$, $\beta_1 \neq 0$ with probability 95% (in the frequentist interpretation).

Confidence Intervals The **standard error** of $\hat{\beta}_j$ reflects how the estimate would vary under various Tr; it can be used to compute a $(1-\alpha)\%$ **confidence interval** for the true β_j:

$$\text{CI}(\beta_j; 1-\alpha) \equiv \hat{\beta}_j \pm z_{\alpha/2} \cdot \text{se}(\hat{\beta}_j);$$

at $\alpha = 0.05$, $z_{\alpha/2} = 1.96 \approx 2$, so that

$$\text{CI}(\beta_j; 0.95) \equiv \hat{\beta}_j \pm 2\text{se}(\hat{\beta}_j).$$

In the Gapminder example, we have

coeff.	**est.**	**s.e.**	**t***	**95% CI**
β_0	79.677	0.7985	99.786	$[78.1, 81.3,]$
β_1	−0.276	0.0248	−11.138	$[-0.33, -0.23]$
β_2	0.443	0.4131	−1.075	$[-1.27, 0.38]$

In **frequentist** statistics, the confidence interval has a particular interpretation – it does not mean, as one might wish, that there is a 95% chance, say, that the true β_j is found in the CI; rather, it suggests that the approach used to build the 95% CI will yield an interval in which the true β_j will reside approximately 95% of the time.[22]

22: Compare with the Bayesian notion of a **credible interval** (see Chapter 26).

The resulting confidence intervals also depend on the underlying model. For instance, the 95% CI for β_1 in the **full model** is $[-0.33, -0.23]$ (see above), whereas the corresponding CI in the **reduced model**

$$\hat{Y} = \gamma_0 + \gamma_1 X_1$$

is $[-0.33, -0.27]$.

The estimates are necessarily distinct as well:

```
reduced.model = lm(life_expectancy ~ infant_mortality)
summary(reduced.model)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-14.9729  -1.9716   0.1726   2.9727  11.0275

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)      78.99279    0.48357  163.35   <2e-16 ***
infant_mortality -0.29888    0.01313  -22.76   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.174 on 164 degrees of freedom
Multiple R-squared:  0.7595,    Adjusted R-squared:  0.758
F-statistic: 517.9 on 1 and 164 DF,  p-value: < 2.2e-16
```

Note that $\hat{\beta}_1 = -0.2763 \neq -0.2989 = \hat{\gamma}_1$.

Feature Selection How would we determine if all the predictors help explain the response Y, or if only a (proper) subset of the predictors is needed? The most direct approach to solve this problem (in the linear regression context) is to run **best subsets** regression.

The procedure is as follows: fit an OLS model for all possible subsets of predictors and select the **optimal model** based on a criterion that balances **training error** with **model size**.

There are 2^{p+1} such models (a quantity that quickly becomes unmanageable). In practice, we need to automate and speed-up the search through a collection of predictor subsets. OLS approaches include **forward selection** and **backward selection** (we discuss these in detail in Chapter 23, *Feature Selection and Dimension Reduction*).

Forward selection is a bottom-up approach:

1. start with the **null model** $\mathcal{M}_0 : Y = \beta_0 + \varepsilon$;
2. fit p simple linear regressions $Y = \beta_0 + \beta_j X_j + \varepsilon$ and add to the null model the predictor X_{j_1} resulting in the lowest SSE:

$$\mathcal{M}_1 : Y = \beta_0 + \beta_{j_1} X_{j_1} + \varepsilon;$$

3. add to that model the predictor X_{j_2} that results in the lowest SSE among all the two-variable models:

$$\mathcal{M}_2 : Y = \beta_0 + \beta_{j_1} X_{j_1} + \beta_{j_2} X_{j_2} + \varepsilon;$$

4. the process continues until a stopping criterion is met.

Backward selection is a top-down approach, and it works in reverse, removing predictors from the full model.

In both approaches, there are at most

$$p + (p-1) + \cdots + 2 + 1 = \frac{p(p+1)}{2} \ll 2^{p+1} \text{ (when } p \text{ is large)}$$

regressions to run. These methods are, frankly, not ideal in the machine learning framework (we will shortly see alternatives).

Other Questions

- How do we handle qualitative variables? (dummy binary variables);
- How do we handle interaction terms? (add features);
- How do we handle outliers? (median regression, Theil-Sen estimate);
- How do we handle non-constant variance of error terms? (data transformations, weighted least square regression, Bayesian regression);
- How do we handle high-leverage observations? (robust regression);
- How do we handle collinearity? (principal component analysis, generalized linear models, partial least square regression);
- How do we handle multiple tests? (Bonferroni correction: for q independent tests with the same data, set significance level to α/q to get joint significance equivalent to α for a single test).

20.2.3 Generalizations of OLS

The OLS assumptions are convenient from a mathematical perspective, but they are not always met in practice.

One way out of this problem is to use **remedial measures** to transform the data into a compliant set; another one is to extend the assumptions and to work out the corresponding mathematical formalism:

- **generalized linear models** (GLM) implement responses with non-normal conditional distributions;
- **classifiers** (logistic regression, decision trees, support vector machines, naïve Bayes, neural networks) extend regression to categorical responses (see Chapter 21);
- **non-linear methods** such as splines, generalized additive models (GAM), nearest neighbour methods, kernel smoothing methods are used for responses that are not linear combinations of the predictors (see Section 20.5);
- **tree-based methods** and **ensemble learning methods** (bagging, random forests, boosting) are used for predictor interactions (see Chapter 21);
- **regularization methods** (ridge regression, LASSO, elastic net) facilitate the process of model selection and feature selection (see the subsection on *Shrinkage Methods*).

Generalized Linear Models GLM extend the OLS paradigm[23] by accommodating response variables with **non-normal** conditional distributions. Apart from the error structure, a GLM is essentially a linear model:

23: Ordinary least squares.

$$Y_i \sim \mathcal{D}(\mu_i), \quad \text{where} \quad g(\mu_i) = \mathbf{x}_i^\top \boldsymbol{\beta}.$$

A GLM consists of:

- a systematic component $\mathbf{x}_i^\top \boldsymbol{\beta}$;
- a random component specified by the distribution $\mathcal{D}$ for Y_i, and
- a link function g.

The **systematic component** is specified in terms of the linear predictor for the i^{th} observation $\eta_i = \mathbf{x}_i^\top \boldsymbol{\beta}$; the general ideas and concepts of OLS carry over to GLM, with the added presence of the link function and the distribution of the response y_i.

In principle, the link function g could be any function linking the linear predictor η_i to the distribution of the response Y_i; in practice, however, g should be **smooth** and **monotonic**.[24]

24: Or at least differentiable and invertible.

We could specify any distribution $\mathcal{D}$ for the response Y_i, but they are usually selected from the **exponential family** of distributions.[25] OLS is an example of GLM, with:

25: These are distributions have probability density functions that satisfy

$$f(\mathbf{x} \mid \vec{\theta}) = h(\mathbf{x}) g(\vec{\theta}) \exp(\vec{\phi}(\vec{\theta}) \cdot \vec{T}(\mathbf{x})).$$

This includes the normal, binomial, Poisson, Gamma distributions, etc. These are all distributions with **conjugate priors** (see Chapter 26).

- systematic component $\eta_i = \mathbf{x}_i^\top \boldsymbol{\beta}$;
- random component $Y_i \sim \mathcal{N}(\mu_i, \sigma^2)$;
- link function $g(\mu) = \mu$.

For a more substantial example, consider the following situation. In the early stages of a rumour spreading, the rate at which new individual learn the information increases exponentially over time. If μ_i is the expected number of people who have heard the rumour on day t_i, a model of the form $\mu_i = \gamma \exp(\delta t_i)$ might be appropriate:

$$\underbrace{\ln(\mu_i)}_{\text{link}} = \ln\gamma + \delta t_i = \beta_0 + \beta_1 t_i = \underbrace{(1, t_i)^{\top}(\beta_0, \beta_1)}_{\text{systematic component}} .$$

Furthermore, since we measure a count of individuals, the Poisson distribution could be a reasonable choice:

$$Y_i \sim \underbrace{\text{Poisson}(\mu_i),}_{\text{random component}} \quad \ln(\mu_i) = (1, t_i)^{\top}(\beta_0, \beta_1).$$

The main advantages of GLM are that:

- there is no need to transform the response Y if it does not follow a normal distribution;
- if the link produces **additive effects**, the assumption of homoscedasticity does not need to be met;
- the choice of the link is separate from the choice of random component, providing modeling flexibility;
- models are still fitted *via* a **maximum likelihood** procedure;
- **inference tools** and **model checks** (Wald ratio test, likelihood ratio test, deviance, residuals, CI, etc.) still apply;
- they are easily implemented (`proc genmod`, `glm()`, etc.), and
- the framework unites various regression modeling approaches (OLS, logistic, Poisson, etc.) under a single umbrella.

20.2.4 Shrinkage Methods

We will discuss the curse of dimensionality (CoD), subset selection, and dimension reduction in Chapter 23. Another approach to dealing with high-dimensionality is provided by the **least absolute shrinkage and selection operator** (LASSO) and its variants.

In what follows, assume that the training set consists of N **centered** and **scaled** observations $\mathbf{x}_i = (x_{i,1}, \cdots, x_{i,p-1})$, with responses y_i.

Let $\hat{\beta}_{\text{OLS},j}$ be the jth OLS coefficient, and set a threshold $\lambda > 0$, whose value depends on the training dataset Tr. Recall that $\hat{\boldsymbol{\beta}}_{\text{OLS}}$ is the exact solution to the OLS problem

$$\hat{\boldsymbol{\beta}}_{\text{OLS}} = \arg\min_{\boldsymbol{\beta}}\{\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2\} = \arg\min_{\boldsymbol{\beta}}\{\text{SSE}\}.$$

In general, **no restrictions** are assumed on the values of the coefficients $\hat{\beta}_{\text{OLS},j}$ – large magnitudes imply that corresponding features **play an important role** in predicting the target. This observation forms the basis of a series of useful OLS variants.

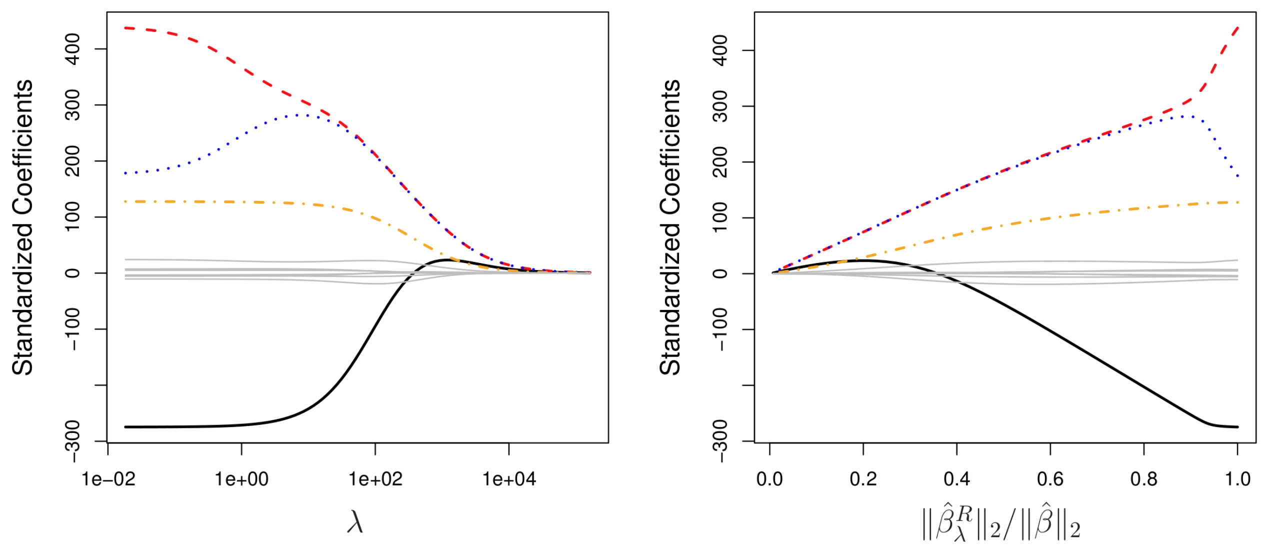


Figure 20.8: Ridge regression coefficients in a generic problem; note how the coefficients converge to 0 when the threshold lambda increases (left); the ratio between the magnitude of the ridge regression parameter and the corresponding OLS parameter is shown on the right [3].

Ridge Regression RR is a method to **regularize** the OLS regression coefficients. Effectively, it shrinks the OLS coefficients by penalizing solutions with large magnitudes – if the magnitude of a specific coefficient is large, then it must have great relevance in predicting the target variable.

This leads to a modified OLS problem:

$$\hat{\boldsymbol{\beta}}_{\text{RR}} = \arg\min_{\boldsymbol{\beta}}\{ \underbrace{\|\mathbf{Y}-\mathbf{X}\boldsymbol{\beta}\|_2^2}_{\text{SSE}} + \underbrace{N\lambda\|\boldsymbol{\beta}\|_2^2}_{\text{shrinkage penalty}} \}.$$

This quantity is small when SSE is small (i.e., the model is a good fit to the data) and when the **shrinkage penalty** is small (i.e., when each β_j is small). RR solutions are typically obtained *via* numerical methods.[26]

26: For **orthonormal covariates** (which is to say, $\mathbf{X}^\top\mathbf{X} = I_p$), we have, in fact:

$$\hat{\beta}_{\text{RR},j} = \frac{\hat{\beta}_{\text{OLS},j}}{1+N\lambda}.$$

The hyperparameter λ controls the relative impact of both components. If λ is small, then the shrinkage penalty is small even if the individual coefficients β_j are large; if λ is large, then the shrinkage penalty is only small when all coefficients β_j are small (see Figure 20.8).

Setting the "right" value for λ is crucial; it can be done via cross-validation (see [3, pp.227-228] and Section 20.3 (*Cross-Validation*) for details). The OLS estimates are **equivariant**: if $\hat{\beta}_j$ is the estimate for the coefficient β_j of X_j, then $\hat{\beta}_j/c$ is the estimate for the coefficient of the scaled variable cX_j. RR coefficients do not have this property, however, which is why the dataset must be centered and scaled to start with.

Finally, note that RR estimates help to mitigate the bias-variance trade-off and reduce issues related to overfitting.[27]

27: Even if they do not reduce the dimensions of the dataset.

Regression With Best Subset Selection BS runs on the same principle but uses a different penalty term, which effectively sets some of the coefficients to 0 (this could be used to select the features with non-zero coefficients, potentially).

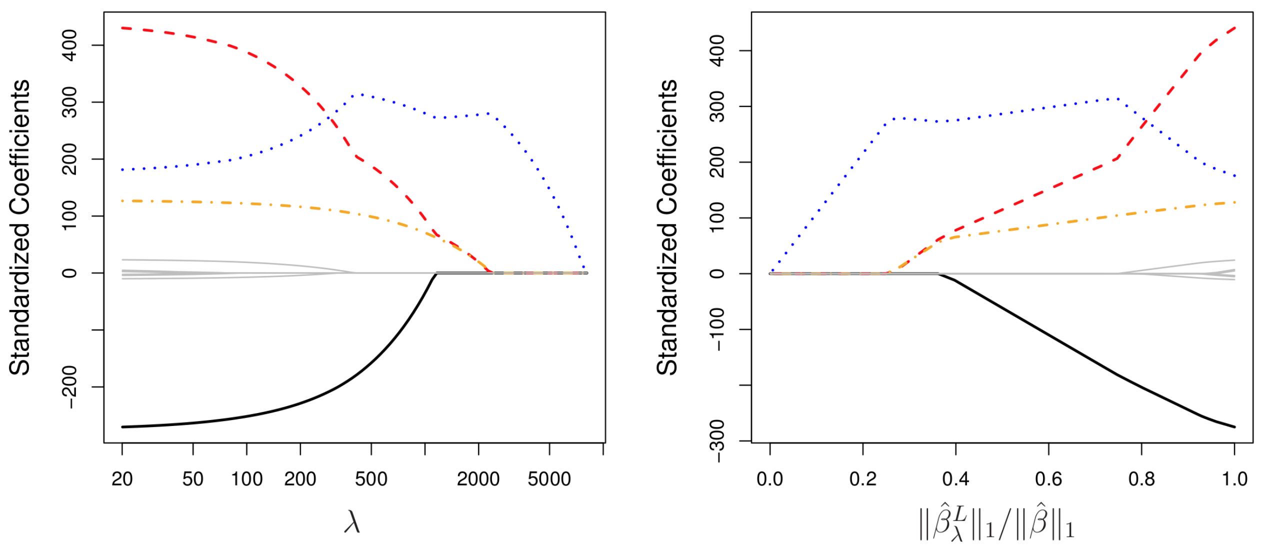


Figure 20.9: LASSO coefficients in a generic problem; note how the coefficients goes directly to 0 after a certain threshold lambda (left); the ratio between the magnitude of the LASSO parameter and the corresponding OLS parameter is shown on the right [3].

The problem consists in solving another modified version of the OLS scenario, namely

$$\hat{\boldsymbol\beta}_{\text{BS}} = \arg\min_{\boldsymbol\beta}\{\underbrace{\|\mathbf{Y}-\mathbf{X}\boldsymbol\beta\|_2^2}_{\text{SSE}} + \underbrace{N\lambda\|\boldsymbol\beta\|_0}_{\text{shrinkage}}\}, \qquad \|\boldsymbol\beta\|_0 = \sum_j \operatorname{sgn}(|\beta_j|).$$

Solving the BS problem typically (also) requires numerical methods and cross-validation.[28] A slight modification to the RR shrinkage penalty can overcome the lack of equivariance.

28: For orthonormal covariates, we have

$$\hat\beta_{\text{BS},j} = \begin{cases} 0 & \text{if } |\hat\beta_{\text{LS,j}}| < \sqrt{N\lambda} \\ \hat\beta_{\text{LS,j}} & \text{if } |\hat\beta_{\text{LS,j}}| \geq \sqrt{N\lambda} \end{cases}$$

LASSO This approach is an alternative to RR obtained by solving

$$\hat{\boldsymbol\beta}_{\text{L}} = \arg\min_{\boldsymbol\beta}\{\underbrace{\|\mathbf{Y}-\mathbf{X}\boldsymbol\beta\|_2^2}_{\text{SSE}} + \underbrace{N\lambda\|\boldsymbol\beta\|_1}_{\text{shrinkage}}\};$$

the penalty effectively forces coefficients which combine the propertiesof RR and BS, selecting at most $\max\{p, N\}$ features, and usually no more than one per group of highly correlated variables (the other coefficients are forced down to 0 when λ is large enough, see Figure 20.9).[29]

29: For orthonormal covariates, we have

$$\hat\beta_{\text{L},j} = \hat\beta_{\text{OLS},j}\cdot\max\left(0, 1-\frac{N\lambda}{|\hat\beta_{\text{OLS},j}|}\right).$$

Why do we get $\hat\beta_{\text{L},j} = 0$ for some j, but not for the RR coefficients? The RR and LASSO formulations are equivalent to

$$\hat{\boldsymbol\beta}_{\text{RR}} = \arg\min_{\boldsymbol\beta}\{\text{SSE} \mid \|\boldsymbol\beta\|_2^2 \leq s\}, \text{ for some } s;$$

$$\hat{\boldsymbol\beta}_{\text{L}} = \arg\min_{\boldsymbol\beta}\{\text{SSE} \mid \|\boldsymbol\beta\|_1 \leq s\}, \text{ for some } s.$$

Graphically, this looks like the images shown in Figure 20.10.

The RR coefficients $\hat{\boldsymbol\beta}_{\text{RR}}$ are found at the first intersection of the ellipses of constant SSE around the OLS coefficient $\hat{\boldsymbol\beta}$ with the 2−sphere $\|\boldsymbol\beta\|_2^2 \leq s$; that intersection is usually away from the axes;[30] this is not usually the case for the intersection of the 1−sphere $\|\boldsymbol\beta\|_1 \leq s$.

30: Due to the lack of "sharp" points.

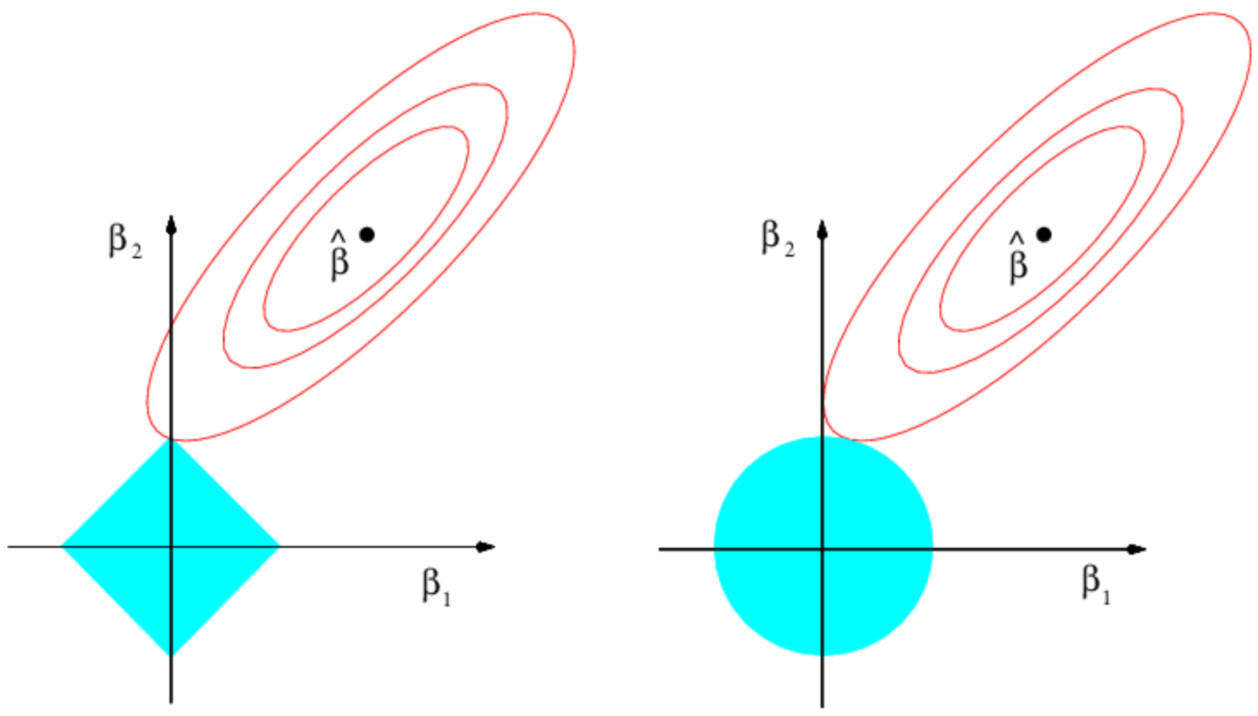


Figure 20.10: Level curves and neighbourhoods for LASSO (left) and ridge regression (right) [3].

The LASSO thus typically produces simpler models, but predictive accuracy matters too (in the form of MSE_{Te}, say).[31]

31: Depending on the data, either of the two approaches can be optimal, thanks to the *No Free Lunch Theorem*.

Generalizations If the response is related to a relatively small number of predictors (whether this is the case or not is not something we usually know *a priori*), LASSO is recommended. The use of other penalty functions (or combinations thereof) provides various extensions, such as: **elastic nets**; **group, fused and adaptive lassos**; **bridge regression**, etc.

The modifications described above were defined assuming an underlying linear regression model, but they generalize to arbitrary regression/classification models as well. For a **loss** (cost) function $\mathscr{L}(\mathbf{Y}, \mathbf{y}(\mathbf{W}))$ between the actual target and the values predicted by the model parameterized by $\mathbf{W}$, and a **penalty** vector $\mathbf{R}(\mathbf{W}) = (R_1(\mathbf{W}), \cdots, R_k(\mathbf{W}))^\top$, the **regularized parametrization $\mathbf{W}^*$** solves the **general regularization** problem

$$\mathbf{W}^* = \arg\min_{\mathbf{W}}\{\mathscr{L}(\mathbf{Y}, \mathbf{y}(\mathbf{W})) + N\lambda^\top \mathbf{R}(\mathbf{W})\},$$

which can be solved numerically, assuming some nice properties on $\mathscr{L}$ and $\mathbf{R}$ [236]; as before, cross-validation can be used to determine the optimal vector λ [2].

Gapminder Example In `R`, regularization is implemented in the package `glmnet` (among others). In `glmnet()` the parameter `alpha` controls the elastic net mixture: LASSO (`alpha = 1`), RR (`alpha = 0`).

Say we are interested in modeling life expectancy Y in the 2011 Gapminder dataset as a function of population, infant mortality, fertility, gdp, and continental membership (we use the entire set as a training set Tr).

A priori, an OLS model on this data would take the form

$$Y = \alpha_0 + \alpha_1\text{population} + \alpha_2\text{infant mortality} + \alpha_3\text{fertility} + \alpha_4\text{gpd} + \alpha_5\text{Africa} + \alpha_6\text{Americas} + \alpha_7\text{Asia} + \alpha_8\text{Europe} + \alpha_9\text{Oceania}.$$

We start by creating dummy variables for the continents:

```
gapminder.2011.f <- fastDummies::dummy_cols(gapminder.2011,
                        select_columns = 'continent')
```

Next, we select the appropriate variables for the response and the training set, and scale and center the data (it must be in a matrix format to be compatible with `glmnet()`:

Setting up the Gapminder dataset

```
library(dplyr)
y <- gapminder.2011.f |> select(life_expectancy) |>
            as.matrix()
x <- gapminder.2011.f |> select(c("population",
            "infant_mortality","fertility","gdp",
            "continent_Africa","continent_Americas",
            "continent_Asia","continent_Europe",
            "continent_Oceania")) |>
            scale(center = TRUE, scale = TRUE) |>
            as.matrix()
```

Finally, we run the regression and extract the LASSO coefficients for hyperparameter $\lambda = 1$:

LASSO coefficients

```
glmnet1 <- glmnet::glmnet(x=x, y=y, type.measure='mse', alpha=1)
(c1 <- coef(glmnet1, x=xx, y=y,s=1,exact=TRUE))
```

```
10 x 1 sparse Matrix of class "dgCMatrix"
                             s1
(Intercept)         70.82349398
population            .
infant_mortality    -5.57897055
fertility             .
gdp                   .
continent_Africa    -1.13074639
continent_Americas    .
continent_Asia        .
continent_Europe      .
continent_Oceania   -0.03096299
```

Thus

$$Y = 70.82 - 5.58(\text{infant mortality}) - 1.13(\text{Africa}) - 0.03(\text{Oceania}).$$

For RR ($\alpha = 0$), we obtain, with the same hyperparameter $\lambda = 1$:

Ridge regression

```
glmnet0 <- glmnet::glmnet(x=x, y=y, type.measure='mse', alpha=0)
(c0 <- coef(glmnet0, x=xx, y=y,s=1,exact=TRUE))
```

```
10 x 1 sparse Matrix of class "dgCMatrix"
                           s1
(Intercept)        70.8234940
population         -0.3471671
infant_mortality   -4.4002779
fertility          -0.6348077
gdp                 0.5803223
continent_Africa   -1.6275714
continent_Americas  0.5475769
continent_Asia      0.6117358
continent_Europe    1.0141934
continent_Oceania  -0.6855980
```

which is to say:

$$Y = 70.82 - 0.34(\text{population}) - 4.4(\text{infant mortality}) - 0.63(\text{fertility}) + 0.58(\text{gdp})$$
$$- 1.62(\text{Africa}) + 0.55(\text{Americas}) + 0.61(\text{Asia}) + 1.01(\text{Europe}) - 0.68(\text{Oceania}),$$

which is compatible with the above discussion.

The coefficient values themselves are not as important as their signs and the fact that they are roughly similar in both models.

It is important to note, however, that the choice of $\lambda = 1$ was arbitrary, and that we have not been evaluating the result on test data Te. We will revisit these issues in Section 20.3 (*Cross-Validation*).

20.3 Resampling Methods

How do we determine the variability of a regression fit? It can be done by drawing different samples from the available data, fitting a regression model to each sample, and then examining the extent to which the various fits differ from one another.

Resampling methods provide additional information about a fitted model, by applying the same fitting approach to various sub-samples of the training set Tr. We will consider three such methods:

- **cross-validation**, which estimates the test error associated with a modeling approach in order to evaluate model performance;
- the **bootstrap**, which provides a measure of accuracy, standard deviation, bias, etc. of various model parameter estimates, and
- the **jackknife**, which is a simpler approach with the same aims as the bootstrap.

The **test error** associated with a statistical learning model is the average error arising when predicting the response for observations that were not used to train the model.

The **training error**, on the other hand, is computed directly by comparing the model's predictions to the actual responses in Tr. In general, the training error underestimates the test error, dramatically so when the model complexity increases (see **variance-bias trade-off**, Figure 20.5).

A possible way out of this conundrum is to set aside a large-enough testing set Te, but that's not always possible if the original dataset is not that large in the first place.[32]

32: Some methods make direct adjustments to the training error rate in order to estimate the test error (e.g., Mallow's C_p statistic, R_a^2, AIC, BIC, etc.)

In the statistical learning framework, we estimate the test error by **holding a subset** Va $\subseteq$ Tr **out** from the fitting process (which takes place on Tr $\setminus$ Va). The **validation approach** is a simple strategy that is used to estimate the test error associated with a particular statistical model on a set of observations.

Formally, the latter is split into a **training set** Tr and a **validation set** Va (the hold-out set). The model is fit on the training set; the fitted model is used to make predictions on the validation set. The resulting validation set error provides an estimate for the test error.

This approach is easy to implement and interpret, but it has a number of drawbacks, most importantly:

- the validation error is highly dependent on the choice of the validation set, and is thus quite **volatile**;
- the model is fitted on a proper subset of the available observations, and we might expect that this would lead to **the validation error being larger than the test error in general**, and
- a number of classical statistical models can provide test error estimates without having to resort to the validation set approach.

20.3.1 Cross-Validation

K-**fold cross-validation** is a widely-used approach to estimate the test error without losing some observations to a hold-out set.[33]

33: It can also provide a basis for model selection.

The procedure is simple:

1. Divide the dataset **randomly** into K (roughly) equal-sized **folds** (typically, $K = 4, 5, 10$).
2. Each fold plays, in succession, the role of the **validation set**. If there are N observations in the dataset, partition
$$\{1,\ldots,N\} = \underbrace{\mathcal{C}_1}_{\text{fold } 1} \sqcup \cdots \sqcup \underbrace{\mathcal{C}_K}_{\text{fold } K}.$$
If $|\mathcal{C}_k| = n_k$, we expect $n_k \approx \frac{N}{K}$ for all $k = 1,\ldots,K$.
3. For all $k = 1,\ldots,K$, fit a model on observations $\{1,\ldots,N\} \setminus \mathcal{C}_k$ and denote the error on $\mathcal{C}_k$ by E_k.[34]
4. Write $\overline{E}$ for the average of the E_k.

34: For a regression model, there are many options but we typically use
$$E_k = \sum_{i\in\mathcal{C}_k} \frac{(y_i - \hat{y}_i)^2}{n_k}.$$

5. The **cross-validation estimate** of the test error is

$$\mathrm{CV}_{(K)}=\sum_{k=1}^{K}\frac{n_k}{N}E_k,$$

with standard error

$$\widehat{\mathrm{se}}\left(\mathrm{CV}_{(K)}\right)=\sqrt{\frac{1}{K-1}\sum_{k=1}^{K}(E_k-\overline{E})^2}.$$

These steps could also be **replicated** n times to generate a distribution of an evaluation metric, such as the standard error.[35]

35: See Figure 19.33 for an illustration.

The resulting mean can prove useful in order to determine how well a statistical learning procedure will perform on unseen data. If, however, we are interested in selecting a method from a list of methods, or a flexibility level among a family of approaches, we do not care about the specific value of $\mathrm{CV}_{(K)}$ so much as where it is minimized.[36]

36: The estimate is usually biased, anyway.

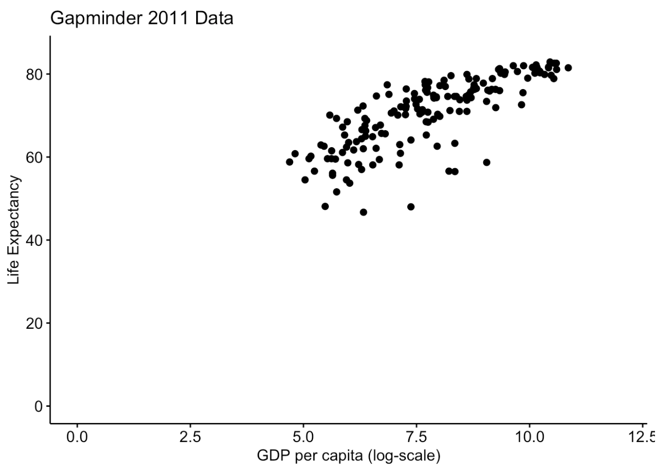
From the perspective of **bias reduction** (in the estimate for the test error), the best choice is $K = N$, but this is mitigated by the variance-bias trade-off. With $K = N$, we have N models and N estimates for the test error, but these estimates are **highly correlated** and the mean of highly correlated estimates has **high variance**.[37]

37: See Section 20.3, *Jackknife*, for details.

Gapminder Example We use cross-validation in the `Gapminder` dataset to estimate the test error $\mathrm{MSE}_{\mathrm{Te}}$ when predicting life expectancy as a regression against the logarithm of the GDP per capita for the 2011 data.

Gapminder subset

```
gapminder.2011.cv <- gapminder.2011 |>
  dplyr::mutate(lgdppc = log(gdp/population)) |>
  select(life_expectancy,lgdppc)

ggpubr::ggscatter(gapminder.2011.cv, x="lgdppc",
        y="life_expectancy", palette="jco", size = 2,
        xlab="GDP per capita (log-scale)", xlim=c(0,12),
        ylab = "Life Expectancy",  ylim=c(0,85),
        title = "Gapminder 2011 Data")
```



We split the dataset into $K = 10$ random folds, each containing 16 or 17 observations, and fit 10 linear regression models using the 149 or 150 remaining observations.[38]

38: Note that the estimates for β_0, β_1, and MSE_{Te} are likely to be correlated from one fold to the next, since the respective training sets share a fair number of observations.

The indices for each of the folds are computed below:

Setting-up the folds

```
set.seed(0) # for replicability
true.order = sample.int(nrow(gapminder.2011.cv),
        nrow(gapminder.2011.cv),replace=FALSE)

index=list()
for(k in 1:6){
  index[[k]] = true.order[((k-1)*17+1):(k*17)]
}
for(k in 7:10){
  index[[k]] = true.order[(102+(k-6-1)*16+1):(k*16+6)]
}
```

Each fold is used, in turn, as a testing set while the remaining folds form the training set. We fit an OLS model on each training set, and evaluate the MSE performance of the model on the appropriate fold testing set.

Compute the test MSE for each fold

```
training.gap = list()
testing.gap = list()
model.lm.gap = list()
pred.lm.gap = list()
beta.0 = c()
beta.1 = c()
MSE.cv = c()
n.row = c()

for(k in 1:10){
  n.row[k]=length(index[[k]])
  training.gap[[k]] = gapminder.2011.cv[-index[[k]],]
  testing.gap[[k]] = gapminder.2011.cv[index[[k]],]
  model.lm.gap[[k]] = lm(life_expectancy~lgdppc,
            data=training.gap[[k]])
  beta.0[k] = model.lm.gap[[k]][[1]][1]
  beta.1[k] = model.lm.gap[[k]][[1]][2]
  pred.lm.gap[[k]] = predict(model.lm.gap[[k]],
            newdata=testing.gap[[k]])
  tmp = data.frame(pred.lm.gap[[k]],testing.gap[[k]][1])
  MSE.cv[k] = 1/nrow(tmp)*sum((tmp[,1]-tmp[,2])^2)
}
```

The number of observations in each fold, as well as the regression parameters and the MSE on each fold testing set are shown below:

```
(results = data.frame(n.row,beta.0,beta.1,MSE.cv))
```

n.row	beta.0	beta.1	MSE.cv
17	37.69812	4.254105	33.859051
17	36.22257	4.442061	21.415376
17	37.59386	4.247255	45.620933
17	36.66761	4.345484	29.584469
17	37.49685	4.268917	24.127300
17	36.49849	4.386124	19.398769
16	36.78991	4.380887	48.157391
16	36.91113	4.331365	23.142625
16	37.41767	4.274771	7.837172
16	37.68955	4.254600	19.743046

The 10−fold cross-validation estimate of MSE_{Te} is thus

$$\overline{\text{MSE}_{\text{Te}}} = \frac{1}{10}\sum_{k=1}^{10}\text{MSE}_{\text{Te}_k} = 27.29;$$

$$\text{CV}_{(K)} = \sum_{k=1}^{10}\frac{n_k}{166}\text{MSE}_{\text{Te}_k} = 27.35;$$

$$\widehat{\text{se}}\left(\text{CV}_{(K)}\right) = \sqrt{\frac{1}{10-1}\sum_{k=1}^{10}(\text{MSE}_{\text{Te}_k} - \overline{\text{MSE}_{\text{Te}}})^2} = 12.38;$$

these can be computed as below.

CV results

```
mean.MSE = mean(results$MSE.cv)
cv.k = sum(results$n.row*results$MSE.cv/sum(results$n.row))
se.cv.k = sqrt(1/(nrow(results)-1)*sum((results$MSE.cv-
                                mean.MSE)^2))
```

Thus, $27.35 \pm 2(12.38) \equiv (2.59, 52.11)$ is a 95% CI for the MSE_{Te}.

We can also get 10−fold cross-validation estimates of β_0, β_1: we have

$$\beta_{0_{(K)}} = \sum_{k=1}^{10}\frac{n_k}{166}\beta_{0;k} = 37.10$$

$$\widehat{\text{se}}\left(\beta_{0_{(K)}}\right) = \sqrt{\frac{1}{10-1}\sum_{k=1}^{10}(\beta_{0;k} - \overline{\beta_0})^2} = 0.54,$$

so $\text{CI}(\beta_0; 0.95) \equiv 37.10 \pm 2(0.54) \equiv (36.00, 38.18)$ and

$$\beta_{1_{(K)}} = \sum_{k=1}^{10}\frac{n_k}{166}\beta_{1;k} = 4.32$$

$$\widehat{\text{se}}\left(\beta_{1_{(K)}}\right) = \sqrt{\frac{1}{10-1}\sum_{k=1}^{10}(\beta_{1;k} - \overline{\beta_1})^2} = 0.07,$$

so $\text{CI}(\beta_1; 0.95) \equiv 4.32 \pm 2(0.07) \equiv (4.18, 4.56)$, as computed below.

CV estimates for the regression coefficients

```
mean.beta_0 = mean(results$beta_0)
cv.beta_0.k = sum(results$n.row*results$beta_0/
            sum(results$n.row))
se.cv.beta_0.k = sqrt(1/(nrow(results)-1)*sum(
                (results$beta_0-mean.beta_0)^2))

mean.beta_1 = mean(results$beta_1)
cv.beta_1.k = sum(results$n.row*results$beta_1/
            sum(results$n.row))
se.cv.beta_1.k = sqrt(1/(nrow(results)-1)*sum(
                (results$beta_1-mean.beta_1)^2))
```

LASSO and Regression Ridge Revisited How would we pick the optimal hyperparameter λ in shrinkage regressions? Let us revisit the example from Section 20.2 (*Shrinkage Methods*).

As before, we are interested in modeling life expectancy Y in the 2011 Gapminder dataset as a function of population, infant mortality, fertility, gdp, and continental membership.[39] We run a 5-fold cross-validation LASSO regression for a variety of hyperparameter values λ, and evaluate the CV test error for each λ using MSE. The optimal λ is the one that minimizes the CV test error.

39: We use `gapminder.2011.f`, `x`, and `y` as in that section.

Let us start with the LASSO (`alpha=1`):

```
glmnet1 <- glmnet::cv.glmnet(x=x, y=y, type.measure='mse',
            nfolds=5, alpha=1)
(c1 <- coef(glmnet1, s='lambda.min', exact=TRUE))
```

```
                            s1
(Intercept)        70.8234940
population          .
infant_mortality   -5.7375945
fertility           .
gdp                 0.1616446
continent_Africa   -1.7592037
continent_Americas  .
continent_Asia      .
continent_Europe    0.1219114
continent_Oceania  -0.7977736
```

The optimal λ in this case is:

```
(lambda1 = glmnet1$lambda.min)
```

```
[1] 0.3118295
```

We repeat the process for RR (`alpha = 0`):

```
glmnet0 <- glmnet::cv.glmnet(x=x, y=y, type.measure='mse',
            nfolds=5, alpha=0)
(c0 <- coef(glmnet0, s='lambda.min', exact=TRUE))
```

```
                           s1
(Intercept)        70.8234940
population         -0.3466483
infant_mortality   -4.6968992
fertility          -0.4240814
gdp                 0.5813385
continent_Africa   -1.6192452
continent_Americas  0.5467797
continent_Asia      0.6295896
continent_Europe    1.0091460
continent_Oceania  -0.7207190
```

The optimal λ in this case is:

```
(lambda0 = glmnet0$lambda.min)
```

```
[1] 0.7373175
```

Cross-Validation with `Python` Let us take a look at how we could estimate the test error *via* cross-validation manually in `Python`. The following modules will be necessary: `statsmodels` to run linear models (in particular to define **formulas** for linear regression), `numpy` for numerical operations, and `pandas` for data frame manipulations.

Python modules for CV

```
import statsmodels.formula.api as smf
import numpy as np
import pandas as pd
import random
random.seed(0) # for replicability
```

We use the `calculus.csv` dataset from Section 1.6, whose structure is as shown below. We will try to predict students' grades in terms of the other predictors, using linear regression. In particular, we are interested in which model does a better job of predicting the grades.

```
df = pd.read_csv('calculus.csv')
df.head()
```

```
      ID Sex  Grade   GPA  Year
0  10001   F     47  5.02     2
1  10002   M     57  3.82     1
2  10003   M     91  7.70     1
3  10004   M     71  4.82     1
4  10005   F     83  7.91     1
```

We start by obtaining a random permutation of the observations (the `pandas` method `iloc()` selects values for specified indices).

```
nrows = len(df)
permuted = df.iloc[np.random.permutation(nrows)]
permuted.head()
```

```
       ID Sex  Grade    GPA  Year
60  10061   F     97  11.45     2
61  10062   M     70   3.65     1
28  10029   M     98  11.90     1
49  10050   F     92  11.05     1
50  10051   M     79   6.87     2
```

In this example, we separate the sample indices into $k = 5$ folds for cross-validation using the `numpy` function `array_split()`.

```
k = 5
chunks = np.array_split(range(nrows), k)
```

We iterate over each fold as a **test set** while using the remaining folds as a **training set**.

Say `chunk[i]` is the current test set; we can obtain the corresponding training set as follows:

```
training = permuted.iloc[ np.concatenate( [ chunks[j]
                for j in range(k) if j != i]) ]
```

We then perform a linear regression over this training set (with the `statsmodels` methods `ols()` and `fit()`) and compute the MSE over the test set using the predicted values. Remember, this is for a single fold:

```
fit = smf.ols(formula=m, data = training).fit()
test = permuted.iloc[chunks[i]]
pred = fit.predict( test )
testerror = ((pred - test['Grade'])**2).mean()
```

In the chunk of code above, `formula=m` is an R-style formula. In the following, we go through a number of possible formulas, for all folds.

```
f = ['Grade ~ GPA + C(Year) + C(Sex)',
     'Grade ~ GPA + C(Year)',
     'Grade ~ GPA + C(Sex)', 'Grade ~ GPA' ]

for m in f:
    testerror = 0.0
    for i in range(k):
        training = permuted.iloc[ np.concatenate(
            [ chunks[j] for j in range(k) if j != i]) ]
        fit = smf.ols(formula=m, data = training).fit()
        test = permuted.iloc[chunks[i]]
        pred = fit.predict( test )
        testerror += ((pred - test['Grade'])**2).mean()
    testerror /= k
    print(testerror, m)
```

```
118.2165188650409 Grade ~ GPA + C(Year) + C(Sex)
117.4815061224269 Grade ~ GPA + C(Year)
115.77980266850878 Grade ~ GPA + C(Sex)
114.87270405373037 Grade ~ GPA
```

The best model is given by the formula `Grade ~ GPA`.

20.3.2 Bootstrap

The **bootstrap procedure** uses re-sampling of the available data to **mimic the process of obtaining new replicates**, which allows us to estimate the variability of a statistical model parameter of interest **without the need to generate new observations**.

Replicates are obtained by repeatedly sampling observations from the original dataset **with replacement**. A **bootstrap dataset** Tr^* for a training set Tr with N observations is a sample of N such observations, drawn **with replacement**.

The process is repeated M times to obtain bootstrap samples Tr_i^* and parameter estimates $\hat{\alpha}_i^*$, for $i = 1, \ldots, M$, from which we derive a **bootstrap estimate**

$$\hat{\alpha}^* = \frac{1}{M}\sum_{i=1}^{M}\hat{\alpha}_i^*,$$

with standard error

$$\widehat{\text{se}}\,(\hat{\alpha}^*) = \sqrt{\frac{1}{M-1}\sum_{i=1}^{M}(\hat{\alpha}_i^* - \hat{\alpha}^*)^2}.$$

The bootstrap can also be used to build **approximate frequentist confidence intervals** for the parameter α.[40] We can even construct a covariance structure for the parameters, given enough replicates.

40: Note that this is not as straightforward as one might think, so caution is advised.

Finally, it should be noted that in more complex scenarios, the appropriate bootstrap procedure might be more sophisticated than what has been described here.[41]

41: For instance, sampling with replacement at the observation level would not preserve the covariance structure of time series data.

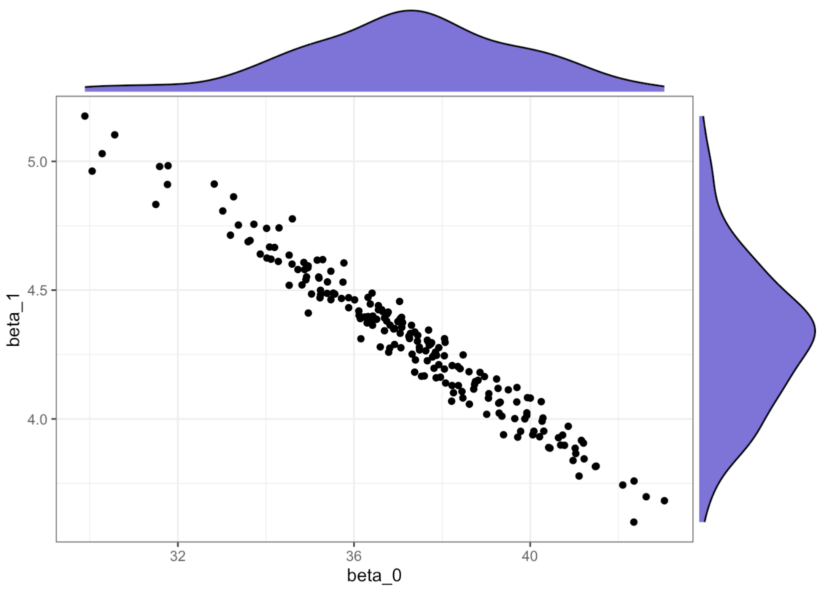
Gapminder Example We use the bootstrap procedure for the regression problem with life expectancy and the log of the GDP per capita in the 2011 Gapminder data.

We draw, with replacement, $M = 200$ bootstrap samples of size $N = 166$ from the original dataset. For each sample $1 \leq i \leq M$, we find the OLS fit and retain the intercept $\beta_{0,i}$ and slope $\beta_{1,i}$.

```
beta_0 = c()
beta_1 = c()

set.seed(0) # for replicability
for(k in 1:200){
  index = sample.int(nrow(gapminder.2011.cv),
        nrow(gapminder.2011.cv),replace=TRUE)
  training.gap = gapminder.2011.cv[-index,]
  model.lm.gap = lm(life_expectancy~lgdppc,
                    data=training.gap)
  beta_0[k] = model.lm.gap[[1]][1]
  beta_1[k] = model.lm.gap[[1]][2]
}

results.boot = data.frame(beta_0,beta_1)
```

We display the joint distribution of $\boldsymbol{\beta} = (\beta_0, \beta_1)^\top$, together with the marginal distributions for each parameter.

```
library(ggplot2)
p <- ggplot(results.boot, aes(x=beta_0, y=beta_1)) +
  geom_point() +
  theme(legend.position="none")
ggExtra::ggMarginal(p, type="density", fill = "slateblue")
```



We see that $\boldsymbol{\beta}$ roughly follows a multivariate normal $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with

$$\boldsymbol{\mu} \approx \hat{\boldsymbol{\mu}}^* = \begin{pmatrix} 37.22 \\ 4.31 \end{pmatrix}, \ \boldsymbol{\Sigma} \approx \hat{\boldsymbol{\Sigma}}^* = \begin{pmatrix} 6.32 & -0.72 \\ -0.72 & 0.08 \end{pmatrix},$$

as computed below:

```
boot.beta_0 = mean(results.boot$beta_0)
boot.beta_1 = mean(results.boot$beta_1)
cov(results.boot)
```

The vector $\hat{\boldsymbol{\mu}}^*$ provides the bootstrap estimates; the corresponding estimates for the standard errors are $\widehat{\text{se}}(\hat{\boldsymbol{\mu}}^*) = (2.51, 0.29)^\top$, and

$$\text{CI}(\beta_0; 0.95) = 37.22 \pm 2(2.51) \equiv (32.19, 42.25),$$
$$\text{CI}(\beta_1; 0.95) = 4.31 \pm 2(0.29) \equiv (3.73, 4.89);$$

the standard errors are computed as below:[42]

42: Note that the bootstrap CI are wider than the corresponding cross-validation CI.

```
se.boot.beta_0 = sqrt(1/(nrow(results.boot)-1)*
  sum((results.boot$beta_0-mean(results.boot$beta_0))^2))
se.boot.beta_1 = sqrt(1/(nrow(results.boot)-1)*
  sum((results.boot$beta_1-mean(results.boot$beta_1))^2))
```

20.3.3 Jackknife

The **jackknife estimator** arises from cross-validation when $K = N$;[43] the sole difference being in the standard error estimate

43: The jackknife procedure is also known as **leave one out validation**.

$$\widehat{\text{se}}(\hat{\alpha}^*) = \sqrt{\frac{N-1}{N}\sum_{i=1}^{N}(\hat{\alpha}_i^* - \overline{\hat{\alpha}^*})^2}.$$

Gapminder Example We use the jackknife procedure on the same task as in the previous section.

For each fold $1 \leq k \leq N$, we find the OLS fit on TR_k and retain the intercept $\beta_{0,k}$ and slope $\beta_{1,k}$. The code is exactly as in the bootstrap case, with one exception: we replace the line

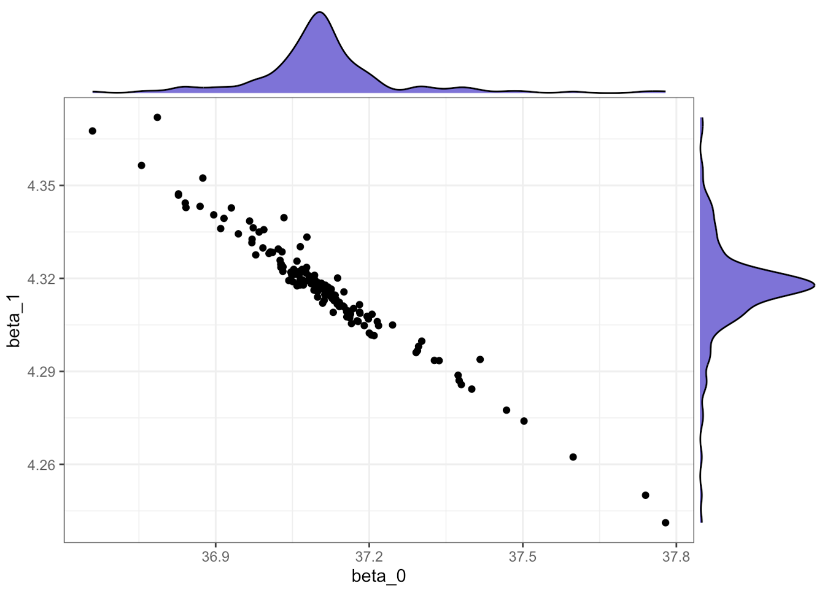
```
  index = sample.int(nrow(gapminder.2011.cv),
        nrow(gapminder.2011.cv),replace=TRUE)
```

by

```
  index = k
```

We display the joint distribution of $\boldsymbol{\beta} = (\beta_0, \beta_1)^\top$, together with the marginal distributions for each parameter.

```
library(ggplot2)
p <- ggplot(results.jack, aes(x=beta_0, y=beta_1)) +
  geom_point() +
  theme(legend.position="none")
ggExtra::ggMarginal(p, type="density", fill = "slateblue")
```



We see that $\boldsymbol{\beta}$ roughly a multivariate normal $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with

$$\boldsymbol{\mu} \approx \hat{\boldsymbol{\mu}}^* = \begin{pmatrix} 37.11 \\ 4.32 \end{pmatrix}, \ \boldsymbol{\Sigma} \approx \hat{\boldsymbol{\Sigma}}^* = \begin{pmatrix} 0.021 & -0.002 \\ -0.002 & 0.0003 \end{pmatrix},$$

as can be computed below:

```
jack.beta_0 = mean(results.jack$beta_0)
jack.beta_1 = mean(results.jack$beta_1)
cov(results.jack)
```

The vector $\hat{\boldsymbol{\mu}}^*$ provides the jackknife estimates; the corresponding estimates for the standard errors are $\widehat{\text{se}}(\hat{\boldsymbol{\mu}}^*) = (1.86, 0.21)^\top$, and

$$\begin{aligned} \text{CI}(\beta_0; 0.95) &= 37.11 \pm 2(1.86) \equiv (33.38, 40.83); \\ \text{CI}(\beta_1; 0.95) &= 4.32 \pm 2(0.21) \equiv (3.890, 4.744). \end{aligned}$$

The standard errors are computed as below:

```
se.jack.beta_0 = sqrt(1/nrow(results.jack)*
  (nrow(results.jack)-1)*sum((results.jack$beta_0-
     mean(results.jack$beta_0))^2))
se.jack.beta_1 = sqrt(1/nrow(results.jack)*
  (nrow(results.jack)-1)*sum((results.jack$beta_1-
    mean(results.jack$beta_1))^2))
```

In this case, the jackknife estimates are tighter than the corresponding bootstrap estimates, but looser than the cross-validation estimates. Will this always be the case?

20.4 Model Selection

A linear model

$$Y = \vec{X}^{\top}\boldsymbol{\beta} + \varepsilon$$

should be seen as an attempt to approximate the regression function

$$y = f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}].$$

But what we gain in convenience of fit (and structure) by using a linear model, we may lose in modeling accuracy.

In this context, we assume a **linear relationship** between the **response** Y and the **predictors** $X_1, \ldots, X_p$, which we (typically) fit using the **(ordinary) least squares** (OLS) framework, which is to say

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}}\{\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2\},$$

for the **response vector Y** and **design matrix X** provided by a **training set** Tr; additional assumptions on the **error components** $\boldsymbol{\varepsilon}$ usually require

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma\mathbf{I}_n),$$

where n represents the number of observations in Tr.

Fundamentally, there are 3 ways in which the OLS framework can be extended:

1. additive but non-linear models (see Section 20.5, *Generalized Additive Models*);
2. non-linear models (see Section 20.5 and Chapter 21), and
3. replacing LS with alternative fitting procedures (see Section 20.2, *Shrinkage Methods*).

The latter approach can produce **better accuracy** than OLS without sacrificing too much in the way of **model interpretability**.[44]

44: In practice, linear models have distinct advantages over more sophisticated models, mainly in the areas of superior interpretability and (frequently) appropriate predictive performances (especially for linearly separable data). These "Old Faithful" models will still be there if fancy deep learning models fail analysts in the future.

But in the OLS framework, prediction accuracy suffers when $p > n$, due to **curse of dimensionality** (see Section 23.2, *Curse of Dimensionality*); model interpretability can be improved by removing **irrelevant features** or by reducing p.

The 3 classes of methods to do so are:

- shrinkage and regularization methods;
- dimension reduction, and
- subset selection/feature selection.

For **shrinkage/regularization methods**, we fit a model involving all p predictors, but the estimated coefficients are shrunk towards 0 relative to the OLS parameter estimates, which has the effect of reducing variance and simultaneously perform variable selection (see Section 20.2, *Shrinkage Methods*).

In **dimension reduction**, we project the p predictors onto a manifold $\mathcal{H}$, with $\dim(\mathcal{H}) = M \ll p$; in numerous circumstances, $\mathcal{H}$ is a subspace of $\mathbf{R}^p$ and we can fit an OLS model on the projected coordinates (see Section 23.2, *Dimension Reduction*).

In **subset selection**, we identify a subset of the p predictors for which there is evidence of a (strong-ish) link with the response, and we fit a model to this reduced set using the OLS framework. Given p predictors (some of which may be interaction terms), there are 2^p OLS models that can be fit on a training set Tr.

Which of those models should be selected as the **best model**?

20.4.1 Best Subset Selection

In the **best subset selection** BSS approach, the search for the best model is usually broken down into 3 stages:

1. let $\mathcal{M}_0$ denote the **null model** (without predictor) which simply predicts the sample mean for all observations;
2. for $k = 1, \dots, p$ (as long as the model can be fit):
   a. fit every model that contains exactly k predictors (there are $\binom{p}{k}$ of them);
   b. pick the model with smallest SSE (largest R^2) and denote it by $\mathcal{M}_k$;
3. select a unique model from $\{\mathcal{M}_0, \dots, \mathcal{M}_p\}$ using $\text{CV}_{(K)}$, C_p (AIC), BIC, R_a^2, or any other appropriate metric.[45]

BSS is conceptually simple, but with 2^p models to try out, it quickly becomes computationally infeasible for large p ($p > 40$, say). When p is large, the chances of finding a model that performs well according to step 3 but poorly for new data increase, which can lead to **overfitting** and **high-variance** estimates, which were exactly the problems we were trying to avoid in the first place.[46]

45: We cannot use SSE or R^2 as metrics in this last step, as we would always select $\mathcal{M}_p$ since SSE decreases monotonically with k and R^2 increases monotonically with k. Low SSE/high R^2 are associated with a low training error, whereas the other metrics attempt to say something about the test error, which is what we are after: after all, a model is good if it makes good predictions!

46: Here, we are assuming that all models are OLS models, but subset selection algorithms can be used for other families of supervised learning methods; all that is required are appropriate training error estimates for step 2b and test error estimates for step 3.

20.4.2 Stepwise Selection

Stepwise selection (SS) methods attempt to overcome this challenge by only looking at a restricted set of models. **Forward stepwise selection** (FSS) starts with the null model $\mathcal{M}_0$ and adding predictors one-by-one until it reaches the full model $\mathcal{M}_p$:

1. Let $\mathcal{M}_0$ denote the null model;
2. for $k = 0, \dots, p-1$ (as long as the model can be fit):
   a. consider the $p-k$ models that add a single predictor to $\mathcal{M}_k$;
   b. pick the model with smallest SSE (largest R^2) and denote it by $\mathcal{M}_{k+1}$;
3. select a unique model from $\{\mathcal{M}_0, \dots, \mathcal{M}_p\}$ using $\text{CV}_{(K)}$, C_p (AIC), BIC, R_a^2, or any other appropriate metric.

Backward stepwise selection (also BSS, unfortunately) works the other way, starting with the full model $\mathcal{M}_p$ and removing predictors one-by-one until it reaches the null model $\mathcal{M}_0$:

1. Let $\mathcal{M}_p$ denote the full model;
2. for $k = p, \dots, 1$ (as long as the model can be fit):
   a. consider the k models that remove a single predictor from $\mathcal{M}_k$;

   b. pick the model with smallest SSE (largest R^2) and denote it by $\mathscr{M}_{k-1}$;

3. select a unique model from $\{\mathscr{M}_0, \ldots, \mathscr{M}_p\}$ using $\text{CV}_{(K)}$, C_p (AIC), BIC, R_a^2, or any other appropriate metric.

The computational advantage of SS over B(est)SS is evident: instead of having to fit 2^p models, SS only requires

$$1 + p + (p-1) + \cdots + 2 + 1 = \frac{p^2 + p + 2}{2}$$

models to be fit to Tr. However, there is no guarantee that the "best" model (among the 2^p BSS models) is found in the reduced set of SS models.

SS can be used in settings where p is too large for BSS to be computationally feasible. Note that for OLS models, backward stepwise selection only works if $p \leq n$ (otherwise OLS might not have a unique parameter solution); if $p > n$, only forward stepwise selection is viable.

Hybrid selection (HS) methods attempt to mimic BSS while keeping model computation in a manageable range, not unlike in SS. More information on this topic is available in [3].

20.4.3 Selecting the Optimal Model

The full model always has largest R^2/smallest SSE.[47] In order to estimate the test error,[48] we can either:

47: As it is a measure of the training error, and as such, is subject to the overfitting property found in the bias-variance trade-off diagram of Figure 20.5.

48: And thus pick the optimal model in the list $\{\mathscr{M}_0, \ldots, \mathscr{M}_p\}$.

- **adjust** the training error to account for the bias induced by overfitting, or
- **directly** estimate the test error using a validation set or cross-validation.

Adjustment Statistics Commonly, we use one of the following adjustment statistics: Mallow's C_p, the Akaike information criterion (AIC), the Bayesian information criteria (BIC), or the adjusted coefficient of determination R_a^2; C_p, AIC, and BIC must be **minimized**, while R_a^2 must be **maximized**.

The adjustment statistics require the following quantities:

- N, the number of observations in Tr;
- p, the number of predictors under consideration;
- $d = p + 2$,
- $\hat{\sigma}^2$, the estimate of $\text{Var}(\varepsilon)$ (irreducible error);
- SSE and SST, the residual and the total sum of squares.

Mallow's C_p statistic is given by

$$C_p = \frac{1}{N}(\text{SSE} + 2d\hat{\sigma}^2) = \text{MSE}_{\text{Tr}} + \underbrace{\frac{2d\hat{\sigma}^2}{N}}_{\text{adjustment}} .$$

As d increases, so does the adjustment term. Note that if $\hat{\sigma}^2$ is an unbiased estimate of $\text{Var}(\varepsilon)$, C_p is an unbiased estimate of MSE_{Te}.

The **Akaike information criterion** (AIC) is given by

$$\text{AIC} = -2\ln L + \underbrace{2d}_{\text{adjustment}},$$

where L is the maximized value of the likelihood function for the estimated model. If the errors are normally distributed, this requires finding the maximum of

$$\begin{aligned} L &= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\hat{\sigma}} \exp\left(-\frac{(Y_i - \mathbf{X}_i^{\top}\boldsymbol{\beta})^2}{2\hat{\sigma}^2}\right) \\ &= \frac{1}{(2\pi)^{n/2}\hat{\sigma}^n} \exp\left(-\frac{1}{2\hat{\sigma}^2}\sum_{i=1}^{N}(Y_i - \mathbf{X}_i^{\top}\boldsymbol{\beta})^2\right), \end{aligned}$$

or, upon taking the logarithm,

$$\ln L = \text{constant} - \frac{1}{2\hat{\sigma}^2}\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2,$$

and so

$$\arg\max_{\boldsymbol{\beta}}\{\ln L(\boldsymbol{\beta})\} = \arg\min_{\boldsymbol{\beta}}\{\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2\}.$$

However,

$$\begin{aligned} \text{AIC} &= -2\ln L + 2d = \text{constant} + \frac{1}{\hat{\sigma}^2}\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + 2d \\ &= \text{constant} + \frac{\text{SSE}}{\hat{\sigma}^2} + 2d \\ &= \text{constant} + \frac{N}{\hat{\sigma}^2}\cdot\frac{1}{N}\left(\text{SSE} + 2d\hat{\sigma}^2\right) = \text{constant} + \frac{N}{\hat{\sigma}^2}C_p. \end{aligned}$$

Evidently, when the error structure is normal, minimizing AIC is equivalent to minimizing C_p.

The **Bayesian information criterion** uses a different adjustment term:

$$\text{BIC} = \frac{1}{N}(\text{SSE} + d\hat{\sigma}^2\ln N) = \text{MSE}_{\text{Tr}} + \underbrace{d\hat{\sigma}^2\frac{\ln N}{N}}_{\text{adjustment}}.$$

This adjustment penalizes models with large number of predictors; minimizing BIC results in selecting models with fewer variables than those obtained by minimizing C_p, in general.

The **adjusted coefficient of determination** R_a^2 is the Ur-example of an adjusted statistic:

$$R_a^2 = 1 - \frac{\text{SSE}/(N-p-1)}{\text{SST}/(N-1)} = 1 - (1 - R^2)\frac{N-1}{N-p-1}.$$

Maximizing R_a^2 is equivalent to minimizing $\text{SSE}/(N-p-1)$; note that R_a^2 penalizes models with **unnecessary** variables.[49]

49: Note that in this subsection's formalism, we have $p+1$ predictors for the linear model: $X_1, \ldots, X+p$ and a constant term X_0.

Validation and Cross-Validation (Reprise) As above, we want to select $\mathcal{M}_{k^*}$ from a sequence of models $\{\mathcal{M}_1, \mathcal{M}_2, \ldots\}$. The procedure is simple: we compute MSE_{Va} on some validation set or $\text{CV}_{(K)}$ for each $\mathcal{M}_k$, and we find the k^* for which the value is **smallest** (see Section 20.3, *Cross-Validation*).

The main advantages of this approach are that:

- there is no need to estimate the irreducible error $\text{Var}(\varepsilon) = \sigma^2$;
- the method produces an estimate for MSE_{Te} "for free," and
- it can be used when the number of parameters is hard to pinpoint (in deep learning networks, for instance).

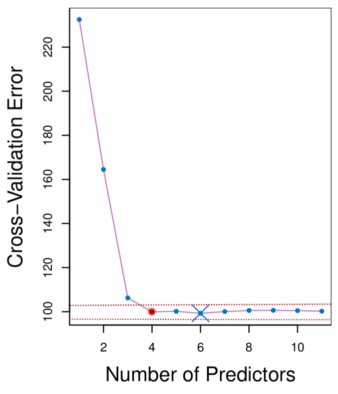
Historically, adjustment approaches have been preferred because cross-validation was computationally demanding, especially when p, n were large, but that is not as much of a problem in modern times.

Consequently, cross-validation is championed as the optimal model selection approach, using the **one standard error rule**: calculate the standard error of $\widehat{\text{MSE}}_{\text{Te}}$ for each model size, and select the **smallest model** for which $\widehat{\text{MSE}}_{\text{Te}}$ is within one standard error from the lowest point on the cross-validation error curve.

Roughly speaking, this is equivalent to **Occam's Razor**[50] on models that have similar predictive power.

50: "When presented with competing hypotheses about the same prediction, one should select the solution with the fewest assumptions."

In the image below (modified from [3]), the lowest point is reached when $p = 6$ (blue "X") and the dashed red lines represent the 1-standard error limits; according to the rule described above, we would select the model with $p = 4$ parameters (red dot).



SS methods are used extensively in practice, but there are serious limitations to this approach:

- all intermediate tests are **biased**, as they are based on the same data;
- R_a^2 only takes into account the number of features in the final model, not the degrees of freedom that have been used up during the entire process;
- if the cross-validation error is used, stepwise selection should be repeated for each sub-model.

All in all, SS is a classic example of p−hacking: we are getting results without setting hypotheses up first.

Example In spite of the warning mentioned above, it could still be useful to know how to perform stepwise selection. In what follows, we search for the best FSS and BSS linear models to predict the credit card balance for observations contained in the training set Credit.csv.

```
Credit <- read.csv("Credit.csv", stringsAsFactors = TRUE)
str(Credit)
```

```
'data.frame':   400 obs. of  12 variables:
 $ X        : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Income   : num  14.9 106 104.6 148.9 55.9 ...
 $ Limit    : int  3606 6645 7075 9504 4897 8047 3388 7114 ...
 $ Rating   : int  283 483 514 681 357 569 259 512 266 491 ...
 $ Cards    : int  2 3 4 3 2 4 2 2 5 3 ...
 $ Age      : int  34 82 71 36 68 77 37 87 66 41 ...
 $ Education: int  11 15 11 11 16 10 12 9 13 19 ...
 $ Gender   : Factor w/ 2 levels "Female","Male": 2 1 2 1 2 2 ...
 $ Student  : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 1 ...
 $ Married  : Factor w/ 2 levels "No","Yes": 2 2 1 1 2 1 ...
 $ Ethnicity: Factor w/ 3 levels "African American",..: 3 2 2 ...
 $ Balance  : int  333 903 580 964 331 1151 203 872 279 ...
```

We remove the id variable X, and create dummy variables for the categorical levels.

```
Credit <- Credit[,-c(1)]
Credit$Gender.dummy <- ifelse(Credit$Gender == "Female",1,0)
Credit$Student.dummy <- ifelse(Credit$Student == "Yes",1,0)
Credit$Married.dummy <- ifelse(Credit$Married == "Yes",1,0)
Credit$Ethnicity.AA.dummy <- ifelse(Credit$Ethnicity == "African American",1,0)
Credit$Ethnicity.A.dummy <- ifelse(Credit$Ethnicity == "Asian",1,0)
Credit <- Credit[,c(1:6,12:16,11)]
summary(Credit)
```

```
     Income           Limit            Rating          Cards      
 Min.   : 10.35   Min.   :  855   Min.   : 93.0   Min.   :1.000  
 1st Qu.: 21.01   1st Qu.: 3088   1st Qu.:247.2   1st Qu.:2.000  
 Median : 33.12   Median : 4622   Median :344.0   Median :3.000  
 Mean   : 45.22   Mean   : 4736   Mean   :354.9   Mean   :2.958  
 3rd Qu.: 57.47   3rd Qu.: 5873   3rd Qu.:437.2   3rd Qu.:4.000  
 Max.   :186.63   Max.   :13913   Max.   :982.0   Max.   :9.000  

      Age          Education      Gender.dummy     Student.dummy
 Min.   :23.00   Min.   : 5.00   Min.   :0.0000   Min.   :0.0  
 1st Qu.:41.75   1st Qu.:11.00   1st Qu.:0.0000   1st Qu.:0.0  
 Median :56.00   Median :14.00   Median :1.0000   Median :0.0  
 Mean   :55.67   Mean   :13.45   Mean   :0.5175   Mean   :0.1  
 3rd Qu.:70.00   3rd Qu.:16.00   3rd Qu.:1.0000   3rd Qu.:0.0  
 Max.   :98.00   Max.   :20.00   Max.   :1.0000   Max.   :1.0  
 Married.dummy    Ethnicity.AA.dummy Ethnicity.A.dummy    Balance
 Min.   :0.0000   Min.   :0.0000     Min.   :0.000     Min.   :   0.00
 1st Qu.:0.0000   1st Qu.:0.0000     1st Qu.:0.000     1st Qu.:  68.75
 Median :1.0000   Median :0.0000     Median :0.000     Median : 459.50
 Mean   :0.6125   Mean   :0.2475     Mean   :0.255     Mean   : 520.01
 3rd Qu.:1.0000   3rd Qu.:0.0000     3rd Qu.:1.000     3rd Qu.: 863.00
 Max.   :1.0000   Max.   :1.0000     Max.   :1.000     Max.   :1999.00
```

We will work with a scaled version of the dataset.

```
Credit.scaled <- scale(Credit)
parameters <- attributes(Credit.scaled)
Credit.scaled <- data.frame(Credit.scaled)
var.names <- colnames(Credit.scaled)
```

We start by implementing step 2 of the FSS algorithm.

```
model <- c()
ind   <- c()

for(i in 1:(ncol(Credit.scaled)-1)){
r2 <- c()
  for(j in setdiff((1:(ncol(Credit.scaled)-1)),c(ind))){
    model <- lm(Balance ~ .,
                data = Credit.scaled[,c(ind,j,12)])
    r2[j] <- summary(model)$r.squared
  }
ind[i] <- which.max(r2)
}

var.names[ind]
```

```
 [1] "Rating"       "Income"             "Student.dummy"
 [4] "Limit"        "Cards"              "Age"
 [7] "Gender.dummy" "Ethnicity.AA.dummy" "Married.dummy"
[10] "Education"    "Ethnicity.A.dummy"
```

The best 1-parameter model $\mathcal{M}_1$ uses `Rating`, the best 2-parameter model $\mathcal{M}_2$ built from $\mathcal{M}_1$ uses `Rating` and `Income`, and so on.

Next, we implement step 3 by computing the adjustment statistics (AIC, BIC, R_a^2) and the cross-validation error (with $K = 5$ folds) for each of $\mathcal{M}_0, \mathcal{M}_1, \ldots$.[51]

51: The latter uses the function `cv.lm()` available in the `lmvar` package in R.

We deal with $\mathcal{M}_0$ first.

```
model <- c()
aic   <- c()
bic   <- c()
r2a   <- c()
cv.m  <- c()
```

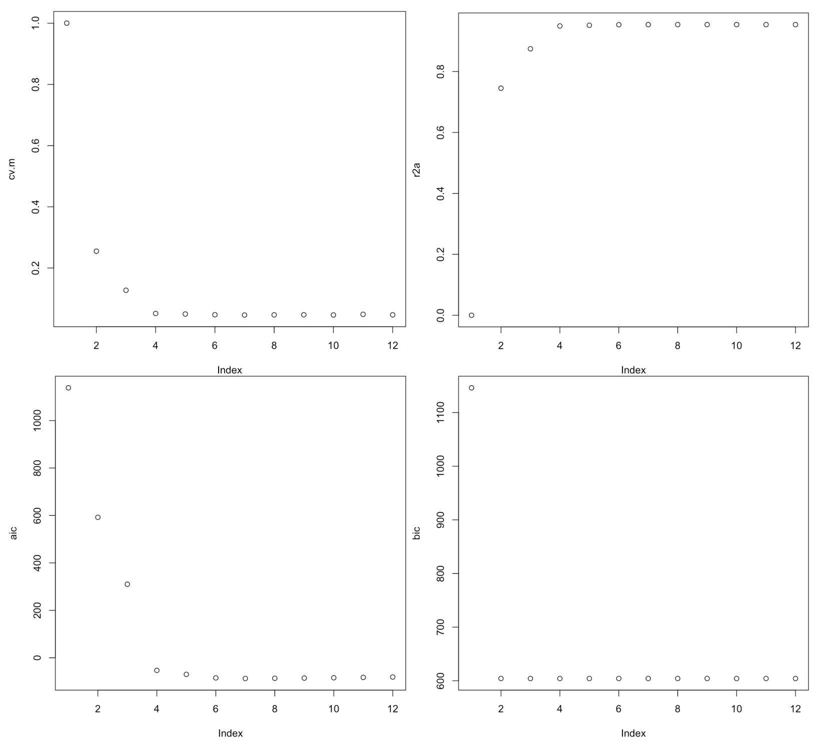
```
model[[1]] <- lm(Balance ~ 1,
                 data=Credit.scaled, y=TRUE, x=TRUE)
cv.m[1]    <- lmvar::cv.lm(model[[1]],k=5)$MSE[[1]]
r2a[1]     <- summary(model[[1]])$adj.r.squared
aic[1]     <- AIC(model[[1]])
bic[1]     <- BIC(model[[1]])
```

The remaining models are similarly handled:

```
for(i in 1:(ncol(Credit.scaled)-1)){
  model[[i+1]] <- lm(Balance ~., data=Credit.scaled[,
           c(ind[c(1:i)],12)], y=TRUE, x=TRUE)
  cv.m[i+1]    <- lmvar::cv.lm(model[[i+1]],k=5)$MSE[[1]]
  r2a[i+1]     <- summary(model[[i+1]])$adj.r.squared
  aic[i+1]     <- AIC(model[[i+1]])
  bic[i+1]     <- BIC(model[[1+1]])
}
```

Let us plot the outcome for each adjustment statistic (and the CV estimation of the test error):

```
plot(cv.m)
plot(r2a)
plot(aic)
plot(bic)
```



The best FSS model using the CV estimate of the test error is:

```
ind.cv  <- which.min(cv.m)
var.names[ind[1:ind.cv]]
```

```
[1] "Rating"        "Income"        "Student.dummy" "Limit"
[5] "Cards"         "Age"
```

The best FSS model using R_a^2 is:

```
ind.r2a <- which.max(r2a)
var.names[ind[1:ind.r2a]]
```

```
[1] "Rating"             "Income"             "Student.dummy"
[4] "Limit"              "Cards"              "Age"
[7] "Gender.dummy"       "Ethnicity.AA.dummy" "Married.dummy"
```

The best FSS model using AIC is:

```
ind.aic <- which.min(aic)
var.names[ind[1:ind.aic]]
```

```
[1] "Rating"        "Income"        "Student.dummy" "Limit"
[5] "Cards"         "Age"           "Gender.dummy"
```

The best FSS model using BIC is:

```
ind.bic <- which.min(bic)
var.names[ind[1:ind.bic]]
```

```
[1] "Rating" "Income"
```

Are there overlaps? The same can be done for BSS, instead:

```
model.BSS <- c()
ind.BSS   <- list()

for(i in 1:(ncol(Credit.scaled)-1)){
  r2 <- c()
  list.of.indices <- combn(1:(ncol(Credit.scaled)-1), i)
  for(j in 1:ncol(list.of.indices)){
    model.BSS <- lm(Balance ~ .,
        data=Credit.scaled[,c(list.of.indices[,j],12)])
    r2[j] <- summary(model.BSS)$r.squared
  }
  ind.BSS[[i]] <- list.of.indices[,which.max(r2)]
}

model.BSS <- c()
aic.BSS   <- c()
bic.BSS   <- c()
r2a.BSS   <- c()
```

```
cv.m.BSS  <- c()

model.BSS[[1]] <- lm(Balance ~ 1, data=Credit.scaled,
                     y=TRUE, x=TRUE)
cv.m.BSS[1] <- lmvar::cv.lm(model.BSS[[1]],
                     k=5)$MSE[[1]]
r2a.BSS[1] <- summary(model.BSS[[1]])$adj.r.squared
aic.BSS[1] <- AIC(model.BSS[[1]])
bic.BSS[1] <- BIC(model.BSS[[1]])

for(i in 1:(ncol(Credit.scaled)-1)){
 model.BSS[[i+1]] <- lm(Balance ~.,
  data=Credit.scaled[,c(ind.BSS[[i]],12)], y=TRUE, x=TRUE)
 cv.m.BSS[i+1] <- lmvar::cv.lm(model.BSS[[i+1]],k=5)$MSE[[1]]
 r2a.BSS[i+1] <- summary(model.BSS[[i+1]])$adj.r.squared
 aic.BSS[i+1] <- AIC(model.BSS[[i+1]])
 bic.BSS[i+1] <- BIC(model.BSS[[1+1]])
}
```

```
ind.cv.BSS  <- which.min(cv.m.BSS)
var.names[ind.BSS[[ind.cv.BSS]]]
ind.r2a.BSS <- which.max(r2a.BSS)
var.names[ind.BSS[[ind.r2a.BSS]]]
ind.aic.BSS <- which.min(aic.BSS)
var.names[ind.BSS[[ind.aic.BSS]]]
ind.bic.BSS <- which.min(bic.BSS)
var.names[ind.BSS[[ind.bic.BSS]]]
```

```
[1] "Income"        "Limit"         "Rating"        "Cards"
[5] "Age"           "Student.dummy"

[1] "Income"             "Limit"              "Rating"
[4] "Cards"              "Age"                "Gender.dummy"
[7] "Student.dummy"      "Married.dummy"      "Ethnicity.AA.dummy"

[1] "Income"        "Limit"         "Rating"        "Cards"
[5] "Age"           "Gender.dummy"  "Student.dummy"

[1] "Income" "Rating"
```

Any surprises?

20.5 Nonlinear Modeling

In practice the linearity assumption is **almost never met** and the regression function

$$y = f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}]$$

has to be approximated by some other technique. Or does it?

The linearity assumption is often "good enough" in spite of it not being met, and, coupled with its convenience of use and its multiple extensions, it is rarely a waste of time to give that approach a try.

When heavier machinery is required, it pays to consider the following OLS generalizations, which offer a lot of flexibility without sacrificing ease of interpretability, before jumping to so-called **black box models** (SVM, ANN, ensemble learning, etc.) of Chapter 21:

- curve fitting (polynomial regression, step functions, splines, etc.);
- local regression methods, or
- generalized additive models.

20.5.1 Basis Function Models

If we have reason to suspect that the response Y is not a linear combination of the predictors, we might benefit from using a **derived set of predictors** (see [3, Section 7.3]).

Polynomial Regression We can extend the simple linear regression model $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$, $i = 1, \ldots, N$, by allowing for **polynomial basis terms** in the regression function:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_d x_i^d + \varepsilon_i, \quad i = 1, \ldots, N.$$

The regression function is non-linear in terms of the observations x_i, but it is linear in terms of the coefficients β_j.[52] We thus create new variables $X_1 = X$, $X_2 = X^2$, and so on, and estimate the regression function $y = f(\mathbf{x})$ *via* $\hat{f}(\mathbf{x}) = \mathbf{x}^\top \hat{\boldsymbol{\beta}}$, where the coefficients $\hat{\boldsymbol{\beta}}$ are learned using the training set Tr.

52: In terms of $\{x, x^2, \ldots, x^d\}$.

Typically, the coefficient values are of little interest – it is the predictions $\hat{f}(\mathbf{x})$ that are sought.

It is easy to obtain and estimate for $\text{Var}(\hat{f}(\mathbf{x}))$ since $\hat{f}(\mathbf{x})$ is linear in the coefficients $\hat{\beta}_i$, $i = 0, \ldots, d$:

$$\begin{aligned}\text{Var}(\hat{f}(\mathbf{x})) = \text{Var}(\mathbf{x}^\top \hat{\boldsymbol{\beta}}) &= \sum_{i,j=0}^{d} \text{Cov}(\hat{\beta}_i \tilde{x}_i, \hat{\beta}_j \tilde{x}_j) \\ &= \sum_{i,j=0}^{d} \tilde{x}_i \tilde{x}_j \text{Cov}(\hat{\beta}_i, \hat{\beta}_j) = \mathbf{x}^\top \text{Cov}(\hat{\boldsymbol{\beta}}) \mathbf{x} = \sigma^2 \mathbf{x}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}.\end{aligned}$$

The estimated variance of the approximation at $\mathbf{x}$ is thus

$$\hat{\text{Var}}(\hat{f}(\mathbf{x})) = \frac{\text{SSRes}}{n-d-1} \mathbf{x}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x} = \frac{\|\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}\|_2^2}{n-d-1} \mathbf{x}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x},$$

with $\text{se}(\hat{f}(\mathbf{x})) = \sqrt{\hat{\text{Var}}(\hat{f}(\mathbf{x}))}$, so that

$$\hat{f}(\mathbf{x}) \pm 2 \cdot \text{se}(\hat{f}(\mathbf{x}))$$

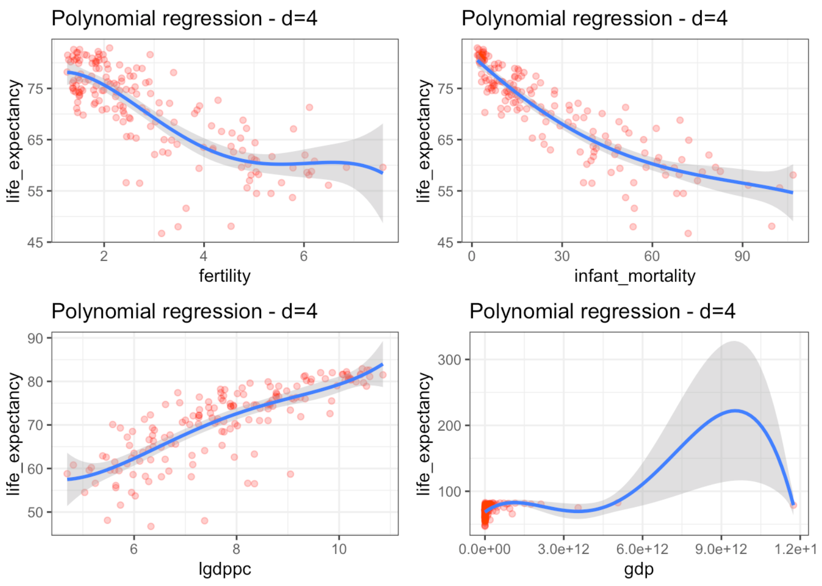
constitutes a 95% C.I. for $\hat{f}(\mathbf{x})$, assuming normality of the error terms.

Gapminder Example The charts below show polynomial regressions ($d = 4$) and confidence intervals for life expectancy against 4 different predictors in the 2011 Gapminder data (assuming that the training set Tr is the entire dataset).[53]

53: In this section, we assume that `ggplot2` and `dplyr` have already been loaded.

```
plot1 <- ggplot(gapminder.2011, aes(x=fertility,
    y=life_expectancy)) + theme_bw() +
  geom_point(color=’red’, alpha=0.3) +
  stat_smooth(method=’lm’, formula = y~poly(x,4)) +
  ggtitle("Polynomial regression - d=4")

plot2 <- ggplot(gapminder.2011, aes(x=infant_mortality,
    y=life_expectancy)) + theme_bw() +
  geom_point(color=’red’, alpha=0.3) +
  stat_smooth(method=’lm’, formula = y~poly(x,4)) +
  ggtitle("Polynomial regression - d=4")
```

```
plot3 <- gapminder.2011 |>
  mutate(lgdppc=log(gdp/population)) |>
  ggplot(aes(x=lgdppc, y=life_expectancy)) +
  geom_point(color=’red’, alpha=0.3) +
  stat_smooth(method=’lm’, formula = y~poly(x,4)) +
  ggtitle("Polynomial regression - d=4") +
  theme_bw()

plot4 <- ggplot(gapminder.2011, aes(x=gdp,
    y=life_expectancy)) +
  geom_point(color=’red’, alpha=0.3) +
  stat_smooth(method=’lm’, formula = y~poly(x,4)) +
  ggtitle("Polynomial regression - d=4") +
  theme_bw()

gridExtra::grid.arrange(plot1, plot2, plot3, plot4, ncol=2)
```



In this example, we picked $d = 4$. How do we select d, in general? We can either pick a reasonable small d (often below 4) or use cross-validation to select a d that minimizes the estimated MSE_{Te}.

Note that it is easy to incorporate more than one predictor and interaction terms into the model.

The nature of polynomials ($|f(\mathbf{x})| \to \infty$ when $\|\mathbf{x}\| \to \infty$) is such that tail behaviour is usually quite horrible (look at the bottom-right example above). Consequently, polynomial regression should be used **very carefully**, staying within the domain and making sure to centre the predictors to reduce variance inflation.

Step Functions Polynomial regression is an attractive approach because of the ease with which we can use the apparatus of OLS, but the elephant in the room is that we are imposing a **global structure** on the non-linear function $y = f(\mathbf{x})$, and that cannot always be justified.

Step functions can be used to keep things "local". Let c_i, $i = 1, \dots, K$ lie in range(X) and consider the following $K + 1$ new predictors:

$$\begin{aligned} C_0(X) &= \mathcal{I}(X < c_1) \\ C_i(X) &= \mathcal{I}(c_i \le X < c_{i+1}),\ i = 1, \dots, K-1 \\ C_K(X) &= \mathcal{I}(c_k \le X), \end{aligned}$$

where $\mathcal{I}$ is the **indicator function**

$$\mathcal{I}(\alpha) = \begin{cases} 0, & \alpha \text{ is false} \\ 1, & \alpha \text{ is true} \end{cases}$$

For any X, $C_0(X) + C_1(X) + \cdots + C_K(X) = 1$, since X lies in exactly one of the intervals

$$(-\infty, c_1),\ [c_1, c_2),\ \cdots,\ [c_{K-1}, C_K),\ [C_K, \infty).$$

The **step function regression model** is

$$Y_i = \beta_0 + \beta_1 C_1(X_i) + \cdots + \beta_K C_K(X_i) + \varepsilon_i, \quad i = 1, \dots, N;$$

it can also be obtained using the OLS framework.[54]

54: Thus a 95% C.I. can be built just as with polynomial and other regressions.

For a given X, at most one of $C_1(X), \dots, C_K(X)$ is $\neq 0$; thus, when $X < c_1$, $C_j(X) = 0$ for all $j = 1, \dots, K$, and so

$$\beta_0 = \text{Avg}\{Y \mid X < c_1\}.$$

For $X \in [c_j, c_{j+1})$, $\hat{y} = \beta_0 + \beta_j$, so β_j represents the **average increase in** Y **for** $[c_j, c_{j+1})$ **relative to** $(-\infty, c_1)$.

The only major challenge with step function regression is that there is no easy way to find the number K and select the position of the breakpoints $c_1, \dots, c_K$, unless there are natural gaps in the predictors. We will discuss a strategy to determine the number and location of knots when we discuss classification and regression trees in Chapter 21.

We did not discuss how step function regression or polynomial regression could be achieved in higher dimensions, but the principle remains the same (except that the number of parameters increases drastically, which can create some overfitting issues).

Gapminder Example The charts below show step function regressions and confidence intervals for life expectancy against 4 different predictors in the 2011 Gapminder data .[55]

55: Assuming that the training set Tr is the entire dataset.

We start by building a $K = 3$ knots step function model for life expectancy against fertility, using the (arbitrary) knot values at 2, 4, and 6:

```
gapminder.2011 <- gapminder.2011 |>
  mutate(fert0=I(fertility<2),
         fert1=I(2<=fertility & fertility<4),
         fert2=I(4<=fertility & fertility<6),
         fert3=I(6<=fertility))
model.sf.1 = lm(life_expectancy ~ fert0 + fert1 + fert2,
                data=gapminder.2011)
summary(model.sf.1)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-24.0485  -2.8300   0.2515   3.9669  12.1515

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  60.5444     1.9554  30.963  < 2e-16 ***
fert0TRUE    16.8106     2.0969   8.017 2.04e-13 ***
fert1TRUE    10.2040     2.0845   4.895 2.36e-06 ***
fert2TRUE     0.7814     2.2212   0.352    0.725
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.866 on 162 degrees of freedom
Multiple R-squared:  0.5308,    Adjusted R-squared:  0.5221
F-statistic: 61.09 on 3 and 162 DF,  p-value: < 2.2e-16
```

The corresponding step function is defined with:

```
g.1 <- function(x){
    model.sf.1$coefficients[1] +
    model.sf.1$coefficients[2]*I(x<2) +
    model.sf.1$coefficients[3]*I(2<=x & x<4) +
    model.sf.1$coefficients[4]*I(6<=x)
  }
```

We next build a $K = 4$ knots step function model for life expectancy against infant mortality, using the (arbitrary) knot values 10, 20, 40, 70:

```
gapminder.2011 <- gapminder.2011 |>
  mutate(inf0=I(infant_mortality<10),
    inf1=I(10<=infant_mortality & infant_mortality<20),
    inf2=I(20<=infant_mortality & infant_mortality<40),
    inf3=I(40<=infant_mortality & infant_mortality<70),
    inf4=I(70<=infant_mortality))

model.sf.2 = lm(life_expectancy ~ inf0 + inf1 + inf2 +
                inf3, data=gapminder.2011)
summary(model.sf.2)

g.2 <- function(x){
    model.sf.2$coefficients[1] +
    model.sf.2$coefficients[2]*I(x<10) +
    model.sf.2$coefficients[3]*I(10<=x & x<20) +
    model.sf.2$coefficients[4]*I(20<=x & x<40) +
    model.sf.2$coefficients[5]*I(40<=x & x<70)
  }
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-13.9800  -2.3800   0.3725   2.7622   9.3200

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   56.675      1.207  46.939  < 2e-16 ***
inf0TRUE      22.017      1.340  16.437  < 2e-16 ***
inf1TRUE      17.963      1.390  12.928  < 2e-16 ***
inf2TRUE      11.782      1.429   8.247 5.46e-14 ***
inf3TRUE       5.305      1.399   3.791 0.000211 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.183 on 161 degrees of freedom
Multiple R-squared:  0.763, Adjusted R-squared:  0.7571
F-statistic: 129.5 on 4 and 161 DF,  p-value: < 2.2e-16
```

We next build a $K = 3$ knots step function model for life expectancy against the log of gdp per capita, using the (arbitrary) knot values at 6, 8, 10:

```
gapminder.2011 <- gapminder.2011 |>
  mutate(lgdppc=log(gdp/population)) |>
  mutate(lgdppc0=I(lgdppc<6),
         lgdppc1=I(6<=lgdppc & lgdppc<8),
         lgdppc2=I(8<=lgdppc & lgdppc<10),
         lgdppc3=I(10<=lgdppc))

model.sf.3 = lm(life_expectancy ~ lgdppc0 + lgdppc1 + lgdppc2,
                data=gapminder.2011)
summary(model.sf.3)
```

```
g.3 <- function(x){
    model.sf.3$coefficients[1] +
    model.sf.3$coefficients[2]*I(x<6) +
    model.sf.3$coefficients[3]*I(6<=x & x<8) +
    model.sf.3$coefficients[4]*I(8<=x & x<10)
  }
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-21.771  -1.831   0.550   3.691   9.789

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   81.100      1.270  63.857  < 2e-16 ***
lgdppc0TRUE  -20.788      1.708 -12.174  < 2e-16 ***
lgdppc1TRUE  -12.629      1.453  -8.692 3.75e-15 ***
lgdppc2TRUE   -6.012      1.509  -3.984 0.000102 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.82 on 162 degrees of freedom
Multiple R-squared:  0.5382,    Adjusted R-squared:  0.5296
F-statistic: 62.93 on 3 and 162 DF,  p-value: < 2.2e-16
```

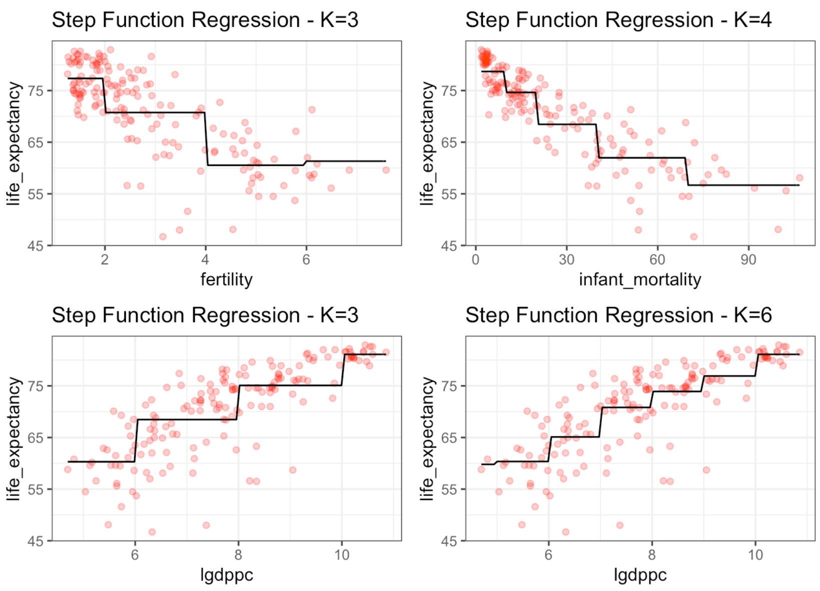
Finally, we build a $K = 6$ knots step function model for life expectancy against the log of gdp per capita, using the (arbitrary) knot values at 5, 6, 7, 8, 9, 10:

```
gapminder.2011 <- gapminder.2011 |>
  mutate(lgdppc=log(gdp/population)) |>
  mutate(lgdppc0=I(lgdppc<5),
         lgdppc1=I(5<=lgdppc & lgdppc<6),
         lgdppc2=I(6<=lgdppc & lgdppc<7),
         lgdppc3=I(7<=lgdppc & lgdppc<8),
         lgdppc4=I(8<=lgdppc & lgdppc<9),
         lgdppc5=I(9<=lgdppc & lgdppc<10),
         lgdppc6=I(10<=lgdppc))

model.sf.4 = lm(life_expectancy ~ lgdppc0 + lgdppc1 +
       lgdppc2 + lgdppc3 + lgdppc4 + lgdppc5,
       data=gapminder.2011)
summary(model.sf.4)

g.4 <- function(x){
    model.sf.4$coefficients[1] +
    model.sf.4$coefficients[2]*I(x<5) +
    model.sf.4$coefficients[3]*I(5<=x & x<6) +
    model.sf.4$coefficients[4]*I(6<=x & x<7) +
    model.sf.4$coefficients[5]*I(7<=x & x<8) +
    model.sf.4$coefficients[6]*I(8<=x & x<9) +
    model.sf.4$coefficients[7]*I(9<=x & x<10)
  }
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-22.8250  -1.3500   0.5964   3.1841  12.2929

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   81.100      1.204  67.367  < 2e-16 ***
lgdppc0TRUE  -21.300      4.082  -5.217 5.59e-07 ***
lgdppc1TRUE  -20.746      1.648 -12.585  < 2e-16 ***
lgdppc2TRUE  -15.993      1.593 -10.042  < 2e-16 ***
lgdppc3TRUE  -10.275      1.487  -6.912 1.10e-10 ***
lgdppc4TRUE   -7.187      1.559  -4.610 8.24e-06 ***
lgdppc5TRUE   -4.190      1.724  -2.431   0.0162 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.517 on 159 degrees of freedom
Multiple R-squared:  0.5927,    Adjusted R-squared:  0.5774
F-statistic: 38.57 on 6 and 159 DF,  p-value: < 2.2e-16
```

The step functions in each of the 4 cases are displayed below:

```
plot1 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  stat_function(fun=g.1) +
  ggtitle("Step Function Regression - K=3")

plot2 <- ggplot(gapminder.2011,aes(x=infant_mortality,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  stat_function(fun=g.2) +
  ggtitle("Step Function Regression - K=4")

plot3 <- ggplot(gapminder.2011,aes(x=lgdppc,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  stat_function(fun=g.3) +
  ggtitle("Step Function Regression - K=3")

plot4 <- ggplot(gapminder.2011,aes(x=lgdppc,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  stat_function(fun=g.4) +
  ggtitle("Step Function Regression - K=6")

gridExtra::grid.arrange(plot1, plot2, plot3, plot4, ncol=2)
```



The step functions do capture the general trends, but how easily interpretable are they?

20.5.2 Splines

We can combine polynomial regression and step functions to obtain a more flexible **curve fitting** approach.

Regression Splines Instead of fitting a polynomial over the entire range of the predictor X, we use different polynomials (of degree up to 3, usually) in regions R_k,[56] such as:

56: Defined by **knots** in the 1-dimensional case.

$$Y_i = \begin{cases} \beta_{0,1} + \beta_{1,1}X_i + \beta_{2,1}X_i^2 + \beta_{3,1}X_i^3 + \varepsilon_i, & \text{if } X_i \in R_1 \\ \beta_{0,2} + \beta_{1,2}X_i + \beta_{2,2}X_i^2 + \beta_{3,2}X_i^3 + \delta_i, & \text{if } X_i \in R_2 \end{cases}$$

Various constraints can be imposed on the polynomials:

- none;
- continuity at each region's borders;
- C^1 (continuously differentiable) at each region's borders; etc.

In a sense to be defined shortly, **splines** have the "maximum" amount of continuity. Note that using more regions leads to a more flexible fit.

In what follows, we assume that the domain is split into $K + 1$ regions, bounded by **knots** (there are thus K such knots). If we impose **no restriction** on the functions, we are trying to fit $K + 1$ piecewise cubic functions to the data; each polynomial has 4 parameters to be estimated, leading to $4(K + 1)$ effective parameters.

If we impose a **continuous fit**,[57] we reduce the number of effective parameters. We can also require a **continuously differentiable** fit,[58] further reducing the number of effective parameters.

57: The polynomials must agree at the knots.

58: The derivatives must also agree at the knots.

A **cubic spline** (with only $K + 4$ parameters to fit) is a regression spine which is C^2 on its domain.

Let ξ be a knot and X be a predictor value. The **positive part** function is defined by

$$w_+ = \begin{cases} w & \text{if } w > 0 \\ 0 & \text{else} \end{cases}$$

Formally, the **linear spline** requires $\xi_1, \ldots, \xi_K$ knots and has $K+1$ effective parameters. The model can be expressed simply using positive parts:

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 (X_i - \xi_1)_+ + \cdots + \beta_{K+1}(X_i - \xi_K)_+ + \varepsilon_i;$$

the **cubic spline** is:

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 X_i^2 + \beta_3 X_i^3 + \beta_4 (X_i - \xi_1)_+^3 + \cdots + \beta_{K+3}(X_i - \xi_K)_+^3 + \varepsilon_i,$$

and the **natural cubic spline** is a cubic spline between ξ_1 and ξ_K, with linear extrapolation beyond ξ_1 and ξ_K; this adds 4 extra constraints to the cubic spline and allows for more knots while keeping the number of effective parameters identical to that of the linear spline.

In all instances, the machinery of OLS remains available: predictions, diagnostics, remedial measures, confidence intervals, and extension to logistic regression, as needed.

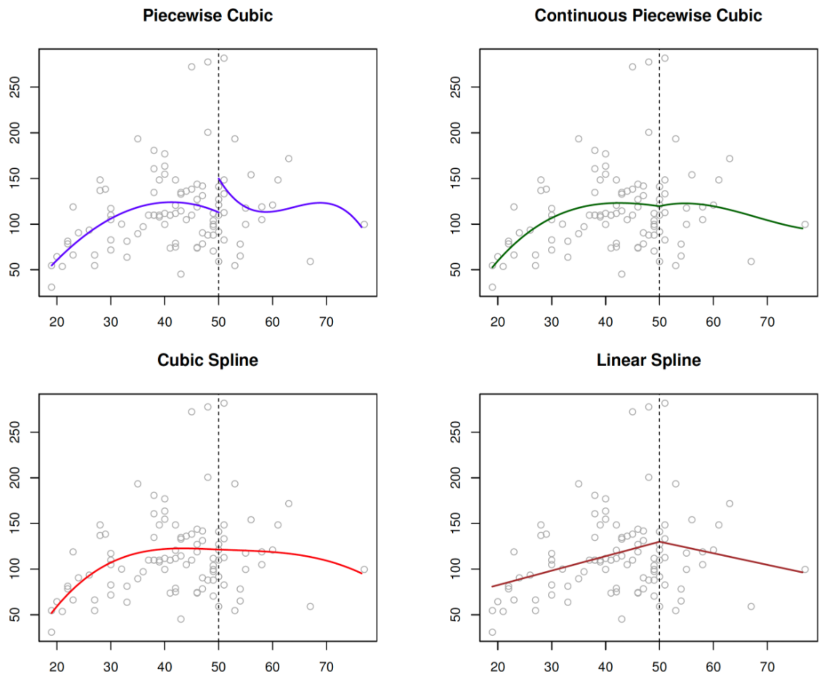


Figure 20.11: Various splines on a 1-dimensional dataset, with a single knot [3].

Gapminder Example The charts below show cubic splines for life expectancy against fertility in the 2011 Gapminder data.[59]

59: Assuming again that the training set Tr is the entire dataset.

Cubic splines are modeled using the `splines` package `bs()` function. In theory, we place more knots in locations where the spline function is believed to vary more rapidly, and fewer knots where it is more stable.[60]

60: In practice, the knots are placed uniformly at **quantiles** of the predictor variable X, based on their number.

The syntax for the OLS model formula in `R` follows the form

```
response ~ splines::bs(predictor, df)
```

where the **degrees of freedom** `df` are linked to the number of parameters to estimate (in the case of cubic spline, df = $K + 3$). We start by building a cubic spline with $K = 0$ knot.[61]

61: So $K + 3 = 3$ degrees of freedom.

```
lm(life_expectancy ~ splines::bs(fertility, df = 3))
```

```
Coefficients:
                    (Intercept)  splines::bs(fertility, df = 3)1
                          79.28                           -11.47
splines::bs(fertility, df = 3)2  splines::bs(fertility, df = 3)3
                         -27.41                           -16.65
```

Here is a cubic spline with $K = 10$ knots, with their locations:[62]

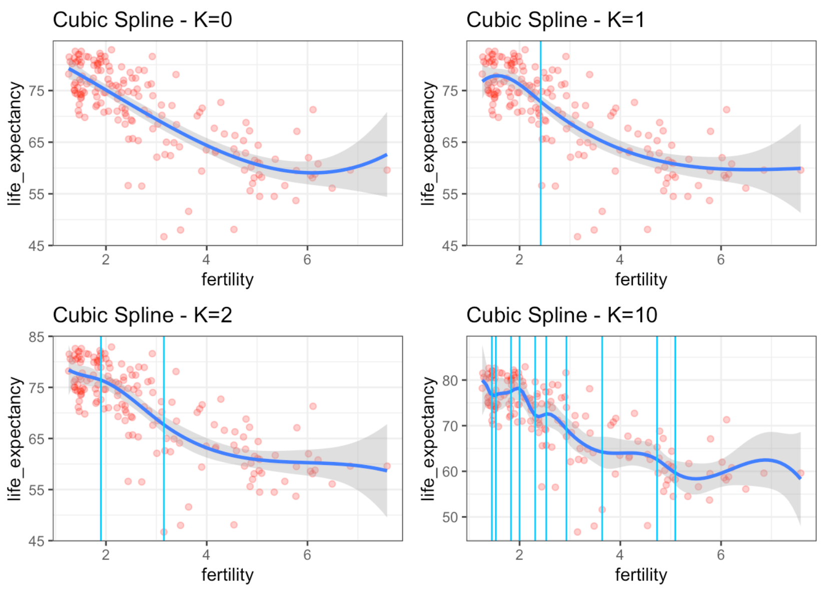
62: We can find the knot locations of a cubic spline with $K = 1, 2$ knots by computing `fm1`, `fm2`, test1, `test2`, `g1`, and `g2` in the same manner (these quantities are required in the display code on the next few pages).

```
fm10 <- lm(life_expectancy ~ splines::bs(fertility, df = 13))
test10 <- eval(attr(fm10$terms, "predvars"))
(g10 <- as.numeric(attr(test10[[2]],"knots")))
```

```
 [1] 1.45 1.53 1.83 2.00 2.31 2.53 2.93 3.64 4.73 5.09
```

We can display cubic splines with $K = 0, 1, 2, 10$ knots as below:

```
plot1 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm", formula=y~splines::bs(x,df=3)) +
  ggtitle("Cubic Spline - K=0")

plot2 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm", formula=y~splines::bs(x,df=4)) +
  geom_vline(xintercept = g1, colour = "deepskyblue") +
  ggtitle("Cubic Spline - K=1")

plot3 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm", formula=y~splines::bs(x,df=5)) +
  geom_vline(xintercept = g2, colour = "deepskyblue") +
  ggtitle("Cubic Spline - K=2")

plot4 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm", formula=y~splines::bs(x,df=13)) +
  geom_vline(xintercept = g10, colour = "deepskyblue") +
  ggtitle("Cubic Spline - K=10")

gridExtra::grid.arrange(plot1, plot2, plot3, plot4, ncol=2)
```



Natural cubic splines are modeled using the `splines` package `ns()` function; the knots are, again, placed uniformly at **quantiles** of the predictor variable X, based on their number.[63]

63: The knot locations are thus the same as in the cubic spline case.

The syntax for the OLS model formula in `R` follows the form

```
response ~ splines::ns(predictor, df)
```

where the **degrees of freedom** `df` are linked to the number of parameters to estimate (in the case of natural cubic spline, $\text{df} = K + 1$). We start by building a natural cubic spline with $K = 0$ knot.[64]

64: So $K + 1 = 1$ degrees of freedom.

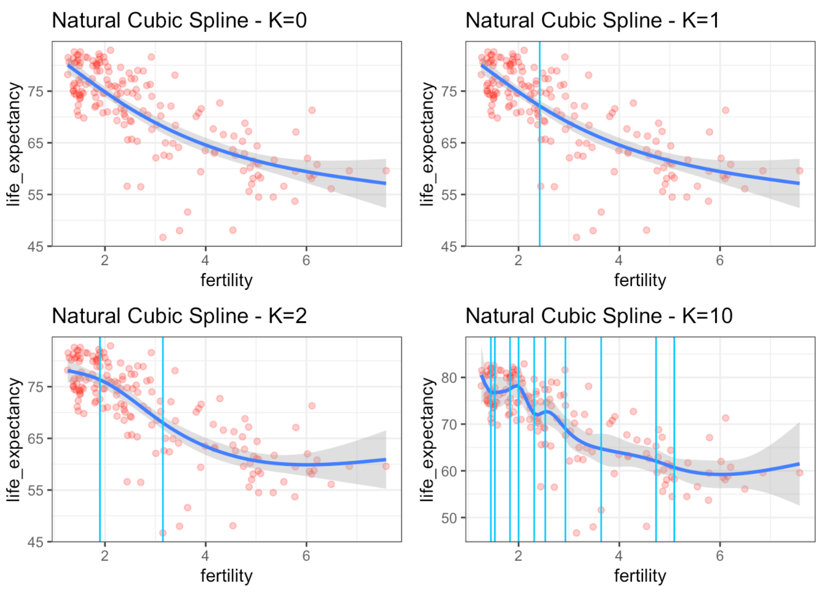
```
lm(life_expectancy ~ splines::ns(fertility, df = 1))
```

```
Coefficients:
                          (Intercept)  splines::ns(fertility, df = 1)
                                78.09                          -34.27
```

The natural cubic splines with $K = 0, 1, 2, 10$ are displayed below:

```
plot1 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm", formula=y~splines::ns(x,df=2)) +
  ggtitle("Natural Cubic Spline - K=0")

plot2 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm",
   formula=  y ~ splines::ns(x, knots = g1, df = 2)) +
  geom_vline(xintercept = g1, colour = "deepskyblue") +
  ggtitle("Natural Cubic Spline - K=1")
```

```
plot3 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm",
   formula=  y ~ splines::ns(x, knots = g2, df = 3)) +
  geom_vline(xintercept = g2, colour = "deepskyblue") +
  ggtitle("Natural Cubic Spline - K=2")

plot4 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm",
   formula=  y ~ splines::ns(x, knots = g10, df = 11)) +
  geom_vline(xintercept = g10, colour = "deepskyblue") +
  ggtitle("Natural Cubic Spline - K=10")

gridExtra::grid.arrange(plot1, plot2, plot3, plot4, ncol=2)
```



Do you notice any difference in the shape of the cubic splines vs. that of the natural cubic splines?[65]

65: Cross-validation (again!) can be used to determine the optimal K: compute the estimated error for various K (10-fold CV, say), and pick the K^* that minimizes the error.

Regression splines often give better results than polynomial regression because they induce **flexibility** *via* a large number of parameters K with low polynomial degree $d \leq 3$, rather than through high d of the latter (and the wild variability that such polynomials have, especially near the boundaries of the predictor's range, as can be observed in the polynomial regression examples above).

Multivariate Adaptive Regression Splines We can reduce the polynomial degree to $d \leq 2$ without losing too much curve fitting accuracy by considering bases consisting of functions of the forms:

$$1, (x - \xi_k)_\pm, (x - \xi_{k_1})_\pm (x - \xi_{k_2})_\pm,$$

where $(x-t)_\pm$ is one of the two **hinge functions**:

$$(x-t)_+ = \begin{cases} x-t & \text{if } x > t \\ 0 & \text{else} \end{cases} \qquad (x-t)_- = \begin{cases} t-x & \text{if } x < t \\ 0 & \text{else} \end{cases}$$

$(x-1)_\pm$, $(x-1)_+(x-5)_+$, $(x-1)_+(x-8)_-$ are shown in Figure 20.12.

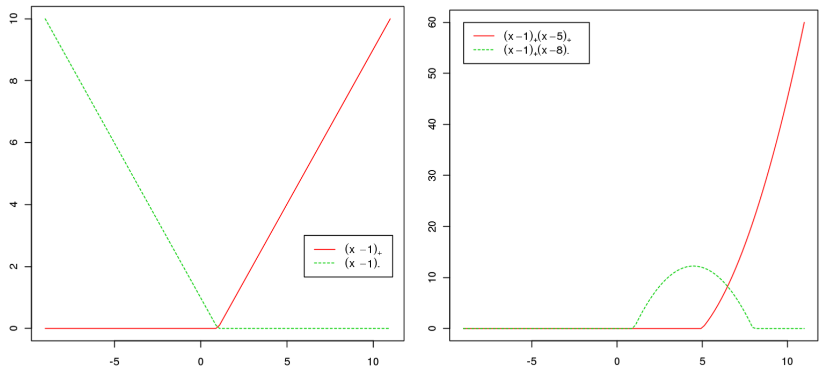


Figure 20.12: A few hinge functions.

A **multivariate adaptive regression spline** (MARS) is expressed as

$$y_i = \sum_{k=1}^{K} \beta_k h_k(x_i) + \varepsilon_i, \quad i = 1, \ldots, N,$$

where h_k is either a constant function, a hinge function, or a product of hinge functions.

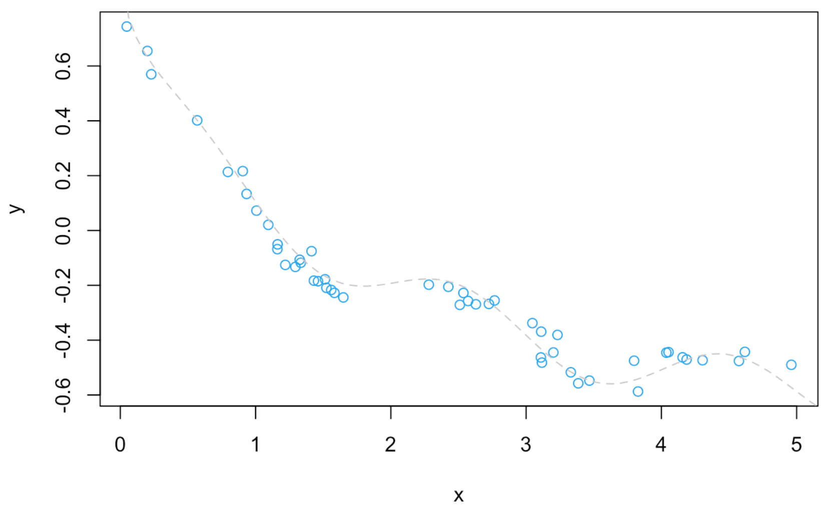
MARS adds terms to its model in an iterative fashioin; once a stopping criterion is met, unwanted terms are removed. The model growth's parallels the growth of tree-based models, which we will discuss in Chapter 21, and it has the same advantage that the knots are selected automatically.

Artificial Dataset Example Let us take a look at a synthetic dataset, based off of:

$$y = f(x) = \frac{\sin(\pi x)}{10} - \sqrt{x} + \exp(x/10) + \varepsilon,$$

where $\varepsilon \sim \mathcal{N}(0, 0.04^2)$.

```
set.seed(1234)
fx=function(x){
  sin(pi*x)/10-sqrt(x)+exp(x/10)
  }

x=sort(runif(50, 0, 5))
noise=rnorm(50, 0, 0.04)
y=fx(x)+noise

plot(x, y, col=4)
x.vec=seq(0,6, length.out=100)
lines(x.vec, fx(x.vec), col="grey", lty=2)
```



We can fit the data using package `mda`'s `mars()` function, in `R`.[66] Let us use only functions of degree 1 (linear functions and linear hinges, but no interaction terms) for the time being:

66: This is a licensed implementation of MARS. There is another implementation in the `earth` package: **enhanced adaptive regression through hinges**, or EARTH.

```
MARS.1 = mda::mars(x, y, degree=1)
```

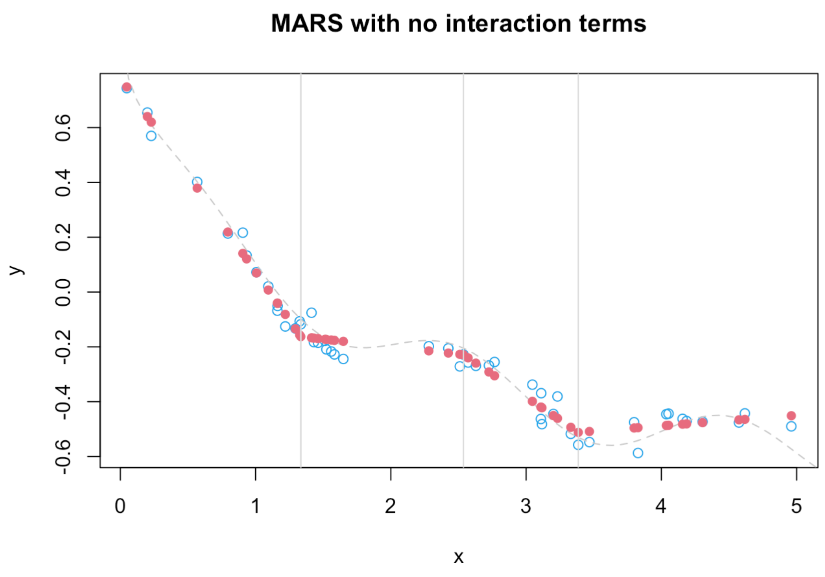
The output is rather lengthy and is suppressed for readability.[67]

67:

- `$call` provides the model;
- `$fitted.values` contains the estimated values $\hat{y}_i$;
- `$residuals` contain the residuals $\hat{y}_i - y_i$, and
- `$x` gives the hinge functions used in the final model.

Let's see how good a job MARS did:

```
plot(x, y, col=4, main="MARS with no interaction terms")
x.vec=seq(0,6, length.out=100)
lines(x.vec, fx(x.vec), col="grey", lty=2)
points(x, MARS.1$fitted.values, col=2, pch=16)
abline(v = MARS.1$cuts[MARS.1$selected.terms[-1]],
       col = "light grey")
```



Not bad, all things considered.

The EARTH output is identical, and would be obtained thus:

```
EARTH.1 = earth::earth(x, y, degree=1)
summary(EARTH.1)
```

```
             coefficients
(Intercept)   -0.16254461
h(1.3341-x)    0.70785002
h(x-1.3341)   -0.05502561
h(x-2.53653)  -0.27853205
h(x-3.38547)   0.37209809

Selected 5 of 6 terms, and 1 of 1 predictors
Termination condition: RSq changed by less than 0.001 at 6 terms
Importance: x
Number of terms at each degree of interaction: 1 4 (additive model)
GCV 0.002341396    RSS 0.07871774    GRSq 0.9759754    RSq 0.9831798
```

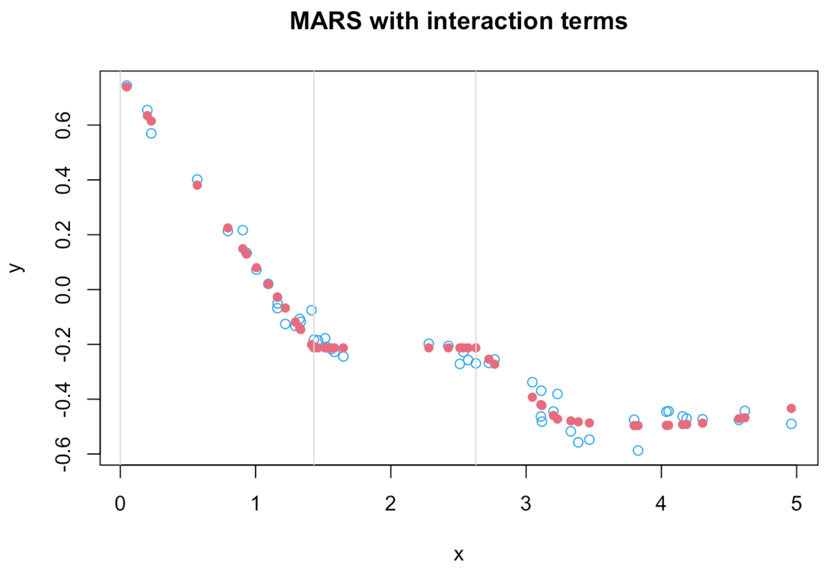
What about interaction terms? In order for MARS or EARTH to consider such terms, we must first provide a second predictor.

```
xnew = x*x
data = data.frame(x,xnew,y)
EARTH.2 = earth::earth(y ~ x + xnew , data=data, degree=2)
summary(EARTH.2)
```

```
               coefficients
(Intercept)     -0.21273261
h(1.43112-x)     0.68806424
h(x-2.62849)    -0.43057541
h(xnew-10.446)   0.05531043

Selected 4 of 6 terms, and 2 of 2 predictors
Termination condition: RSq changed by less than 0.001 at 6 terms
Importance: x, xnew
Number of terms at each degree of interaction: 1 3 (additive model)
GCV 0.00273995    RSS 0.09437758    GRSq 0.971886    RSq 0.9798336
```

What does the plot look like? Can you spot the non-linear components?

```
plot(x, y, col=4, main="MARS with interaction terms")
x.vec=seq(0,6, length.out=100)
points(x, EARTH.2$fitted.values, col=2, pch=16)
abline(v = EARTH.2$cuts[EARTH.2$selected.terms[-1]],
       col = "light grey")
EARTH.2$cuts
```



Housing Dataset Example In this section, we analyze a housing dataset related to house selling prices in Ames, Iowa (VE_Housing.csv, modified from [233]). We start by reading in the data:

```
dat.Housing=read.csv("VE_Housing.csv", header=TRUE,
                     stringsAsFactors = TRUE)
dim(dat.Housing)
```

```
[1] 1460   81
```

Next, we count the number of missing values for each variable, excluding those variables with complete rows.

```
missing = attributes(which(apply(is.na(dat.Housing), 2,
           sum)>0))$names
apply(is.na(dat.Housing[,missing]), 2, sum)
```

```
 LotFrontage        Alley   MasVnrType   MasVnrArea     BsmtQual     BsmtCond
         259         1369            8            8           37           37
BsmtExposure BsmtFinType1 BsmtFinType2   Electrical  FireplaceQu   GarageType
          38           37           38            1          690           81
 GarageYrBlt GarageFinish   GarageQual   GarageCond       PoolQC        Fence
          81           81           81           81         1453         1179
 MiscFeature
        1406
```

The housing dataset thus consists of $n = 1460$ observations with $p = 79$ predictors. There are two other variables: `Id` and `SalePrice`, representing the index variable and the response variable, respectively.[68] Furtheremore, the variables

68: Use `colnames()` or `str()` to list all the variables.

- `LotFrontage`
- `Alley`

- `FireplaceQu`
- `PoolQC`
- `Fence`, and
- `MiscFeature`

all have anywhere from 259 to 1406 missing observations. The proportions of missing values in these variables are probably too high for imputation (see Chapter 15 for details), so we elect to remove them from further analyses.

Note that the remaining major missing variables are all related to *Garage* and *Basement*, with corresponding variables missing for the same houses. Given that there are other variables associated with these, we suspect these variables will not play a crucial role in model building, and we also elect to remove them from the analyses.

For the remaining three variables with missing values (`MasVnrType`, `MasVnrArea`, and `Electrical`), the number of missing observations are so small that we could easily

- impute these values, or
- perform list-wise deletion.

For the purposes of this example, we will select the latter options and delete all columns with missing values.

```
dat.Housing.new = dat.Housing[,
       !colnames(dat.Housing)%in%missing]
dim(dat.Housing.new)
```

```
[1] 1460   62
```

We also remove the index variable `ID`:

```
dat.Housing.new = subset(dat.Housing.new, select = -c(Id))
```

In order to evaluate the effectiveness of the eventual model (i.e., to have good predictive power without overfitting the data), we split the Housing dataset into training and testing sets. The model is then developed using the training set (i.e., optimized using a subset of data), and then later tested for its prediction power using the testing set.

We select roughly 80% of the observations (1160) for the training set:

```
set.seed(1234) # for replicability
n.train=1160
ind.train=sample(1:nrow(dat.Housing.new), n.train)
```

The training and testing sets are thus:

```
dat.train=dat.Housing.new[ind.train,]
dat.test=dat.Housing.new[-ind.train,]
```

We train EARTH (with interactions) on the training data:

```
EARTH.3 <- earth::earth(SalePrice~., data=dat.train,
                        degree=2)
summary(EARTH.3)
```

```
                                        coefficients
(Intercept)                                317.95604
Exterior1stBrkFace                          18.17930
FoundationPConc                             34.63490
h(14442-LotArea)                            -0.00198
h(LotArea-14442)                             0.00048
...
h(7-OverallCond) * h(316-WoodDeckSF)        -0.01602
h(2005-YearBuilt) * h(1056-BsmtFinSF1)       0.00030
h(YearBuilt-2005) * h(1056-BsmtFinSF1)      -0.00928

Selected 36 of 39 terms, and 19 of 188 predictors
Termination condition: RSq changed by less than 0.001 at 39 terms
Importance: OverallQual, GrLivArea, YearBuilt, SaleTypeWD, BsmtFinSF1, ...
Number of terms at each degree of interaction: 1 18 17
GCV 396.2527    RSS 392191.8    GRSq 0.9377484    RSq 0.9467931
```

We now predict `SalePrice` on the testing data:

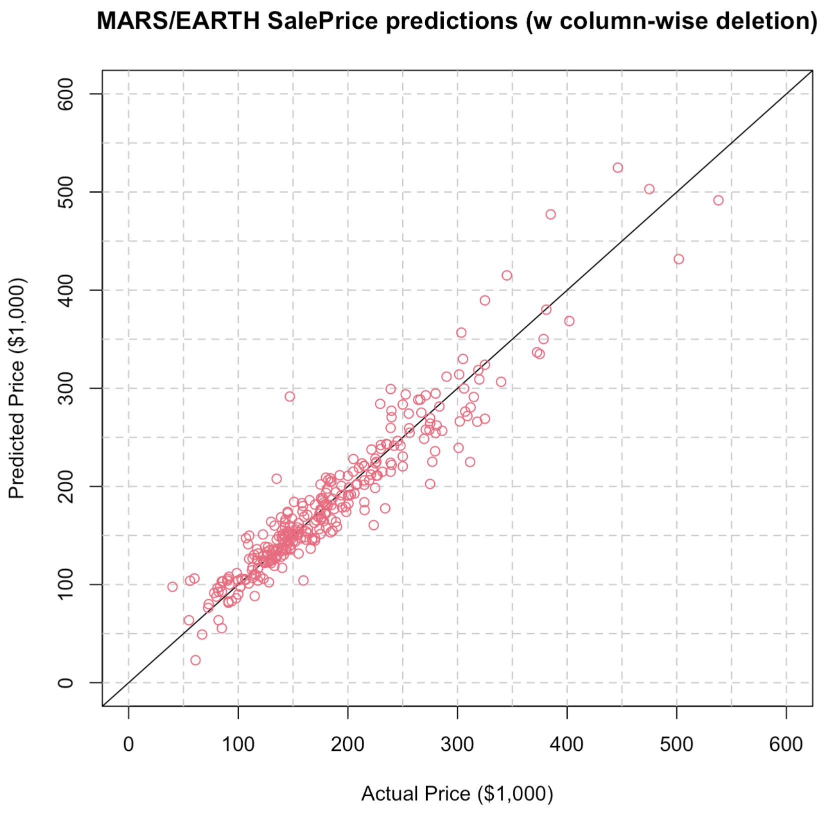
```
yhat.EARTH.3 = predict(EARTH.3, dat.test)
```

We can evaluate the quality of the predictions on the testing set either by computing MSE_{Te} directly (≈ 628), but this value is more or less useless on its own.

We get a better sense for the quality of the prediction on the testing set by comparing the actual `SalePrice` values to the EARTH predicted `SalePrice` values:

```
xlimit = ylimit = c(0,600)
plot(NA, col=2, xlim=xlimit, ylim=ylimit,
     ylab="Predicted Price ($1,000)",
     xlab="Actual Price ($1,000)",
     main="MARS/EARTH SalePrice predictions
           (w column-wise deletion)")
abline(h=seq(0,600, length.out=13), lty=2, col="grey",
       v=seq(0,600, length.out=13))
abline(a=0, b=1)
points(dat.test$SalePrice, yhat.EARTH.3, col=2)
```

(see plot on the next page) What do you think? Is the model likely to prove useful?



Smoothing Splines Given a training set Tr with N observations, we have seen that **regression splines** use the following approach:

1. identify K knots $\xi_1, \ldots, \xi_K$;
2. produce some basis functions $\{b_1(x), \ldots, b_K(x)\}$, and
3. use OLS to estimate the coefficients of

$$Y_i = \beta_0 + \beta_1 b_1(X_i) + \cdots + \beta_K b_k(X_i) + \varepsilon_i, \quad i = 1, \ldots, N.$$

But we can use another mathematical approach in order to produce a spline. In general, we need to find a function g that provides a good fit for the available data; in other words, we are looking for a g for which

$$\text{SSE} = \sum_{i=1}^{N} (Y_i - g(X_i))^2 \quad \text{is "small".}$$

But we also need g to be constrained, otherwise any smooth function interpolating (X_y, Y_i), $i = 1, \ldots, N$ would yield SSE $= 0$, at the cost of **severe overfitting** and loss of **interpretability**, as in Figure 20.13. The flip side is that too many constraints can result in the data being **underfit**.

The **smoothing spline** approach seeks to solve the following problem:

$$g_\lambda = \arg\min_h \left\{ \underbrace{\sum_{i=1}^{N} (Y_i - h(X_i))^2}_{\text{SSE loss}} + \lambda \underbrace{\int_{\Omega(X)} [h''(t)]^2 \, dt}_{\text{penalty term}} \right\},$$

where $\lambda \geq 0$ is a **tuning parameter** and $\Omega(X)$ represents the range of the predictor X.

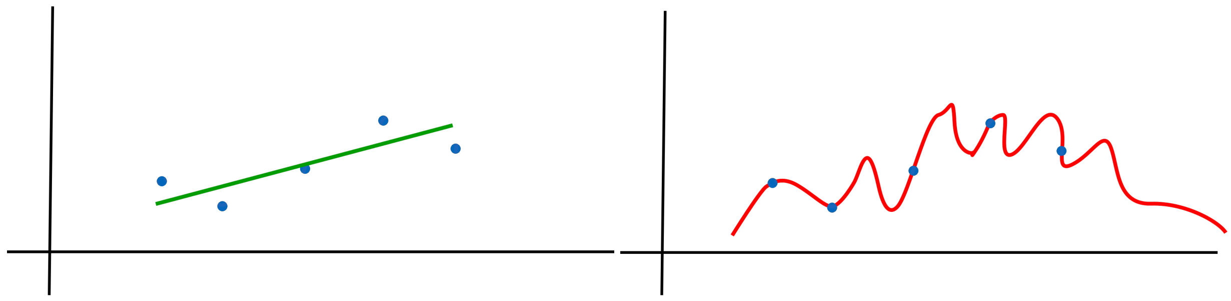


Figure 20.13: A spline with too few constraint overfits the data (right).

The **penalty term** measures the roughness of the spline function h; if h is quite "wiggly", the penalty will be (relatively) large, and **vice-versa** (and similarly for g).[69]

69: If h represents a straight line, say, the penalty term would be zero.

When $\lambda \longrightarrow 0$, the penalty term has little effect, so we would expect g_λ to be "jumpy" in such cases and that it would interpolate the observations exactly, leading to overfitting.

When $\lambda \longrightarrow \infty$, the penalty term dominates and g_λ is a function for which $\int [g''_\lambda(t)]^2 dt \longrightarrow 0$ over $\Omega(X)$, so $g_\lambda \longrightarrow$ linear OLS solution over $\Omega(X)$, leading to underfitting.

As we have seen over and over again, the tuning parameter λ controls the bias-variance trade-off, expressed, in this case, as a battle between rigidity and model complexity.

The **optimal smoothing spline** g_λ is a natural cubic spline with a knot at every data point $\xi_i = x_i$, $i = 1, \ldots, N$, with continuous 0th, 1st, 2nd derivatives throughout the range $\Omega(X) = [\min \xi_i, \max \xi_i]$, and is linear outside $\Omega(X)$, but, importantly, **it is not the one that would be obtained from building a regression spline**, as it also depends on the turning parameter λ.

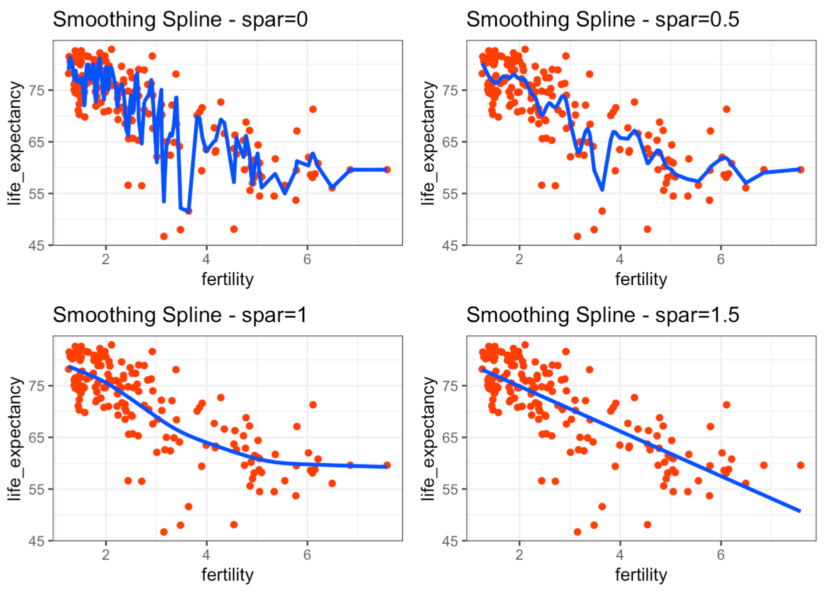
What is the best choice for λ? At first glance, this would seem to be another job for cross-validation, but there is another option: we can specify the smoothing spline through the **effective degrees of freedom**, which decrease from N to 2 as λ goes from 0 to ∞ (note, however, that R's `smooth.spline()` uses a different parameterization).

Gapminder Example The charts below show the smoothing spline for life expectancy against fertility in the 2011 Gapminder data, for 4 different smoothing parameter values, using `stats`'s `smooth.spline()` function. Note that the entire set is used as training data.

```
x=gapminder.2011$fertility
y=gapminder.2011$life_expectancy

ss00 = stats::smooth.spline(x, y, spar=0)
ss05 = stats::smooth.spline(x, y, spar=0.5)
ss10 = stats::smooth.spline(x, y, spar=1)
ss15 = stats::smooth.spline(x, y, spar=1.5)
```

In order to be able to display the smoothing splines over the datapoints, we use the broom::augment() function, which provide the value of the spline at the various fertility values in the dataset.

```
plot1 <- ggplot(broom::augment(ss00, gapminder.2011),
                aes(x=fertility, y=life_expectancy)) +
  geom_point(colour="red") + theme_bw() +
  geom_line(aes(y = .fitted), colour="blue", size=1.1) +
  ggtitle("Smoothing Spline - spar=0")

plot2 <- ggplot(broom::augment(ss05, gapminder.2011),
                aes(x=fertility, y=life_expectancy)) +
  geom_point(colour="red") + theme_bw() +
  geom_line(aes(y = .fitted), colour="blue", size=1.1) +
  ggtitle("Smoothing Spline - spar=0.5")

plot3 <- ggplot(broom::augment(ss10, gapminder.2011),
                aes(x=fertility, y=life_expectancy)) +
  geom_point(colour="red") + theme_bw() +
  geom_line(aes(y = .fitted), colour="blue", size=1.1) +
  ggtitle("Smoothing Spline - spar=1")

plot4 <- ggplot(broom::augment(ss15, gapminder.2011),
                aes(x=fertility, y=life_expectancy)) +
  geom_point(colour="red") + theme_bw() +
  geom_line(aes(y = .fitted),  colour="blue", size=1.1) +
  ggtitle("Smoothing Spline - spar=1.5")

gridExtra::grid.arrange(plot1, plot2, plot3, plot4, ncol=2)
```



Note the evolution of a flexible but highly non-interpretable model (the wiggly curve associated to spar=0) into a rigid but highly interpretable model (the line associated to spar=1.5) as the spar values increase.

20.5.3 Generalized Additive Models

While polynomial regression and splines can be applied to predictor sets, they are best-suited to predicting a response Y on the basis of a **single predictor** X (the model complexity increases quickly if more than one predictor is present).

Generalized additive models (GAM) allow for flexible non-linearities in several variables while retaining the **additive structure** of linear models:

$$y_i = \beta_0 + f_1(x_{i,1}) + \cdots + f_p(x_{i,p}) + \varepsilon_i, \quad i = 1, \ldots, N$$

where each of the f_j can be derived using any of the methods previously discussed; if

$$\begin{aligned} f_1(x_i) &= \beta_{1,1}b_{1,1}(x_{i,1}) + \cdots + \beta_{1,L_1}b_{1,L_1}(x_{i,1}) \\ &\vdots \\ f_p(x_i) &= \beta_{p,1}b_{p,1}(x_{i,p}) + \cdots + \beta_{p,L_p}b_{p,L_p}(x_{i,p}), \end{aligned}$$

say, we would fit the data using OLS (but this cannot be done if one of the components is a smoothing spline, for instance, or if it is non-linear in some other way).

In practice, using natural cubic splines for the quantitative components seem to work as well as smoothing spline, when it comes to making predictions.[70]

70: GAM can also be used for classification via log-odds:

$$\ln\left(\frac{p_1(\mathbf{x})}{1-p_1(\mathbf{x})}\right) = \beta_0 + f_1(x_1) + \cdots + f_p(x_p).$$

GAM are implemented in `R` using the `mgcv::gam()` function; a typical call might look like:

```
mgcv::gam(y ~ s(x1,df=5) +
              lo(x2,spar=0.5) +
              bs(x3,df=4) +
              ns(x4,df=5):ns(x5,df=5) +
              x6, data=dat)
```

which would indicate that the contribution of:

- X_1 is given by smoothing spline with 5 degrees of freedom,
- X_2 is given by a local regression with `spar`=0.5,
- X_3 is given by a cubic spline with 4 degrees of freedom,
- the fourth component is an interaction term based on natural splines for X_4 and X_5 (each with 5 degrees of freedom), and
- X_6 is directly added to the model.

GAM provide a useful compromise between linear models and fully non-parametric models.

Advantages:

- GAM can fit a non-linear f_j to each predictor X_j, so that they could capture trends that linear regression would miss;
- GAM can reduce the number of data transformations to try out manually on each predictor X_j;

- non-linear fits may improve accuracy of predictions for the response Y;
- GAM are useful for inference due to their additivity – the effect of X_j on Y (while keeping other predictors fixed) can be analyzed separately;
- the overall smoothness of the model can be summarized *via* effective degrees of freedom/parameters.

Disadvantages:

- GAM still suffer from the curse of dimensionality;
- GAM are restricted to additive models – interaction terms can be added manually by introducing new predictors $X_j \times X_k$, as can interaction functions $f_{j,k}(X_j, X_k)$ (using local regression or MARS, say), but they quickly get out of hand (due to *Curse of Dimensionality* issues).

Gapminder Example The charts below show the individual contributions of fertility, infant mortality, GDP, and continental membership to life expectancy in the 2011 Gapminder data.[71]

71: Using the entire set as training data.

```
library(mgcv)
b <- gam(gapminder.2011$life_expectancy ~
         s(gapminder.2011$fertility) +
         s(gapminder.2011$infant_mortality) +
         s(gapminder.2011$gdp) +
         gapminder.2011$continent)
summary(b)
```

```
Family: gaussian
Link function: identity

Formula:
gapminder.2011$life_expectancy ~ s(gapminder.2011$fertility) +
    s(gapminder.2011$infant_mortality) + s(gapminder.2011$gdp) +
    gapminder.2011$continent

Parametric coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)        68.1186     0.7470  91.190  < 2e-16 ***
continentAmericas   4.4787     1.1161   4.013 9.30e-05 ***
continentAsia       4.7110     0.9993   4.714 5.35e-06 ***
continentEurope     3.4179     1.3209   2.588   0.0106 *
continentOceania   -0.2891     1.3798  -0.210   0.8343
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
                                     edf Ref.df      F p-value
s(gapminder.2011$fertility)        1.000  1.000  4.474   0.036 *
s(gapminder.2011$infant_mortality) 3.027  3.800 40.541  <2e-16 ***
s(gapminder.2011$gdp)              1.478  1.779  0.367   0.575
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.828   Deviance explained = 83.8%
GCV = 13.199  Scale est. = 12.363    n = 166
```

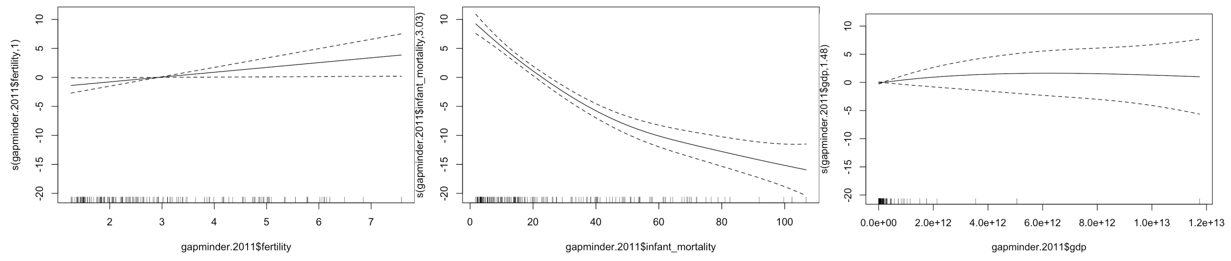
We see in the outcome that the intercept is $\beta_0 = 68.1186$ and that

$$\beta_{\text{continent}} = \begin{cases} 0 & \text{Africa} \\ 4.4787 & \text{America} \\ 4.7110 & \text{Asia} \\ 3.4179 & \text{Europe} \\ -0.2891 & \text{Oceania} \end{cases}$$

so that predictions take the form

$$\begin{aligned}\text{life expectancy} \approx \beta_0 + f_1(\text{fertility}) + f_2(\text{infant mortality}) \\ + f_3(\text{gdp}) + \beta_{\text{continent}}.\end{aligned}$$

```
plot.gam(b)
```



For instance, the life expectancy for an American country with fertility= 3, infant mortality= 1, GDP= 6×10^{12} would be approximately

$$68.1 + 0 + 10 + 2 + 4.5 = 84.6.$$

Take the time to read the `mgcv` and the `gam` documentation to better understand how these work in practice (in particular, how to make predictions on test/new observations).

20.6 Example: Algae Blooms

We continue the algae blooms analysis started in Section 15.7 (based on a case study by L.Torgo [103]). The objective is to predict various algae levels in water samples; we continue the analysis with the data frame `algae_blooms.sna2`.

20.6.1 Value Estimation Modeling

For **supervised learning** tasks, the *bias-variance* trade-off means that we need to set aside a **testing set** on which the models (which were learned on the **training set**) are evaluated.

There are no hard-and-fast rules to determine how to split the data; if the signal is very strong, a small training set should capture its important features, but we do not typically know how strong the signal is before we start the modeling process. On the other hand, if the training set is too large, we run the risk of overfitting the data. Weighing both of these considerations, we elect to use a 65%/35% training/testing split.

The training data should also be **representative**, to avoid injecting biases in the model (in case the data was provided according to some systematic but unknown rule).

There are numerous ways to do this,[72] but we can do so using a simple random sample of 218 observations.[73]

To avoid issues related to replicability, we use a single training set.[74]

72: See Chapter 11, *Survey Sampling Methods*, for instance.

73: We could also have stratified according to season, size, etc.

74: The code that would allow for a different random sample every time the code is run has been commented out in the following code box.

```
# ind <- sample(1:dim(algae_blooms.sna2)[1], 218)
ind <- 1:218
algae.train <- algae_blooms.sna2[ind,] # training set
algae.test <- algae_blooms.sna2[-ind,] # testing set
set.seed(0)  # for replicability
```

Generalized Linear Model We implement a linear model to predict a2 (to pick but one of the response variables) against all the predictor variables, but only using the training set.[75]

75: Before getting too excited about using various machine learning methods, it is worth seeing what the traditional approaches yield.

```
linear.model.a2 <- lm(a2 ~ season + size + speed + mxPH +
        mnO2 + Cl + NO3 + NH4 + oPO4 + PO4 + Chla,
        data=algae.train)
```

A summary of the results can be given by calling the `summary` method on the resulting object.

```
summary(linear.model.a2)
```

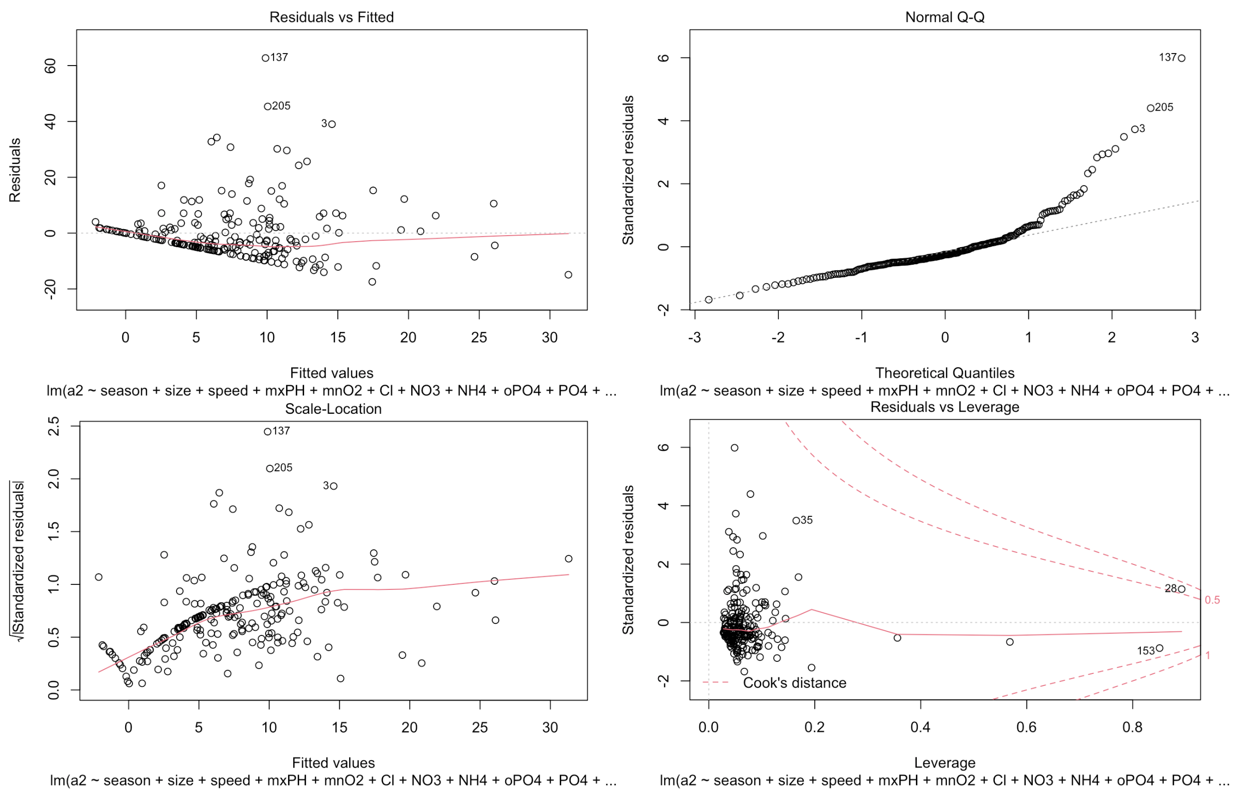
```
Residuals:
    Min      1Q  Median      3Q     Max
-17.436  -5.281  -2.613   2.026  62.712

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)  -3.083e+01  1.257e+01  -2.452 0.015056 *
seasonsummer -1.166e-01  2.112e+00  -0.055 0.956035
seasonautumn  1.071e+00  2.370e+00   0.452 0.651934
seasonwinter -1.451e+00  2.000e+00  -0.726 0.468935
sizemedium   -2.628e+00  1.895e+00  -1.387 0.166896
sizelarge    -3.210e+00  2.412e+00  -1.331 0.184767
speedmedium   3.887e+00  2.485e+00   1.564 0.119325
speedhigh    -1.104e+00  2.772e+00  -0.398 0.690751
mxPH          4.859e+00  1.559e+00   3.117 0.002092 **
mnO2         -1.841e-01  3.924e-01  -0.469 0.639474
Cl           -7.432e-03  2.006e-02  -0.371 0.711351
```

```
NO3          2.132e-01  3.028e-01   0.704 0.482249
NH4         -5.979e-04  5.355e-04  -1.117 0.265510
oPO4         2.290e-03  9.876e-03   0.232 0.816875
PO4         -1.559e-03  5.936e-03  -0.263 0.793090
Chla         1.652e-01  4.614e-02   3.579 0.000432 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.74 on 202 degrees of freedom
Multiple R-squared:  0.206, Adjusted R-squared:  0.147
F-statistic: 3.493 on 15 and 202 DF,  p-value: 2.498e-05
```

We see that the adjusted R^2 coefficient is fairly small, which is not ideal. Furthermore, the residuals should have a mean of 0 and be "small", which is not quite what we are seeing here; the $F-$statistic seems to indicate that there is some (linear) dependence on the predictor variables.

We can get a better handle on the regression diagnostics by calling the `plot()` method on the object.

```
plot(linear.model.a2)
```



All in all, the linear model is ... not great. The significance of some of the coefficients is questionable, however, and we might wonder what effect their inclusion might have.

```
anova(linear.model.a2)
```

```
Analysis of Variance Table
Response: a2
           Df  Sum Sq Mean Sq F value    Pr(>F)
season      3   112.3   37.42  0.3243 0.8078029
size        2   436.0  217.99  1.8892 0.1538604
speed       2  1552.8  776.42  6.7287 0.0014825 **
mxPH        1  2223.5 2223.54 19.2698 1.829e-05 ***
mnO2        1     0.5    0.54  0.0047 0.9455025
Cl          1     0.3    0.33  0.0029 0.9572795
NO3         1    43.9   43.91  0.3806 0.5380001
NH4         1   193.8  193.82  1.6797 0.1964428
oPO4        1     0.1    0.09  0.0008 0.9775762
PO4         1     4.8    4.82  0.0417 0.8383141
Chla        1  1478.2 1478.18 12.8103 0.0004316 ***
Residuals 202 23308.8  115.39
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
```

We might be interested in the results of a linear regression with the NH4 predictor removed, say.

```
linear.model.a2.mod <- update(linear.model.a2, . ~ . -NH4)
summary(linear.model.a2.mod)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-16.801  -5.500  -2.647   2.504  63.259

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)  -30.996221  12.580354  -2.464 0.014577 *
seasonsummer  -0.107996   2.113697  -0.051 0.959301
seasonautumn   0.806683   2.359593   0.342 0.732799
seasonwinter  -1.397244   2.000275  -0.699 0.485648
sizemedium    -2.378831   1.882645  -1.264 0.207838
sizelarge     -3.086404   2.411377  -1.280 0.202029
speedmedium    3.637403   2.476278   1.469 0.143408
speedhigh     -1.382060   2.762133  -0.500 0.617364
mxPH           4.821140   1.559264   3.092 0.002268 **
mnO2          -0.074216   0.380118  -0.195 0.845398
Cl            -0.001602   0.019376  -0.083 0.934181
NO3           -0.013968   0.224437  -0.062 0.950437
oPO4          -0.001285   0.009348  -0.137 0.890775
PO4           -0.001518   0.005940  -0.256 0.798576
Chla           0.165865   0.046166   3.593 0.000411 ***
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1

Residual standard error: 10.75 on 203 degrees of freedom
Multiple R-squared:  0.2011,    Adjusted R-squared:  0.146
F-statistic: 3.649 on 14 and 203 DF,  p-value: 2.009e-05
```

The fit is not that much better, but an ANOVA on the 2 suggested models shows that we are at least ≈ 88% certain that the models are different.

```
anova(linear.model.a2,linear.model.a2.mod)
```

```
Analysis of Variance Table
Model 1: a2 ~ season + size + speed + mxPH + mnO2 + Cl + NO3 + NH4 +
    oPO4 + PO4 + Chla
Model 2: a2 ~ season + size + speed + mxPH + mnO2 + Cl + NO3 +
    oPO4 + PO4 + Chla
  Res.Df   RSS Df Sum of Sq      F Pr(>F)
1    202 23309
2    203 23453 -1   -143.86 1.2467 0.2655
```

The `step()` function uses AIC to perform a **model search** (using **backward elimination**). The "best" linear model for a2 is thus:

```
final.linear.model.a2 <- step(linear.model.a2)
summary(final.linear.model.a2)
```

```
Start:  AIC=1050.52
a2 ~ season + size + speed + mxPH + mnO2 + Cl + NO3 + NH4 + oPO4 + PO4 + Chla

         Df Sum of Sq   RSS    AIC
- season  3    157.44 23466 1046.0
- oPO4    1      6.20 23315 1048.6
- PO4     1      7.96 23317 1048.6
- Cl      1     15.85 23325 1048.7
- mnO2    1     25.40 23334 1048.8
- NO3     1     57.19 23366 1049.0
- size    2    282.28 23591 1049.1
- NH4     1    143.86 23453 1049.9
<none>                23309 1050.5
- speed   2    967.47 24276 1055.4
- mxPH    1   1121.22 24430 1058.8
- Chla    1   1478.18 24787 1061.9

Step:  AIC=1045.98
a2 ~ size + speed + mxPH + mnO2 + Cl + NO3 + NH4 + oPO4 + PO4 + Chla

        Df Sum of Sq   RSS    AIC
- oPO4   1      2.54 23469 1044.0
- PO4    1      4.10 23470 1044.0
- mnO2   1      6.61 23473 1044.0
- Cl     1     15.59 23482 1044.1
- size   2    257.60 23724 1044.4
- NO3    1     47.04 23513 1044.4
- NH4    1    114.06 23580 1045.0
<none>               23466 1046.0
- speed  2   1035.56 24502 1051.4
- mxPH   1   1052.01 24518 1053.5
- Chla   1   1477.06 24943 1057.3
```

```
Step:  AIC=1044.01
a2 ~ size + speed + mxPH + mnO2 + Cl + NO3 + NH4 + PO4 + Chla

        Df Sum of Sq   RSS    AIC
- PO4    1      1.62 23470 1042.0
- mnO2   1      7.17 23476 1042.1
- Cl     1     14.19 23483 1042.1
- NO3    1     44.93 23514 1042.4
- size   2    266.73 23736 1042.5
- NH4    1    114.91 23584 1043.1
<none>               23469 1044.0
- speed  2   1050.55 24519 1049.5
- mxPH   1   1099.78 24569 1052.0
- Chla   1   1480.47 24949 1055.3

Step:  AIC=1042.02
a2 ~ size + speed + mxPH + mnO2 + Cl + NO3 + NH4 + Chla

        Df Sum of Sq   RSS    AIC
- mnO2   1      6.59 23477 1040.1
- Cl     1     17.42 23488 1040.2
- size   2    265.19 23736 1040.5
- NO3    1     51.04 23521 1040.5
- NH4    1    140.72 23611 1041.3
<none>               23470 1042.0
- speed  2   1050.42 24521 1047.6
- mxPH   1   1105.21 24576 1050.0
- Chla   1   1482.34 24953 1053.4

Step:  AIC=1040.08
a2 ~ size + speed + mxPH + Cl + NO3 + NH4 + Chla

        Df Sum of Sq   RSS    AIC
- Cl     1     13.41 23490 1038.2
- size   2    260.65 23738 1038.5
- NO3    1     44.48 23522 1038.5
- NH4    1    135.66 23613 1039.3
<none>               23477 1040.1
- speed  2   1121.64 24599 1046.3
- mxPH   1   1103.17 24580 1048.1
- Chla   1   1492.55 24970 1051.5

Step:  AIC=1038.21
a2 ~ size + speed + mxPH + NO3 + NH4 + Chla

        Df Sum of Sq   RSS    AIC
- NO3    1     36.13 23526 1036.5
- size   2    275.91 23766 1036.8
- NH4    1    128.31 23619 1037.4
<none>               23490 1038.2
- speed  2   1172.78 24663 1044.8
- mxPH   1   1089.85 24580 1046.1
- Chla   1   1490.94 24981 1049.6
```

```
Step:  AIC=1036.54
a2 ~ size + speed + mxPH + NH4 + Chla

        Df Sum of Sq   RSS    AIC
- size   2    244.91 23771 1034.8
- NH4    1     93.48 23620 1035.4
<none>               23526 1036.5
- speed  2   1164.36 24691 1043.1
- mxPH   1   1053.88 24580 1044.1
- Chla   1   1611.04 25138 1049.0

Step:  AIC=1034.8
a2 ~ speed + mxPH + NH4 + Chla

        Df Sum of Sq   RSS    AIC
- NH4    1     82.62 23854 1033.6
<none>               23771 1034.8
- mxPH   1    850.56 24622 1040.5
- speed  2   1085.45 24857 1040.5
- Chla   1   1540.50 25312 1046.5

Step:  AIC=1033.56
a2 ~ speed + mxPH + Chla

        Df Sum of Sq   RSS    AIC
<none>               23854 1033.6
- speed  2   1021.27 24875 1038.7
- mxPH   1    928.72 24783 1039.9
- Chla   1   1479.59 25334 1044.7

Call:
lm(formula = a2 ~ speed + mxPH + Chla, data = algae.train)

Residuals:
    Min      1Q  Median      3Q     Max
-16.195  -6.008  -2.530   2.024  63.589

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -27.13270   11.07921  -2.449 0.015134 *
speedmedium   4.17176    2.34330   1.780 0.076453 .
speedhigh    -0.32929    2.41899  -0.136 0.891850
mxPH          3.89794    1.35358   2.880 0.004387 **
Chla          0.15945    0.04387   3.635 0.000349 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.58 on 213 degrees of freedom
Multiple R-squared:  0.1874,    Adjusted R-squared:  0.1721
F-statistic: 12.28 on 4 and 213 DF,  p-value: 5.289e-09
```

It is still not a great fit (the adjusted R^2 is quite small); we conclude that the linear model is not ideal to predict a2.

```
anova(final.linear.model.a2)
```

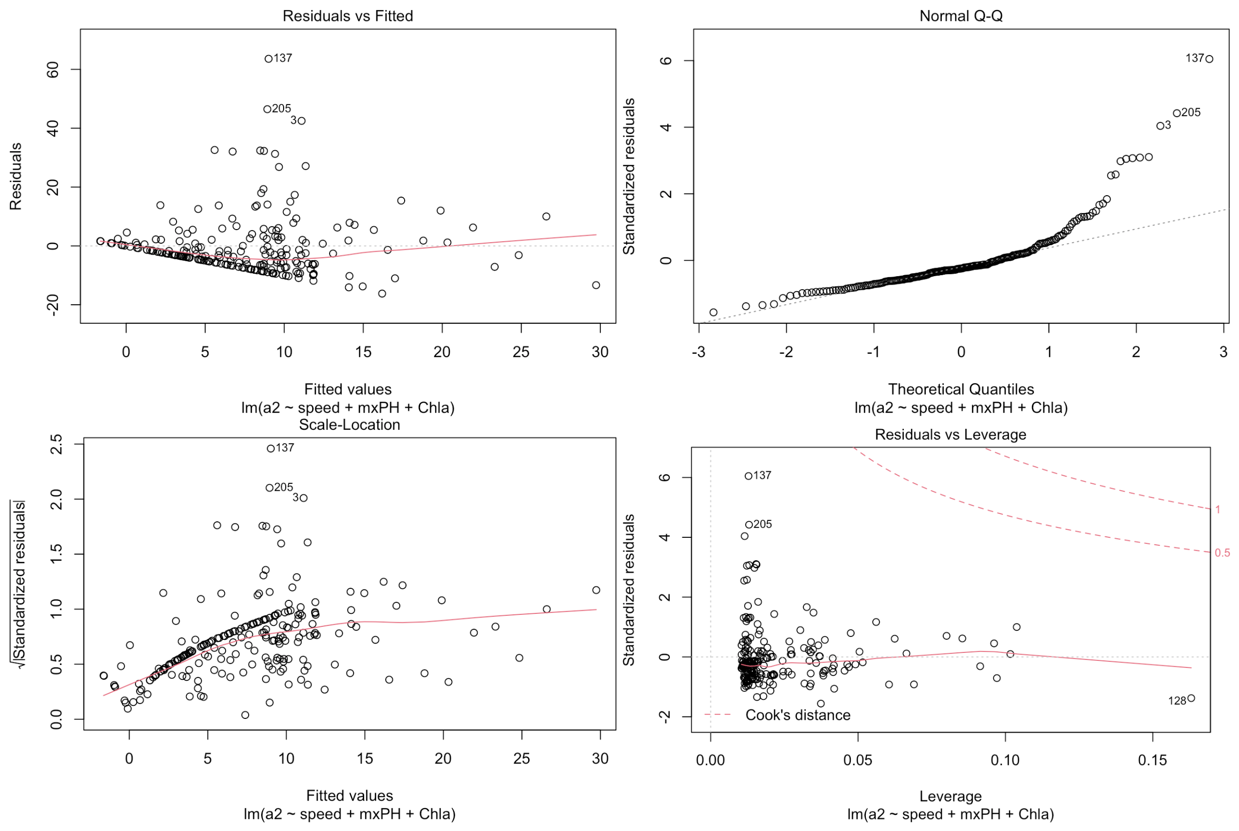
```
Analysis of Variance Table
Response: a2
           Df  Sum Sq Mean Sq F value    Pr(>F)
speed       2  1994.8  997.42  8.9063 0.0001929 ***
mxPH        1  2026.6 2026.63 18.0964 3.145e-05 ***
Chla        1  1479.6 1479.59 13.2117 0.0003488 ***
Residuals 213 23854.1  111.99
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In spite of the final model's poor quality, it is significantly different from the full model.

```
anova(linear.model.a2,final.linear.model.a2)
```

```
Model 1: a2 ~ season + size + speed + mxPH + mnO2 + Cl +
    NO3 + NH4 + oPO4 + PO4 + Chla
Model 2: a2 ~ speed + mxPH + Chla
  Res.Df   RSS  Df Sum of Sq      F Pr(>F)
1    202 23309
2    213 23854 -11   -545.26 0.4296 0.9416
```

```
plot(final.linear.model.a2)
```



Regression Tree Model An alternative to regression is the use of **regression trees**, implemented in the function `rpart()`.[76]

76: Its syntax is similar to `lm()`.

```
regression.tree.a2 <-rpart::rpart(a2 ~ season + size +
    speed + mxPH + mnO2 + Cl + NO3 + NH4 + oPO4 + PO4 +
    Chla, data=algae.train)
```

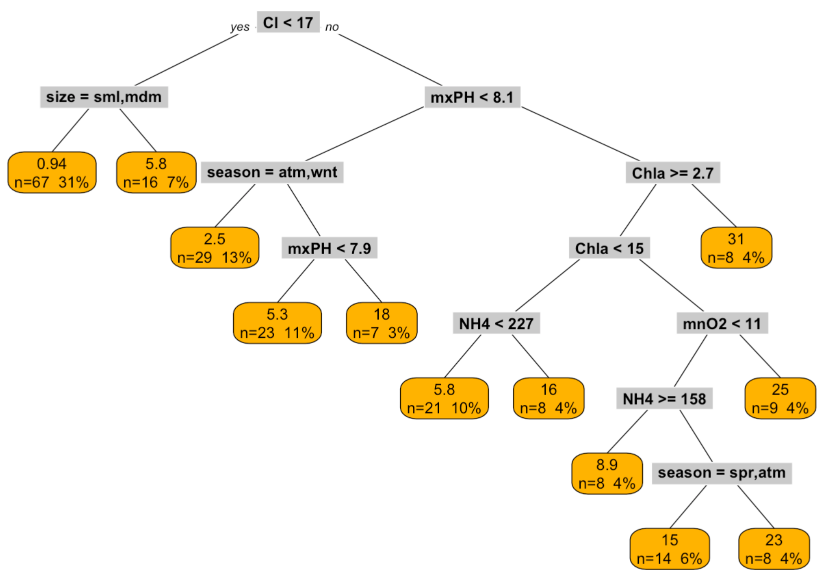
The outcome can be displayed by calling the object directly.

```
regression.tree.a2
```

```
node), split, n, deviance, yval
      * denotes terminal node

  1) root 218 29355.1300  7.6366970
    2) Cl< 16.6875 83  1193.6400  1.8891570
      4) size=small,medium 67   398.6457  0.9447761 *
      5) size=large 16   485.0194  5.8437500 *
    3) Cl>=16.6875 135 23733.9200 11.1703700
      6) mxPH< 8.065 59  3831.8290  5.3864410
       12) season=autumn,winter 29   561.8414  2.5172410 *
       13) season=spring,summer 30  2800.4720  8.1600000
         26) mxPH< 7.9375 23   889.9730  5.3173910 *
         27) mxPH>=7.9375 7  1114.0000 17.5000000 *
      7) mxPH>=8.065 76 16396.0400 15.6605300
       14) Chla>=2.65 68  9694.0890 13.8544100
         28) Chla< 14.8875 29  2747.5810  8.7172410
           56) NH4< 226.875 21   558.4257  5.7857140 *
           57) NH4>=226.875 8  1534.9490 16.4125000 *
         29) Chla>=14.8875 39  5612.0940 17.6743600
           58) mnO2< 11.05 30  3139.0940 15.4233300
            116) NH4>=158.409 8   577.1000  8.9000000 *
            117) NH4< 158.409 22  2097.7700 17.7954500
              234) season=spring,autumn 14   674.7521 14.6642900 *
              235) season=summer,winter 8  1045.5550 23.2750000 *
           59) mnO2>=11.05 9  1814.2760 25.1777800 *
       15) Chla< 2.65 8  4594.6690 31.0125000 *
```

The tree structure can be hard to determine when there is a large number of nodes; we can improve on the visuals by using the `R` library `rpart.plot`.

```
rpart.plot::prp(regression.tree.a2, extra=101,
                box.col="orange", split.box.col="gray")
```



Details on the regression tree can be obtained by calling the `summary()` method on the object.

```
summary(regression.tree.a2)
```

```
           CP nsplit rel error    xerror      xstd
1  0.15082765      0 1.0000000 1.0059069 0.1990911
2  0.11943572      1 0.8491724 0.9492709 0.1815913
3  0.07178590      2 0.7297366 0.8655117 0.1688012
4  0.04545758      3 0.6579507 0.9007445 0.1699016
5  0.02243987      4 0.6124932 0.9597254 0.1737117
6  0.02228595      5 0.5900533 0.9472199 0.1658890
7  0.02156378      6 0.5677673 0.9472199 0.1658890
8  0.01581407      8 0.5246398 0.9287217 0.1629262
9  0.01285848      9 0.5088257 0.9255472 0.1613858
10 0.01055949     10 0.4959672 0.9320459 0.1622581
11 0.01000000     11 0.4854077 0.9389544 0.1625727

Variable importance
  Chla     NH4      Cl    mxPH    oPO4     PO4     NO3   speed
    19      14      14      13      11       9       6       5
  mnO2  season    size
     4       3       2
```

Note that `rpart()` grows a tree on the training data until one of the following criterion is met: - decrease in deviance goes below a certain threshold (`cp`) - number of samples in a node is below some other threshold (`minsplit`) - depth of the tree crosses yet another threshold (`maxdepth`)

The library also implements a **pruning method** based on *cost complexity*: finding the value of `cp` which best balances **predictive accuracy** and **tree size**.[77]

77: We will revisit these notions in Section 21.4, *Tree-Based Methods*.

```
rpart::printcp(regression.tree.a2)
```

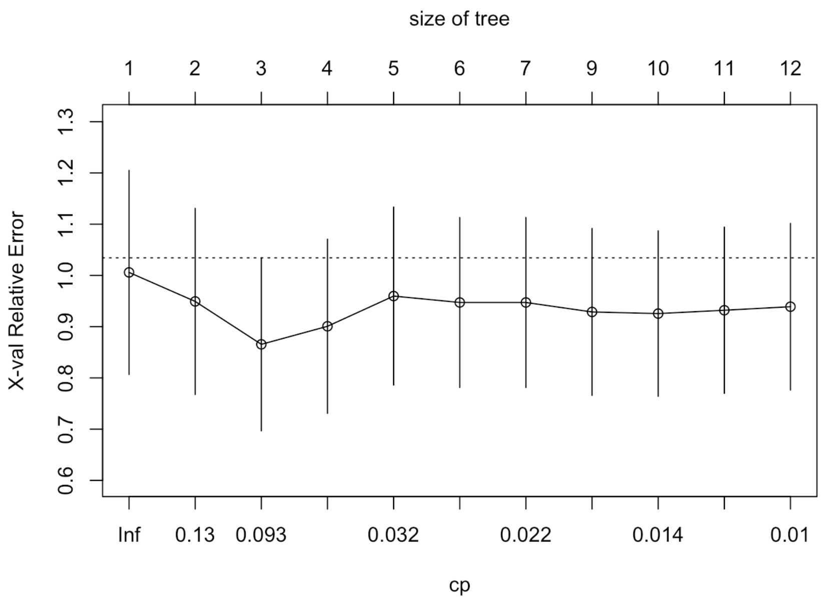
```
Variables actually used in tree construction:
[1] Chla   Cl     mnO2   mxPH   NH4    season size

Root node error: 29355/218 = 134.66

n= 218

         CP nsplit rel error  xerror    xstd
1  0.150828      0   1.00000 1.00591 0.19909
2  0.119436      1   0.84917 0.94927 0.18159
3  0.071786      2   0.72974 0.86551 0.16880
4  0.045458      3   0.65795 0.90074 0.16990
5  0.022440      4   0.61249 0.95973 0.17371
6  0.022286      5   0.59005 0.94722 0.16589
7  0.021564      6   0.56777 0.94722 0.16589
8  0.015814      8   0.52464 0.92872 0.16293
9  0.012858      9   0.50883 0.92555 0.16139
10 0.010559     10   0.49597 0.93205 0.16226
11 0.010000     11   0.48541 0.93895 0.16257
```

The tree returned by `rpart()` is the final one (`cp`= 0.01 is the default value); it requires 11 decision tests, and has a relative error of 0.485. Internally, `rpart()` uses 10-fold cross-validation to estimate that the tree has an average relative error of 0.98 ± 0.18.[78]

78: These values might change when from one run to the next due to the stochastic nature of the internal cross-validation routines.

In this framework, the optimal tree minimizes the value of `xerror`. Alternatively, one could use the **1 − SE** rule to find the minimal `xerror` + `xstd` tree.

```
rpart::plotcp(regression.tree.a2)
```

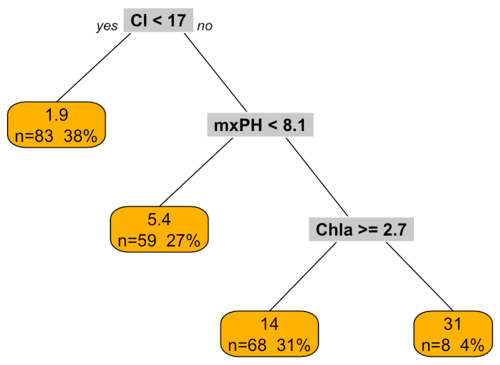


The scree plot above suggests that cp= 0.08 (that value may change when you run yours due to the stochastic nature of the internal cross-validation algorithm) is a special value for tree growth, so we could prune the tree using that specific value.

```
(regression.tree.a2.mod <- rpart::prune(
                            regression.tree.a2,cp=0.05))
```

```
node), split, n, deviance, yval
      * denotes terminal node

 1) root 218 29355.130  7.636697  
   2) Cl< 16.6875 83  1193.640  1.889157 *
   3) Cl>=16.6875 135 23733.920 11.170370  
     6) mxPH< 8.065 59  3831.829  5.386441 *
     7) mxPH>=8.065 76 16396.040 15.660530  
      14) Chla>=2.65 68  9694.089 13.854410 *
      15) Chla< 2.65 8  4594.669 31.012500 *
```

```
rpart.plot::prp(regression.tree.a2.mod, extra=101,
                box.col="orange", split.box.col="gray")
```



The entire process is automated in the wrapper method `rpartXse()` provided with the `DMwR` library;[79] we (abitrarily) use se= 0.2.

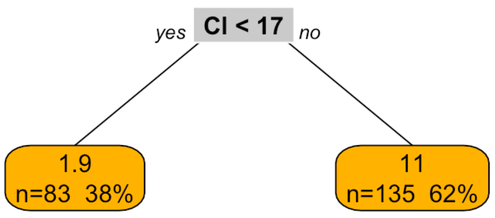
79: This library had to be installed from source files as it was not available on the Comprehensive R Archive Network as of January 2023.

```
library(DMwR)
(regression.tree.a2.final <- DMwR::rpartXse(a2 ~ season +
    size + speed + mxPH + mnO2 + Cl + NO3 + NH4 + oPO4 +
    PO4 + Chla, data=algae.train, se=0.2))
summary(regression.tree.a2.final)
```

```
node), split, n, deviance, yval
      * denotes terminal node

1) root 218 29355.13  7.636697  
  2) Cl< 16.6875 83  1193.64  1.889157 *
  3) Cl>=16.6875 135 23733.92 11.170370 *
Call:
```

```
rpart(formula = form, data = data, cp = cp, minsplit = minsplit)
  n= 218

         CP nsplit rel error    xerror      xstd
1 0.1508276      0 1.0000000 1.0130822 0.2001495
2 0.1194357      1 0.8491724 0.9320224 0.1752140

Variable importance
   Cl   PO4  oPO4  Chla   NH4 speed
   28    17    16    14    14    12

Node number 1: 218 observations,    complexity param=0.1508276
  mean=7.636697, MSE=134.6565
  left son=2 (83 obs) right son=3 (135 obs)
  Primary splits:
      Cl   < 16.6875 to the left,  improve=0.15082760, (0 missing)
      mxPH < 7.94    to the left,  improve=0.14900670, (0 missing)
      NO3  < 0.18    to the right, improve=0.11564070, (0 missing)
      oPO4 < 45.1    to the left,  improve=0.11106510, (0 missing)
      Chla < 12.21   to the left,  improve=0.09817759, (0 missing)
  Surrogate splits:
      PO4   < 70.465  to the left,  agree=0.844, adj=0.590, (0 split)
      oPO4  < 19.8635 to the left,  agree=0.835, adj=0.566, (0 split)
      NH4   < 46.35   to the left,  agree=0.807, adj=0.494, (0 split)
      Chla  < 2.225   to the left,  agree=0.807, adj=0.494, (0 split)
      speed splits as  RRL,         agree=0.775, adj=0.410, (0 split)

Node number 2: 83 observations
  mean=1.889157, MSE=14.38121

Node number 3: 135 observations
  mean=11.17037, MSE=175.8068
```

```
rpart.plot::prp(regression.tree.a2.final, extra=101,
                box.col="orange", split.box.col="gray")
```



The resulting tree is not nearly as complex as the original tree (hence discourages overfitting) but is still more complex than the pruned tree (which should improve predicting accuracy).

20.6.2 Model Evaluation

At this stage, we know that the linear model is not great for a2, and we have seen how to grow a regression tree for a2 but we have not yet discussed whether this model is a good fit, to say nothing of the remaining 6 algae concentrations. Can we get a better handle on these models' performance (i.e., comparing the model predictions to the real values of the target variable in the test data)?

We have discussed various metrics that can be used to determine how the values compare in Chapter 19; in this case, we elect to use the **normalized mean squared error** (NMSE):

$$\frac{\text{MSE}}{\text{mean}\left\{\left(\overline{\text{real}}-\text{real}_i\right)^2 ; i=1,\ldots,N\right\}}.$$

As the ratio of MSE to a baseline predictor (the mean of the value of the target), NMSE is **unitless**. NMSE values between 0 and 1 (smaller is better) indicate that the model performs better than the baseline; greater than 1 indicate that the model's performance is sub-par.

We use the `performanceEstimation` library to run 5×10−fold cross-validations to determine which of the models (linear model and 4 regression trees parametrized by `se`) yields an optimal (smaller) NMSE value when trying to predict `a2`.

```
library(performanceEstimation)
kCV.results.algae.a2 <- performanceEstimation(
  PredTask(a2 ~ season + size + speed + mxPH + mnO2 + Cl +
    NO3 + NH4 + oPO4 + PO4 + Chla, data=algae.train, "a2"),
  c(Workflow(learner="lm",post="onlyPos"),
    workflowVariants(learner="rpartXse",
       learner.pars=list( se=c(0,0.25,0.5,0.75,1) ))),
  EstimationTask(metrics="nmse",
                 method=CV(nReps=5,nFolds=10))
)
```

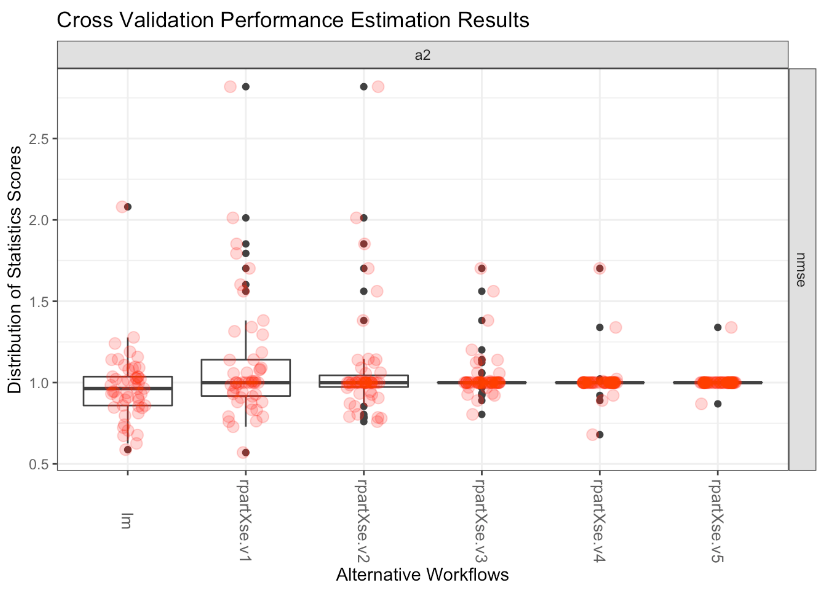
A summary and plot of the cross-validation results for NMSE can be displayed using calls to `summary()` and `plot()`.

```
summary(kCV.results.algae.a2)
```

```
== Summary of a  Cross Validation Performance Estimation Experiment ==
Task for estimating  nmse  using 5 x 10 - Fold Cross Validation (seed =

* Predictive Tasks ::  a2
* Workflows  ::  lm, rpartXse.v1, rpartXse.v2, rpartXse.v3,
                 rpartXse.v4, rpartXse.v5

-> Task:  a2
  *Workflow: lm
               nmse
avg     0.9723125
std     0.2221976
med     0.9634147
iqr     0.1771688
```

```
min     0.5878283
max     2.0801221
invalid 0.0000000

  *Workflow: rpartXse.v1
             nmse
avg     1.1148436
std     0.3871551
med     1.0000000
iqr     0.2226673
min     0.5701226
max     2.8186400
invalid 0.0000000

  *Workflow: rpartXse.v2
              nmse
avg     1.08587675
std     0.35111303
med     1.00000000
iqr     0.07178237
min     0.76004730
max     2.81864005
invalid 0.00000000

  *Workflow: rpartXse.v3
                nmse
avg     1.035773e+00
std     1.470430e-01
med     1.000000e+00
iqr     2.220446e-16
min     8.044770e-01
max     1.701835e+00
invalid 0.000000e+00

  *Workflow: rpartXse.v4
                nmse
avg     1.011250e+00
std     1.214329e-01
med     1.000000e+00
iqr     2.220446e-16
min     6.800497e-01
max     1.701835e+00
invalid 0.000000e+00

  *Workflow: rpartXse.v5
                nmse
avg     1.004167e+00
std     5.174279e-02
med     1.000000e+00
iqr     2.220446e-16
min     8.692699e-01
max     1.339067e+00
invalid 0.000000e+00
```

```
plot(kCV.results.algae.a2)
```



It is not obvious which of the models has smaller values of NMSE, although it does seem that the latter versions of the regression tree models are not substantially better than the baseline model.

The first regression tree model sometimes produces very small NMSE values, but that is offset by some of the larger values it also produces.[80]

80: Similarly for the linear model.

At any rate, visual evidence seems to suggest that the **linear model** is the best predictive model for `a2` given the training data (in this version of kCV), which is corroborated by a call to `topPerformers()`.

```
topPerformers(kCV.results.algae.a2)
```

```
$a2
     Workflow Estimate
nmse       lm    0.972
```

This might seem disheartening at first given how poorly the linear model performed, but it might be helpful to remember that there is no guarantee that a decent predictive model even exists in the first place.

Furthermore, regression trees and linear models are only two of a whole collection of possible models. How do **support vector machines** perform the task, for instance?[81]

81: See Chapter 21 for an in-depth discussion on the topic.

This time, however, we will learn models and perform evaluation for all target variables (`a1`-`a7`) simultaneously. This does not mean that we are looking for a single model which will optimize all learning tasks at once, but rather that we can prepare and evaluate the models for each target variable with the same bit of code.

This first require some code to create the appropriate model formulas (`a1 ~ .` , ... ,`a7 ~ .`) and the appropriate training data.

```
gg<-function(x,list.of.variables){
        PredTask(as.formula(paste(x,"~ .")), algae.train[,c(list.of.variables,x )], x, copy=TRUE)
}
(data.sources <- sapply(names(algae.train[12:18]), gg, names(algae.train[1:11])))
```

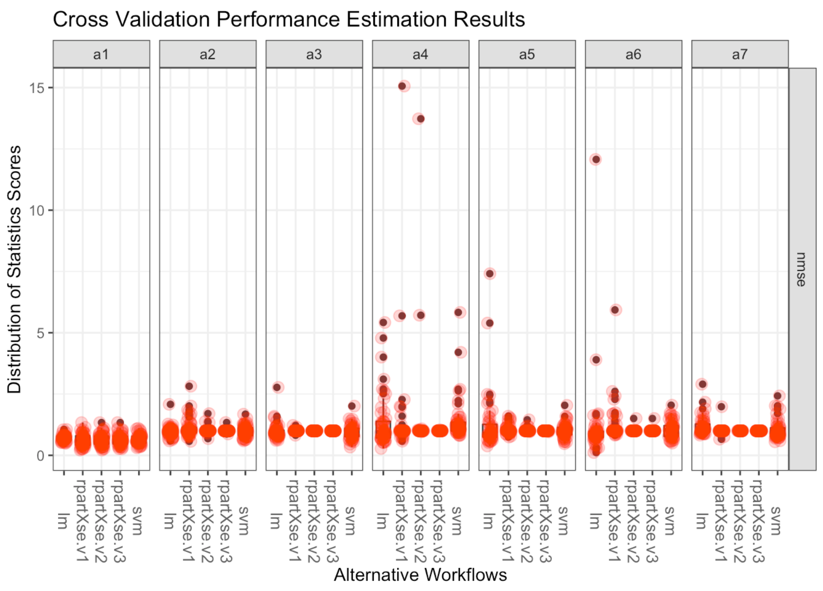
```
$a1
Prediction Task Object:
    Task Name         :: a1
    Task Type         :: regression
    Target Feature    :: a1
    Formula           :: a1 ~ .

$a2
Prediction Task Object:
    Task Name         :: a2
    Task Type         :: regression
    Target Feature    :: a2
    Formula           :: a2 ~ .

$a3
Prediction Task Object:
    Task Name         :: a3
    Task Type         :: regression
    Target Feature    :: a3
    Formula           :: a3 ~ .

$a4
Prediction Task Object:
    Task Name         :: a4
    Task Type         :: regression
    Target Feature    :: a4
    Formula           :: a4 ~ .

$a5
Prediction Task Object:
    Task Name         :: a5
    Task Type         :: regression
    Target Feature    :: a5
    Formula           :: a5 ~ .

$a6
Prediction Task Object:
    Task Name         :: a6
    Task Type         :: regression
    Target Feature    :: a6
    Formula           :: a6 ~ .

$a7
Prediction Task Object:
    Task Name         :: a7
    Task Type         :: regression
    Target Feature    :: a7
    Formula           :: a7 ~ .
```

We shall use `e1071`'s implementation of `svm()`, with various values of the `svm()`-specific parameters `cost` and `gamma`.

```
library(e1071)
kCV.results.algae.all <- performanceEstimation(
    data.sources,
    c(Workflow(learner="lm", post="onlyPos"),
      Workflow(learner="svm", learner.pars=list(
          cost=c(10,1,0.1), gamma=0.1)),
    workflowVariants(learner="rpartXse", learner.pars=list(
          se=c(0,0.7,1)))),
    EstimationTask(metrics="nmse",
                   method=CV(nReps=5, nFolds=10)))
```

The rest of the evaluation proceeds much as before, except that we can display results for the 7 target variables simultaneously.

```
plot(kCV.results.algae.all)
```



```
rankWorkflows(kCV.results.algae.all,top=3)
```

```
$a1$nmse
     Workflow  Estimate
1 rpartXse.v1 0.6163406
2 rpartXse.v2 0.6278027
3 rpartXse.v3 0.6430736

$a2$nmse
     Workflow  Estimate
1          lm 0.9723125
2         svm 0.9954432
3 rpartXse.v3 1.0041667
```

```
$a3$nmse
     Workflow  Estimate
1         svm 0.9497730
2          lm 0.9801662
3 rpartXse.v2 1.0000000

$a4$nmse
     Workflow Estimate
1 rpartXse.v3 1.001453
2 rpartXse.v2 1.351494
3          lm 1.357243

$a5$nmse
     Workflow  Estimate
1         svm 0.9968475
2 rpartXse.v3 0.9990465
3 rpartXse.v2 1.0194733

$a6$nmse
     Workflow Estimate
1 rpartXse.v2 1.010069
2 rpartXse.v3 1.010069
3         svm 1.054975

$a7$nmse
     Workflow Estimate
1 rpartXse.v2  1.00000
2 rpartXse.v3  1.00000
3 rpartXse.v1  1.00797
```

```
topPerformers(kCV.results.algae.all)
```

```
$a1
        Workflow Estimate
nmse rpartXse.v1    0.616

$a2
     Workflow Estimate
nmse       lm    0.972

$a3
     Workflow Estimate
nmse      svm     0.95

$a4
        Workflow Estimate
nmse rpartXse.v3    1.001

$a5
     Workflow Estimate
nmse      svm    0.997
```

```
$a6
        Workflow Estimate
nmse rpartXse.v2     1.01

$a7
        Workflow Estimate
nmse rpartXse.v2        1
```

For a1, the models seem to perform reasonably well, but it is not as rosy for the other target variables, where the baseline model is sometimes better.[82]

82: Again, this could be built-in in the data, but we might benefit from incorporating more models.

```
library(randomForest)
kCV.algae.all.rf <- performanceEstimation(
    data.sources,
    c(Workflow(learner="lm", post="onlyPos"),
      Workflow(learner="svm", learner.pars=list(
            cost=c(10,1,0.1), gamma=0.1)),
      workflowVariants(learner="rpartXse",
            learner.pars=list(se=c(0,0.7,1))),
      workflowVariants(learner="randomForest",
            learner.pars=list(ntree=c(200,500,700)))),
    EstimationTask(metrics="nmse", method=CV(nReps=5,
                                         nFolds=10))
)
```

```
rankWorkflows(kCV.algae.all.rf,top=3)
```

```
$a1$nmse
         Workflow  Estimate
1 randomForest.v2 0.5217204
2 randomForest.v3 0.5228744
3 randomForest.v1 0.5264328

$a2$nmse
         Workflow  Estimate
1 randomForest.v3 0.7798749
2 randomForest.v2 0.7806831
3 randomForest.v1 0.7849360

$a3$nmse
         Workflow  Estimate
1 randomForest.v3 0.9377108
2 randomForest.v2 0.9400108
3 randomForest.v1 0.9431801

$a4$nmse
         Workflow Estimate
1     rpartXse.v3 1.001453
2 randomForest.v3 1.006496
3 randomForest.v1 1.006806
```

```
$a5$nmse
         Workflow  Estimate
1 randomForest.v1 0.7626241
2 randomForest.v2 0.7675794
3 randomForest.v3 0.7681834

$a6$nmse
         Workflow  Estimate
1 randomForest.v2 0.8590227
2 randomForest.v3 0.8621478
3 randomForest.v1 0.8663869

$a7$nmse
     Workflow Estimate
1 rpartXse.v2  1.00000
2 rpartXse.v3  1.00000
3 rpartXse.v1  1.00797
```

`rankWorkflows()` does not report on the standard error, so we cannot tell whether the differences between the score of the best model and the other models is statistically significant.

`randomForest.v3` seems to have the best ranking across all learning tasks, so we will use it as the baseline model.

```
p <- pairedComparisons(kCV.algae.all.rf,
                       baseline="randomForest.v3")
p$nmse$F.test
p$nmse$BonferroniDunn.test
```

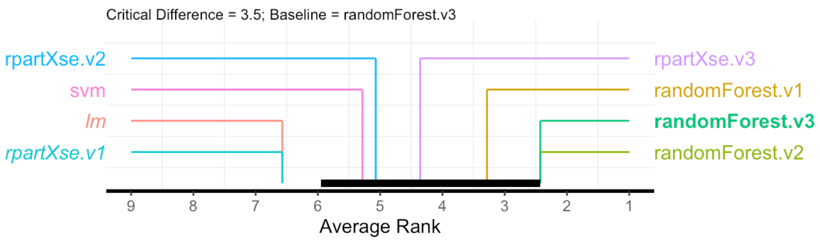
```
$chi
[1] 22.86905

$FF
[1] 5.251025

$critVal
[1] 0.7071231

$rejNull
[1] TRUE

$critDif
[1] 3.52218

$baseline
[1] "randomForest.v3"

$rkDifs
             lm             svm     rpartXse.v1     rpartXse.v2     rpartXse.v3
      4.1428571       2.8571429       4.1428571       2.6428571       1.9285714
randomForest.v1 randomForest.v2
      0.8571429       0.0000000
```

```
$signifDifs
             lm             svm     rpartXse.v1     rpartXse.v2     rpartXse.v3
           TRUE           FALSE            TRUE           FALSE           FALSE
randomForest.v1 randomForest.v2
          FALSE           FALSE
```

We can reject with 95% certainty the hypothesis that the performance of the baseline method (`randomForest.v3`) is the same as that of the linear model and the first 2 regression trees, but not that it is better than `svm`, `rpartXse.v3`, and the other 2 random forests.

The information is also displayed in the Bonferroni-Dunn CD diagram below.

```
CDdiagram.BD(p)
```



20.6.3 Model Predictions

Finally, we might actually be interested in generating predictions for each of the target variables in the testing set. This simply requires that the best performers for each target response be brought together in an `R` object.

```
best.performers <- sapply(taskNames(kCV.algae.all.rf),
        function(x) topPerformer(kCV.algae.all.rf,
                metric="nmse", task=x))
best.performers
```

```
$a1
Workflow Object:
    Workflow ID       ::  randomForest.v2
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> ntree=500
         learner  -> randomForest

$a2
Workflow Object:
    Workflow ID       ::  randomForest.v3
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> ntree=700
         learner  -> randomForest

$a3
```

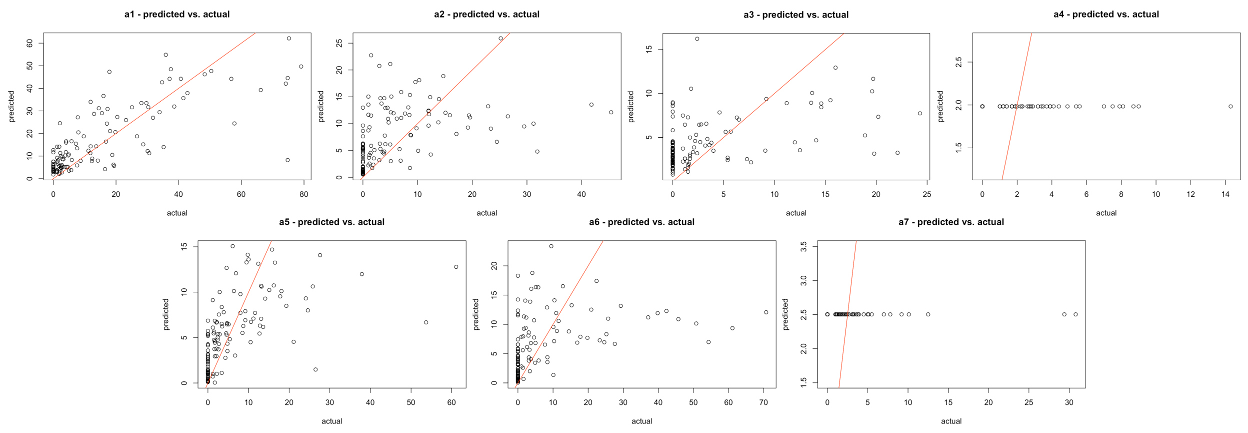
```
Workflow Object:
    Workflow ID       ::  randomForest.v3
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> ntree=700
         learner  -> randomForest

$a4
Workflow Object:
    Workflow ID       ::  rpartXse.v3
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> se=1
         learner  -> rpartXse

$a5
Workflow Object:
    Workflow ID       ::  randomForest.v1
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> ntree=200
         learner  -> randomForest

$a6
Workflow Object:
    Workflow ID       ::  randomForest.v2
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> ntree=500
         learner  -> randomForest

$a7
Workflow Object:
    Workflow ID       ::  rpartXse.v2
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> se=0.7
         learner  -> rpartXse
```

The observations that form the **testing set** are placed in an object, as below:

```
test.observations <- array(dim=c(nrow(algae.test),7,2),
    dimnames=list(rownames(algae.test), paste("a",1:7),
        c("actual","predicted")))
```

The function `runWorkflow()` will compute the predicted values for each of the targets' best performers. We can then plot the predicted and actual values for each of the testing set targets.

```
for(j in 1:7){
    results <- runWorkflow(best.performers[[j]],
        as.formula(paste(names(best.performers)[j],"~ .")),
        algae.train[,c(1:11,11+j)],
        algae.test[,c(1:11,11+j)])
    test.observations[,j,"actual"] <- results$trues
    test.observations[,j,"predicted"] <- results$preds
}

df.a1 <- as.data.frame(test.observations[,1,])
df.a2 <- as.data.frame(test.observations[,2,])
df.a3 <- as.data.frame(test.observations[,3,])
df.a4 <- as.data.frame(test.observations[,4,])
df.a5 <- as.data.frame(test.observations[,5,])
df.a6 <- as.data.frame(test.observations[,6,])
df.a7 <- as.data.frame(test.observations[,7,])

plot(df.a1,main="a1 - predicted vs. actual")
abline(0,1,col="red")
plot(df.a2,main="a2 - predicted vs. actual")
abline(0,1,col="red")
plot(df.a3,main="a3 - predicted vs. actual")
abline(0,1,col="red")
plot(df.a4,main="a4 - predicted vs. actual")
abline(0,1,col="red")
plot(df.a5,main="a5 - predicted vs. actual")
abline(0,1,col="red")
plot(df.a6,main="a6 - predicted vs. actual")
abline(0,1,col="red")
plot(df.a7,main="a7 - predicted vs. actual")
abline(0,1,col="red")
```



The models simply are not that great, but we already expected that. The average prediction for each target is shown below.

```
(average.prediction <- apply(algae.train[,12:18],2, mean))
```

```
   a1    a2    a3    a4    a5    a6    a7
17.47  7.64  4.13  1.98  4.96  5.81  2.50
```

Finally, you might be interested in the NMSE metrics for the predicted values and how they compare to the NMSE metrics on the training set. Is any of this surprising?

```
apply((test.observations[,,"actual"]-
        test.observations[,,"predicted"])^2, 2, sum) /
    apply((scale(test.observations[,,"actual"],
        average.prediction,FALSE))^2, 2, sum)
```

```
  a1   a2   a3   a4   a5   a6   a7
0.40 0.88 0.78 1.00 0.71 0.84 1.00
```

20.7 Exercises

1. Let (X, Y) be a bivariate normal random variable with parameters
$$\mu_X = 12, \mu_Y = -7, \sigma_X^2 = 1, \sigma_Y^2 = 2, \sigma_{XY} = 4.$$
Consider the parameter
$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}.$$
Using a bootstrap procedure with $N = 100$ samples and $M = 200$ replicates, provide a confidence interval for the true value of α.
2. Explicitly obtain the polynomial regression models in the Gapminder Example, for $d = 2, 3, 4$.
3. Play around with a variety of knots in the step function regression models for the Gapminder Example, and build the corresponding confidence intervals (including those of the example). How would you determine the number and location of the knots?
4. Determine the optimal number of knots K for cubic splines and natural cubic splines for the Gapminder Example, using cross-validation.
5. Build piecewise cubic splines and continuous piecewise cubic splines for the Gapminder Example. Use cross-validation to determine the optimal number of knots.
6. Predict life expectancy of countries in 2011 using the various spline models (in the text and in the exercises) on the Gapminder dataset, with training/testing pairs. Evaluate your models. Which ones perform best?
7. Predict life expectancy of countries in 2011 using various GAM models on the Gapminder dataset, with training/testing pairs. Evaluate your models. Which ones perform best?
8. Consider the dataset `algae_blooms.csv`, as in Section 20.6. Run the analysis with a scaled dataset. Run the analysis with a PCA-reduced dataset. Do the results change significantly?
9. Consider the following datasets:
    - `GlobalCitiesPBI.csv`
    - `2016collisionsfinal.csv`
    - `polls_us_election_2016.csv`
    - `HR_2016_Census_simple.xlsx`, and

- UniversalBank.csv.

For each of these datasets, identify a response variable (or more than one, if the fancy strikes you) and predictors, and build models to predict the response(s) using the various methods discussed in this chapter. Evaluate and rank the resulting models. You may need to clean, transform, and visualize the data along the way.

10. Complete the definition of the Python function kfoldCV(k, data, yname, formulas) where k is the number of folds, data is the data set, yname is the column name of the dependent variable, and formulas is a list of formulas. The function should return the tuple fit, f where fit is the OLS model for the formula f in formulas that has the minimum k−fold CV estimate. Use it on the mpg data set with $k = 10$ to obtain a good model for predicting mpg.

```
import seaborn as sns

df = sns.load_dataset('mpg')
df.head()

def kfoldCV(k, data, yname, formulas):
    fit = None
    # Your code here.  Don't forget to obtain a
    # random permutation of the observations

    for f in formulas:
        # Your code here
        None

    return fit, f
```