STATISTICAL METHODS FOR SUPERVISED LEARNING

Patrick Boily[1,2,3]

Abstract
In this document, we present the mathematical foundations of machine learning, with a special emphasis on value estimation (regression) and classification.

Keywords
Multivariate linear and polynomial regression, logistic regression, k—nearest neighbours, regularization methods, naïve Bayes classification, cross-validation, bootstrap procedures, tree-based methods, support vector machines, neural networks.

Acknowledgement
Our approach generally follows that of James, Witten, Hastie, and Tibshirani's *Introduction to Statistical Learning With R* [15].

[1]Department of Mathematics and Statistics, University of Ottawa, Ottawa, Canada
[2]Data Action Lab, Ottawa, Canada
[3]Idlewyld Analytics and Consulting Services, Wakefield, Canada
Email: pboily@uottawa.ca

Contents

1. Introduction

In [2], we have provided a (basically) math-free general overview of machine learning.

In this document, we present an introductory mathematical treatment of the discipline (with a focus on supervised learning), starting with its underlying formalism.

1.1 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]

1.1.1 Supervised Learning Framework

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

If Y is quantitative (price, height, etc.), then the problem of predicting Y in terms of X is a **regression** task; if X takes on values in a finite unordered set (survived/died, colours, vegetation types, etc.), it is a **classification** task.

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

This is typically achieved with the use of **training data**, which is to say observations or instances

obs.	predictors			resp.
1	$x_{1,1}$	$\cdots$	$x_{1,p}$	y_1
$\vdots$	$\vdots$		$\vdots$	$\vdots$
N	$x_{N,1}$	$\cdots$	$x_{N,p}$	y_N

which we often denote by $[\mathbf{X} \mid \mathbf{Y}]$ (the column denoting the observation IDs is dropped).

The objectives of supervised learning are usually to:

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

1.1.2 Unsupervised Learning

In this environment, there are no outcome variables, only the features on a set of observations $\mathbf{X}$.[2] The obectives are much more **vague** – analysts could seek to:

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

A spotlight is shone on such methods in [1, 27].

1.1.3 Statistical Learning vs. Machine Learning

The term "statistical learning" is not used frequently in practice (except by mathematicians and statisticians, perhaps); the current tendency is to speak of **machine learning**.

If a distinction must be made, we could argue that:

- 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 [22] for another take on this).

1.1.4 Motivating Example

Throughout, we will illustrate the concepts and notions *via* the `Gapminder` dataset, which records socio-demographic information relating to the planet's nations, ranging from 1960 to 2011 [23, 24].

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.

[2] The response variable $\mathbf{Y}$ that was segregated away from $\mathbf{X}$ in the supervised learning case could now be one of the variables in $\mathbf{X}$.

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.

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 discrepancies.

1.1.5 Systematic Component and Regression Function

It is the systematic component that is used for **predictions** and **inferences**.

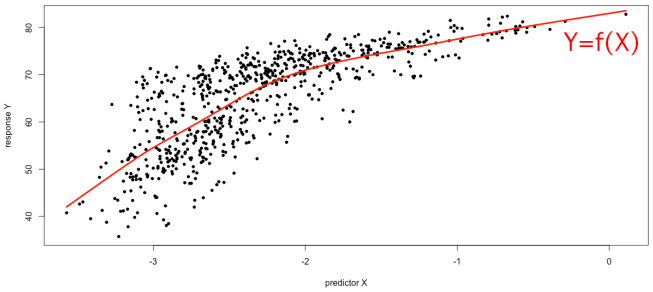
As long as f is "good", we can:

- make predictions for the response Y at new points $\vec{X} = \vec{x}$;
- 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.

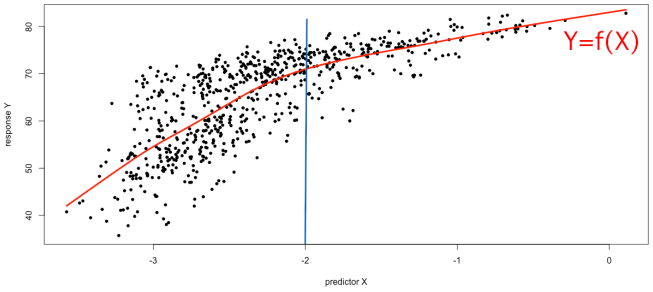
Imagine a model with one predictor X and a target Y, with systematic component f, so that

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

How can we find the ideal f?



What is a good value of $f(-2)$, say?



Ideally, we would like to have $f(-2) = \mathrm{E}[Y \mid X = -2]$.[3] For any x in the range of X, the function

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

is the **regression function of Y on X**.

In the general setting with p predictors, the regression function is

$$\begin{aligned} f(\vec{x}) = f(x_1, \ldots, x_p) &= \mathrm{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] \\ &= \mathrm{E}[Y \mid \vec{X} = \vec{x}]. \end{aligned}$$

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

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

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,[4] then

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

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 f of $\hat{f}$ that minimizes this reducible component, in some way.

1.1.6 Estimating the Regression Function

In theory, we know that the regression function is

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

in practice, however, there might be too few (or even no) observations at $\vec{X} = \vec{x}$ to trust the estimate provided by the sample mean.

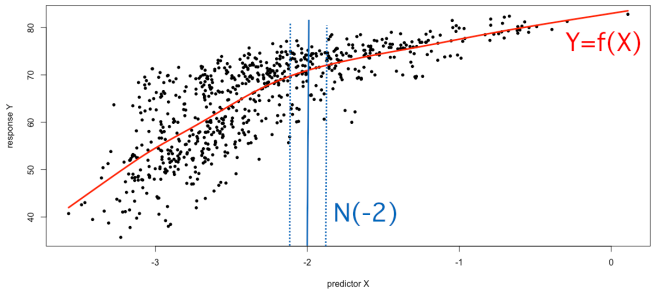
One solution is to approximate the expectation by a **nearest neighbour average**

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

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

[3] Why?

[4] In particular, if $\hat{Y} = f(\vec{X})$, then $\hat{Y} \approx Y = f(\vec{X}) + \varepsilon$).



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 $N(\vec{x})$ is defined as the nearest 5% of observations to $\vec{x}$, say[5], then we need to leave the "local" neighbourhood of $\vec{x}$ to build $N(\vec{x})$, which could compromise the quality of $\hat{f}(\vec{x})$ as an approximation to $f(\vec{x})$.

We provide more details in Section 5, but this is a topic about which it is worth being well-read (see [10] for a formal treatment).

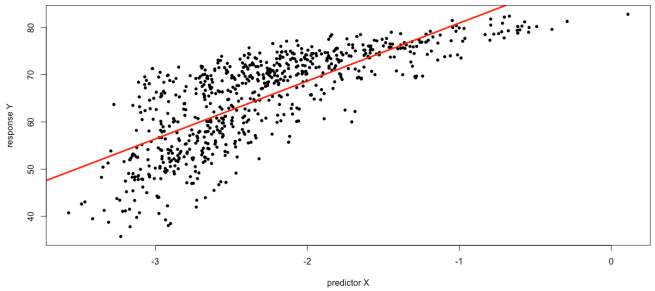
The various statistical learning methods attempt to provide estimates of the regression function by minimizing the reducible component through **parametric** or **non-parametric** approaches.[6]

For instance, the **classical linear regression** approach is parametric (see Section 2.1): it assumes that the true regression function f is linear and suggests the estimate

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

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 is almost never correct, but it often provides an **interpretable**[7] approximation to the true regression function f.



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

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

[5] The proportion must be large enough to bring the variance down.

[6] In this context, parametric means that assumptions are made about the form of the regression function f, non-parametric that no such assumptions are made.

[7] We will revisit this concept at a later stage.

say, then the linear regression approach would assume that

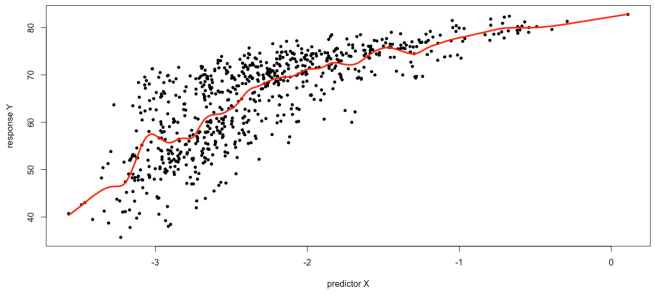
$$f_L(\text{fertility, infant mortality, gdp}) = \beta_0 + \beta_1 \cdot \text{fertility} + \beta_2 \cdot \text{infant mortality} + \beta_3 \cdot \text{gdp}.$$

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, as previously mentioned, 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 more difficult to learn and they tend to **overfit the data** (i.e. they mistake noise in the data for a signal to model, see Section 1.3 for details).

On the other hand, a **spline** is a non-parametric model (see Section 6.2): it makes no assumption about the form of f, and it seeks to estimate it by getting close to the data points without being too **rough** or too **wiggly**.



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 (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** (the set of parameters to learn is small and we can make sense of them, which leads us to understand how the predictors interact to produce outputs).

Ensemble learning methods (Section 7.5), support vector machines (Section 7.2), neural network [5, 9], and splines are high-flexibility/low-interpretability approaches; the LASSO (Section 5.3) and OLS are low-flexibility/high-interpretability approaches; generalized additive models (Section 6.3) and regression trees (Section 7.1) are medium-flexibility/medium-interpretability approaches. Importantly, 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.

1.2 Model Evaluation

In an ideal world (from a model performance point of view), we would be able to identify the modeling approach that performs "best", and use it for all problems.

In practice, however, the preceding discussion son trade-offs hows 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.

But another issue lurks just around the corner, even when we settle on an "optimal" performance evaluation measure.

No-Free Lunch Theorem: no single method is optimal over all possible datasets.[8]

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**,[9] various metrics are commonly used:

- **mean squared error** (MSE): $\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{f}(\vec{x}_i))^2$;
- **mean absolute error** (MAE): $\frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{f}(\vec{x}_i)|$;
- **normalized mean squared error** (NMSE):

$$\frac{\sum_{i=1}^{N}(y_i - \hat{f}(\vec{x}_i))^2}{\sum_{i=1}^{N}(y_i - \overline{y})^2};$$

- **normalized mean absolute error** (NMAE):

$$\frac{\sum_{i=1}^{N}|y_i - \hat{f}(\vec{x}_i)|}{\sum_{i=1}^{N}|y_i - \overline{y}|};$$

[8] In reality, achine learning is simply applied optimization; the proof of this important result is outside the scope of this document (but see [32,33] for details).

[9] We discuss the classification setting on p. 7 and in Sections 3 and 7.

- **mean average percentage error** (MAPE):

$$\frac{1}{N}\sum_{i=1}^{N}\frac{|y_i-\hat{f}(\vec{x}_i)|}{y_i};$$

- **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.

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.[10]

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.[11]

Let $\text{Tr}=\{(\vec{x}_i,y_i)\}_{i=1}^{N}$ 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, $\hat{f}(\vec{x}_i)\approx y_i$ for all $i=1,\dots,N$, and

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

is small. If it is, 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

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

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

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

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

over the testing set $\text{Te}=\{(\vec{x}_j,y_j)\}_{j=N+1}^{M+N}$, where, a priori, none of the test observations were in Tr.[12]

The general situation is illustrated in Figures 1 and 2.

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

[11] Although it would be surprising if the performance on the test data performance 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.

[12] New test observations may end up assuming the same values as some of the training observations, but that is an accident of sampling or due to the reality of the scenario under consideration.

1.3 Bias-Variance Trade-Off

The "U" shape of the testing MSE in Figure 2 is generic – something of this nature occurs for nearly all datasets and choice of statistical learning family of methods (although the actual shape could be quite different). Underfitting and overfitting is a fact of the machine learning life.

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

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

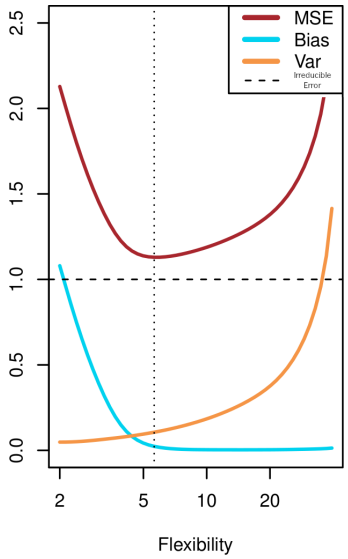
The **expected test MSE at** $\vec{x}^*$ can be decomposed into 3 fundamental quantities

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

As before, $\text{Var}(\varepsilon)$ is the **irreducible error**, due to the inherent noise in the data; the **variance component error** $\text{Var}(\hat{f}(\vec{x}^*))$ 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 [10, 15] for details).

The **overall expected test MSE** $\text{E}[\text{MSE}_{\text{Te}}]$ is the average of $\text{E}[\text{MSE}_{\text{Te}}(\vec{x}^*)]$ over all allowable $\vec{x}^*$ in the testing space. Note that, $\text{E}[\text{MSE}_{\text{Te}}]\geq\text{Var}(\varepsilon)$, 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 [15]).



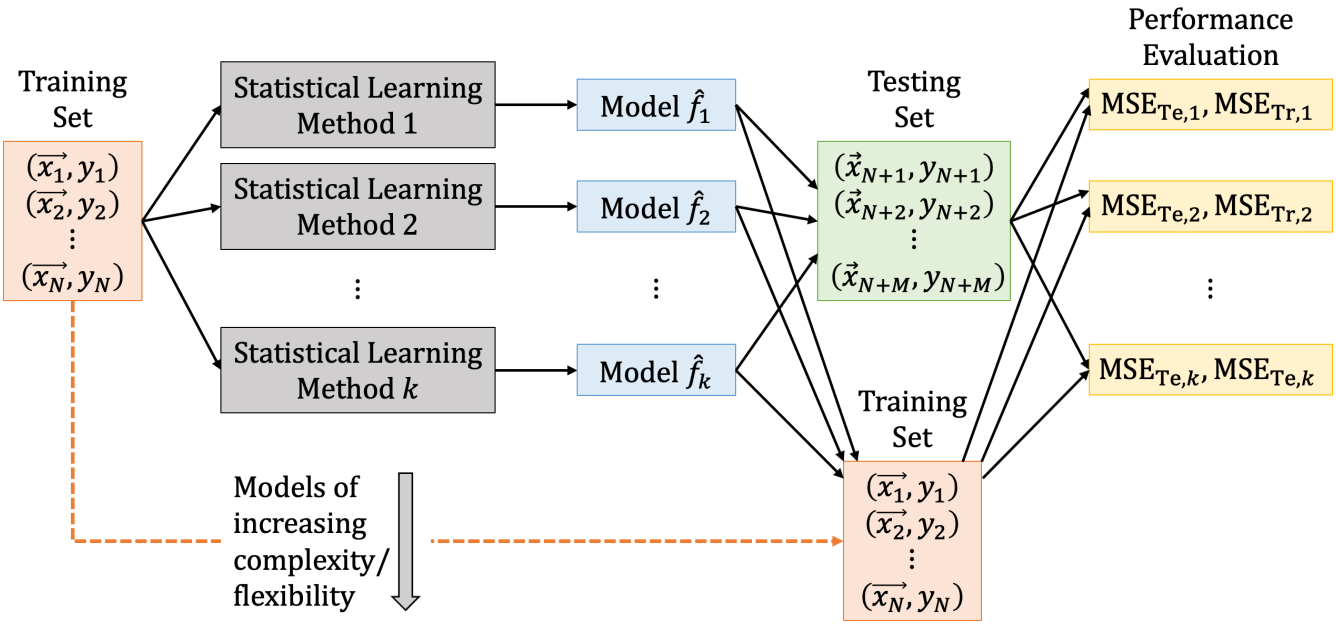


Figure 1. The training/testing paradigm. Training data is fed into a variety of statistical learning methods, possibly arranged in increasing order of complexity, yielding a sequence of models. These models are then used to make predictions on the testing set (using only the predictors variables); the 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.

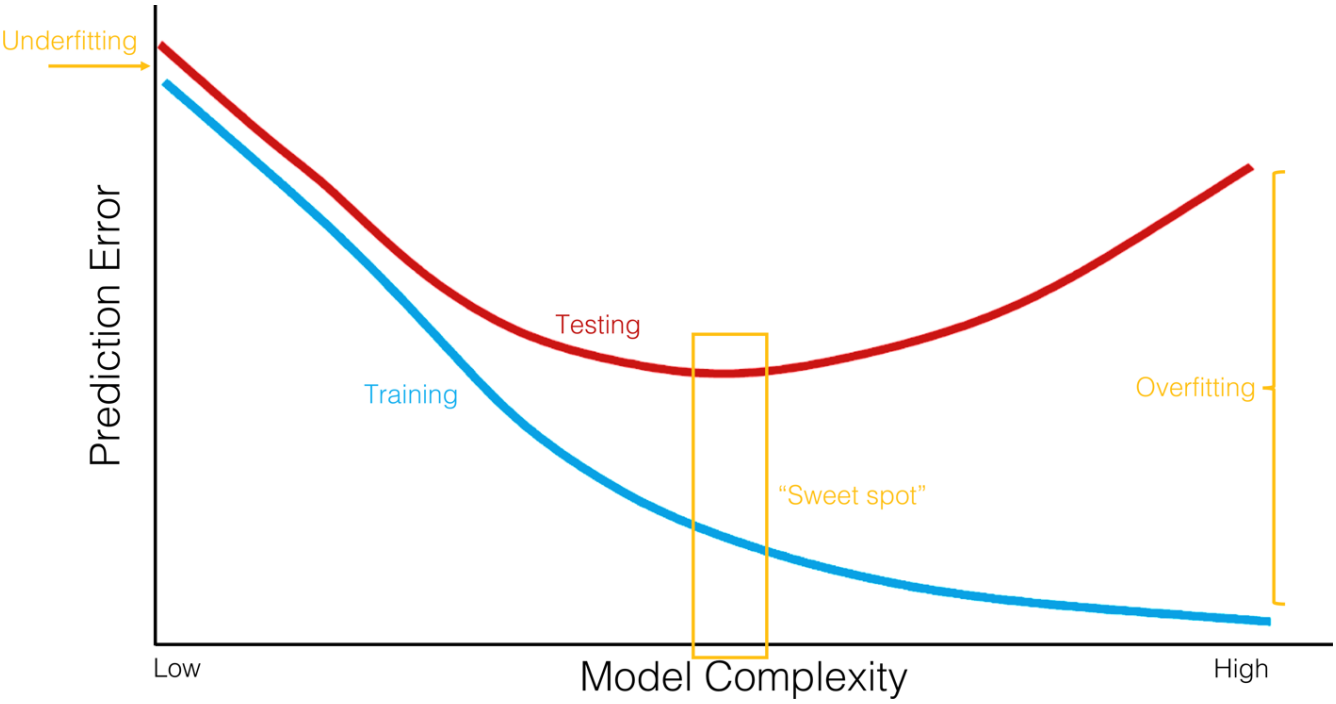


Figure 2. Generic illustration of the bias-variance trade-off; when the complexity of the model increases, the training error decreases, but the testing error eventually starts increasing. Generally, models that are too simple will have "large" prediction errors on both the training and the testing sets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

1.2 Model Evaluation (Revisited)

In a **classification setting**, the response Y is categorical, which is to say that $Y \in \mathscr{C}$, where $\mathscr{C} = \{C_1, \ldots, C_K\}$, but the objectives remain the same:

- build a classifier $C(\vec{x}^*)$ that assigns a label $C_k \in \mathscr{C}$ to test observations $\vec{x}^*$;
- understand the role of the predictors in this assignment, and
- assess the uncertainty and the accuracy of the classifier.

The main difference with the regression setting (and it's a big one), is that we do not have access to an MSE-type metric to evaluate the classifier's performance.

The counterpart of the regression function

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

is defined as follows: for $1 \leq k \leq K$, let $p_k(\vec{x}) = P(Y = C_k \mid \vec{X} = \vec{x})$ (in other words, pick the most numerous categorical label of observations for which the signature vector is $\vec{x}$) – the **Bayes optimal classifier** at $\vec{x}$ is the function

$$C(\vec{x}) = C_j \quad \text{if } p_j(\vec{x}) = \max\{p_1(\vec{x}), \ldots, p_K(\vec{x})\}.$$

As was the case or regression, it could be that there are too few observations at $\vec{X} = \vec{x}$ to estimate the probability exactly, in which case we might want to allow for **nearest neighbour averaging**:

$$\hat{C}(\vec{x}) = C_j, \quad \text{if } \tilde{p}_j(\vec{x}) = \max\{\tilde{p}_1(\vec{x}), \ldots, \tilde{p}_K(\vec{x})\},$$

where $\tilde{p}_k(\vec{x}) = P(Y = C_k \mid \vec{X} \in N(\vec{x}))$ and $N(\vec{x})$ is a neighbourhood of $\vec{x}$.[13]

The quantity that plays an analogous role to the MSE for $\tilde{C}(\vec{x})$ is the **misclassification error rate**:

$$\mathrm{ERR}_{\mathrm{Te}} = \frac{1}{M} \sum_{j=N+1}^{M} \mathscr{I}[y_j \neq \tilde{C}(\vec{x}_j)],$$

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

$$\mathscr{I}[\text{condition}] = \begin{cases} 0 & \text{if the condition is false} \\ 1 & \text{otherwise} \end{cases}$$

The Bayes optimal classifier $C(\vec{x})$ is the optimal classifier with respect to $\mathrm{ERR}_{\mathrm{Te}}$.

The **Bayes error rate**

$$\eta_{\vec{x}} = 1 - \mathrm{E}\left[\max_k P(Y = C_k \mid \vec{X} = \vec{x})\right]$$

corresponds to the irreducible error and provides a lower limit on any classifier's expected error.

[13] The curse of dimensionality is also in play when p becomes too large.

Most classifiers build structured models $\hat{C}(\vec{x})$ which **directly** approximate the Bayes optimal classifier $C(\vec{x})$ (such as support vector machines or naïve Bayes classifiers), but some classifiers build structured models $\hat{p}_k(\vec{x})$ for the **conditional probabilities** $p_k(\vec{x})$, $1 \leq j \leq K$, which are then used to build $\hat{C}(\vec{x})$ (such as logistic regression, generalized additive models, k—nearest neighbours).

The latter models are said to be **calibrated** (i.e., the relative values of $\hat{p}_k(\vec{x})$ represent relative probabilities), whereas the former are **non-calibrated** (only the most likely outcome is provided; it is impossible to say to what extent a given outcome is more likely than another).

The bias-variance trade-off is also observed in classifiers, although the decomposition is (of necessity) different (see [10] for details).

In a k—**nearest neighbours** classifier, for instance, the prediction for a new observation $(x^*, y^*) \in \mathrm{Te}$ is obtained by finding the most frequent class label of the k nearest neighbours to x^* in Tr on which the model $\hat{C}_{k\mathrm{NN}}(\vec{x})$ is built.

As the number of nearest neighbours under consideration increases, the complexity of the the model $\hat{C}_{k\mathrm{NN}}(\vec{x})$ decreases, and *vice-versa*.

We would thus expect:

- a model with a large k to underfit the data;
- a model with a small k to overfit the data, and
- models in the "Goldilock zone" to strike a balance between **prediction accuracy** and **interpretability of the decision boundary** (see Figure 3).

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

- for **numerical responses**, the optimal regression function $Y = f(\vec{X}) + \varepsilon$ is $f(x) = \mathrm{E}[Y \mid \vec{X} = \vec{x}]$;
- for **categorical responses**, the optimal classifier $Y = C(\vec{X})$ is the Bayes optimal classifier;
- models are **learned** over training data Tr;
- in practice, we learn the best model from a **restricted** group of model families;
- the best model $\hat{f}(\vec{x})$ (resp. $\hat{C}(\vec{x})$) minimizes the reducible part of the prediction error $\mathrm{MSE}_{\mathrm{Te}}$ (resp. $\mathrm{ERR}_{\mathrm{Te}}$), **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 (resp. Bayes error rate).

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).

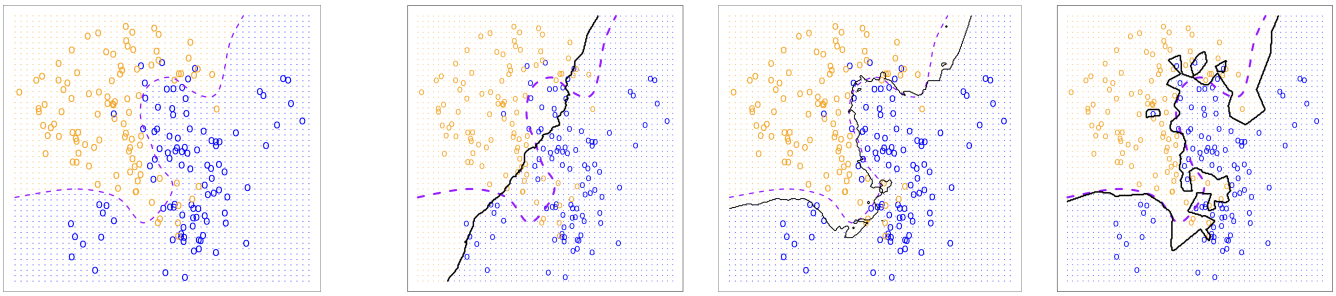


Figure 3. Illustration of the accuracy-boundary interpretability trade-off for classifiers on an artificial dataset [15]; Bayes optimal classifier $C(\vec{x})$ (leftmost), underfit $\hat{C}_{100\text{NN}}(\vec{x})$ model (2nd leftmost), Goldilock $\hat{C}_{10\text{NN}}(\vec{x})$ model (3rd leftmost), overfit $\hat{C}_{1\text{NN}}(\vec{x})$ model (4th leftmost). Notice the interplay between prediction accuracy and complexity of the decision boundary (the dashed curve in the last three graphs shows the Bayes optimal boundary).

2. Regression

Recall that, in the regression setting, the goal is to estimate the regression function

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

the solution to the regression problem

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

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

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

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 **various families of models**.

Remember, however, that no matter the $\hat{f}$,

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

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

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

which is to say that we assume that $\hat{f}_{OLS}$ is nearly globally linear (we neglect the intercept term, in one 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 $\mathrm{MSE}_{\text{Te}}(\hat{f})$ is often "small-ish".

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 predictors?
- Is the relationship linear?
- are there combinations of factors that are linked with life expectancy?

How do we answer such questions?

2.1 Regression Modeling

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:

- these regression models being straightforward to **interpret** and to **train**;
- the prediction error MSE_{Te} having a closed-form linear expression, and
- the OLS solution being computable using simple matrix manipulations.

2.1.1 Matrix Formulation

Consider a dataset $\text{Tr} = \{(\vec{x_1}, y_1), \ldots, (\vec{x_n}, y_n)\}$ with n observations and p features. The corresponding **design matrix**, **response vector**, and **coefficient vector** are, respectively,

$$\mathbf{X} = \begin{pmatrix} 1 & x_{1,1} & \cdots & x_{1,p} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n,1} & \cdots & x_{n,p} \end{pmatrix}, \quad \mathbf{Y} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}.$$

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.

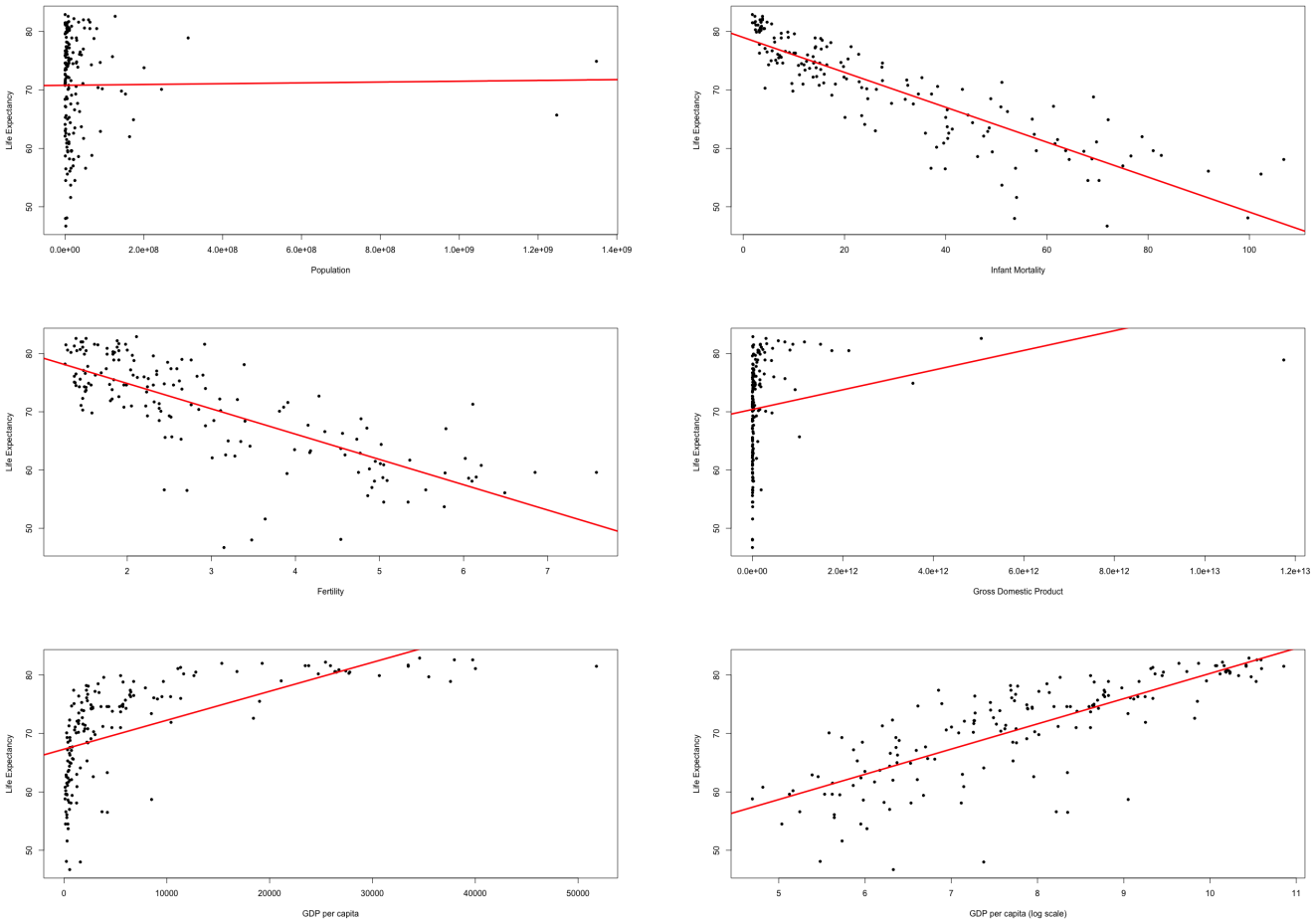


Figure 4. Scatterplots of various predictors against life expectancy, with line of best fit, in 2011, for the `Gapminder` dataset. Can they be used to answer the questions on the previous page?

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

$$\hat{f}(\vec{x}) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p.$$

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 $\mathscr{L}(\boldsymbol{\beta}) = \mathbf{0}$.

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 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

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

where $\hat{\sigma}^2$ is the variance of the residuals.[14] We say that "**we have learned the coeficient vector $\hat{\beta}$ on the training data** Tr **using linear regression**".

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

$$(1, x_1, \ldots, x_p)^\top.$$

At times, we will also use the notation to represent $\vec{x}$; the context should make clear which version is meant.

[14] 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.

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

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

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

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

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.

2.1.2 Motivating Example

We shall study a subset of the `Gapminder` dataset: the observations for 2011, the predictor variables infant mortality X_1, and fertility X_2, the derived variable GDP per capita X_3, and the response variable life expectancy Y.

The training data contains $n = 166$ observations and $p = 2$ predictor features. The design matrix $\mathbf{X}$ is thus of dimension 166×3, and

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

and

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

from which we conclude that

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

The fitted surface is thus

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

for a test 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 at the top of the next column – one should resist the temptation to predict y^* for $\mathbf{x}^* = (100, 2)$, say.

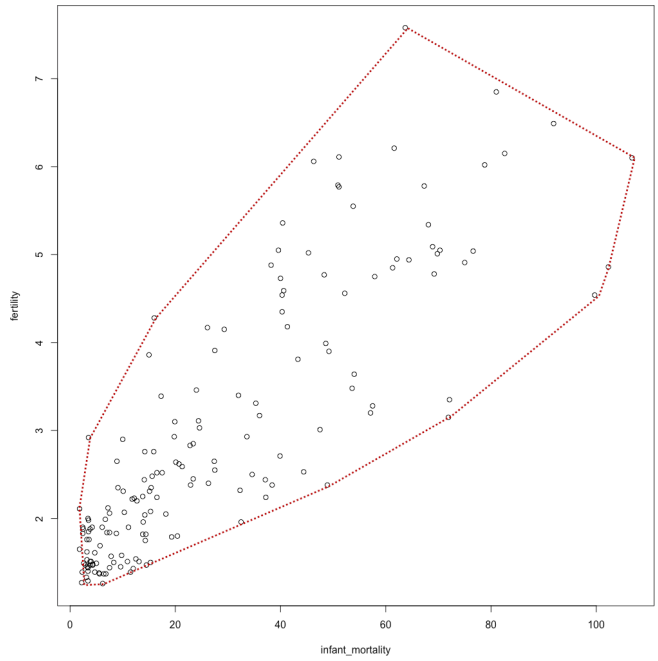
Since the family of OLS learners is a subset of all possible learners, the best we can say about $\hat{f}_{\text{OLS}}$ is that

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

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,[15] we typically also introduce a series of assumptions on the data.

[15] That is, when impose structure on the learners.



2.1.3 Least Squares Assumptions

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;
- **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

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

Typically, although that is not a requirement, it is 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 in Sections 2.1.5 and 6.

In the meantime, 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?

2.1.4 Least Squares Properties

Let us assume that the OLS assumptions are satisfied. What can we say about the linear regression results?[16]

Coefficient of Determination Let

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

and

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

The coefficient of determination of the OLS regression is the quotient

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

The coefficient of determination identifies the proportion of the variation of the data which is 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

$$R_a^2 = 1 - \frac{n-1}{n-p-1}(1-R^2) = 1 - \frac{\text{SSRes}/(n-p-1)}{\text{SSTot}/(n-1)}$$

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 plays also plays a role in the **feature selection** process.

In the Gapminder example, we have

$$\text{SSRes} = 2837.7, \quad \text{SSTot} = 11882.18,$$

so that

$$R^2 = 1 - \frac{\text{SSRes}}{\text{SSTot}} = 1 - \frac{2837.7}{11882.18} = 0.761$$

and

$$\begin{aligned} R_a^2 &= 1 - \frac{n-1}{n-p-1}(1-R^2) \\ &= 1 - \frac{166-1}{166-2-1}(1-0.761) = 0.757, \end{aligned}$$

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

[16] See [16], say, for a refresher.

Significance of Regression We can determine if at least one of the predictors $X_1, \ldots, X_p$ is useful in predicting the response Y by pitting

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

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

$$F^* = \frac{(\text{SSTot} - \text{SSRes})/p}{\text{SSRes}/(n-p-1)} \sim F_{p,n-p-1}.$$

At significance level α, if $F^* \geq F_{p,n-p-1;\alpha}$ (the $1-\alpha$ quantile of the F distributions with p and $n-p-1$ 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

$$F^* = \frac{(11882.18 - 2837.7)/2}{2837.7/(166-2-1)} = 258.169.$$

At a significance level $\alpha = 0.05$, the critical value of the $F_{2,163}$ distribution is $F_{2,163;0.05} = 3.051819$.

Since $F^* \geq F_{2,163;0.05}$, at least one of $\beta_1, \beta_2 \neq 0$, with probability 95%.

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** [26]). 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 interpretion is risky, since when X_j changes, so do the other predictors.[17]

More importantly, the interpretation can also be read as a claim of causality, which **should be avoided** when dealing with observational data.

> "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 [3])

[17] 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?

In the Gapminder the correlation between the variables X_1 and X_2 is $\rho(X_1, X_2) =$. The predictors are 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 $\text{se}(\hat{\beta}_j) = \sqrt{\hat{\sigma}^2 (\mathbf{X}^\top \mathbf{X})^{-1}_{j,j}}$, and $\hat{\sigma}^2 = \frac{\text{SSRes}}{n-p-1}$, 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, $n = 166$, $p = 2$, $\hat{\beta}_1 = -0.276$ and $(\mathbf{X}^\top \mathbf{X})^{-1}_{1,1} = 0.00003534$, so that

$$\hat{\sigma}^2 = \frac{2837.7}{163} = 17.5, \ \text{se}(\hat{\beta}_1) = \sqrt{17.5 \cdot 3.5 \times 10^{-5}} = 0.025$$

and

$$t^* = -\frac{0.276}{0.025} = -11.14.$$

At a significance level $\alpha = 0.05$, the critical value of the T_{164} distribution is $t_{164;0.025} = -1.97$. Since $|t^*| \geq |t_{164;0.025}|$, $\beta_1 \neq 0$ with probability 95%.

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:

$$\text{C.I.}(\beta_j; \alpha) \equiv \hat{\beta}_j \pm z_{\alpha/2} \cdot \text{se}(\hat{\beta}_j);$$

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

$$\text{C.I.}(\beta_j; 0.05) \equiv \hat{\beta}_j \pm 2\,\text{se}(\hat{\beta}_j).$$

In the Gapminder example, we have

coeff.	est.	s.e.	t^*	95% C.I.
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 C.I.; rather, it suggests that the approach used to build the 95% C.I. will yield an interval in which the true β_j will reside approximately 95% of the time.[18]

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

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

is $[-0.33, -0.27]$ (the estimates are necessarily distinct as well: $\hat{\beta}_1 = -0.2763 \neq -0.2989 = \hat{\gamma}_1$).

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 subset of predictor subsets. OLS approaches include **forward selection** and **backward selection**.[19]

Forward selection is a bottom-up approach:

1. start with the **null model** $\mathscr{M}_0 : Y = \beta_0 + \varepsilon$;
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 SSRes:

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

3. add to that model the variable that results in the lowest SSRes among all the two-variable models:

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

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

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

regressions to run.

These methods are, frankly, not ideal in the machine learning framework (see Section 5 for alternatives).

[18] Compare with the Bayesian notion of a **credible interval** [8].

[19] We will discuss these approaches in detail in Section 5.2.

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, genearlized 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).

2.1.5 Generalizations of Least Squares

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 Sections 3 and 7);
- 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 6);
- tree-based methods and ensemble learning methods (bagging, random forests, boosting) are used for predictor interactions (see Sections 7.1 and 7.5);
- regularization methods (ridge regression, LASSO, elastic net) facilitate the process of model selection and feature selection (see 5.3).

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

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

A GLM consists of:

- a systematic component $\mathbf{x}_i^\top \boldsymbol{\beta}$;
- a random component specified by the distribution $\mathscr{D}$ 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** (or at least differentiable) and **monotonic** (and so invertible).

We could specify any distribution $\mathscr{D}$ for the response Y_i, but they are usually selected from the **exponential family** of distributions.[20]

OLS is an example of GLM, with:

- 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, C.I., etc.) still apply;
- they are easily implemented in SAS (`proc genmod`), R (`glm()`), etc., and
- the framework unites various regression modeling approaches (OLS, logistic, Poisson, etc.) under a single umbrella.

[20] These are distributions have probability density functions that satisfy

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

This includes the normal, binomial, Poisson, Gamma distributions, etc. These are all distributions with **conjugate priors** (see [8]).

2.2 Comparison Between kNN and OLS

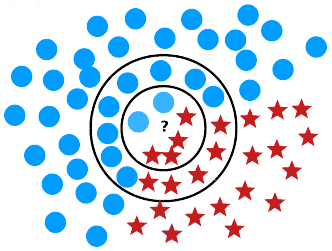
We are going to try to get a better intuitive sense of the bias-variance trade-off by comparing ordinary least squares (OLS), a rigid yet simple model (as measured by the number of **effective parameters**), with k—nearest neighbours (kNN), a very flexible yet more complex model (again, according to the number of effective parameters).

Given an input vector $\mathbf{z} \in \mathbb{R}^p$, the k—nearest neighbours model predicts the response Y as the average

$$\hat{Y} = \text{Avg}\{Y(\mathbf{x}) \mid \mathbf{x} \in N_k(\mathbf{z})\} = \frac{1}{k} \sum_{\mathbf{x} \in N(\mathbf{z})} Y(\mathbf{x}),$$

where $Y(\mathbf{x})$ is the known response for predictor $\mathbf{x} \in \text{Tr}$ and $N_k(\mathbf{z})$ is the set of the k training observations nearest to $\mathbf{z}$.[21]

Of course, the prediction may depend on the value of k: in the classification image below, the 6NN prediction would be a red star, whereas the 19NN model prediction would be a blue disk.[22]



The following classification example (based on [10]) illustrates some of the trade-off consequences. Consider a training dataset Tr consisting of 200 observations with features $(x_1, x_2) \in \mathbb{R}^2$ and responses $y \in \{\text{BLUE}_{(=0)}, \text{ORANGE}_{(=1)}\}$.

Throughout, let $[\cdot] : \mathbb{R} \to \{\text{BLUE}, \text{ORANGE}\}$ denote the function

$$[w] = \begin{cases} \text{BLUE} & w \leq 0.5 \\ \text{ORANGE} & w > 0.5 \end{cases}$$

Linear Fit Fit an OLS model

$$\hat{y}(\mathbf{x}) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$$

on Tr; the class prediction is $\hat{g}(\mathbf{x}) = [\hat{y}(\mathbf{x})]$.

The **decision boundary**

$$\partial_{\text{OLS}} = \{(x_1, x_2) \mid \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0.5\}$$

is shown in Figure 5 (on the left); it is a straight line which can be described using only 2 **effective parameters**.[23]

There are several **misclassifications** on both sides of ∂_{OLS}; even though errors seem to be unavoidable, the OLS model is likely to be **too rigid**.

kNN Fit If $\hat{y}(\mathbf{x})$ represents the proportion of ORANGE points in $N_k(\mathbf{x})$, then the class prediction is $\hat{g}(\mathbf{x}) = [\hat{y}(\mathbf{x})]$.

The decision boundaries $\partial_{1\text{NN}}$ and $\partial_{15\text{NN}}$ are displayed in Figure 5 (in the middle and on the right, respectively). They are both irregular: $\partial_{1\text{NN}}$ is overfit, whereas $\partial_{15\text{NN}}$ is probably not so (although neither is great for **interpretability**).

The effective parameters are not as obviously defined for this model; one approach is to view kNN as a model that fits 1 parameter (a mean) to each **ideal** (non-overlapping) **neighbourhood** in the data, so that the number of effective parameters is roughly equal to the number of such neighbourhoods:

$$\frac{N}{k} \approx \begin{cases} 13 & \text{when } k = 15 \\ 200 & \text{when } k = 1 \end{cases}$$

The kNN models are thus fairly complex, in comparison with the OLS model.

There are no misclassification for $k = 1$, and several in the case $k = 15$ (but not as many as with the OLS model). The 15NN model seems to strike a balance between various competing properties; it is likely nearer the "sweet spot" of the test error curve.[24]

Conclusions The OLS model is **stable** (adding a few training observations is unlikely to alter the fit substantially), but **biased** (the assumptions of a valid linear fit is questionable); the kNN models are **unstable** (adding a few training observations is likely to alter the fit substantially, especially for small values of k), but **unbiased** (no apparent assumptions are made about the data).

So which approach is best? That depends entirely on what the ultimate task is: description, prediction, etc.

In predictive data science, machine learning, and artificial intelligence, the validity of modeling assumptions take a backseat to a model's ability to make good predictions on new (and unseen) observations.

Naturally, we would expect that models whose assumptions are met are more likely to make good predictions than models for whom that is not the case, but it does not need to be the case.

The theory of linear models is mature and extensive, and we could have discussed a number of its other features.

Machine learning methods are not meant to replace or supplant classical statistical analysis methods, but rather, to **complement them**. They simply provide different approaches to **gain insights from data**.

[21] The notion of proximity depends on the distance metric in use; the Euclidean case is the most common, but it does not have to be that one.

[22] For classification, kNN models use the mode instead of the average.

[23] Their number is a measure of a model's **complexity**.

[24] Remember that we have not evaluated the performance of the models on a testing set; we have only described some of its behaviour on the training set Tr.

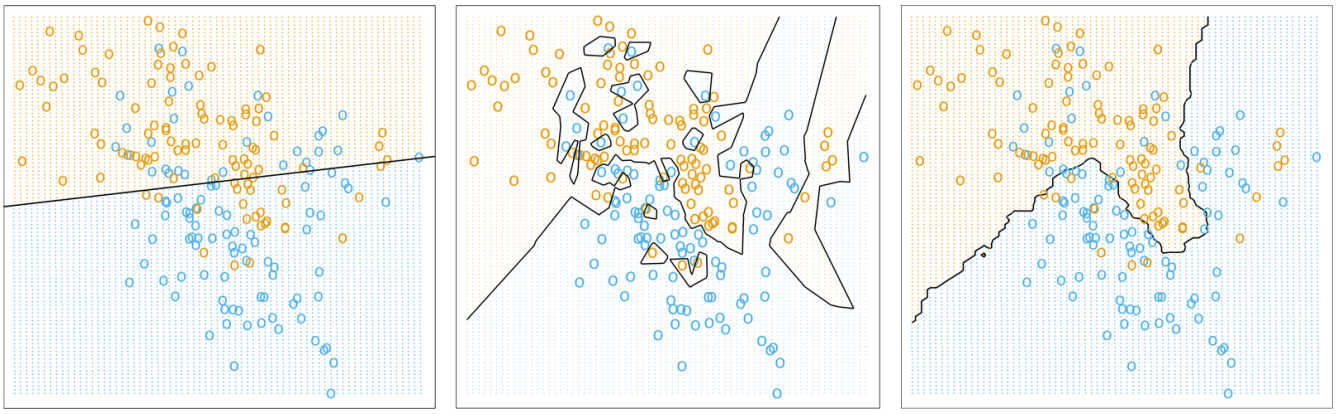


Figure 5. Classification based on OLS (left), 1NN (middle), and 15NN (right) [10].

3. Classification

Qualitative variables take values (the **levels**) in an **unordered** set $\mathscr{C} = \{C_1, \ldots, C_K\}$.

For instance,

- hair colour $\in$ {black, red, blond, grey, other}
- email message $\in$ {ham, spam}
- life expectancy $\in$ {high, low}

For a training set Tr with observations $(\vec{X}, Y) \in \mathbb{R}^p \times \mathscr{C}$, the **classification problem** is to build a classifier $\hat{C} : \mathbb{R}^p \to \mathscr{C}$ to approximate the optimal Bayes classifier $C : \mathbb{R}^p \to \mathscr{C}$ (see Section 1.3, p. 1.3).

In many instances, we might be more interested in the probabilities

$$\pi_k(\mathbf{x}) = P\{\hat{C}(\mathbf{x}) = C_k\}, \quad k = 1, \ldots, K$$

than in the classification predictions themselves.

Typically, the classifier $\hat{C}$ is **built** on a training set

$$\text{Tr} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{N}$$

and **evaluated** on a testing set

$$\text{Te} = \{(\mathbf{x}_i, y_i)\}_{i=N+1}^{M}.$$

Example

Let us revisit the `Gapminder` dataset, again focusing on observations from 2011, with the difference that life expectancy is now recorded as "high" (1) if it falls above 72.45 (the median in 2011), and as "low" (0) otherwise.

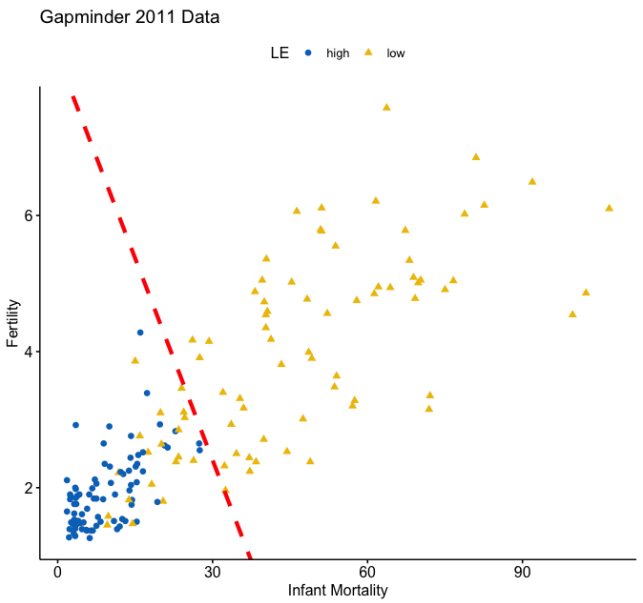
We could run a simple linear regression of Y on $\vec{X}$ over Tr and obtain the model

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 \cdot \text{infant mortality} + \hat{\beta}_2 \cdot \text{fertility},$$

from which we would classify an observation's life expectancy as

$$\hat{C}(\vec{X}) = \begin{cases} \text{high} & \text{if } \hat{Y} > 0.5 \\ \text{low} & \text{else} \end{cases}$$

In this specific example, the OLS approach does a good job, as the data is roughly **linearly separable** (see below).



This will not, however, usually be the case. In this example, the optimal regression function is

$$f(\vec{x}) = \text{E}[Y \mid \vec{X} = \vec{x}] = P(Y = 1 \mid \vec{X} = \vec{x}) = p_1(\vec{x})$$

because Y is a binary variable; this might lead us to believe that $f(\vec{x})$ could also be used to directly classify and determine the class probabilities for the data, in which case there would be no need for a **separate classification apparatus**.

There is one major drawback with this approach: if linear regression is used to model the data (which is to say, if we assume that $f(\vec{x}) \approx \vec{x}^\top \vec{\beta}$), we need to insure that $\hat{f}_{\text{OLS}}(\vec{x}) \in [0, 1]$ for all $\vec{x} \in \text{Te}$. This, in general, cannot be guaranteed.

Another problem arises if we study the residual situation further. If we model $Y = \{0, 1\}$ with an OLS regression, we

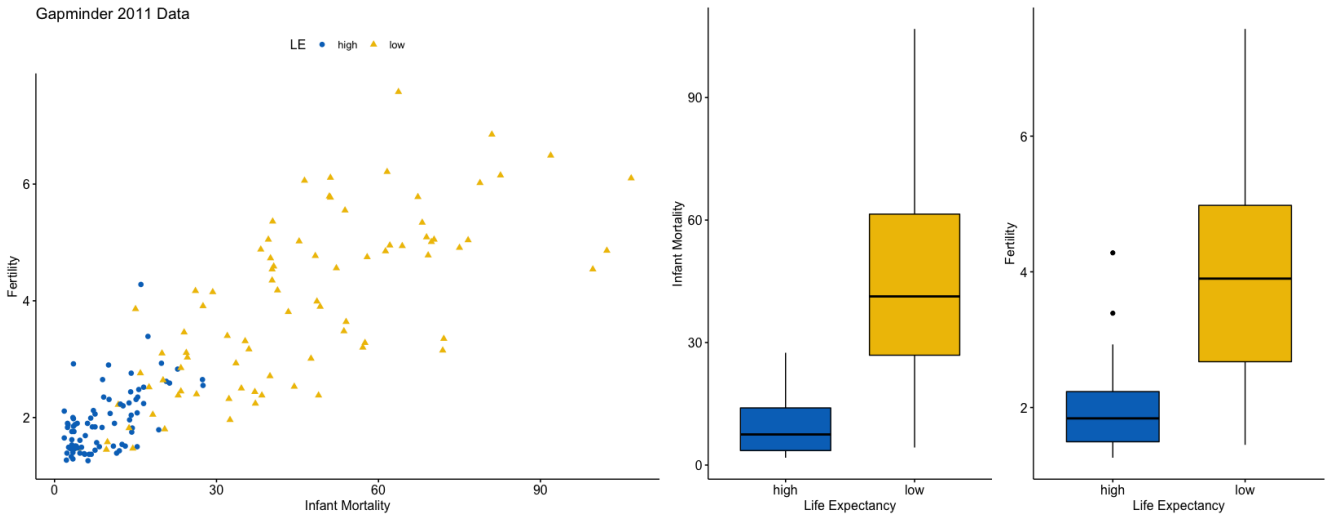


Figure 6. Visualization of life expectancy (as a class), infant mortality, and fertility in the 2011 Gapminder dataset.

have

$$Y_i = \mathbf{x}_i^\top \boldsymbol{\beta} + \varepsilon_i.$$

Thus

$$\varepsilon_i = Y_i - \mathbf{x}_i^\top \boldsymbol{\beta} = \begin{cases} 1 - \mathbf{x}_i^\top \boldsymbol{\beta} & \text{if } Y_i = 1 \\ -\mathbf{x}_i^\top \boldsymbol{\beta} & \text{if } Y_i = 0 \end{cases}$$

But OLS assumes that $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$, and so is not appropriate to model the response.

Furthermore, $\text{Var}(Y_i) = p_1(\mathbf{x}_i)(1 - p_1(\mathbf{x}_i))$ and

$$\text{Var}(\varepsilon_i) = \text{Var}(Y_i - p_1(\mathbf{x}_i)) = \text{Var}(Y_i) = p_1(\mathbf{x}_i)(1 - p_1(\mathbf{x}_i)),$$

which is not constant as it depends on $\mathbf{x}_i$.

The OLS assumptions are thus violated at every turn[25] – OLS is not a good fit/modeling approach to estimate

$$p_k(\vec{x}) = P(Y = C_k \mid \vec{X} = \vec{x}).$$

3.1 Logistic Regression

The problems presented above point to OLS not being an ideal method for classification, but the linear regression still provided a good separator in the Gapminder example.

This suggests that we should not automatically reject the possibility of first transforming the data and then seeing if OLS might not be an appropriate modeling strategy on the transformed data.

[25] There is another way in which OLS would fail, but it has nothing to do with the OLS assumptions per se. When the set of qualitative responses contains more than 2 level (such as $\mathscr{C} = \{\text{low}, \text{medium}, \text{high}\}$, for instance), the response is usually encoded using numerals to facilitate the implementation of the analysis:

$$Y = \begin{cases} 0 & \text{if low} \\ 1 & \text{if medium} \\ 2 & \text{if high} \end{cases}$$

This encoding suggests an **ordering** and a **scale** between the levels (the difference between "high" and "medium" is equal to the difference between "medium" and "low", and half again as large as the difference between "high" and "low", say). OLS is not appropriate in this context.

3.1.1 Formulations

In **logistic regression**, we are seeking an invertible function $g : \mathbb{R} \to [0,1]$, $g(y^*) = y$, such that $g^{-1}(y_i) = y_i^*$. This quantity must behave like a probability; in the 2-class setting, we use $g_L(y^*)$ to approximate the probability

$$p_1(\vec{x}) = P(Y = 1 \mid \vec{X} = \vec{x}).$$

The idea is to run OLS on a transformed training set

$$\text{Tr}^* = \{(\mathbf{x}_i, y_i^*)\}_{i=1}^N,$$

and to transform the results back using $y_i = g(y_i^*)$.

There are many such functions: the **probit** model,[26] which we will not discuss, and the **logit** model regression model are two common approaches.

3.1.2 Logit Model

The logit model uses the transformation

$$y = g_L(y^*) = \frac{e^{y^*}}{1 + e^{y^*}}.$$

It is such that

$$g_L^{-1}(0) = -\infty, \quad g_L^{-1}(1) = \infty, \quad g_L^{-1}(0.5) = 0, \quad \text{etc.}$$

We solve for y^* in order to get a transformed response $y^* \in \mathbb{R}$ (instead of one restricted to $[0,1]$):

$$p_1(\vec{x}) = \frac{e^{y^*}}{1 + e^{y^*}} \iff y^* = g_L^{-1}(y) = \ln\left(\frac{p_1(\vec{x})}{1 - p_1(\vec{x})}\right)$$

It is the **log-odds** transformed observations that we attempt to fit with an OLS model:

$$\hat{Y}^* = \ln\left(\frac{p_1(\vec{x})}{1 - p_1(\vec{x})}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p = \vec{X}^\top \vec{\beta}.$$

In order to make a prediction for $p_1(\vec{x})$, we estimate y^* and use the logit transformation to obtain y.

[26] The probit transformation uses $g_P(y^*) = \Phi(y^*)$, where Φ is the cumulative distribution function of $\mathcal{N}(0,1)$.

For instance, if $\mathbf{x}^\top\hat{\boldsymbol{\beta}} = 0.68$, then

$$\hat{y}^* = \ln\left(\frac{\hat{p}_1(\mathbf{x})}{1-\hat{p}_1(\mathbf{x})}\right) = 0.68$$

and

$$\hat{p}_1(\mathbf{x}) = \frac{e^{y^*}}{1+e^{y^*}} = \frac{e^{0.68}}{1+e^{0.68}} = 0.663.$$

Depending on the **decision rule threshold** γ, we may thus predict that $\hat{C}(\mathbf{x}) = C_1$ if $p_1(\mathbf{x}) > \gamma$ and $\hat{C}(\mathbf{x}) = C_2$ otherwise.

The technical challenge is in obtaining the coefficients $\hat{\boldsymbol{\beta}}$; they are found by **maximizing the likelihood** (see [13])

$$\begin{aligned} L(\boldsymbol{\beta}) &= \prod_{y_i=1} p_1(\mathbf{x}_i)\prod_{y_i=0}(1-p_1(\mathbf{x}_i)) \\ &= \prod_{y_i=1}\frac{\exp(\mathbf{x_i}^\top\boldsymbol{\beta})}{1+\exp(\mathbf{x_i}^\top\boldsymbol{\beta})}\prod_{y_i=0}\frac{1}{1+\exp(\mathbf{x_i}^\top\boldsymbol{\beta})}, \end{aligned}$$

that is to say,

$$\begin{aligned} \hat{\boldsymbol{\beta}} &= \underset{\boldsymbol{\beta}}{\text{argmax}}\{L(\boldsymbol{\beta})\} = \underset{\boldsymbol{\beta}}{\text{argmax}}\{\ln L(\boldsymbol{\beta})\} \\ &= \underset{\boldsymbol{\beta}}{\text{argmax}}\left\{\sum_{y_i=1}\ln p_1(\mathbf{x}_i) + \sum_{y_i=0}\ln(1-p_1(\mathbf{x}_i))\right\} \\ &= \ldots \text{ (terms in } \boldsymbol{\beta} \text{ and the observations } \mathbf{x}_i). \end{aligned}$$

The optimizer is found using numerical methods; in `R`, the function `glm()` computes the maximum likelihood estimate directly.

Example

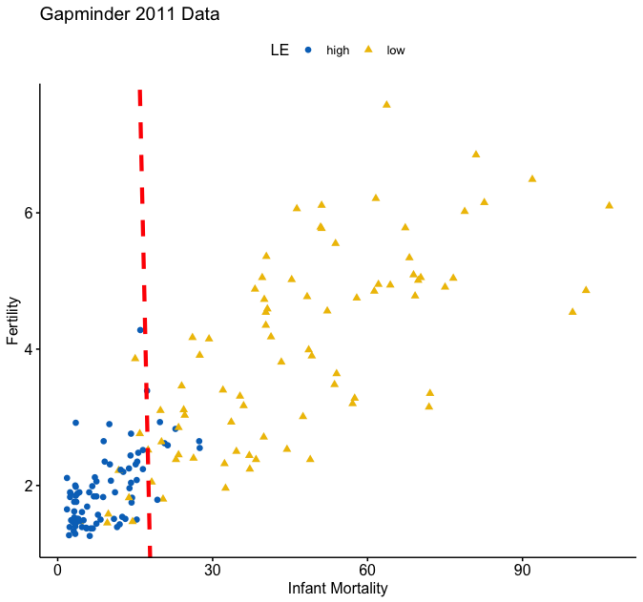
Using the `Gapminder` data from the start of the section, we obtain the following model:

$$\hat{y}^* = \ln\left(\frac{P(Y=\text{high}\mid\vec{X})}{1-P(Y=\text{high}\mid\vec{X})}\right) = 4.59 - 0.22X_1 - 0.06X_2.$$

For a decision rule threshold of $\gamma = 0.5$, the decision boundary is shown at the top of the next column (compare with the linear regression boundary).

What is the estimated probability that the life expectancy is high in a country whose infant mortality is 15 and whose fertility is 4? By construction,

$$\begin{aligned} p_1(Y=\text{high}|X_1=15,X_2=4) &\approx g_L([1,15,24]^\top\hat{\boldsymbol{\beta}}) \\ &= \frac{\exp(4.59-0.22(15)-0.06(24))}{1+\exp(4.59-0.22(15)-0.06(24))} \\ &= \frac{\exp(0.9526322)}{1+\exp(0.9526322)} = 0.72. \end{aligned}$$



At this point, we might be wondering how all of this squares up with the statistical learning framework we have been describing. No testing set has made an appearance, no misclassification or mean squared error rate has been calculated.

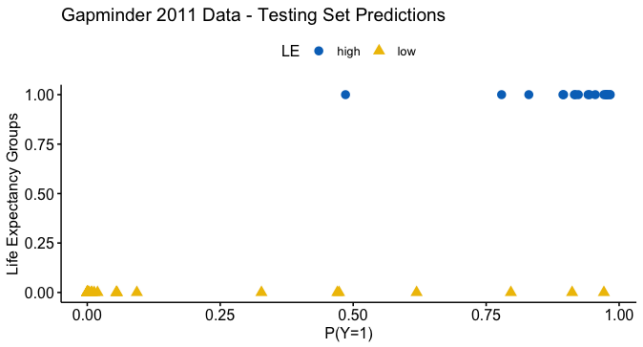
We **randomly** select 116 observations and train a logistic regression model on this training set Tr:

$$\hat{y}^* = 6.12 - 0.21x_1 - 0.67x_2.$$

Now, compute

$$\hat{p}_i = P(Y_i = \text{high}|X_1 = x_{1,i}, X_2 = x_{2,i}) = \frac{\exp(\hat{y}_i^*)}{1+\exp(\hat{y}_i^*)}$$

on the observations in the testing set Te (see result below).



We obtain

$$\text{MSE}_{\text{Te}} = \frac{1}{50}\sum_{i=1}^{50}(\hat{p}_i - \mathbb{1}[Y_i = \text{high}])^2 = 0.075.$$

Is that a good test error? It is difficult to answer without more context.

Perhaps a more intuitive way to view the situation is to make actual predictions. For $\alpha \in [0,1]$, define

$$\text{pred}_i(\alpha) = \begin{cases} \text{high} & \text{if } \hat{p}_i > \alpha \\ \text{low} & \text{else} \end{cases}$$

In the specific version of Te used in this example, 36% of the nations had a high life expectancy. If we set $\alpha = 0.81$, then the model predicts that 36% of the test nations will have a high life expectancy, and the **confusion matrix** on Te (see p. 20) is shown below:

$\alpha = 0.81$		**prediction** 0	1
actual	0	30	2
	1	2	16

But why do we pick $\alpha = 0.81$ instead of $\alpha = 0.5$, say (which may be the only rational choice in the absence of information)?

In the latter case, 42% of nations are predicted to have high life expectancy, and the confusion matrix on Te is as below.

$\alpha = 0.5$		**prediction** 0	1
actual	0	28	4
	1	2	17

We will revisit this question in Section 3.3.

3.2 Discriminant Analysis

In logistic regression, we model $P(Y = C_k \mid \mathbf{x})$ directly *via* the logistic function

$$p_1(\mathbf{x}) = \frac{\exp(\mathbf{x}^\top \hat{\boldsymbol{\beta}})}{1 + \exp(\mathbf{x}^\top \hat{\boldsymbol{\beta}})}.$$

We have discussed some of the properties of the process in the previous section. It should be noted that logistic regression is sometimes contra-indicated:

- when the classes are **well-separated**, the coefficient estimates may be **unstable** (adding as little as one additional point to Tr could change the coefficients substantially);
- when Tr is small and the distribution of the predictors is roughly **Gaussian** in each of the classes $Y = C_k$, the coefficient estimates may be unstable too;
- when there are more than 2 response levels, it is not always obvious how to select an extension of logistic regression.

In **discriminant analysis** (DA), we model instead

$$P(\mathbf{x} \mid Y = C_k),$$

the distribution of the predictors $\vec{X}$ conditional on the level of Y, and use Bayes' Theorem to obtain

$$P(Y = C_k \mid \mathbf{x}),$$

the probability of observing the response conditional on the predictors.

Let $\mathscr{C} = \{C_1, \ldots, C_K\}$ be the K response levels, $K \geq 2$, and denote by π_k the probability that a random observation lies in C_k, for $k \in \{1, \ldots, K\}$; π_k is the **prior**

$$\pi_k = P(Y = C_k) = \frac{|C_k|}{N}.$$

Let $f_k(\mathbf{x}) = P(\mathbf{x} \mid Y = C_k)$ be the **conditional density function** of the distribution of $\vec{X}$ in C_k; we would expect $f_k(\mathbf{x})$ to be large if there is a high probability that an observation in C_k has a corresponding predictor $\vec{X} \approx \mathbf{x}$, and small otherwise.

According to Bayes' Theorem,

$$\begin{aligned} p_k(\mathbf{x}) = P(Y = C_k \mid \mathbf{x}) &= \frac{P(\mathbf{x} \mid Y = C_k) \cdot P(Y = C_k)}{P(\mathbf{x})} \\ &= \frac{P(\mathbf{x} \mid Y = C_k) \cdot P(Y = C_k)}{\sum_{j=1}^K P(\mathbf{x} \mid Y = C_j) \cdot P(Y = C_j)} \\ &= \frac{\pi_k f_k(\mathbf{x})}{\sum_{j=1}^K \pi_j f_j(\mathbf{x})}. \end{aligned}$$

Given an observation $\mathbf{x} \in$ Te, the DA classifier is

$$\hat{C}_{\text{DA}}(\mathbf{x}) = C_{\text{argmax}_j \{p_{j(\mathbf{x})}\}}.$$

In order to say more about discriminant analysis, we need to make additional assumptions on the nature of the underlying distributions.

3.2.1 Linear Discriminant Analysis

If there is only one predictor ($p = 1$), we make the **Gaussian assumption**,

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left[-\frac{1}{2}\left(\frac{x - \mu_k}{\sigma_k}\right)^2\right],$$

where μ_k and σ_k are the mean and the standard deviation, respectively, of the predictor for all observations in class C_k.[27]

If we further assume that $\sigma_k \equiv \sigma$ for all k, then

$$\begin{aligned} p_k(x) &= \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2\right]}{\sum_{j=1}^K \pi_j \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu_j}{\sigma}\right)^2\right]} \\ &= \frac{\pi_k \exp\left[-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2\right]}{\sum_{j=1}^K \pi_j \exp\left[-\frac{1}{2}\left(\frac{x-\mu_j}{\sigma}\right)^2\right]} \\ &= \frac{\pi_k \exp\left[\frac{\mu_k}{\sigma^2}\left(x - \frac{\mu_k}{2}\right)\right] \exp\left(-\frac{x^2}{2\sigma^2}\right)}{\sum_{j=1}^K \pi_j \exp\left[\frac{\mu_j}{\sigma^2}\left(x - \frac{\mu_j}{2}\right)\right] \exp\left(-\frac{x^2}{2\sigma^2}\right)} \\ &= \pi_k \exp\left[\frac{\mu_k}{\sigma^2}\left(x - \frac{\mu_k}{2}\right)\right] \cdot A(x). \end{aligned}$$

[27] Any other predictor distribution could be used if it is more appropriate for Tr, and we could assume that the standard deviations or the means (or both) are identical across classes.

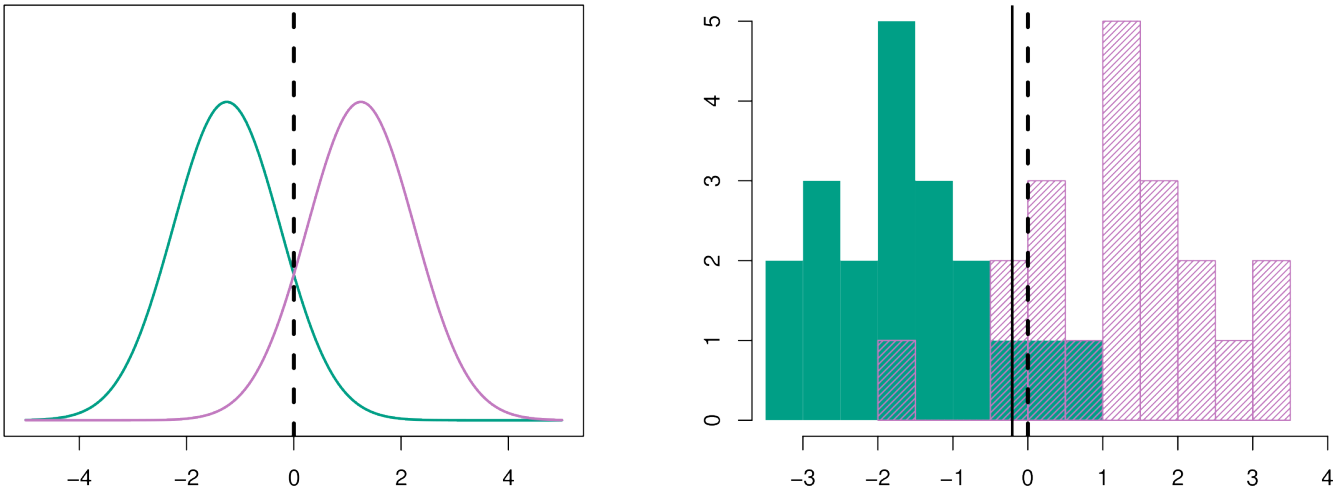


Figure 7. Midpoint of two theoretical normal distributions (dashed line); midpoint of two empirical normal distributions (solid line). Observations to the left of the decision boundary are classified as green, those to the right as purple. [15].

We do not have to compute the actual probabilities $p_k(x)$ directly if we are only interested in classification; in that case, the **discriminant score** for each class may be more useful:

$$\delta_k(x) = \ln p_k(x) = \ln \pi_k + x\frac{\mu_k}{\sigma^2} - \frac{\mu_k}{2\sigma^2} + \ln A(x).$$

As $\ln A(x)$ is the same for all k, we can drop it from the score (it does not contribute to relative differences in class scores); given an observation $x \in \text{Te}$, the **linear discriminant analysis** (LDA) classifier with $p = 1$ is

$$\hat{C}_{\text{LDA}}(\mathbf{x}) = C_{\text{argmax}_j\{\hat{\delta}_{j(\mathbf{x})}\}}.$$

Under other assumptions on the density function, the discriminant score formulation may change. The "linear" in LDA comes from the linearity of the discriminant scores δ_k (after the $\ln A(x)$ term has been dropped).

If $K = 2$ and $\pi_1 = \pi_2 = 0.5$, the midpoint $x^* = \frac{1}{2}(\mu_1 + \mu_2)$ of the predictor means in C_1 and C_2 plays a crucial role. Indeed, the discriminant scores $\delta_1(x)$ and $\delta_2(x)$ meet when

$$x^*\frac{\mu_1}{\sigma^2} - \frac{\mu_1^2}{2\sigma^2} = x^*\frac{\mu_2}{\sigma^2} - \frac{\mu_2^2}{2\sigma^2} \implies x^* = \frac{\mu_1 + \mu_2}{2},$$

as long as $\mu_1 \neq \mu_2$. If $\mu_1 < \mu_2$, say, then the decision rule simplifies to

$$\hat{C}(x) = \begin{cases} C_1 & \text{if } x \leq x^* \\ C_2 & \text{if } x > x^* \end{cases}$$

The principle is illustrated in Figure 7; in practice, we estimate π_k, μ_k and σ from Tr:

$$\hat{\pi}_k = \frac{N_k}{N}, \quad \hat{\mu}_k = \frac{1}{N_k}\sum_{y_i \in C_k} x_i$$

$$\hat{\sigma}^2 = \sum_{k=1}^{K}\frac{N_k - 1}{N - K}\left(\frac{1}{N_k - 1}\sum_{y_i \in C_k}(x_i - \hat{\mu}_k)^2\right).$$

If there are $p > 1$ predictors, we can still make the Gaussian assumption, but adapted to $\mathbb{R}^p$:

$$f_k(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}}\exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^\top \Sigma_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k),\right],$$

where $\boldsymbol{\mu}_k = (\overline{X_1}, \ldots, \overline{X_p})$ and $\Sigma_k(j,i) = \text{Cov}(X_i, X_j)$ for all $\vec{X}$ with $Y = C_k$.

If we further assume that $\sigma_k \equiv \Sigma$ for all k, then we can show that the discriminant score is, again, linear (in $\mathbf{x}$):

$$\delta_{k;\text{LDA}}(\mathbf{x}) = \mathbf{x}^\top \Sigma^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}^\top \Sigma^{-1}\boldsymbol{\mu}_k + \ln \pi_k = c_{k,0} + \mathbf{c}_k^\top \mathbf{x}.$$

We can estimate $\boldsymbol{\mu}_k$ and Σ from the data, from which we can recover the estimates

$$P(Y = C_k \mid \mathbf{x}) \approx \hat{p}_k(\mathbf{x}) = \frac{\exp(\hat{\delta}_{k;\text{LDA}}(\mathbf{x}))}{\sum_{j=1}^{K}\exp(\hat{\delta}_{j;\text{LDA}}(\mathbf{x}))}.$$

The decision rule is as before: given an observation $\mathbf{x} \in \text{Te}$, the LDA classifier with $p > 1$ is

$$\hat{C}_{\text{LDA}} = C_{\text{argmax}_j\{\hat{\delta}_{j;LDA}(\mathbf{x})\}}.$$

3.2.2 Quadratic Discriminant Analysis

The assumption that the conditional probability functions be Gaussians with the same covariance in each training class may be a stretch in some situations.

If $\Sigma_i \neq \Sigma_j$ for at least one pair of classes (i, j), then a similar process gives rise to **quadratic discriminant analysis** (QDA), which reduces to discriminant scores

$$\begin{aligned}\delta_{k;\text{QDA}}(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \Sigma_k^{-1}(\mathbf{x} - \boldsymbol{\mu}) + \ln \pi_k \\ &= -\frac{1}{2}\mathbf{x}^\top \Sigma_k^{-1}\mathbf{x} + \mathbf{x}^\top \Sigma_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}^\top \Sigma_k^{-1}\boldsymbol{\mu}_k + \ln \pi_k.\end{aligned}$$

To learn the LDA model, we must estimate $Kp + \frac{p(p+1)}{2}$ parameters from Tr;[28] to learn QDA, $K\left(p + \frac{p(p+1)}{2}\right)$.[29] QDA is thus more complex (and more flexible) than LDA.

The latter is recommended if Tr is small; the former if Tr is large, but LDA will yield high bias if the $\Sigma_k \equiv \Sigma$ assumption is invalid.

Note that LDA gives rise to nearly **linear separating hypersurfaces** and QDA to **quadratic** ones.

3.2.3 Gaussian Naive Bayes Classification

If we assume further that each Σ_k is **diagonal** (that is, if we assume that the features are independent from each other in each class), we obtain the **Gaussian naïve Bayes classifier** (GNBC), with discriminant scores given by

$$\delta_{k;\text{GNBC}}(\mathbf{x}) = -\frac{1}{2}\sum_{j=1}^{p}\frac{(x_j - \mu_{k,j})^2}{\sigma_{k,j}^2} + \ln \pi_k.$$

The classification process continues as before.

Note that the assumption of independence is usually not met, hence the "naïve" part in the name. In spite of this, GNBC can prove very useful when p is too large, where both LDA and QDA breakdown.

Note that this approach can also be used for mixed feature vectors, by using combinations of p.m.f.s and p.d.f.s in $f_{k,j}(x_j)$, as required. We will re-visit NBC in Section 7.4.

3.2.4 Logistic Regression (Reprise)

We can also recast the 2−class LDA model as

$$\begin{aligned}\ln\left(\frac{p_0(\mathbf{x})}{1-p_0(\mathbf{x})}\right) &= \ln(p_0(\mathbf{x})) - \ln(p_1(\mathbf{x})) \\ &= \delta_0(\mathbf{x}) - \delta_1(\mathbf{x}) = a_0 + \mathbf{a}^\top\mathbf{x},\end{aligned}$$

which has the same form as logistic regression. It is not the same model, however:

- in **logistic regression**, the parameters are estimated using the maximum likelihood on $P(Y \mid \mathbf{X})$;
- in Section 7.4, the parameters are estimated using the full likelihood $P(\mathbf{x} \mid Y)P(\mathbf{x}) = P(\mathbf{x}, Y)$.

3.3 Receiver Operating Characteristic Curve

We finish this section by giving an example of LDA and QDA on the Gapminder example of Section 3.1 (with a training set Tr consisting of $N = 116$ observations and a testing set Te consisting of $M = 50$ observations), and we will hold a preliminary discussion on model evaluation.

Given an observation $\mathbf{x} \in$ Te, we use a decision rule based on the probabilities $\hat{p}_0(\mathbf{x}), \hat{p}_1(\mathbf{x})$ and a decision threshold $\alpha \in (0, 1)$.

[28] p parameters for each $\hat{\boldsymbol{\mu}}_k$ and $1 + 2 + \cdots + p$ parameters for $\hat{\Sigma}$.
[29] p parameters for each $\hat{\boldsymbol{\mu}}_k$ and $1 + 2 + \cdots + p$ parameters for each $\hat{\Sigma}_k$.

On the training set Tr with 116 observations, we have

$$\begin{aligned}&N_0 = 51,\ N_1 = 65,\ \hat{\pi}_0 = 51/116,\ \hat{\pi}_1 = 65/116,\\ &\hat{\boldsymbol{\mu}}_0 = (45.40, 4.08)^\top,\ \hat{\boldsymbol{\mu}}_1 = (9.57, 1.92)^\top\\ &\Sigma_0 = \begin{pmatrix}496.51 & 23.38\\ 23.38 & 2.17\end{pmatrix},\ \Sigma_1 = \begin{pmatrix}42.79 & 2.14\\ 2.14 & 0.31\end{pmatrix}\\ &\hat{\boldsymbol{\mu}} = (25.30, 2.87)^\top,\ \Sigma = \begin{pmatrix}557.89 & 30.51\\ 30.51 & 2.27\end{pmatrix}\end{aligned}$$

Thus,

$$\begin{aligned}\hat{\delta}_{0;\text{LDA}} &= -0.06x_1 + 2.65x_2 - 4.78\\ \hat{\delta}_{1;\text{LDA}} &= -0.11x_1 + 2.31x_2 - 2.28\\ \hat{\delta}_{0;\text{QDA}} &= -4.66 - 0.002x_1^2 + 0.01x_1 + 0.04x_1x_2 + 1.81x_2 - 0.47x_2^2\\ \hat{\delta}_{1;\text{QDA}} &= -6.70 - 0.02x_1^2 - 0.13x_1 + 0.24x_1x_2 + 7.01x_2 - 2.43x_2^2\end{aligned}$$

With the class probability estimates

$$\begin{aligned}\hat{p}_{1;\text{LDA}} &= \frac{\exp(\hat{\delta}_{1;\text{LDA}})}{\exp(\hat{\delta}_{0;\text{LDA}}) + \exp(\hat{\delta}_{1;\text{LDA}})},\\ \hat{p}_{1,\text{QDA}} &== \frac{\exp(\hat{\delta}_{1;\text{QDA}})}{\exp(\hat{\delta}_{0;\text{QDA}}) + \exp(\hat{\delta}_{1;\text{QDA}})}\end{aligned}$$

and the decision threshold $\alpha \in (0, 1)$, the LDA and QDA life expectancy classifiers are defined on Te by

$$\hat{C}_{\alpha;\text{LDA}}(\mathbf{x}) = \begin{cases}1 \text{ (high)} & \text{if } p_{1;\text{LDA}}(\mathbf{x}) \geq \alpha\\ 0 \text{ (low)} & \text{else}\end{cases}$$

and

$$\hat{C}_{\alpha;\text{QDA}}(\mathbf{x}) = \begin{cases}1 \text{ (high)} & \text{if } p_{1;\text{QDA}}(\mathbf{x}) \geq \alpha\\ 0 \text{ (low)} & \text{else}\end{cases}$$

The **confusion matrix** of a classifier on Te is a tool to evaluate the model's performance:

α		**prediction** 0	1
actual	0	TP	FN
	1	FP	TN

Here, TP stands for **true positive**, FN for **true negative**, FP for **false positive**, and TN for **true negative**. There are various **classifier evaluation metrics** (remember that the testing set Te has M observations):

- **accuracy** measures the correct classification rate $\frac{\text{TP+TN}}{M}$;
- **misclassifcation** is $\frac{\text{FP+FN}}{M} = 1 -$ accuracy;
- **false positive rate** (FPR) is $\frac{\text{FP}}{\text{FP+TN}}$;
- **false negative rate** (FNR) is $\frac{\text{FN}}{\text{TP+FN}}$;
- **true positive rate** (TPR) is $\frac{\text{TP}}{\text{TP+FN}}$;
- **true negative rate** (TNR) is $\frac{\text{TN}}{\text{FP+TN}}$;

There are other measures, including the F_1−score, the Matthews' correlation coefficient, etc. [31].

In the Gapminder example, the $\alpha = 0.5$ confusion matrices for the LDA and QDA classifiers are:

LDA		prediction 0	prediction 1
actual	0	22	10
	1	0	18

QDA		prediction 0	prediction 1
actual	0	28	4
	1	2	16

In the LDA case:

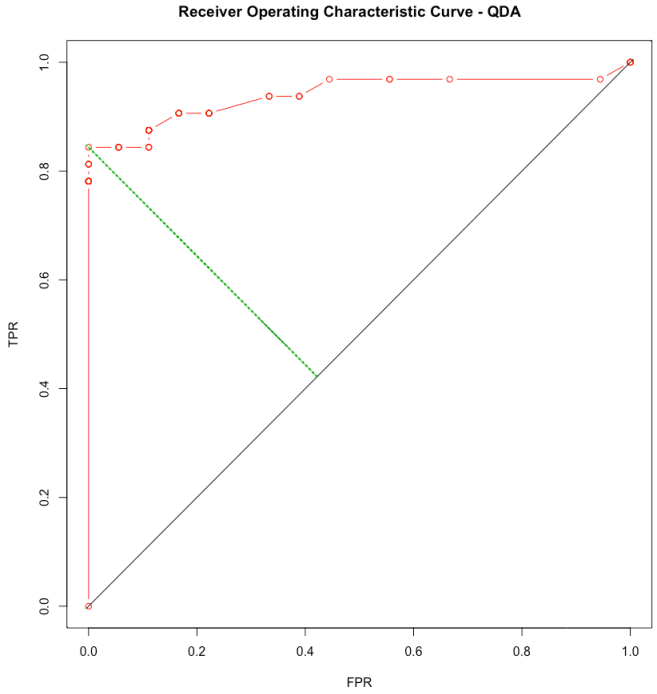
- accuracy $= \frac{22+18}{22+10+0+18} = 80\%$
- misclassification rate $= \frac{10+0}{22+10+0+18} = 20\%$
- FPR $= \frac{0}{0+18} = 0\%$
- FNR $= \frac{10}{22+10} = 31.25\%$
- TPR $= \frac{22}{22+10} = 68.75\%$
- TNR $= \frac{18}{0+18} = 100\%$

In the QDA case:

- accuracy $= \frac{28+16}{28+4+2+16} = 88\%$
- misclassification rate $= \frac{4+2}{28+4+2+16} = 12\%$
- FPR $= \frac{2}{2+16} = 11.1\%$
- FNR $= \frac{4}{28+4} = 12.5\%$
- TPR $= \frac{28}{28+4} = 87.5\%$
- TNR $= \frac{16}{2+16} = 88.9\%$

At first glance, it would certainly seem that the QDA model performs better (at a decision threshold of $\alpha = 0.5$), but the FPR is not ideal. What would be the ideal value of α?

The **receiver operating characteristic** (ROC) curve plots the true positive rate against the false positive rate (in red) for each of the classifiers obtained by varying the decision threshold α from $(0, 1)$ (see below).



A number of models have the same (FPR, TPR) coordinates. The important realization is that classifier that is completely random would lie on the line TPR = FPR (in black). The ideal threshold would then be the one associated with the model which is **farthest** from that line (in green).

Let $\mathbf{u}(\alpha)$ be the vector from $\mathbf{0}$ to the $(\text{FPR}(\alpha), \text{TPR}(\alpha))$ coordinates of the classifier with threshold α, and let $\mathbf{v}(\alpha)$ be the vector through $(\text{FPR}(\alpha), \text{TPR}(\alpha))$ and perpendicular to the line TPR = FPR. The ideal α^* satisfies:

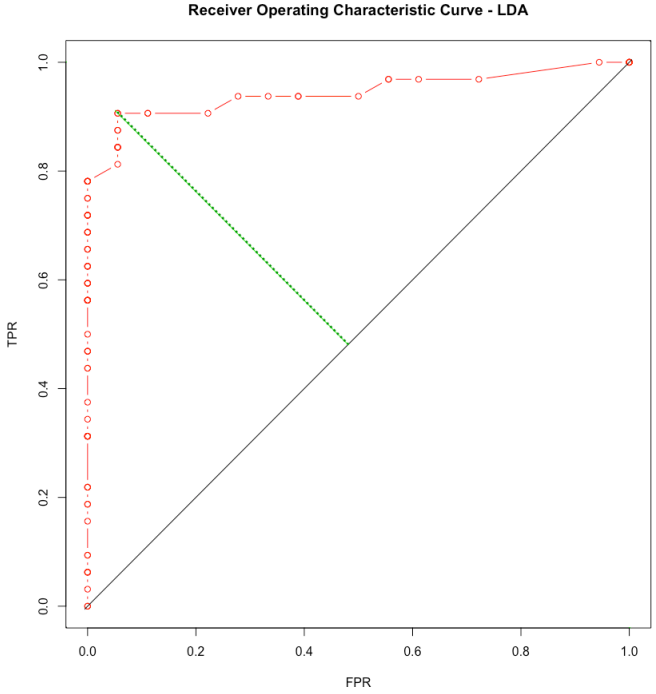
$$\begin{aligned}
\alpha^* &= \underset{\alpha}{\text{argmax}}\{\|\mathbf{v}(\alpha)\|\} = \underset{\alpha}{\text{argmax}}\{\|\mathbf{v}(\alpha)\|^2\} \\
&= \underset{\alpha}{\text{argmax}}\left\{\left\|\mathbf{u}(\alpha) - \text{proj}_{(1,1)}\mathbf{u}(\alpha)\right\|^2\right\} \\
&= \underset{\alpha}{\text{argmax}}\left\{\left\|(\text{FPR}(\alpha), \text{TPR}(\alpha)) - \text{proj}_{(1,1)}(\text{FPR}(\alpha), \text{TPR}(\alpha))\right\|^2\right\} \\
&= \underset{\alpha}{\text{argmax}}\left\{\|(\text{FPR}(\alpha) - \text{TPR}(\alpha), \text{TPR}(\alpha) - \text{FPR}(\alpha))\|^2\right\} \\
&= \underset{\alpha}{\text{argmax}}\{(\text{FPR}(\alpha) - \text{TPR}(\alpha))^2\}.
\end{aligned}$$

With the QDA model, the threshold is $\alpha^*_{\text{QDA}} = 0.28$ (coordinates $(0, 0.844)$); with the LDA model, the threshold is $\alpha^*_{\text{LDA}} = 0.73$ (coordinates $(0.056, 0.906)$). The corresponding confusion matrices are shown below.

α^*_{QDA}		prediction 0	prediction 1
actual	0	27	5
	1	0	18

α^*_{LDA}		prediction 0	prediction 1
actual	0	29	3
	1	1	17

Which model is best? It depends on the context of the task, and on the consequences of the choice. What makes the most sense here? Is there a danger of overfitting? Is parameter tuning acceptable, from a data massaging perspective? What effect does the choice of priors have?



4. 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 is used to estimate the test error associated with a modeling approach in order to evaluate model performance;
- the **bootstrap**, which is used to provide 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.

For quantitative responses, 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 (i.e. the **variance-bias trade-off**, see Figure 2).

A possible solution to 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.[30]

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 leadd to the validation error being larger than the test error in general;
- a number of classical statistical models can provide test error estimates without having to resort to the validation set approach.

4.1 Cross-Validation

K-Fold Cross Validation is a widely-used approach to estimate the test error without losing some observations to a hold-out test.[31]

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
$$\{1, \dots, N\} = \underbrace{\mathscr{C}_1}_{\text{fold } 1} \sqcup \cdots \sqcup \underbrace{\mathscr{C}_K}_{\text{fold } K}.$$
If $|\mathscr{C}_k| = n_k$, we expect $n_k \approx \frac{N}{K}$ for all $k = 1, \dots, K$.
3. For all $k = 1, \dots, K$, fit a model on $\{1, \dots, N\} \setminus \mathscr{C}_k$ and denote the error on $\mathscr{C}_k$ by E_k.[32]
4. Write $\overline{E}$ for the average of the E_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), see Figure 8 for an illustration.

The resulting mean can prove useful in order to determine how well a statistical learning procedure will perform on unseen data.

[30] 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^2_a, AIC, BIC, etc.)

[31] It can also provide a basis for model selection.

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

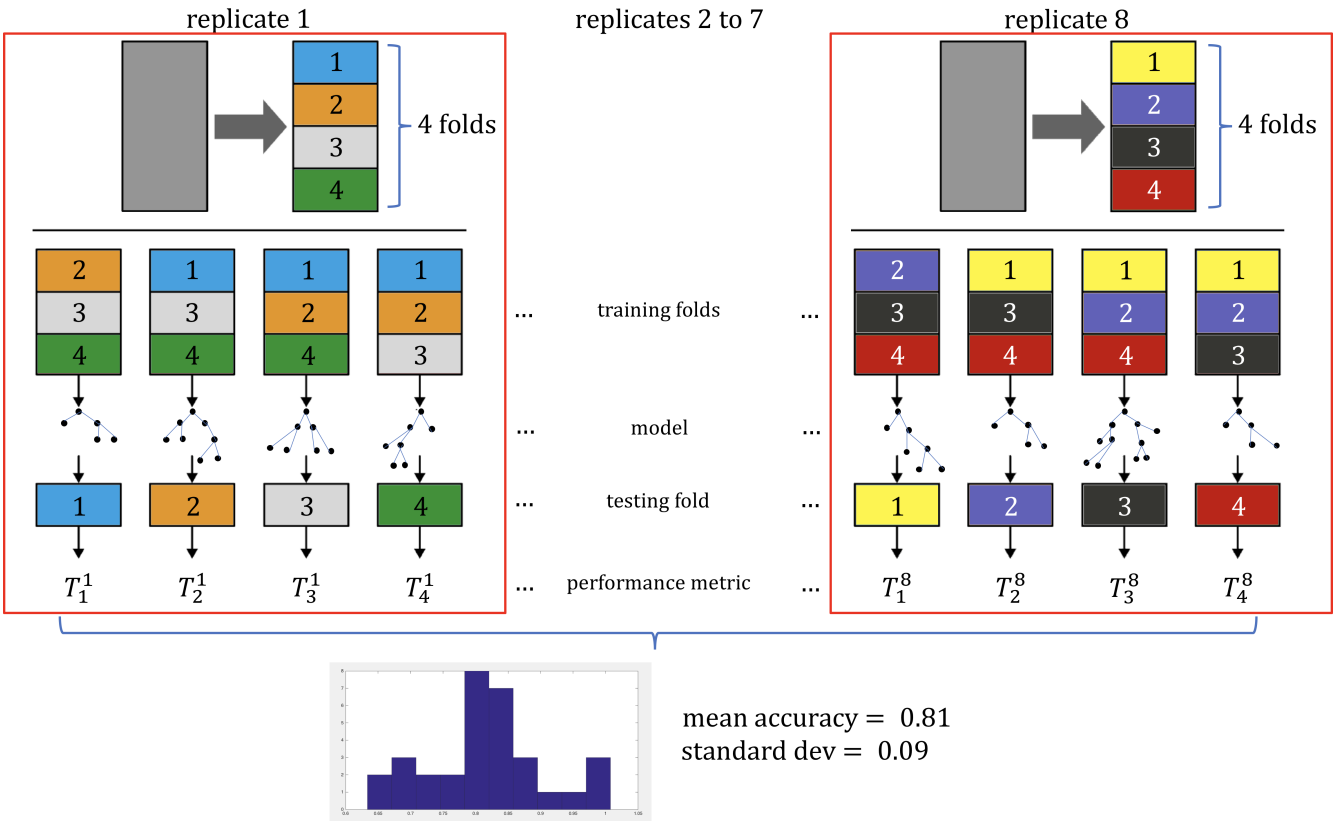


Figure 8. Schematic illustration of cross-fold validation, for 8 replicates and 4 folds; $8 \times 4 = 32$ models from a given family are built on various training sets (consisting of 3/4 of the available data – the training folds). Model family performance is evaluated on the respective holdout folds; the distribution of the performance metric values (in practice, some combination of the mean/median and standard deviation) can be used to compare various model families.

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 $\text{CV}_{(K)}$ so much as where it is minimized.[33]

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** (see Section 4.3 for details).

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

We split the dataset into 10 random folds, each containing 16 or 17 observations, and fit 10 linear regression models using the 149 or 150 remaining observations.[34]

[33] The estimate is usually biased, anyway.

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

k	n_k	$\beta_{0;k}$	$\beta_{1;k}$	MSE_{Te_k}
1	17	37.69	4.25	33.85
2	17	36.22	4.44	21.41
3	17	37.59	4.24	45.62
4	17	36.66	4.34	29.58
5	17	37.49	4.26	24.12
6	17	36.49	4.38	19.39
7	16	36.78	4.38	48.15
8	16	36.91	4.33	23.14
9	16	37.41	4.27	7.83
10	16	37.68	4.25	19.74

The $K-$fold cross-validation estimate of MSE_{Te} is thus

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

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

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

and $27.35 \pm 2(12.38) \equiv (2.59, 52.11)$ is a 95% C.I. for the test error.

We can also get $K-$fold cross-validation estimates of the true β_0 and β_1:

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

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

and $37.10 \pm 2(0.54) \equiv (36.00, 38.18)$ is a 95% C.I. for β_0;

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

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

and $4.32 \pm 2(0.07) \equiv (4.18, 4.56)$ is a 95% C.I. for β_1.

4.2 The 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.[35] We can even construct a covariance structure for the parameters, given enough replicates.

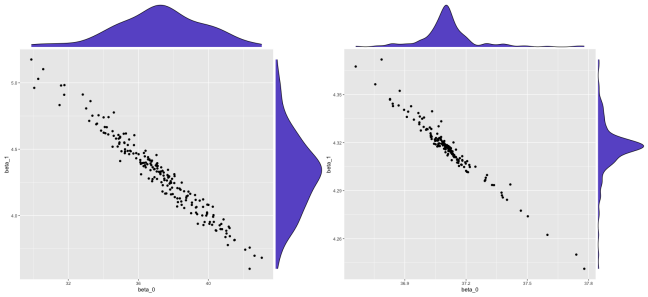
Finally, it should be noted that in more complex scenarios, the appropriate bootstrap procedure might be more sophisticated than what has been described here.[36]

[35] But there are complications, so caution is advised.

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

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 use $M = 200$ bootstrap samples of size $N = 166$, each drawn from the original dataset, with replacement.

The parameter estimates $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \hat{\beta}_1)^\top$ for each bootstrap sample are shown below (on the left).



It would appear that $\boldsymbol{\beta}$ approximately follows a multivariate normal $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with

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

$\hat{\boldsymbol{\mu}}^*$ provides the bootstrap estimates, the corresponding estimates for the standard errors are $\hat{\text{se}}(\hat{\boldsymbol{\mu}}^*) = (2.51, 0.29)^\top$, and the 95% C.I. are:

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

Notice that the bootstrap estimates are wider than the corresponding cross-validation estimates.

4.3 The Jackknife

The **jackknife estimator** arises from cross-validation when $K = N$; the sole difference is in the variance estimate

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

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

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

The parameter estimates $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \hat{\beta}_1)^\top$ for each jackknife sample are shown above (on the right). The jackknife estimates are $\hat{\boldsymbol{\mu}}^* = (37.11, 4.32)^\top$, the corresponding estimates for the standard errors are $\hat{\text{se}}(\hat{\boldsymbol{\mu}}^*) = (0.011, 0.001)^\top$, and the 95% C.I. are:

$$\text{C.I.}(\beta_0; 0.95) = 37.11 \pm 2(0.011) \equiv (37.088, 37.133)$$
$$\text{C.I.}(\beta_1; 0.95) = 4.32 \pm 2(0.001) \equiv (4.314, 4.319).$$

Notice that the jackknife estimates are much tighter than the corresponding bootstrap estimates.

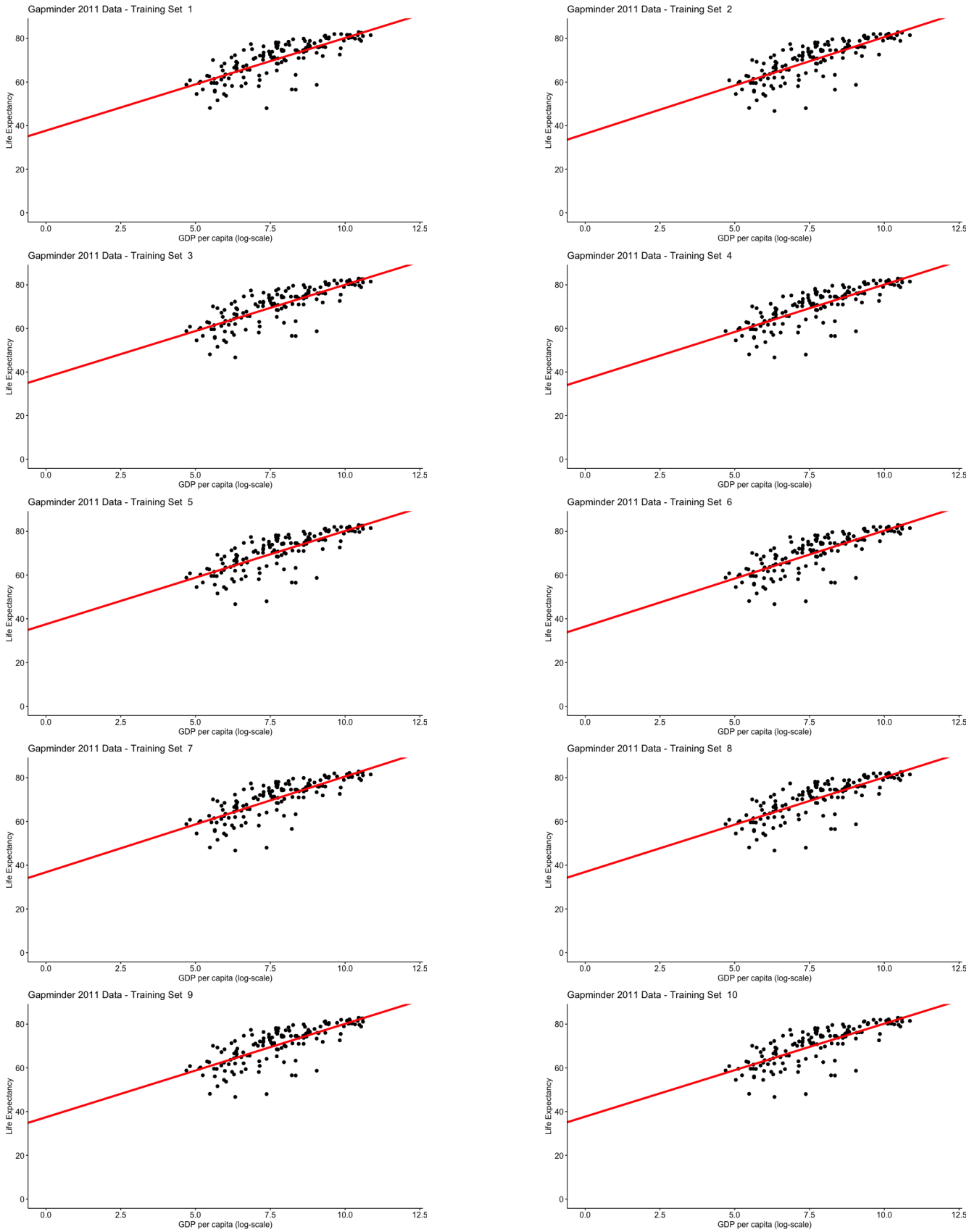


Figure 9. Training set and linear fit for each of the 10 folds in the Gapminder 2011 data. Note the similarity of the fits

5. Model Selection

A linear model

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

should be seen as an attempt to approximate the regression function

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

What we gain in convenience of fit (and structure), we lose in modeling accuracy.

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

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\{\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2\},$$

for the **response vector Y** and **design matrix X** provided by a **training set** Tr.

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

1. additive but non-linear models (see Section 6.3);
2. non-linear models (see Sections 6 and 7);
3. replacing LS with alternative fitting procedures (which we will discuss momentarily).

The latter approach can produce **better accuracy** than OLS without sacrificing too much in the way of **model interpretability**.[37]

But in the OLS framework, prediction accuracy suffers when $p > n$, due to **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:

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

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.

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.

In **dimension reduction**, we project the p predictors onto an $M-$dimensional manifold $\mathscr{H}$, with $M \ll p$; in numerous circumstances, $\mathscr{H}$ is a subspace of $\mathbf{R}^p$ and we can fit an OLS model on the projected coordinates.

[37] 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 model fails analysts in the future.

5.1 Curse of Dimensionality

A model is said to be **local** if it depends solely on the observations near the input vector (k nearest neigbours classification is local, whereas linear regression is global). With a large training set, increasing k in a kNN model, say, will yield enough data points to provide a solid approximation to the theoretical classification boundary.

The **curse of dimensionality** (CoD) is the breakdown of this approach in high-dimensional spaces: when the number of features increases, the number of observations required to maintain predictive power also increases, **but at a substantially higher rate** (see Figure 10 for an illustration of the CoD).

Manifestations of CoD

Let $x_i \sim U^1(0,1)$ be i.i.d. for $i = 1, \ldots, N$. For any $z \in [0,1]$ and $\varepsilon \in (0,1]$ such that

$$I_1(z;\varepsilon) = \{y \in \mathbb{R} : |z - y|_\infty < \varepsilon\} \subseteq [0,1],$$

the expected number of observations x_i in $I_1(z;\varepsilon)$ is

$$\left|I_1(z;\varepsilon) \cap \{x_i\}_{i=1}^N\right| \approx \varepsilon \cdot N,$$

so an $\varepsilon_\infty-$ball subset of $[0,1]^1$ contains approximately ε of the observations in $\{x_i\}_{i=1}^N \subseteq \mathbb{R}$, on average.

Let $\mathbf{x}_i \sim U^2(0,1)$ be i.i.d. for all i. For any $\mathbf{z} \in [0,1]^2$ and $\varepsilon \in (0,1]$ such that

$$I_2(\mathbf{z};\varepsilon) = \{\mathbf{Y} \in \mathbb{R}^2 : \|\mathbf{z} - \mathbf{Y}\|_\infty < \varepsilon\} \subseteq [0,1]^2,$$

the expected number of observations $\mathbf{x}_i$ in $I_2(\mathbf{z};\varepsilon)$ is

$$\left|I_1(\mathbf{z};\varepsilon) \cap \{\mathbf{x}_i\}_{i=1}^N\right| \approx \varepsilon^2 \cdot N,$$

so an $\varepsilon_\infty-$ball subset of $[0,1]^2$ contains approximately ε^2 of the observations in $\{\mathbf{x}_i\}_{i=1}^N \subseteq \mathbb{R}^2$, on average.

In general, the same reasoning shows that an $\varepsilon_\infty-$ball subset of $[0,1]^p \subseteq \mathbb{R}^p$ contains approximately ε^p of the observations in $\{\mathbf{x}_i\}_{i=1}^N \subseteq \mathbb{R}^p$, on average.

Thus, to capture r percent of uniformly distributed observations in a unit $p-$hypercube, a $p-$hypercube with edge

$$\varepsilon_p(r) = r^{1/p}$$

is needed, on average. For instance, to capture $r = 1/3$ of the observations in a unit $p-$hypercube in $\mathbb{R}$, $\mathbb{R}^2$, and $\mathbb{R}^{10}$, a hyper-subset with edge $\varepsilon_1(1/3) \approx 0.33$, $\varepsilon_2(1/3) \approx 0.58$, and $\varepsilon_{10}(1/3) \approx 0.90$, respectively.

The inference is simple: in general, as p increases, the nearest observations to a given point $\mathbf{x}_j \in \mathbb{R}^p$ are in fact quite distant from $\mathbf{x}_j$, in the Euclidean sense, on average – **locality is lost!**[38] This can wreak havoc on models and algorithms that rely on the (Euclidean) nearness of observations (k nearest neighbours, $k-$means clustering, etc.).

[38] The situation can be different when the observations are not i.i.d.

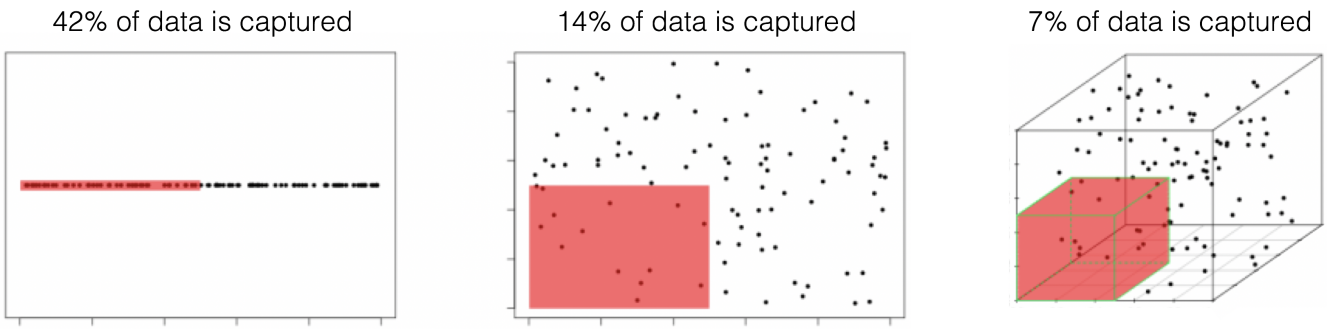


Figure 10. Illustration of the curse of dimensionality; $N = 100$ observations are uniformly distributed on the unit hypercube $[0,1]^d$, $d = 1,2,3$. The red regions represent the smaller hypercubes $[0,0.5]^d$, $d = 1,2,3$. The percentage of captured datapoints is seen to decrease with an increase in d [14].

The CoD manifests itself in various ways. In datasets with a large number of features:

- most observations are **nearer the edge of the sample than they are to other observations**, and
- realistic training sets are necessarily **sparse**.

Imposing restrictions on models can help mitigate the effects of the CoD, but if the assumptions are not warranted the end result may be catastrophic.

5.2 Subset Selection

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**?

5.2.1 Best Subset Selection (BSS)

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

1. let $\mathscr{M}_0$ 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 SSRes (largest R^2) and denote it by $\mathscr{M}_k$;
3. select a unique model from $\{\mathscr{M}_0,\dots,\mathscr{M}_p\}$ using $\text{CV}_{(K)}$, C_p (AIC), BIC, R^2_a, or any other appropriate metric.[39]

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.[40]

5.2.2 Stepwise Selection (SS)

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

1. Let $\mathscr{M}_0$ 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 $\mathscr{M}_k$;
   (b) pick the model with smallest SSRes (largest R^2) and denote it by $\mathscr{M}_{k+1}$;
3. select a unique model from $\{\mathscr{M}_0,\dots,\mathscr{M}_p\}$ using $\text{CV}_{(K)}$, C_p (AIC), BIC, R^2_a, or any other appropriate metric.

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

1. Let $\mathscr{M}_p$ 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 $\mathscr{M}_k$;
   (b) pick the model with smallest SSRes (largest R^2) and denote it by $\mathscr{M}_{k-1}$;
3. select a unique model from $\{\mathscr{M}_0,\dots,\mathscr{M}_p\}$ using $\text{CV}_{(K)}$, C_p (AIC), BIC, R^2_a, or any other appropriate metric.

[39] We cannot use SSRes or R^2 as metrics in this last step, as we would always select $\mathscr{M}_p$ since SSRes decreases monotonically with k and R^2 increases monotonically with k. Low SSRes/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!

[40] 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 2(b) and test error estimates for step 3.

The computational advantage of SS over BSS 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) will be found in the $\frac{p^2+p+2}{2}$ 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.

5.2.3 Hybrid Selection (HS)

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 [15].

5.2.4 Selecting the Optimal Model

The full model always has largest R^2/smallest SSRes (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 2).

In order to estimate the test error (and thus pick the optimal model in the list $\{\mathcal{M}_0,\ldots,\mathcal{M}_p\}$, we can either:

- **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 coefficience of determination R^2_a; C_p, AIC, and BIC must be **minimized**, while R^2_a 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 $\mathrm{Var}(\varepsilon)$ (irreducible error);
- SSRes and SSTot, the residual and the total sum of squares.

Mallow's C_p statistic is given by

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

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

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

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

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

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

or, upon taking the logarithm,

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

and so

$$\underset{\boldsymbol{\beta}}{\mathrm{argmax}}\{\ln L(\boldsymbol{\beta})\}=\underset{\boldsymbol{\beta}}{\mathrm{argmin}}\{\|\mathbf{Y}-\mathbf{X}\boldsymbol{\beta}\|^2\}.$$

However,

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

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

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

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

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^2_a is the Ur-example of an adjusted statistic:

$$R^2_a=1-\frac{\mathrm{SSRes}/(n-p-1)}{\mathrm{SSTot}/(n-1)}=1-(1-R^2)\frac{n-1}{n-p-1}.$$

Maximizing R^2_a is equivalent to minimizing $\frac{\mathrm{SSRes}}{n-p-1}$. Note that R^2_a penalizes models with **unnecessary** variables.

Validation and Cross-Validation (Reprise) As before, 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 select k^* for which the value is **smallest** (see Section 4.1).

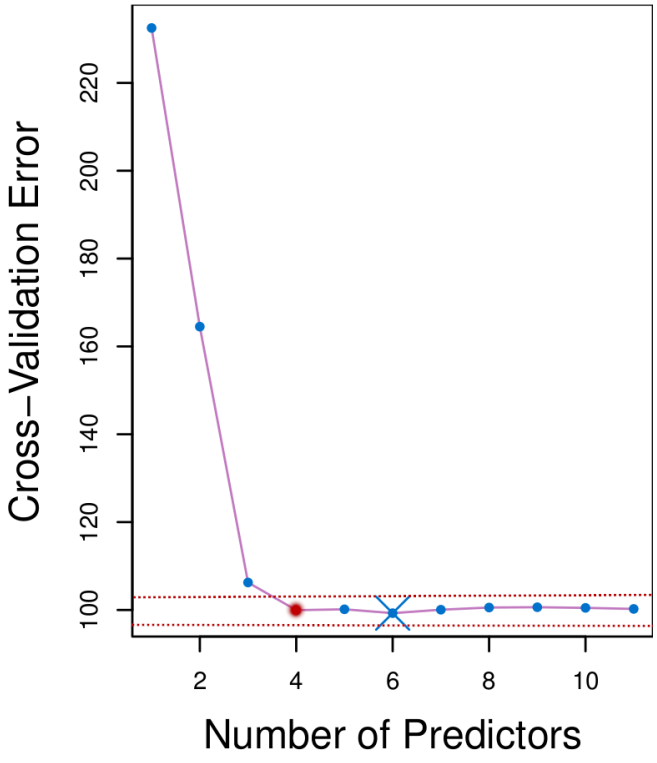
The main advantages of this approach is that there is no need to estimate the irreducible error $\text{Var}(\varepsilon) = \sigma^2$, that the method produces an estimate for MSE_{Te} "for free," and that it can be used when the number of parameters is hard to pinpoint (in deep learning networks, for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

Historically, adjustment approaches were 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 $\hat{\text{MSE}}_{\text{Te}}$ for each model size, and select the **smallest model** for which $\hat{\text{MSE}}_{\text{Te}}$ is within one standard from the lowest point on the cross-validation error curve.

This is equivalent to **Occam's Razor**[41] on models that have equivalent predictive power, roughly speaking.

In the image below (modified from [15]), 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).



[41] "When presented with competing hypotheses about the same prediction, one should select the solution with the fewest assumptions."

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

- all intermediate tests are **biased**, as they are all based on the same data;
- R^2_a 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 or each sub-model.

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

5.3 Feature Selection Methods

Removing **irrelevant** or **redundant** variables is a common data processing task. It is used both to help with the bias-variance trade-off and to mitigate the effects of the curse of dimensionality.

5.3.1 Overview

Feature selection approaches follow 2 axes:

- filter vs. wrapper
- unsupervised vs. supervised

Filter methods inspect each variable individually and core them according to some **importance metric**. The features that are less relevant (i.e., those that score below some fixed threshold or rank) are then removed from the dataset.

Wrapper methods, on the other hand, select feature subsets for which the evaluation criterion used by the eventual analytical method is **optimized**. The process is **iterative** and typically **computationally intensive**: candidate subsets are used in the analysis until one produces an acceptable evaluation metric related to the analysis.[42]

Unsupervised methods determine the importance of a feature solely on the basis of the **values it takes**; **supervised methods** evaluate each feature's importance by studying its relationship with the **target feature** (*via* correlation, etc.).

Wrapper methods (such as BSS, SS) are usually (but not always) supervised methods. Unsupervised filter methods include removing constant variables, variables with "too many" missing values, ID-like variables, features with low variability, etc.[43]

Supervised filter methods have been well-studied and are commonly used. As an example, we can compute the **correlation**

$$\rho_{XY} = \frac{\sum_{i=1}^{N}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \overline{x})^2 \sum_{i=1}^{N}(y_i - \overline{y})^2}}$$

[42] Wrapper methods are so-called as they "wrap" feature selection around the statistical learning algorithm.

[43] Such methods search for irrelevant or noisy features.

between each predictor X and the response variable Y, and retain only those features strongly correlated with Y.[44]

Another such method depend on the **mutual information** shared between a categorical target Y and a categorical predictor X:

$$I(Y;X) = H(Y) - H(Y \mid X),$$

where the **entropy** $H(Y)$ and the **conditional class entropy** $H(Y \mid X)$ are given by

$$H(Y) = -\sum_{C \in \mathscr{C}_Y} P(Y=C)\ln P(Y=C)$$

$$H(Y \mid X) = -\sum_{\substack{C \in \mathscr{C}_Y \\ V \in \mathscr{C}_X}} P(X=V, Y=C)\ln\left[\frac{P(X=V,Y=C)}{P(X=V)}\right],$$

with $\mathscr{C}_X, \mathscr{C}_Y$ being the levels of X, Y, respectively, and $0 \times \ln 0 = 0$, by convention. In a nutshell, $I(Y;X)$ measures the amount of information that can be obtained about Y by knowing X. Features which provide "high" mutual information about the target Y are retained.

For **classification tasks**, other common metrics include: Gain Ratio, Inf Gain, Gini, MDL, etc. For **regression tasks**, the usual metrics are MSE of Mean, MAE of Mean, Relief, etc. More details are available in [18].

5.3.2 Shrinkage Methods

SS methods use OLS to fit a linear model with subsets of predictors. Another approach 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 $\mathbf{x}_i = (x_{i,1}, \cdots, x_{i,p})$, together with target observations 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

$$\hat{\boldsymbol{\beta}}_{\text{OLS}} = \underset{\boldsymbol{\beta}}{\text{argmin}}\{\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2\} = \underset{\boldsymbol{\beta}}{\text{argmin}}\{\text{SSRes}\}.$$

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.

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:

$$\hat{\boldsymbol{\beta}}_{\text{RR}} = \underset{\boldsymbol{\beta}}{\text{argmin}}\{\underbrace{\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2}_{\text{SSRes}} + \underbrace{N\lambda\|\boldsymbol{\beta}\|_2^2}_{\substack{\text{shrinkage} \\ \text{penalty}}}\}.$$

This quantity is small when SSRes 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.[45]

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 11 for an illustration).

Setting the "right" value for λ is crucial; it can be done via cross-validation (see [15, pp.227-228] for details).

The OLS estimate are **equivariant**: if $\hat{\beta}_j$ is the estimate for the coefficient β_j of X_j, then $\frac{\hat{\beta}_j}{c}$ is the estimate for the coefficient of the scaled variable cX_j. RR coefficients do not have this property, 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 (even if they do not reduce the dimensions of the dataset).

Regression with best subset selection 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). The problem consists in solving another modified version of the OLS scenario, namely

$$\hat{\boldsymbol{\beta}}_{\text{BS}} = \underset{\boldsymbol{\beta}}{\text{argmin}}\{\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + N\lambda\|\boldsymbol{\beta}\|_0\},\ \|\boldsymbol{\beta}\|_0 = \sum_j \text{sgn}(|\beta_j|).$$

Solving the BS problem typically (also) requires numerical methods and cross-validation.[46]

A slight modification to the RR shrinkage penalty can overcome the lack of covariance. The **LASSO** is an alternative to RR obtained by solving

$$\hat{\boldsymbol{\beta}}_{\text{L}} = \underset{\boldsymbol{\beta}}{\text{argmin}}\{\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + N\lambda\|\boldsymbol{\beta}\|_1\};$$

the penalty effectively forces coefficients which combine the properties of RR and BS, selecting at most $\max\{p, N\}$

[44] Note that is approach is of limited use if the relationship between X and Y is **non-linear**.

[45] For **orthonormal covariates** $\mathbf{X}^\top\mathbf{X} = I_p$, we have $\hat{\beta}_{\text{RR},j} = \frac{\hat{\beta}_{\text{OLS},j}}{1+N\lambda}$.

[46] For orthonormal covariates, we have

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

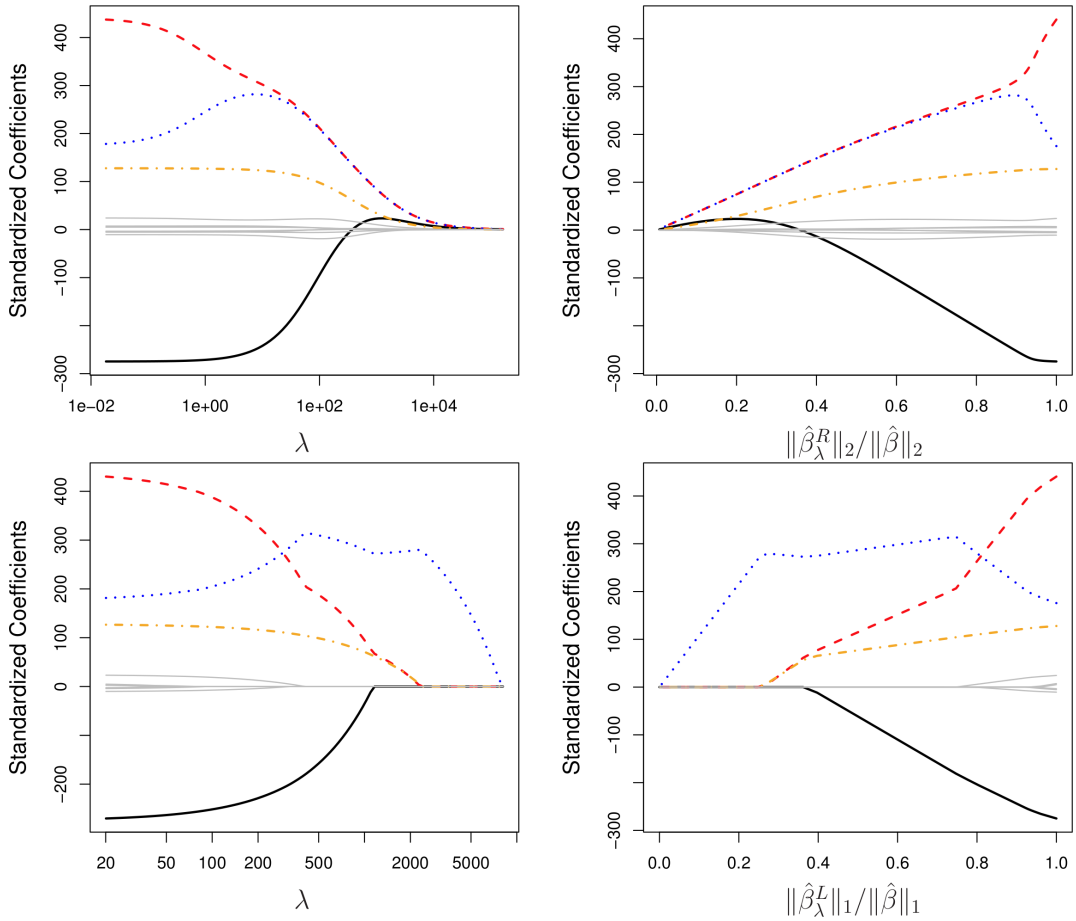


Figure 11. Ridge regression coefficients in a generic problem (top); note how the parameters converge to 0 when the threshold λ increases (left); the ratio between the magnitude of the ridge regression parameter and the OLS parameter is shown on the right. LASSO coefficients on the same data; note how the coefficients go directly to 0 after a certain threshold λ (modified from [15]).

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 11 for an illustration).[47]

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

$$\hat{\boldsymbol{\beta}}_{\text{RR}} = \underset{\boldsymbol{\beta}}{\text{argmin}}\{\text{SSRes} \mid \|\boldsymbol{\beta}\|_2^2 \leq s\} \text{ for some } s$$

$$\hat{\boldsymbol{\beta}}_{\text{L}} = \underset{\boldsymbol{\beta}}{\text{argmin}}\{\text{SSRes} \mid \|\boldsymbol{\beta}\|_1 \leq s\} \text{ for some } s$$

Graphically, this looks like the images shown in Figure 12. The RR coefficients $\hat{\boldsymbol{\beta}}_{\text{RR}}$ are found at the first intersection of the ellipses of constant SSRes 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 (due to the lack of "sharp" points); this is not usually the case for the intersection of the 1−sphere $\|\boldsymbol{\beta}\|_2 \leq s$.

The LASSO thus typically produces simpler models, but predictive accuracy matters too (in the form of MSE_{Te}, say). Depending on the data, either of the two approaches can be optimal, thanks to the No Free Lunch Theorem.

If the response is related to a relatively small number of predictors (which we do not usually know a priori), LASSO is recommended.

The use of other penalty functions (or combinations thereof) provides various extensions: 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 classification/regression models as well. For a **loss** (cost) function $\mathscr{L}(\mathbf{Y}, \hat{\mathbf{y}}(\mathbf{W}))$ between the actual tar-

[47] For orthonormal covariates, we have

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

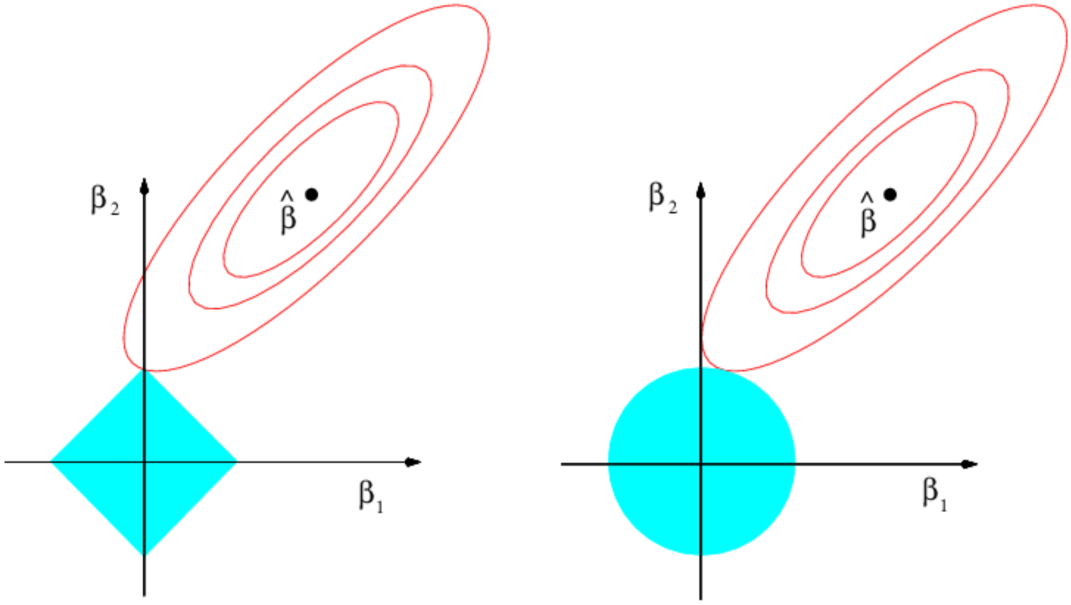


Figure 12. Level curves and neighbourhoods for ridge regression (right) and LASSO (left) [15].

get 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

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

which can be solved numerically, assuming some nice properties on $\mathscr{L}$ and $\mathbf{R}$ [11]; as before, cross-validation can be used to determine the optimal vector $\boldsymbol{\lambda}$ [10].

Example In R, regularization is implemented in the package `glmnet` (among others). An elastic net can be used to select features that are related with the life expectancy Y in the `Gapminder` 2011 observations.

The LASSO method ($\alpha = 1$) leads to the model

$$\begin{aligned} Y = 70.8 &- 5.7(\text{infant mortality}) + 0.2(\text{gdp}) \\ &- 1.8(\text{Africa}) + 0.2(\text{Europe}) - 0.9(\text{Oceania}), \end{aligned}$$

while RR ($\alpha = 0$) leads to the model

$$\begin{aligned} Y = 70.8 &- 0.4(\text{population}) - 4.7(\text{infant mortality}) \\ &- 0.4(\text{fertility}) + 0.6(\text{gdp}) - 1.6(\text{Africa}) \\ &+ 0.5(\text{Americas}) + 0.6(\text{Asia}) + 1.0(\text{Europe}) \\ &- 0.7(\text{Oceania}), \end{aligned}$$

which agrees with the discussion above.

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

5.4 Dimension Reduction Methods

There are many advantages to working with reduced, low-dimensional data:

- **visualisation methods** of all kinds are available and (more) readily applicable to such data in order to extract and present insights;
- high-dimensional datasets are subject to the so-called **curse of dimensionality**(CoD), which asserts that when the number of features in a model increases, the number of observations required to maintain predictive power also increases, but at a **substantially larger rate** (see Figure 10);
- another consequence of the curse is that in high-dimension sets, all observations are roughly **dissimilar** to one another – observations tend to be nearer the dataset's boundaries than they are to one another.

Dimension reduction techniques such as the ubiquitous **principal component analysis**, **independent component analysis**, **factor analysis** (for numerical data), or **multiple correspondence analysis** (for categorical data) project multi-dimensional datasets onto low-dimensional but high-information spaces (see **Manifold Hypothesis** [18]) prior to fitting whatever model is deemed appropriate.

Some information is necessarily lost in the process, but in many instances the drain can be kept under control and the gains made by working with smaller datasets can offset the losses of completeness.

For $m = 1, \ldots, M \leq p$, let $z_m = \vec{X}_j^\top \boldsymbol{\phi}$ be linear combinations of the original predictors $\{X_1, \ldots, X_p\}$.

If we are fitting $y = f(\vec{x}) = \mathrm{E}[Y \mid \vec{X} = \vec{x}]$ using OLS, we can also fit $y_i = \theta_0 + \mathbf{z}_i^\top \boldsymbol{\theta} + \varepsilon_i$, $i = 1, \ldots, N$ using OLS.

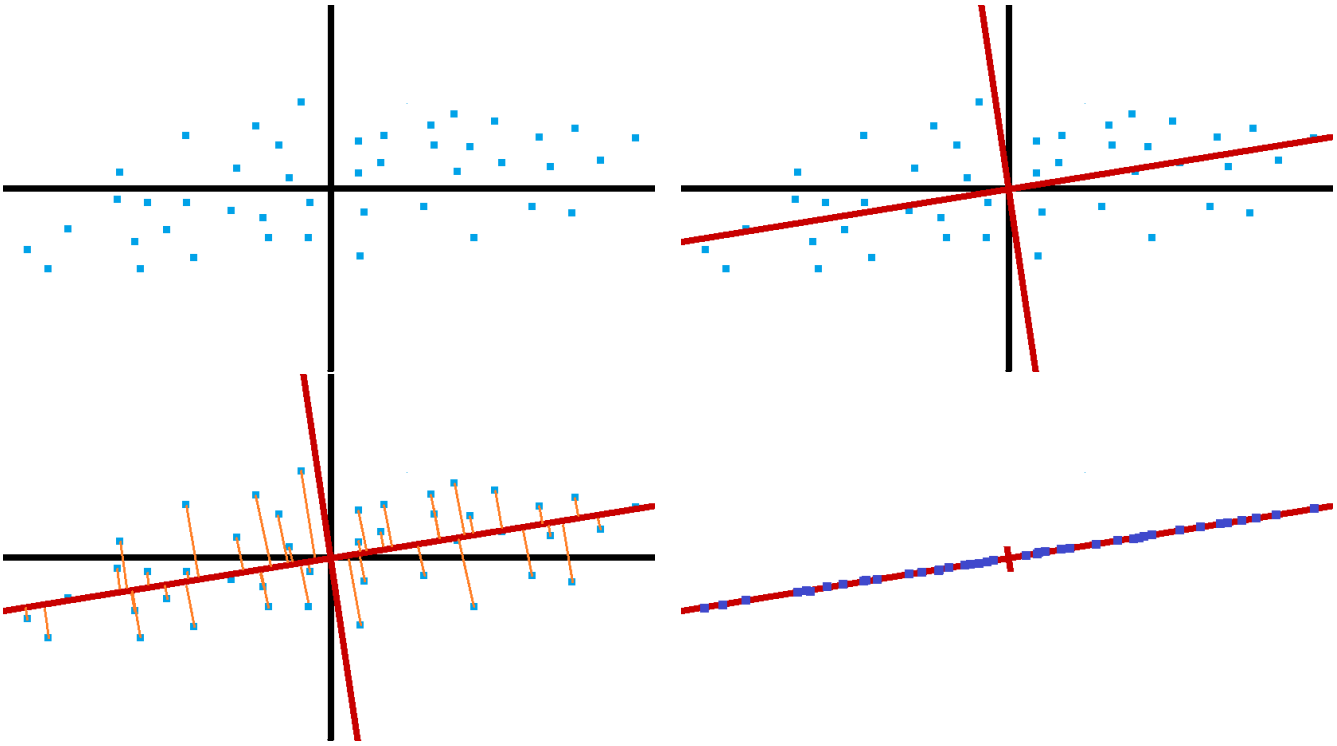


Figure 13. Illustration of PCA on an artificial 2D dataset. The red axes (second image from left) represent the axes of the best elliptic fit. Removing the minor axis by projecting the points on the major axis leads to a dimension reduction and a (small) loss of information (last image on the right).

If the constants $\phi_{m,j}$ are selected **wisely**, then transforming the variables can yield a model that outperforms OLS regression – the predictions might be better than those obtained by fitting $y_i = \beta_0 + \mathbf{x}_i^\top \boldsymbol{\beta} + \varepsilon_i$, $i = 1, \dots, N$.

By definition, $\theta_0 = \beta_0$ and

$$\mathbf{z}_i^\top \boldsymbol{\theta} = \sum_{m=1}^{M} \theta_m z_{i,m} = \sum_{m=1}^{M} \theta_m \mathbf{x}_i^\top \boldsymbol{\phi} = \sum_{m=1}^{M} \theta_m \sum_{j=1}^{p} \phi_{m,j} x_{i,j}$$
$$= \sum_{j=1}^{p} \sum_{m=1}^{M} \theta_m \phi_{m,j} x_{i,j} = \sum_{j=1}^{p} \beta_j x_{i,j} = \mathbf{x}_i^\top \boldsymbol{\beta},$$

where $\beta_j = \sum_{m=1}^{M} \theta_m \phi_{m,j}$, which is to say that the dimension reduction regression is a special case of the original linear regression model, with contstrained coefficients β_j.

Such constraints can help with the bias-variance tradeoff (when $p \gg N$, picking $M \ll p$ can reduce the variance of the fitted coefficients).

The challenge then is to find an appropriate way to pick the $\phi_{m,j}$. We will consider two approaches: **principal components** and **partial least squares**.

5.4.1 Principal Component Analysis

Principal component analysis (PCA) can be used to find the combinations of variables along which the data points are **most spread out**; it attempts to fit a p—**ellipsoid** to a centered representation of the data.

The ellipsoid axes are the principal components of the data. Small axes are components along which the variance is "small"; removing these components leads, in an ideal setting, to a "small" loss of information[48] (see Figure 13)

The procedure is simple:

1. centre and "scale" the data to obtain a matrix $\mathbf{X}$ (warning: not the design matrix);
2. compute the data's covariance matrix $\mathbf{K} = \mathbf{X}^\top \mathbf{X}$;
3. compute $\mathbf{K}$'s eigenvalues $\boldsymbol{\Lambda}$ and its orthonormal eigenvectors matrix $\mathbf{W}$;
4. each eigenvector $\mathbf{w}$ (also known as **loading**) represents an axis, whose variance is given by the associated eigenvalue λ.

The loading that explains the most variance along a single axis (the **first principal component**) is the eigenvector of the empirical covariance matrix corresponding to the largest eigenvalue, and that variance is proportional to the eigenvalue; the second largest eigenvalue and its corresponding eigenvector form the **second principal component** and variance pair, and so on, yielding **orthonormal** principal components $\mathrm{PC}_1, \dots, \mathrm{PC}_r$, where $r = \mathrm{rank}(\mathbf{X})$.[49]

[48] Although there are scenarios where it could be those "small" axes that are more interesting – such as is the case with the "pancake stack" problem.

[49] If some of the eigenvalues are 0, $r < p$, and *vice-versa*, implying that the data was embedded in a r—dimensional manifold to begin with.

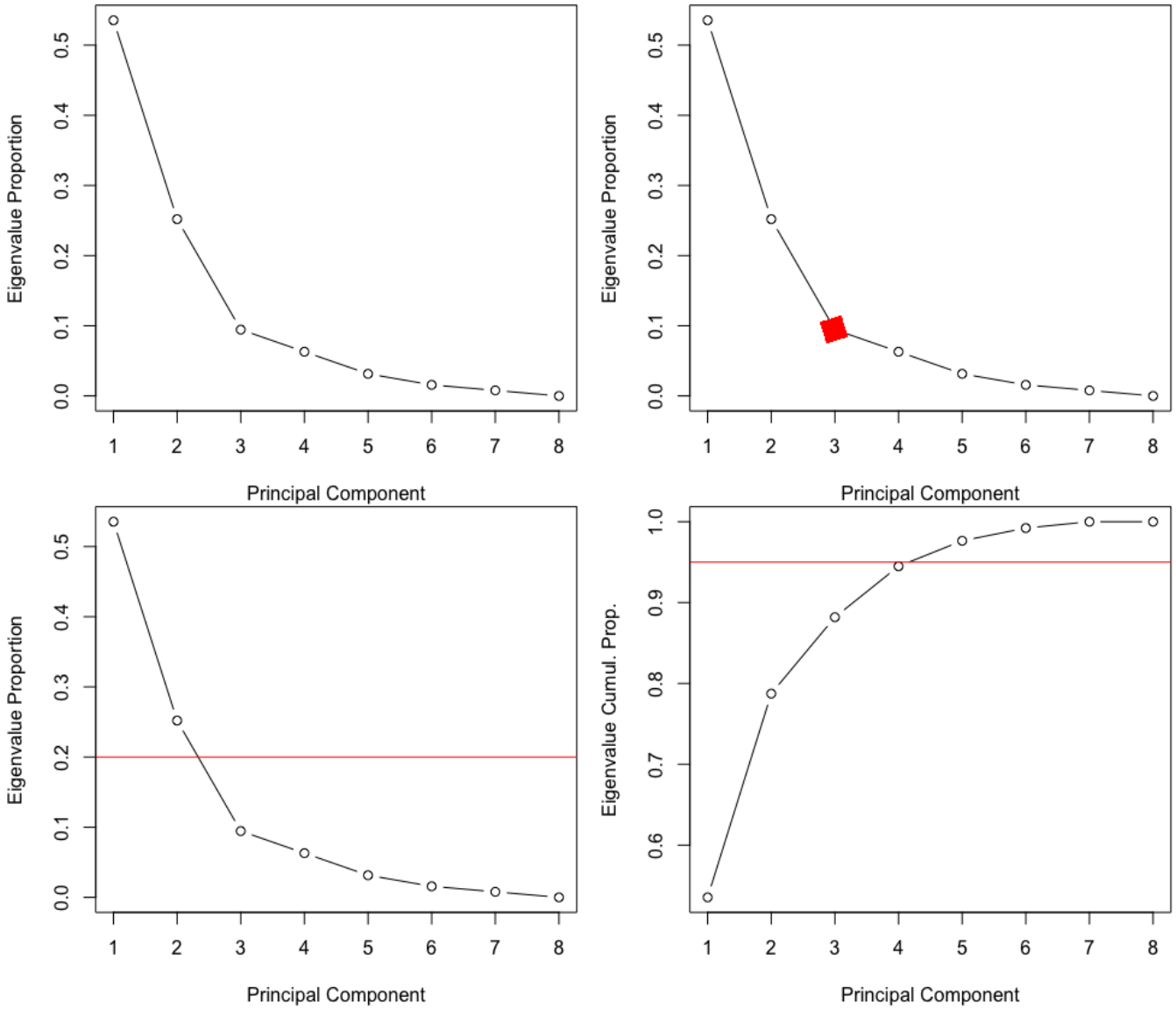


Figure 14. Selecting the number of PCs. The proportion of the variance explained by each (ordered) component is shown in the first 3 charts; the cumulative proportion is shown in the last chart. The kink method is shown in the second image, the individual threshold component in the third, and the cumulative proportion in the fourth.

Principal component analysis can provide an avenue for dimension reduction, by "removing" components with small eigenvalues (see Figure 13): the **proportion of the spread in the data** which can be explained by each principal component (PC) can be placed in a **scree plot** (a plot of eigenvalues against ordered component indices) and retain the ordered PCs:

- for which the eigenvalue is above some threshold (say, 25%);
- for which the cumulative proportion of the spread falls below some threshold (say 95%), or
- prior to a **kink** in the scree plot.

For instance, consider an 8—dimensional dataset for which the ordered PCA eigenvalues are provided below:

PC	1	2	3	4	5	6	7	8
Var	17	8	3	2	1	0.5	0.25	0
Prop	54	25	9	6	3	2	1	0
Cumul	54	79	88	94	98	99	100	100

If only the PCs that explain up to 95% of the cumulative variance are retained, the original data reduces to a 4-dimensional subset; if only the PCs that individually explain more than 25% of the variance are retained, to a 2-dimensional subset; if only the PCs that lead into the first kink in the scree plot are retained, to a 3-dimensional subset (see Figure 14).

PCA is commonly-used, but often without regard to its inherent **limitations**:

- it is dependent on scaling, and so is not uniquely determined;
- with little domain expertise, it may be difficult to interpret the PCs;
- it is quite sensitive to outliers;
- the analysis goals are not always aligned with the principal components, and
- the data assumptions are not always met – in particular, does it always make sense that important data structures and data spread be correlated (the so-called **counting pancakes** problem), or that the components be **orthogonal**?

There are other methods to find the **principal manifolds** of a dataset, including UMAP, self-organizing maps, autoencoders, curvilinear component analysis, manifold sculpting, kernel PCA, etc. PCA.

Formalism because $\mathbf{K}$ is **positive semi-definite** ($\mathbf{K} \geq 0$), the eigenvalues $\lambda_i = s_i^2 \geq 0$ and they can be ordered in a decreasing sequence

$$\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_p), \quad \text{where } \lambda_1 \geq \cdots \lambda_p \geq 0$$

and $\mathbf{W} = [\mathbf{w}_1 | \cdots | \mathbf{w}_p]$.

If $k = \text{rank}(\mathbf{X})$, then there are $p-k$ "empty" principal component (corresponding to null eigenvalues) and k "regular" principal components (corresponding to zero eigenvalues). We write $\mathbf{W}^* = [\mathbf{w}_1 | \cdots | \mathbf{w}_k]$ and $\mathbf{\Lambda}^* = \text{diag}(\lambda_1, \ldots, \lambda_k)$.

If $p - k \neq 0$, then the eigenvalue decomposition of $\mathbf{K}$ is

$$\mathbf{K} = \begin{bmatrix} \mathbf{W}^* & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{\Lambda}^* & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} (\mathbf{W}^*)^\top \\ \mathbf{0} \end{bmatrix} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^\top;$$

if $\mathbf{X}$ is of full rank, then $\mathbf{W}^* = \mathbf{W}$ and $\mathbf{\Lambda}^* = \mathbf{\Lambda}$.

The eigenvectors of K (the $\mathbf{w}_j$) are the **singular vectors** of $\mathbf{X}$: there exist $\mathbf{U}_{N\times N}$ and $\mathbf{\Sigma}_{N\times p}$ such that

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{W}^\top,$$

where

$$\mathbf{U} = \begin{pmatrix} \mathbf{U}^* & \mathbf{0} \end{pmatrix} \quad \text{and} \quad \mathbf{\Sigma} = \begin{pmatrix} \text{diag}(s_i) \\ \mathbf{0} \end{pmatrix}.$$

If $\mathbf{X}$ is of full rank, then $\mathbf{W}$ is **orthonormal** and so represents a **rotation matrix**. As $\mathbf{W}^{-1} = \mathbf{W}^\top$, we must then have $\mathbf{XW} = \mathbf{U\Sigma}$, the **principal component decomposition of X**:

$$\mathbf{T}_{N\times p} = \mathbf{XW}, \begin{bmatrix} \mathbf{t}_1 & \cdots & \mathbf{t}_p \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_N \end{bmatrix}^\top \begin{bmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_p \end{bmatrix}.$$

The link between the principal components and the eigenvectors can be made explicit: the first principal component PC_1 is the loading $\mathbf{w}_1$ (with $\|\mathbf{w}_1\|_2 = 1$) which **maximizes the variance of the first column** of $\mathbf{T}$:

$$\mathbf{w}_1 = \underset{\|\mathbf{w}\|_2=1}{\text{argmax}}\{\text{Var}(\mathbf{t}_1)\} = \underset{\|\mathbf{w}\|_2=1}{\text{argmax}} \left\{ \frac{1}{N-1} \sum_{i=1}^{N} (t_{1,i} - \overline{t_1})^2 \right\}.$$

Since

$$\overline{t_1} = \frac{1}{N} \sum_{j=1}^{N} \text{E}[\mathbf{x}_j^\top \mathbf{w}_1] = \frac{1}{N} \sum_{j=1}^{N} \text{E}\left[\sum_{i=1}^{p} x_{j,i} w_{i,1} \right]$$
$$= \frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{p} w_{i,1} \underbrace{\text{E}[x_{j,i}]}_{\substack{=0 \text{ as } \mathbf{X} \\ \text{is cent.}}} = 0,$$

then $\text{Var}(\mathbf{t}_1) = \frac{1}{N-1}(t_{1,1}^2 + \cdots + t_{n,1}^2)$ and the problem is equivalent to

$$\mathbf{w}_1 = \underset{\|\mathbf{w}\|_2=1}{\text{argmax}}\{t_{1,1}^2 + \cdots + t_{n,1}^2\},$$

By construction, $t_{i,1}^2 = (\mathbf{x}_i^\top \mathbf{w}_1)^2$ for all i, so

$$t_{1,1}^2 + \cdots + t_{n,1}^2 = (\mathbf{x}_1^\top \mathbf{w}_1)^2 + \cdots + (\mathbf{x}_N^\top \mathbf{w}_1)^2 = \|\mathbf{X}\mathbf{w}_1\|_2 = \mathbf{w}_1 \mathbf{X}^\top \mathbf{X} \mathbf{w}_1.$$

Hence,

$$\mathbf{w}_1 = \underset{\|\mathbf{w}\|_2=1}{\text{argmax}}\{\mathbf{w}\mathbf{X}^\top\mathbf{X}\mathbf{w}\} = \underset{\|\mathbf{w}\|_2=1}{\text{argmax}}\{\mathbf{w}^\top\mathbf{K}\mathbf{w}\};$$

this is equivalent to finding the maximizer of $F(\mathbf{w}) = \mathbf{w}^\top\mathbf{K}\mathbf{w}$ subject to the the constrain $G(\mathbf{w}) = 1 - \mathbf{w}^\top\mathbf{w} = 0$. We solve this problem by using the method of Lagarange multipliers; any optimizer $\mathbf{w}^*$ must be either:

1. a critical point of F, or
2. a solution of $\nabla F(\mathbf{w}) + \lambda \nabla G(\mathbf{w}) = \mathbf{0}$, $\lambda \neq 0$.

But $\nabla F(\mathbf{w}) = 2\mathbf{K}\mathbf{w}$ and $\nabla G(\mathbf{w}) = -2\mathbf{w}$; either $\mathbf{w}^* \in \ker(\mathbf{K})$ (case 1) or $2\mathbf{K}\mathbf{w}^* - 2\lambda^*\mathbf{w}^* = \mathbf{0}$ (case 2); either

$$\mathbf{K}\mathbf{w}^* = \mathbf{0} \quad \text{or} \quad (\mathbf{K} - \lambda^* I)\mathbf{w}^* = \mathbf{0}, \ \lambda^* \neq 0.$$

In either case, $\lambda^* \geq 0$ is an eigenvalue of K, with associated eigenvector $\mathbf{w}^*$. There are at most p distinct possibilities $\{(\lambda_j, \mathbf{w}_j)\}_{j=1}^p$, and for each of them

$$\mathbf{w}_j^\top \mathbf{K} \mathbf{w}_j = \mathbf{w}_j^\top \lambda_j \mathbf{w}_j = \lambda_j \mathbf{w}_j^\top \mathbf{w}_j = \lambda_j,$$

since $\mathbf{w}_j^\top \mathbf{w}_j = 1$. Thus,

$$\underset{\|\mathbf{w}\|_2=1}{\text{argmax}}\{\text{Var}(\mathbf{t}_1)\} = \underset{\|\mathbf{w}\|_2=1}{\text{argmax}}\{\lambda_j\} = \mathbf{w}_1 = \text{PC}_1,$$

since $\lambda_1 \geq \lambda \geq 0$ for all eigenvalues λ of $\mathbf{K}$.

A similar argument shows that $\mathbf{w}_j$, $j = 2, \ldots, p$, is the direction along which the variance is the jth highest, assuming that $\mathbf{w}_j$ is orthonormal to all the preceding $\mathbf{w}_\ell$, $\ell = 1, \ldots, j-1$, and that the variance is proportional to λ_j.

The process is repeated at most p times, yielding r nontrivial principal components $\text{PC}_1, \ldots, \text{PC}_r$, where $r \leq p$ is the rank($\mathbf{X}$).

Thus, we see that the rotation matrix $\mathbf{W}$ that maximizes the variance sequentially in the columns of $\mathbf{T} = \mathbf{XW}$ is the matrix of eigenvectors of $\mathbf{K} = \mathbf{X}^\top\mathbf{X}$.

5.4.2 Principal Components Regression

Let us assume that M **principal components** $\{Z_1,\dots,Z_M\}$ have been retained, where

$$Z_i = \mathbf{w}_i^\top (X_1,\dots,Z_p),$$

assuming that the eigenvectors $\mathbf{w}_i$ are ordered according to the corresponding eigenvalues ($\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$):

- the first principal component is the **normalized** (centered and scaled) linear combination of variables with the largest variance;
- the second principal component is the normalized linear combination of variables with the largest variance, subject to having no correlation with all previous components (the first);
- ...
- the Mth principal component is the normalized linear combination of variables with the largest variance, subject to having no correlation with all previous components.

The regression function $f(\vec{x}) = \mathrm{E}[Y \mid \vec{X} = \vec{x}]$ is hopefully well approximated by the function $g(\vec{z}) = \mathrm{E}[Y \mid \vec{Z} = \vec{z}]$, i.e.,

$$\hat{y}_z = g(\mathbf{z}) = \gamma_0 + \gamma_1 z_1 + \cdots + \gamma_M z_M$$

should compare acceptably to

$$\hat{y}_z = f(\mathbf{z}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

The main challenge is to determine the optimal M. If M is too small, we run the risk of having a model with high squared bias and low variance (**underfitting**); if M is too large, not only we we not achieve much in the way of dimension reduction, but we might produce a model with low squared bias and high variance (**overfitting**).

Any method that allows for the estimation of $\mathrm{MSE}_{\mathrm{Te}}$ (such as cross-validation) could be used to select M, but there are other approaches as well (see p.34).

5.4.3 Partial Least Squares

In principal component regression (PCR), the identified **directions** (linear combinations) that best represent the predictors $\{X_1,\dots,X_p\}$ are determined in an **unsupervised** manner: the response Y plays no role in determining the principal components.

As such, there is no guarantee that the directions that best explain the predictors are also the best directions to use to predict the response.

The framework for **partial least squares** is the same as that for PCR, except that the directions Z_i are selected both to explain the predictors and to be related to the response Y.

As in PCR, we start by **normalizing** (centering and scaling) the training set

$$\mathbf{X} = \begin{bmatrix} \mathbf{1} & \mathbf{x}_1 & \cdots & \mathbf{x}_p \end{bmatrix}.$$

The first direction Z_1 is computed using the OLS coefficient estimates of

$$Y_i = \phi_{0,j}^1 + \phi_{1,j} X_{i,j} + \gamma_i, \quad j = 1,\dots,p, i = 1,\dots,N.$$

Note that each $\phi_{1,j}$ is proportional to $\rho_{X_j,Y}$ and that the direction

$$Z_1 = \sum_{j=1}^{p} \phi_{1,j} X_j = \boldsymbol{\phi}_1^\top \vec{X}$$

places the highest weights on the predictors that are most strongly linked to the response.

Now, we run an OLS regression of Y using Z_1 as a predictor:

$$Y_i = \psi_0 + \psi_1 z_{1,i} + \varepsilon_i, \quad i = 1,\dots,N$$

and let $\varepsilon_i = Y_i - \psi_0 - \psi_1 z_{1,i}$ be the component of the data not "explained" by Z_1.

THe second direction Z_2 is computed using the OLS coefficient estimates of

$$\varepsilon_i = \phi_{0,j}^2 + \phi_{2,j} X_{i,j} + \gamma_i, \quad j = 1,\dots,p, i = 1,\dots,N.$$

Note that each $\phi_{2,j}$ is proportional to $\rho_{X_j,\varepsilon}$ and that the direction

$$Z_2 = \sum_{j=1}^{p} \phi_{2,j} X_j = \boldsymbol{\phi}_2^\top \vec{X}$$

places higher weights on the predictors that are most strongly linked to the first residual (which is to say, the component that does not explain Z_1).

The process continues in the same way, building directions $Z_3,\dots,Z_p$ that are strongly linked, in sequence, to the preceding residuals; as the chain starts with the response Y, the directions do take into account both the related response and the predictor structure.[50]

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

- in the bias-variance trade-off, we must strike the right balance when it comes to model complexity, which is usually measured in terms of the number of parameters that must be estimated from tTr;
- while this allows us to compare completely different models with one another, it also suggests that models that use fewer predictors as inputs are not as complex as those that use the full set of predictors;
- the full models are not necessarily the ones that perform best (in terms of $\mathrm{MSE}_{\mathrm{Te}}$), due , in parts, to the curse of dimensionality;
- predictor subset selection methods can be used to select the best model, but the cross-validation approach is preferred;
- other approaches (shrinkage, feature selection, dimension reduction) could also prove competitive.

[50] The problem of selecting M is tackled as it is in PCR.

6. Nonlinear Models and Curve Fitting

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

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

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. see Section 7):

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

6.1 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 [15, Section 7.3]).

6.1.1 Polynomial Regression

We can extend the simple linear 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 (or in terms of the predictors $\{x, x^2, \ldots, x^d\}$).

The solution is to 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 $\hat{\boldsymbol{\beta}}$ are learned using the training set Tr.

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

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

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

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

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

and $\mathrm{se}(\hat{f}(\tilde{\mathbf{x}})) = \sqrt{\hat{\mathrm{Var}}(\hat{f}(\tilde{\mathbf{x}}))}$ and

$$\hat{f}(\tilde{\mathbf{x}}) \pm 2\,\mathrm{se}(\hat{f}(\tilde{\mathbf{x}}))$$

constitutes a 95% C. I. for $\hat{f}(\tilde{\mathbf{x}})$, assuming normality of the error terms.[51]

Example The charts in Figure 15 show polynomial regressions ($d = 4$) and confidence intervals for life expectancy against 4 different predictors in the 2011 Gapminder data.

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 $\mathrm{MSE}_{\mathrm{Te}}$.

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. Polynomial regression should be used **very carefully**, staying within the domain and making sure to centre the predictors to reduce variance inflation.

6.1.2 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 $y = f(\mathbf{x})$.

Step functions can be used to keep things "local".

Let c_i, $i = 1, \ldots, K$ lie in range(X) and consider the following $K+1$ new predictors:

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

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

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

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

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

[51] Polynomial logistic regression can be used for 2–group classification, by modeling

$$\pi_1(x) = P(Y = C_1 \mid X = x) = \frac{\exp(\beta_0 + \beta_1 x + \cdots + \beta_d x^d)}{1 + \exp(\beta_0 + \beta_1 x + \cdots + \beta_d x^d)}.$$

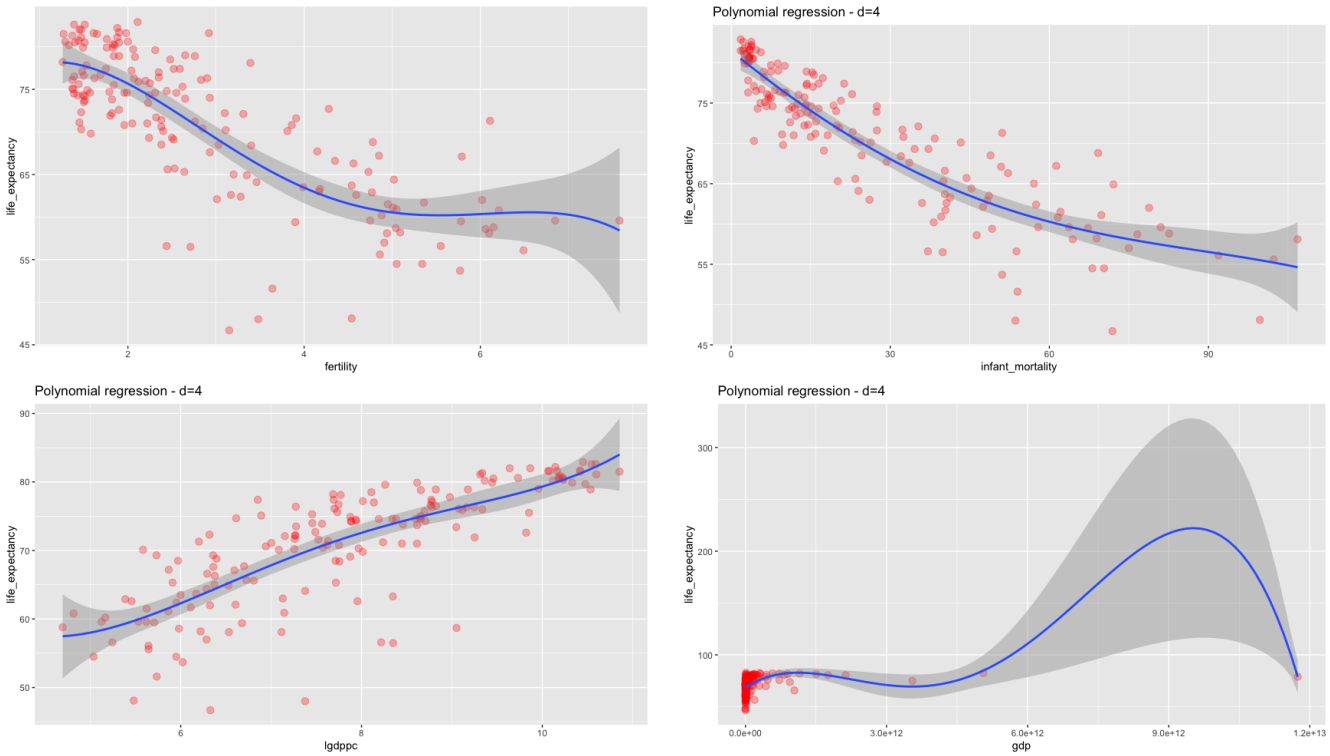


Figure 15. Polynomial regression ($d = 4$) for life expectancy against fertility (top left), infant mortality (top right), log of GDP per capita (bottom left), and GDP (bottom right). Note the extreme behaviour of the polynomial in the fourth case.

The **step function regression model** is

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

it can also be solved using the OLS framework.[52]

For a given X, at most one of $C_1(X), \ldots, C_K(X)$ is $\neq 0$; thus, when $X < c_1$, $C_j(X) = 0$ for all $j = 1, \ldots, K$, 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 $c_1, \ldots, c_K$ unless there are natural gaps in the predictors.[53]

We did not discuss how 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).

Example The charts in Figure 16 show step function regression for life expectancy against different predictors in the 2011 Gapminder data.

[52] Thus a 95% C.I. can be built just as with polynomial regression.
[53] Although see Section 7.1.1.

6.2 Splines

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

6.2.1 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 (defined by **knots** in the 1-dimensional case), such as:

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

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 (see image on the next page, at the bottom of the first column, modified from [15]). 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).

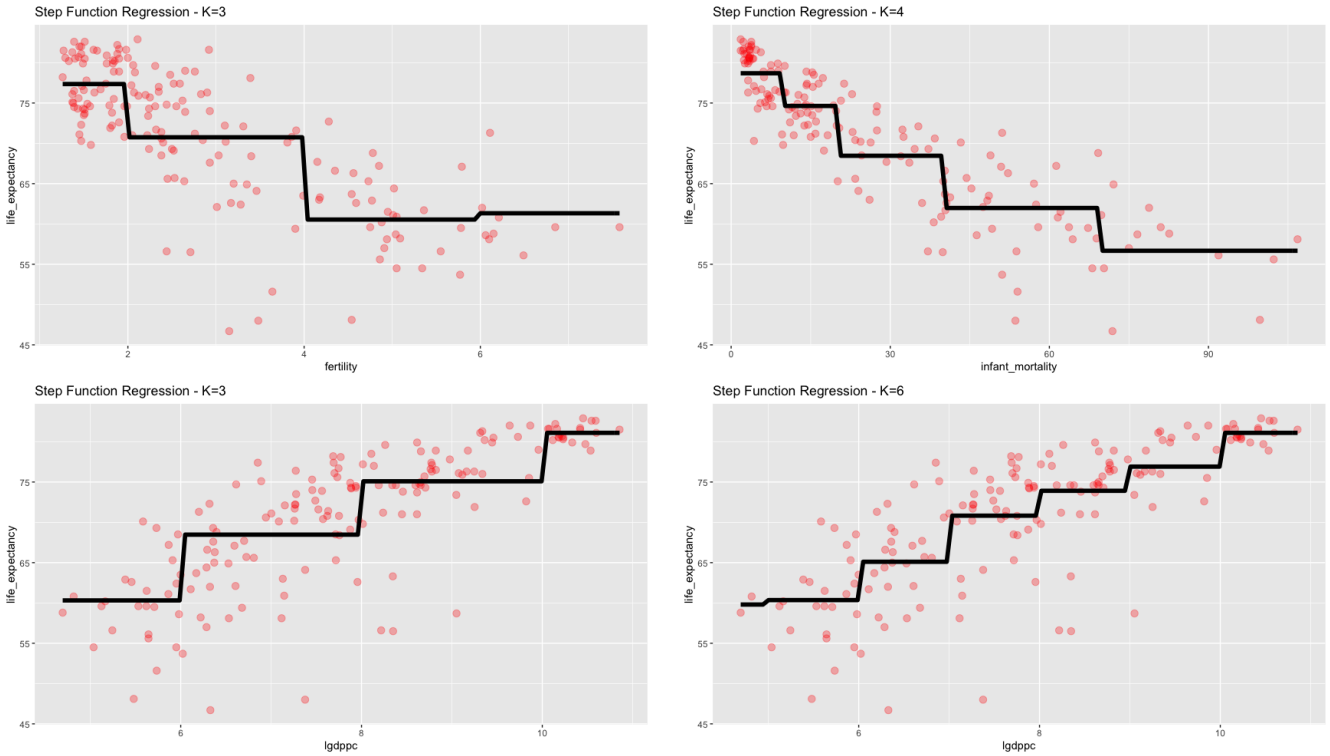
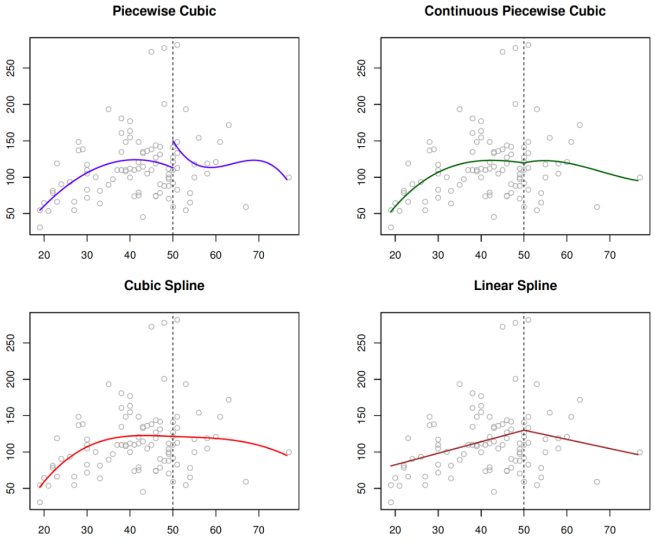


Figure 16. Step function regression for life expectancy against fertility with $K = 3$ (top left), infant mortality with $K = 4$ (top right), log of GDP per capita with $K = 3$ (bottom left), and log of GDP per capita with $K = 6$ (bottom right).

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** (the polynomials must agree at the knots), we reduce the number of effective parameters. We can also require a **continuously differentiable** fit (the derivatives must also agree at the knots), further reducing the number of effective parameters.

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 and is extrapolated linearly 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 is available: predictions, diagnostics, remedial measures, confidence intervals, and extension to logistic regression, as needed.

Example The charts in Figure 17 show 4 cubic splines and 4 natural cubic splines for life expectancy against fertility in the 2011 Gapminder data.

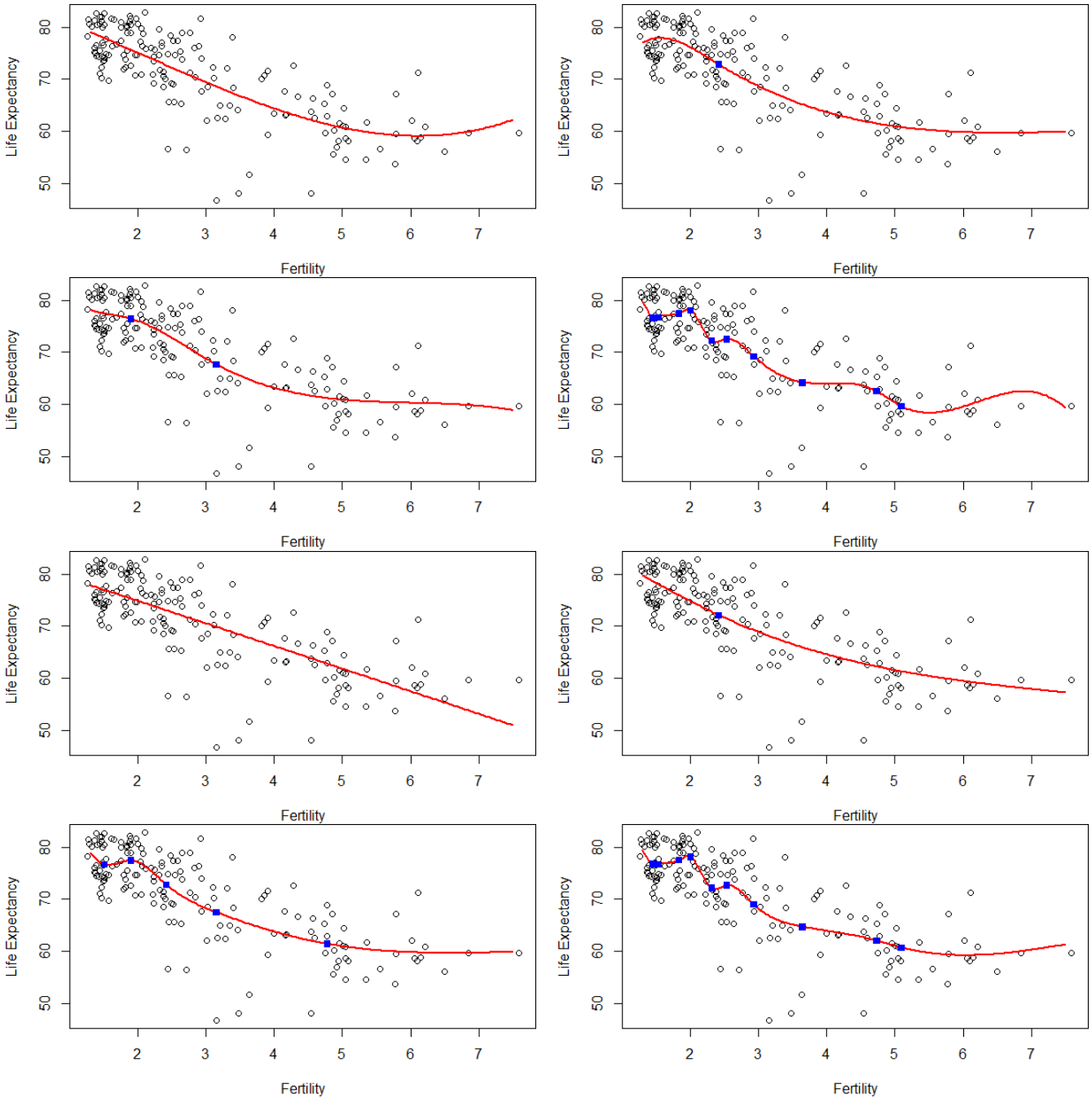


Figure 17. Top 2 rows: cubic splines for life expectancy against fertility with 0 knot (top left), 1 knot (top right), 2 knots (bottom left), and 10 knots (bottom right), at equally-spaced quantiles. Bottom two rows: natural cubic splines for life expectancy against fertility with 0 knot (top left), 1 knot (top right), 5 knots (bottm left), and 10 knots (bottom right), at equally-spaced quantiles. The knots are not equally-spaced values; their position depend on the quantiles of the fertility variable. They are shown as blue squares.

In theory, we place more knots in locations where the spline functions is believed to vary more rapidly, and fewer knots where it is more stable. In practice, the knots are placed uniformly at **quantiles** of the predictor variable X.

Cross-validation (again!) can be used to determine K: compute the estimated SSRes for various K (10-fold CV), 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 Figure 15).

6.2.2 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:

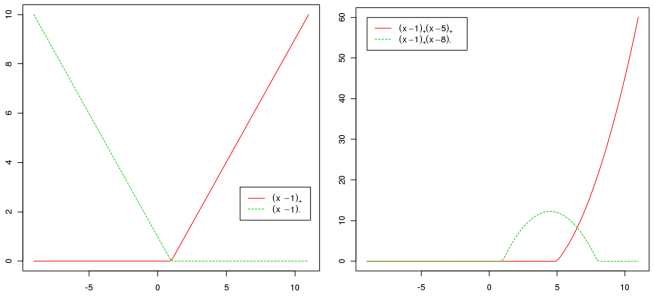
$$1, (x-\xi_k)_\pm, (x-\xi_{k_1})_\pm (x-\xi_{k_2})_\pm,$$

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

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

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

For instance, $(x-1)_\pm$, $(x-1)_+(x-5)_+$, and $(x-1)_+(x-8)_-$ are shown below:



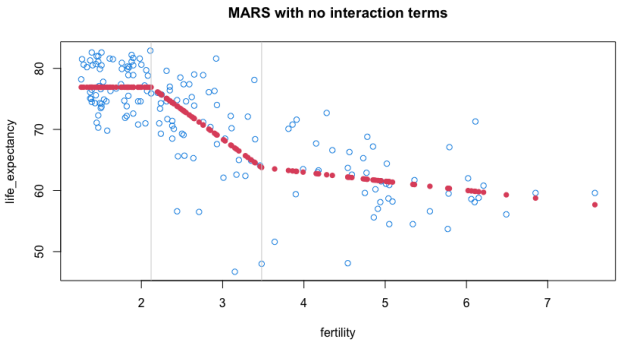
A **multivariate adaptive regression spline** (MARS) model is expressed as

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

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

MARS iteratively adds terms to its model; 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 Section 7.1. One advantage: the knots are selected automatically.

Example The chart below shows the MARS for life expectancy against fertility in the 2011 Gapminder data.



6.2.3 Smoothing Splines

Regression splines use the following approach:

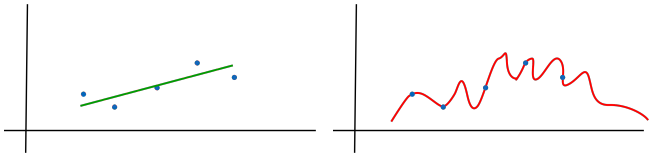
1. identify K knots $\xi_1, \ldots, \xi_K$;
2. produce some basis functions $\{b_1(x), \ldots, b_K(x)\}$, and
3. use OLS to estimate the coefficients of

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

We can use a mathematical approach in order to produce a spline. We need to first find a function g that provides a good fit for the available data; in other words, we are looking for a function g such that

$$\text{SSRes} = \sum_{i=1}^{N} (y_i - g(x_i))^2 \quad \text{is "small".}$$

But we also need to g to be constrained otherwise any smooth function interpolating (x_y, y_i), $i = 1, \ldots, N$ would yield SSRes $= 0$, at the cost of **severe overfitting** and loss of **interpretability**. Keep in mind, however, that too many constraints can result in underfitting the data.



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

$$g_\lambda = \underset{h}{\text{argmin}} \left\{ \underbrace{\sum_{i=1}^{N} (y_i - h(x_i))^2}_{\text{SSRes loss}} + \lambda \underbrace{\int_{\Omega(X)} \left[h''(t)\right]^2 dt}_{\text{penalty term}} \right\},$$

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

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).[54]

[54] If h represents a straight line, say, the penalty term would be zero.

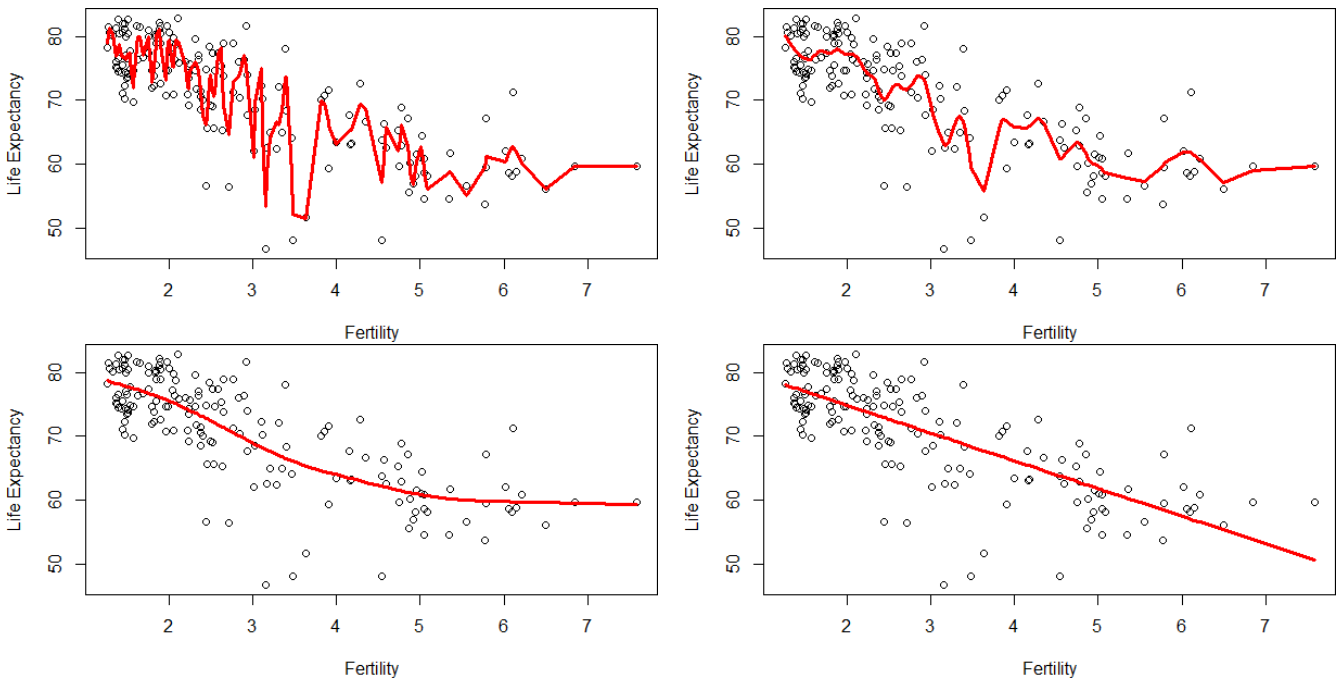


Figure 18. Smoothing splines for life expectancy against fertility with parameter values 0 (top left), 0.5 (top right), 1 (bottom left), and 1.5 (bottom right). Note that the parameter values do not correspond to the tuning parameter (read the `smooth.spline()` documentation for details). See the evolution from "wiggly" model to OLS.

When $\lambda \to 0$, the penalty term has no effect, so we expect g_λ to be "jumpy" and to interpolate the observations exactly (overfitting). When $\lambda \to \infty$, the penalty term dominates and g_λ is a function for which $\int [g''_\lambda(t)]^2 dt \to 0$ over $\Omega(X)$, so $g \to$ linear OLS solution (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 linear outside $\Omega(X)$, but **it is not the one that would be obtained from a regression spline** as it 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 vary from n to 2 as λ goes from 0 to ∞ (R's `smooth.spline()` uses a different parameter), as in Figure 18.

Example The charts in Figure 18 show the smoothing spline for life expectancy against fertility in the 2011 Gapminder data, for 4 smoothing parameter values.

Note the evolution of a flexible but non-interpretable model to a rigid but highly interpretable model as the parameter values increase.

6.3 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 (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 solve the model above using OLS (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.

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

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

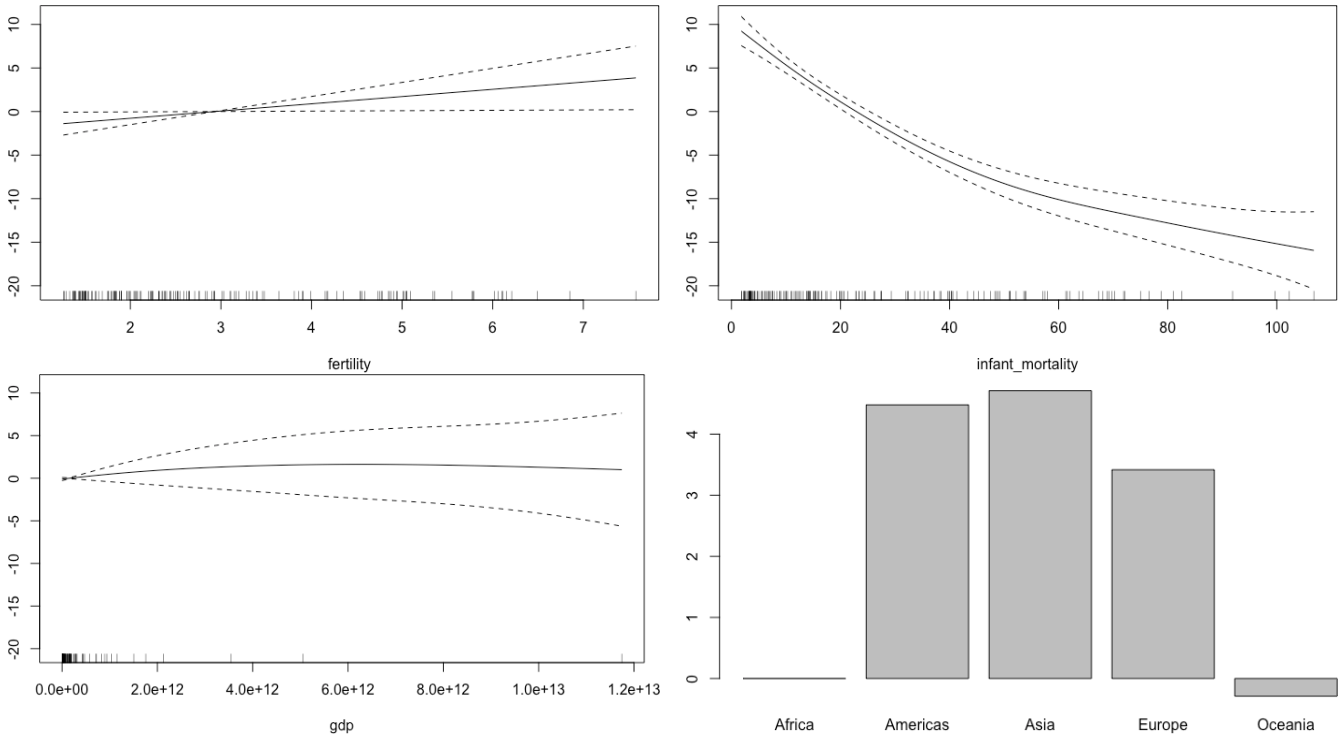


Figure 19. Generalized additive model for life expectancy, with contributions from fertility (top left), infant mortality (top right), GDP (bottom left), and continent (bottom right). The model intercept is 68.1186.

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

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

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

Advantages:

- GAM can fit a non-liner 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), but they quickly get out of hand (due to CoD issues).

Example The charts in Figure 19 show the individual contributions of fertility, infant mortality, GDP, and continental membership to life expectancy in the 2011 Gapminder data. The intercept is $\beta_0 = 68.1186$:

$$\text{life expectancy} \approx \beta_0 + f_1(\text{fertility}) + f_2(\text{infant mortality}) + f_3(\text{gdp}) + \beta_{\text{continent}}.$$

For instance, the life expectancy for an American country with fertility= 3, infant mortality= 1, GDP= 6×10^{12} would be approximately

$$68.1 + 0 + 10 + 2 + 4.5 = 84.6.$$

The rest of the document will focus on non-parametric methods.

7. Other Supervised Approaches

In this section, we present a number of non-parametric approaches, with a focus on classification methods (although we will also discuss regression problems):

- classification and regression trees (CART);
- support vector machines (SVW);
- naïve Bayes classification (NBC);
- artificial neural networks (ANN), and
- ensemble learning.

7.1 Tree-Based Methods

This family of methods involves **stratifying** or **segmenting** the predictor space into a small number of "simple" regions.

The set of **splitting rules** used to segment the space can be summarized using a **tree**, whence their name.

On the one hand, tree-based methods are **simple** and **easy to interpret**; on the other, they are not competitive with the best supervised learning methods when it comes to predictive accuracy.

Nevertheless, there are instances where the ease of interpretability overrules the lessened accuracy.

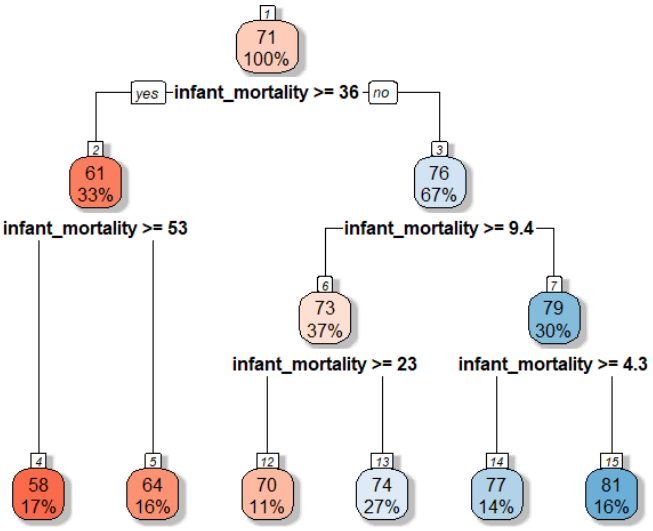
Tree-based methods are applicable both to regression and to classification problems.

7.1.1 Regression Trees

We will introduce the important concepts via the 2011 Gapminder dataset, where the response Y is once again the life expectancy of nations, and the predictors X_1 and X_2 are the fertility rates and infant mortality rates per nation.

In Figure 6, we see that when X_1 and X_2 are both high, Y is low, and when X_1 and X_2 are both low, Y is high. But what is the pattern "in the middle"?

Below, we see a possible **regression tree** for the (full) dataset ($N = 166$ observations).



This can also be re-written as:

```
1) root (166) 70.82349
   2) infant_mortality>=35.65 (54) 60.85370
     4) infant_mortality>=52.9 (28) 58.30714 *
     5) infant_mortality< 52.9 (26) 63.59615 *
   3) infant_mortality< 35.65 (112) 75.63036
     6) infant_mortality>=9.35 (62) 72.89516
      12) infant_mortality>=22.85 (18) 69.50000 *
      13) infant_mortality< 22.85 (44) 74.28409 *
     7) infant_mortality< 9.35 (50) 79.02200
      14) infant_mortality>=4.25 (23) 76.86087 *
      15) infant_mortality< 4.25 (27) 80.86296 *
```

Node 1 is the tree's **root** (initial node) with 166 (100%) observations; the average life expectancy for these observations is 70.82.

The root is also the tree's first **branching point**, separating the observations into two groups: node 2 with 54 observations (33%), given by infant mortality ≥ 35.65, for which the average life expectancy is 60.85, and node 3 with 112 observations (67%), given by infant mortality < 35.65, for which the average life expectancy is 75.63. Note that $54 + 112 = 166$ and that

$$\frac{54(60.81) + 112(75.63)}{54 + 112} = 70.82.$$

Node 2 is an **internal node** – it is further split into two groups: node 4 with 28 observations (17%), given with the additional rule infant mortality ≥ 52.9, for which the average life expectancy is 58.31, and node 5 with 26 observations (16%), given with the additional rule infant mortality < 52.9, for which the average life expectancy is 63.60. Note that $28 + 26 = 54$ and that

$$\frac{28(58.31) + 26(63.60)}{28 + 26} = 60.85.$$

Both nodes 4 and 5 are **leaves** (final nodes, terminal nodes); the tree does not grow any further on that branch.

The tree continues to grow from node 3, eventually leading to 4 leaves on that branch (there is an intermediate branching point). There are 6 leaves in total, 5 branching points (including the root) and the tree's **depth** is 3 (excluding the root).

Note that only one feature is used in the regression tree in this example: to make a prediction for a new observation, only infant mortality is needed. If it was 21, say, the observation's leaf would be node 13 and we would predict that the life expectancy of that nation would be 74.28.

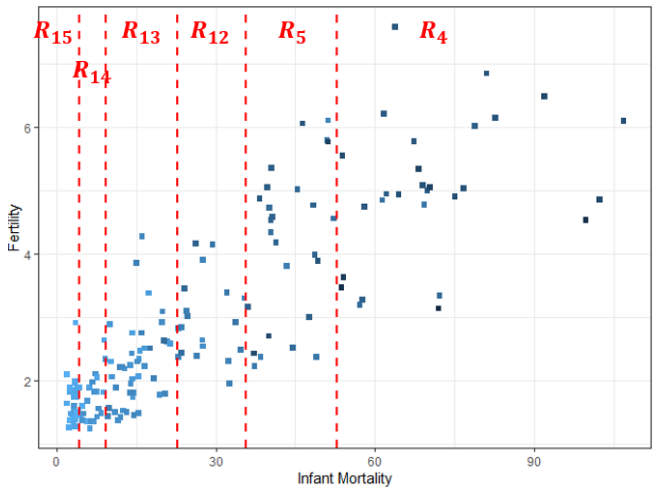
The tree diagram is a useful heuristic, especially as it allows the display results without resorting to a multi-dimensional chart, but it does obscure the **predictor space's stratification**.

In our example, let

$$\begin{aligned}
R_4 &= \{(\text{infant mortality}, \text{fertility}) \mid \text{infant mortality} \geq 52.9\} \\
R_5 &= \{(\text{infant mortality}, \text{fertility}) \mid 36.65 \leq \text{infant mortality} < 52.9\} \\
R_{12} &= \{(\text{infant mortality}, \text{fertility}) \mid 22.85 \leq \text{infant mortality} < 35.65\} \\
R_{13} &= \{(\text{infant mortality}, \text{fertility}) \mid 9.35 \leq \text{infant mortality} < 22.85\} \\
R_{14} &= \{(\text{infant mortality}, \text{fertility}) \mid 4.25 \leq \text{infant mortality} < 9.35\} \\
R_{15} &= \{(\text{infant mortality}, \text{fertility}) \mid \text{infant mortality} < 4.25\}
\end{aligned}$$

Note that only infant mortality is involved in the definition of the tree's **terminal nodes**.

The regions are shown below:



The regression tree model for life expectancy would thus be

$$\hat{y}_{R_i} = \text{Avg}\{y \mid (x_1, x_2) \in R_i\} = \begin{cases} 58.3, & i = 4 \\ 63.6, & i = 5 \\ 69.5, & i = 12 \\ 74.3, & i = 13 \\ 76.9, & i = 14 \\ 80.9, & i = 15 \end{cases}$$

The regression tree tells us that infant mortality is the most important factor in determining life expectancy, with a negative correlation. This interpretation is, of course, a coarse oversimplification, but it highlights the advantage of using a regression tree when it comes to **displaying**, **interpreting**, and **explaining** the results.

This tree is not the only way to stratify the data: in what way is it **optimal**,[55] as opposed to some other tree?

Building A Regression Tree The process is quite simple:

1. Divide the predictor space $\mathscr{X} \subseteq \mathbb{R}^p$ into a distinct union J regions R_j, $j = 1, \dots, J$:
$$\mathscr{X} = R_1 \sqcup \cdots \sqcup R_J;$$
2. for any $\mathbf{x} \in R_j$,
$$\hat{y}(\mathbf{x}) = \text{Avg}\{y(\mathbf{z}) \mid \mathbf{z} \in R_j \in \text{Tr}\}.$$

The second step tells us that trees are locally constant (although it does not necessarily have to be so, that is also the assumption we will follow).

In theory, the R_j could have any shape as long as they form a disjoint cover of $\mathscr{X}$; in practice, we use **hyperboxes** whose boundaries $p-1$ affine spaces that are perpendicular/parallel to the p hyperplanes $X_1 \dots \hat{X_k} \dots X_p$, $k = 1, \dots, p$.

We find the optimal $(R_1, \dots, R_J)$ by minimizing

$$\text{SSRes} = \sum_{j=1}^{J} \sum_{\mathbf{x}_i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response y in $R_j \cap \text{Tr}$.

[55] Recall that all supervised learning tasks are optimization problems.

In an ideal world, we would compute SSRes for all partitions of $\mathscr{X}$ into hyperboxes, and pick the one that minimizes SSRes, but that is not computationally feasible, in general.

Instead, we use a growth algorithmic approach known as **recursive binary splitting**, which is both **top-down** (starts at the root and successively splits $\mathscr{X}$ *via* 2 new branches down the tree) and **greedy** (at each step of the splitting process, the best choice is made there and now, rather than by looking at long-term consequences).

Regression Tree Algorithm The algorithm has 10 steps, but it is fairly straightforward.

1. Let $\hat{y}_0 = \text{Avg}\{y \mid (\mathbf{x}, y) \in \text{Tr}\}$.
2. Set the baseline $\text{SSRes}_0 = \sum_{i=1}^{N} (y_i - \hat{y}_0)^2$.
3. For each $1 \leq k \leq p$, order the predictor values of X_k in Tr: $v_{k,1} \leq v_{k,2} \leq \cdots \leq v_{k,N}$.
4. For each X_k, set $s_{k,\ell} = \frac{1}{2}(v_{k,\ell} + v_{k,\ell+1})$, $\ell = 1, \dots, N-1$.
5. For each $k = 1, \dots, p$, $\ell = 1, \dots, N-1$, define
$$R_1(k,\ell) = \{\vec{X} \in \mathbb{R}^p \mid X_k < s_{k,\ell}\} \quad \text{and} \quad R_2(k,\ell) = \{\vec{X} \in \mathbb{R}^p \mid X_k \geq s_{k,\ell}\}.$$
Note that $\mathscr{X} = R_1(k,\ell) \sqcup R_2(k,\ell)$ for all k, ℓ.
6. For each $k = 1, \dots, p$, $\ell = 1, \dots, N-1$, set
$$\text{SSRes}_1^{k,\ell} = \sum_{m=1}^{2} \sum_{\vec{X}_i \in R_m(k,\ell)} (y_i - \hat{y}_{R_m(k,\ell)})^2,$$
where $\hat{y}_{R_m(k,\ell)} = \text{Avg}\{y(\mathbf{x}) \mid \mathbf{x} \in \text{Tr} \cap R_k(k,\ell)\}$.
7. Find k^*, ℓ^* for which $\text{SSRes}_1^{k,\ell}$ is **minimized**.
8. Define the **children sets** $R_1^L = R_1(k^*, \ell^*)$ and $R_1^R = R_2(k^*, \ell^*)$.
9. While some children sets R_μ^ν still do not meet a **stopping criterion**, repeat steps 3 to 8, searching and minimizing SSRes over $\mathscr{X} \cap R_\mu^\nu$, and producing a binary split $R_{\mu+1}^L, R_{\mu+1}^R$.[56]
10. Once the stopping criterion is met for all children sets, the tree's growth ceases, and $\mathscr{X}$ has been partitioned into J regions (renumbering as necessary)
$$\mathscr{X} = R_1 \sqcup \cdots \sqcup R_J,$$
on which the regression tree predicts the J responses $\{\hat{y}_1, \dots, \hat{y}_J\}$, according to $\hat{y}_j = \text{Avg}\{y(\mathbf{x}) \mid \mathbf{x} \in R_j\}$.

For instance, if the training set was $\text{Tr} = \{(x_{1,i}, x_{2,i}, y_i)\}_{i=1}^N$, the algorithm might provide the regression tree in Figure 20.

Tree Pruning Regression trees grown with the algorithm are prone to overfitting; they can provide good predictions on Tr, but they usually make shoddy predictions on Te, because the resulting tree might be too complex (it fits the noise as well as the signal).

A smaller tree with fewer splits might lead to lower variance and better interpretability, at the cost of a little bias.

Instead of simply growing a tree T_0 until each leaf contains at most M observations, say (or whatever other stopping criterion), it could be beneficial to **prune** it in order to obtain some **optimal subtree**.

[56] Multiple splitting criteria are used in practice, such as insisting that all final nodes contain 10 or fewer observations, etc.

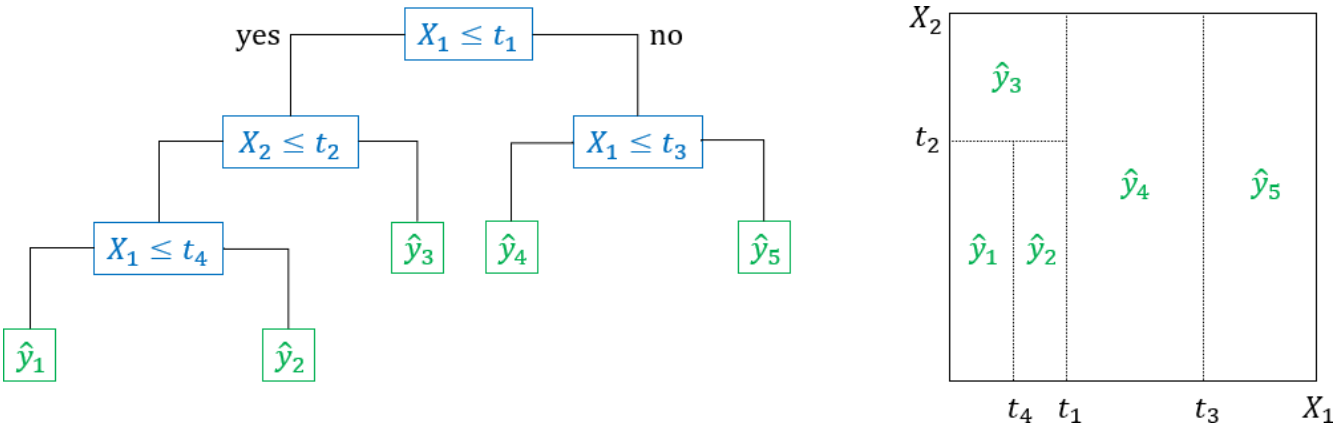


Figure 20. Generic recursive binary partition regression tree for a two-dimensional predictor space, with 5 leaves.

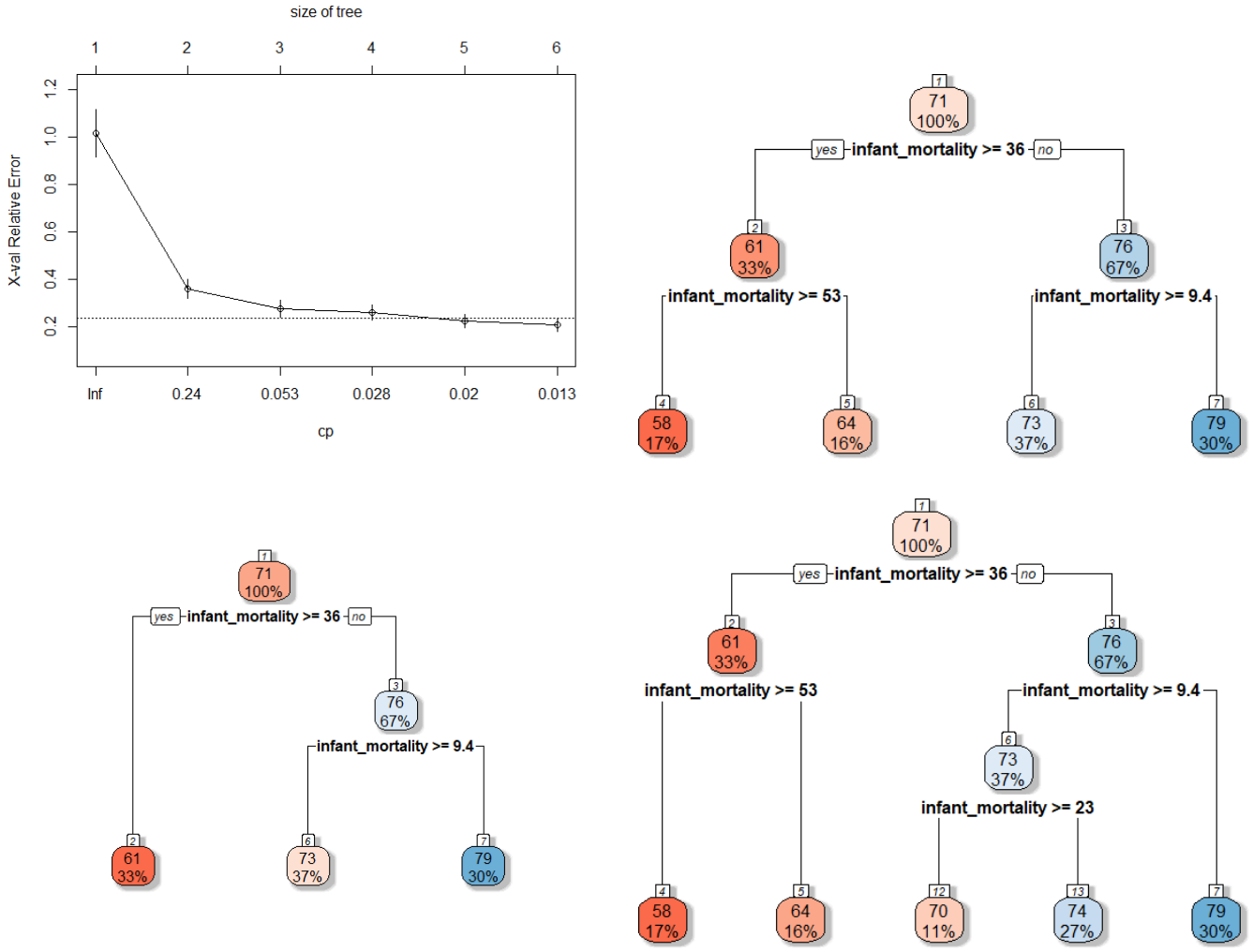


Figure 21. Complexity pruning parmeter (top left), with pruned tree for cp= 0.028 (top right) in the Gapminder 2011 example. Other pruned trees, corresponding to cp= 0.06 (bottom left) and cp= 0.02 (bottom right), are also shown. Note that the tree's complexity increases when cp decreases.

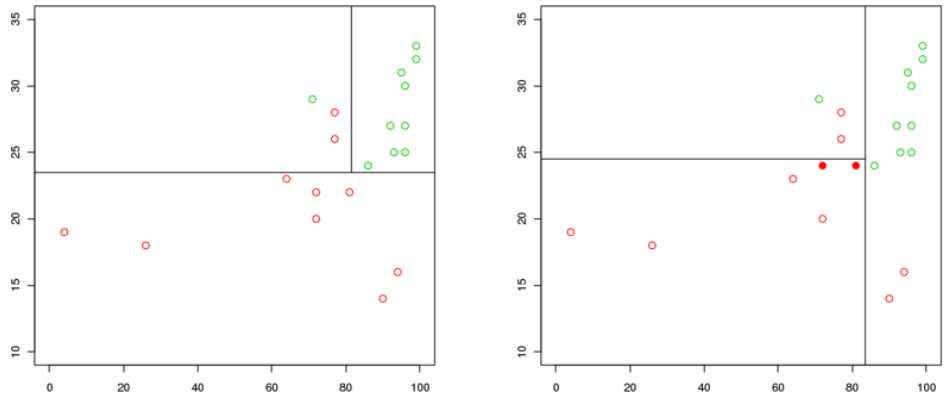


Figure 22. Different tree topologies with small changes in the training set (data modified from [16]).

We use **cost complexity pruning** (CCP) to build a sequence of candidate subtrees indexed by the complexity parameter $\alpha \geq 0$. For each such α, find a subtree $T_\alpha \subseteq T_0$ which minimizes

$$\text{SSRes} + \text{complexity penalty} = \sum_{m=1}^{|T|} \sum_{\mathbf{x}_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|,$$

where $|T|$ represents the number of final nodes in T; when α is large, it is costly to have a complex tree.[57]

Pruning Algorithm Assume that a recursive binary splitting regression tree T_0 has been grown on Tr, using a stopping criterion:

1. apply CCP to T_0 to obtain a "sequence" T_α of subtrees of T_0;
2. divide Tr into K folds;
3. for all $k = 1, \ldots, K$, build a regression tree on $\text{Tr} \setminus \text{Fold}_k$ and evaluate

$$\hat{\text{MSE}}(\alpha) = \underset{1 \leq k \leq K}{\text{Avg}} \{\text{MSE}_k(\alpha) \mid \text{on Fold}_k\};$$

4. return T_{α^*} from step 1, where $\alpha^* = \text{argmin}_\alpha \{\hat{\text{MSE}}(\alpha)\}$.

The Gapminder 2011 tree is pruned in Figure 21.

7.1.2 Classification Trees

The approach for classification is much the same, with a few appropriate substitutions:

1. prediction in a terminal node is either the **class label mode** or the relative frequency of the class labels;
2. SSRes must be replaced by some other fit measure:
   - the **classification error rate**:

$$E = \sum_{j=1}^{J} (1 - \max_k \{\hat{p}_{j,k}\}),$$

   where $\hat{p}_{j,k}$ is the proportion of Tr observations in R_j of class k (this measure is not a recommended choice);
   - the **Gini index**, which measures the total variance across classes

$$G = \sum_{j=1}^{J} \sum_k \hat{p}_{j,k}(1 - \hat{p}_{j,k}),$$

which should be small when the nodes are **pure** ($\hat{p}_{j,k} \approx 0$ or 1 throughout the regions), and
   - the **cross-entropy deviance**

$$D = -\sum_{j=1}^{J} \sum_k \hat{p}_{j,k} \ln \hat{p}_{j,k},$$

which behaves like the Gini index, numerically.

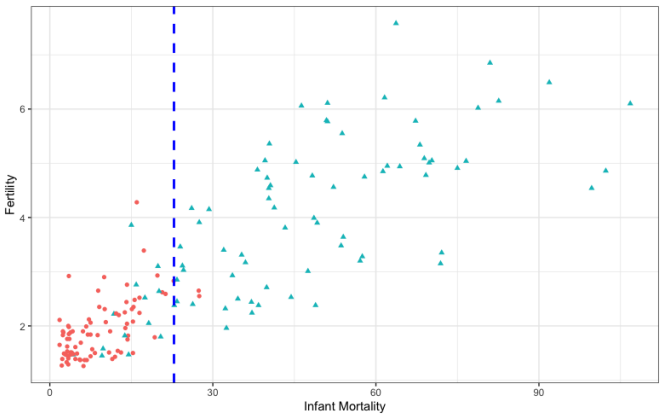
One thing to note is that **classification and regression trees** (jointly known as CART) suffer from high variance and their structure is **unstable** – using different training sets typically gives rise to wildly varying trees.

Sometimes, simply modifying the level of only one of the predictors in only one of the observations can yield a tree with a completely different topology (see Figure 22).

This lack of robustness is a definite strike against the use of CART; despite this, the relative ease of their implementation makes them a popular classification tool.

Example A classification tree for the Gapminder 2011 dataset is shown below:

```
1) root (166) high/low (0.5 0.5)
  2) infant_mortality< 23 (94) high (0.862 0.138)
  3) infant_mortality>=23 (72) low  (0.028 0.972)
```



Note that this tree should not be used for predictions as it was not built on a training subset of the data.

[57] This is similar to the bias-variance trade-off or the regularization framework: a good tree balances considerations of fit and complexity.

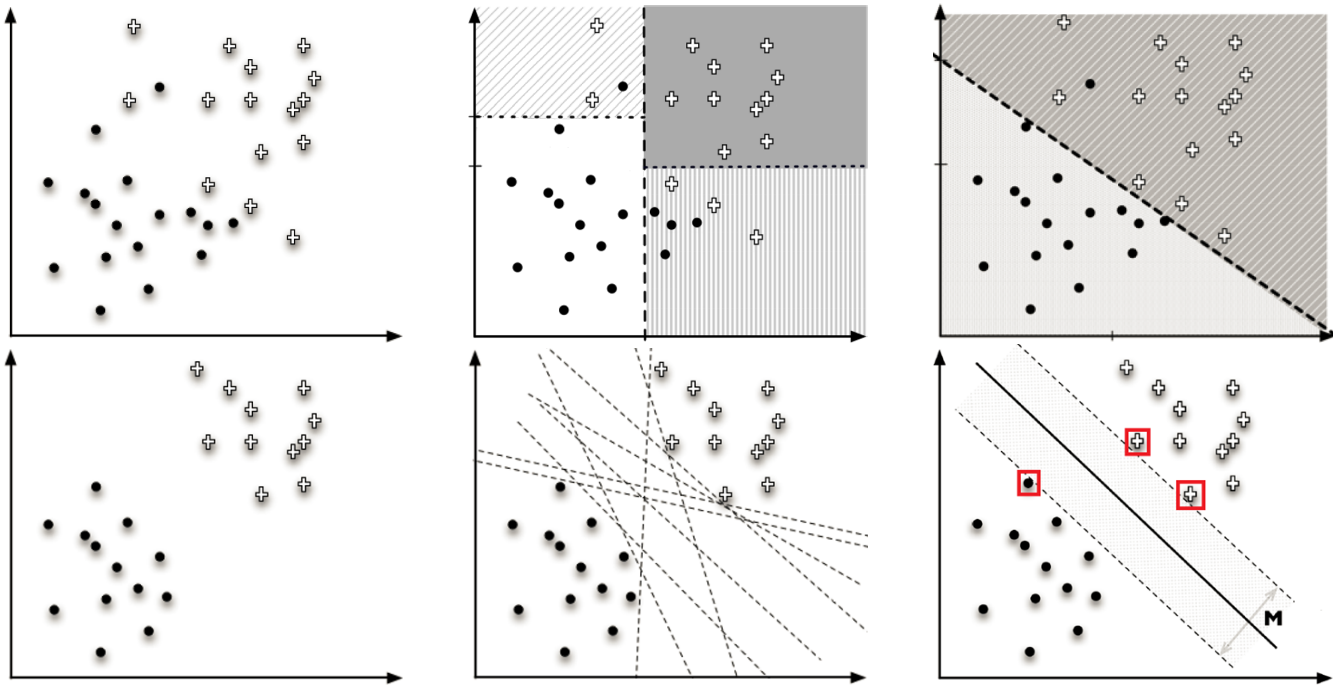


Figure 23. Two-class artificial dataset (top left), with classification tree (top middle) and separating hyperplane (top right); linearly separable subset (bottom left), separating hyperplanes (bottom middle), maximal margin hyperplane with support vectors (bottom right), based on [21].

7.2 Support Vector Machines

The next classifier is more sophisticated, from a mathematical perspective. It was invented by computer scientists in the 1990s.

Support vector machines (SVM) attempt to find hyperplanes that separate the classes in the feature space.

In Figure 23, we see an artificial data with 3 features: X_1 and X_2 (numerical), Y (categorical, represented by different symbols).

We grow a classification tree (perhaps the one shown above): two of the leaves are pure, but the risk of misclassification is fairly large in the other 2 (at least for that tree).[58] Without access to more features, that tree is as good as it gets.[59]

But it is easy to draw a decision curve which improves on the effectiveness of the decision tree (see image on the right): a single observation is misclassified by this rule.[60]

Separating hyperplanes do not always exist; we may need to:

- extend our notion of **separability**, and/or
- extend the feature space so separation becomes possible.

A **hyperplane** $H_{\boldsymbol{\beta},\beta_0} \subseteq \mathbb{R}^p$ is an affine (flat) subset of $\mathbb{R}^p$, with

$$\dim\left(H_{\boldsymbol{\beta},\beta_0}\right) = p-1;$$

in other words,

$$H_{\boldsymbol{\beta},\beta_0} : \beta_0 + \boldsymbol{\beta}^\top \mathbf{x} = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = 0.$$

The vector $\boldsymbol{\beta}$ is normal to $H_{\boldsymbol{\beta},\beta_0}$; if $\beta_0 = 0$, $H_{\boldsymbol{\beta},\beta_0}$ goes through the origin in $\mathbb{R}^p$.

Set $F(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}$; then $F(\mathbf{x}) > 0$ for points on one "side" of $H_{\boldsymbol{\beta},\beta_0}$ and $F(\mathbf{x}) < 0$ for points on the other.[61]

In a binary classification problem $\mathscr{C} = \{C_1, C_2\} = \{\pm 1\}$. If

$$y_i F(\mathbf{x}_i) > 0, \quad \text{for all } (\mathbf{x}_i, y_i) \in \text{Tr}$$

(or, $y_i F(\mathbf{x}_i) < 0$ for all $(\mathbf{x}_i, y_i) \in \text{Tr}$), then $F(\mathbf{x}_i) = 0$ determines a **separating hyperplane** for Tr (which does not need to be unique, see Figure 23), and we say that Tr is **linearly separable**.

Among all separating hyperplanes, the one which provides the widest separation between the two classes is the **maximal margin hyperplane** (MMH); training observations on the boundary of the **separating strip** are called the **support vectors** (see boxed observations in Figure 23).

The classification problem simplifies, as always, to a constrained optimization problem:

$$(\boldsymbol{\beta}^*, \beta_0^*) = \underset{(\boldsymbol{\beta},\beta_0)}{\text{argmax}}\{M_{(\boldsymbol{\beta},\beta_0)}\} \quad \text{s.t. } y_i(\beta_0 + \boldsymbol{\beta}\mathbf{x}_i) \geq M_{(\boldsymbol{\beta},\beta_0)}$$

for all $(\mathbf{x}_i, y_i) \in Tr$, with MMH given by $F(\mathbf{x}) = \beta_0^* + \boldsymbol{\beta}^* \mathbf{x} = 0$.

Any hyperplane can be expressed in an uncountable number of ways; the MMH for which $|F(\mathbf{x}^*)| = 1$ for all support vectors $\mathbf{x}$ provides a **canonical representation**).

[58] The tree is not unique, obviously, but any other tree with separators parallel to the axes will only be marginally better, at best.

[59] To be sure, we could create an intricate decision tree with $> 2^2 = 4$ separating lines, but that is undesirable for a well-fitted tree.

[60] Perfect separation would lead to overfitting.

[61] $F(\mathbf{x})$ for points on $H_{\boldsymbol{\beta},\beta_0}$.

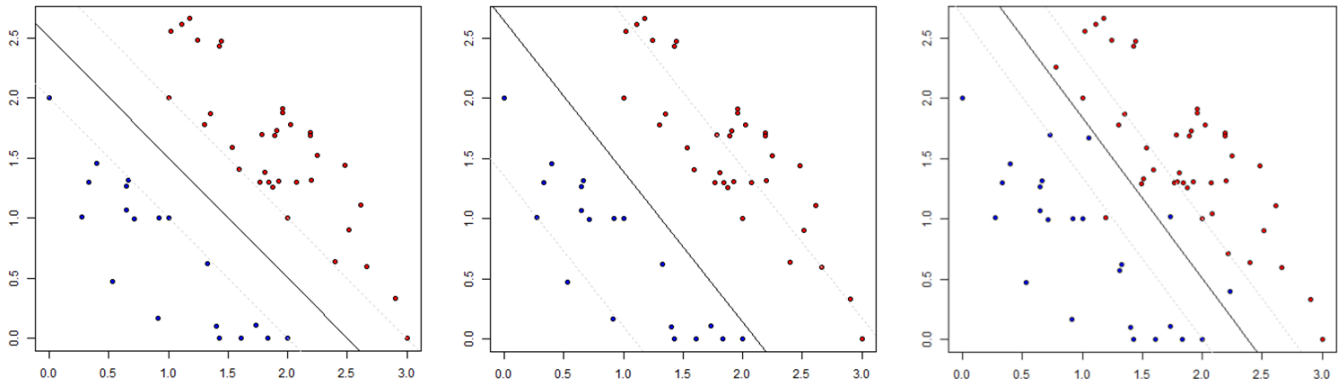
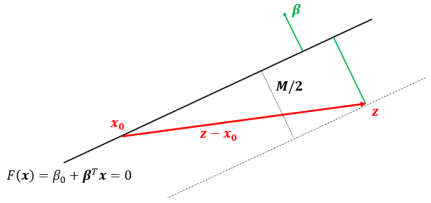


Figure 24. Hard margin for a linearly separable classifier (left); soft margin for a linearly separable classifier (middle); soft margin for a non-linearly separable classifier (right).

From geometry, we know that the distance from the canonical maximal margin hyperplane $H_{\boldsymbol{\beta},\beta_0}$ to any point $\mathbf{z}$ can be computed using vector projections. Let $\mathbf{x}_0$ be a point on MMH, i.e., $F(\mathbf{x}_0) = \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_0 = 0$, as shown below:



In particular, note that $\beta_0 = -\boldsymbol{\beta}^\top \mathbf{x}_0$. Then,

$$\frac{M}{2} = \text{dist}\big(\mathbf{z}, H_{\boldsymbol{\beta},\beta_0}\big) = \big\|\text{proj}_{\boldsymbol{\beta}}(\mathbf{z}-\mathbf{x}_0)\big\| = \left\|\frac{\boldsymbol{\beta}^\top(\mathbf{z}-\mathbf{x}_0)}{\|\boldsymbol{\beta}\|^2}\boldsymbol{\beta}\right\|$$
$$= \frac{|\boldsymbol{\beta}^\top(\mathbf{z}-\mathbf{x}_0)|}{\|\boldsymbol{\beta}\|^2}\|\boldsymbol{\beta}\| = \frac{|\boldsymbol{\beta}^\top\mathbf{z}-\boldsymbol{\beta}^\top\mathbf{x}_0|}{\|\boldsymbol{\beta}\|} = \frac{|F(\mathbf{z})|}{\|\boldsymbol{\beta}\|}.$$

If $\mathbf{z}$ is a support vector, then $F(\mathbf{z}) = 1$, then

$$\frac{M}{2} = \text{dist}\big(\mathbf{z}, H_{\boldsymbol{\beta},\beta_0}\big) = \frac{1}{\|\boldsymbol{\beta}\|}.$$

Maximizing the margin M is thus equivalent to minimizing to minimizing $\frac{\|\boldsymbol{\beta}\|}{2}$, and, since the square function is monotonic,

$$\underset{(\boldsymbol{\beta},\beta_0)}{\text{argmax}}\{M \mid y_i(\beta_0 + \beta^\top \mathbf{x}_i),\ \forall \mathbf{x}_i \in \text{Tr}\}$$

is equivalent to

$$\underset{(\boldsymbol{\beta},\beta_0)}{\text{argmin}}\left\{\frac{1}{2}\|\boldsymbol{\beta}\|^2 \;\middle|\; y_i(\beta_0 + \beta^\top \mathbf{x}_i),\ \forall \mathbf{x}_i \in \text{Tr}\right\}.$$

This constrained quadratic problem (QP) can be solved by Lagrange multipliers (in implementations, it is solved numerically), but a key observation is that it is possible to rewrite

$$\boldsymbol{\beta} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i, \quad \text{with} \quad \sum_{i=1}^N \alpha_i y_i = 0$$

thanks to the **representer theorem**.[62]

[62] Technically speaking we do not need to invoke the representer theorem in the linear separable case. At any rate, the result is out-of-scope for this document.

The original QP becomes

$$\underset{(\boldsymbol{\beta},\beta_0)}{\text{argmin}}\left\{\frac{1}{2}\sum_{i,j=1}^N \alpha_i\alpha_j\mathbf{x}_i^\top\mathbf{x}_j - \sum_{i=1}^N \alpha_i \;\middle|\; \sum_{i=1}^N \alpha_i y_i = 0,\ \forall \mathbf{x}_i, \mathbf{x}_j \in \text{Tr}\right\}.$$

Ultimately, it can be shown that all but L of the coefficients α_i are 0, typically, $L \ll N$. The **support vectors** are those training observations $\mathbf{x}_{i_k}$, $k = 1, \ldots, L$, for which $\alpha_{i_k} \neq 0$.

The **decision function** is defined by

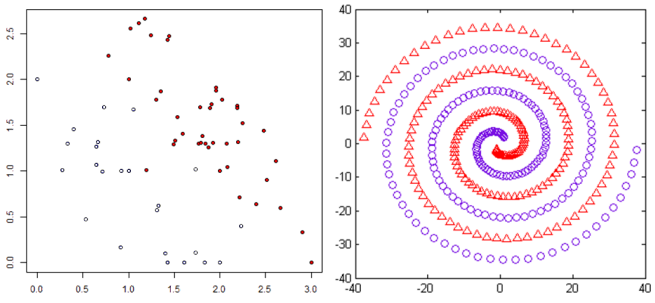
$$T(\mathbf{x}; \boldsymbol{\alpha}) = \sum_{k=1}^L \alpha_{i_k} y_{i_k} \mathbf{x}_{i_k}^\top \mathbf{x} + \beta_0,$$

scaled so that $T(\mathbf{x}_{i_k}; \alpha) = y_{i_k} = \pm 1$ for each support vector $\mathbf{x}_{i_k}$.

The **class assignment** for any $\mathbf{x} \in \text{Te}$ is thus

$$\text{class}(\mathbf{x}) = \begin{cases} +1 & \text{if } T(\mathbf{x}; \alpha) \geq 0 \\ -1 & \text{if } T(\mathbf{x}; \alpha) < 0 \end{cases}$$

In practice (especially when $N < p$), the data is rarely linearly separable into distinct classes (as below, for instance).



Additionally, even when the classes are linearly separable, the data may be **noisy**, which could lead to overfitting, with technically optimal but practically sub-optimal maximal margin solutions (see [21] for examples).

In applications, **support vector classifiers** optimize instead a **soft margin**, one for which some misclassifications are permitted (as in Figure 24).

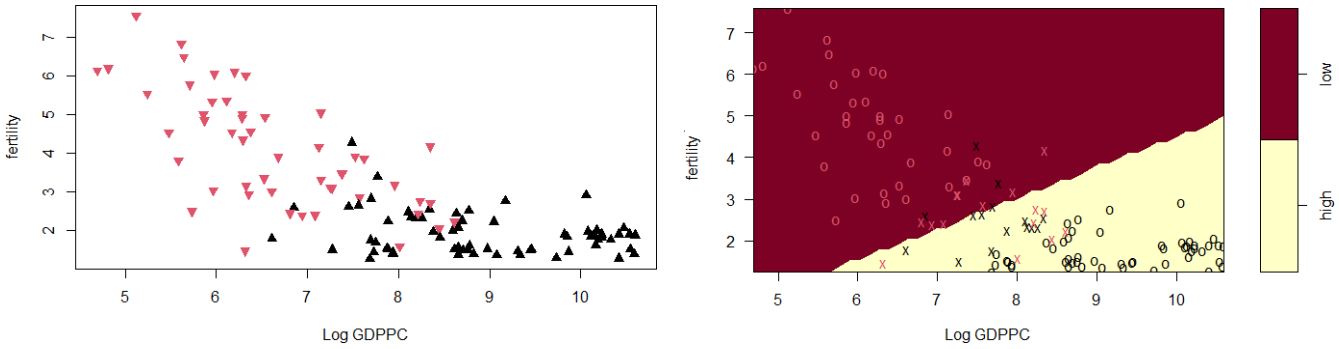


Figure 25. Plot of LE against fertility and log GDP per capita (left), with SVM decision boundary (right).

The soft margin problem can be written as

$$\underset{(\boldsymbol{\beta},\beta_0)}{\operatorname{argmin}} \left\{ \frac{1}{2}\boldsymbol{\beta}^\top\boldsymbol{\beta} \;\middle|\; y_i(\beta_0+\boldsymbol{\beta}\mathbf{x}_i) \geq 1-\varepsilon_i,\ \varepsilon_i \geq 0, \forall \mathbf{x}_i \in \text{Tr},\ \|\boldsymbol{\varepsilon}\| < C \right\},$$

where C is a (budget) **tuning parameter**, $\boldsymbol{\varepsilon}$ is a vector of slack variables, canonically scaled so that $|F(\mathbf{x}^*)| = |\beta_0 + \boldsymbol{\beta}^\top\mathbf{x}^*| = 1$ for any eventual support vector $\mathbf{x}^*$.

Such a model offers greater robustness against unusual observations, while still classifying most training observations correctly:

- if $\varepsilon_i = 0$, then $\mathbf{x}_i \in$ Tr is correctly classified – it falls on the correct side of the hyperplane, and outside the maximum margin;
- if $0 < \varepsilon_i < 1$, then $\mathbf{x}_i \in Tr$ is still correctly classified (falling on the correct side of the hyperplane), but it falls within the margin;
- if $\varepsilon_i \geq 1$, it is incorrectly classified.

If $C = 0$, then no violations are allowed ($\|\boldsymbol{\varepsilon}\| = 0$) and the problem reduces to the hard margin SVM classifier; a solution may not even exist if the data is not linearly separable.

If $C > 0$ is an integer, no more than C training observations can be misclassified; indeed, if $i_1, \ldots, i_C$ are the misclassified indices, then $\varepsilon_{i_1}, \ldots, \varepsilon_{i_C} \geq 1$ and

$$C \geq \sum_{i=1}^{N} \varepsilon_i \geq \sum_{i=1}^{C} \varepsilon_i \geq C.$$

As C increases, tolerance for violations also increases, as does the width of the soft margin; C plays the role of a **regularization parameter**, and is usually selected *via* cross-validation.

Low values of C are associated with harder margins, which leads to low bias but high variance (a small change in the data could create qualitatively different margins); large values of C are associated with wider (softer) margins, leading to more potential misclassifications and higher bias, but also lower variance as small changes in the data are unlikely to change the margin significantly.

We can build a classifier through the representer theorem formulation as before, the only difference being that the **decision function** $T_C(\mathbf{x};\boldsymbol{\alpha})$ is scaled so that $|T(\mathbf{x}_{i_k};\boldsymbol{\alpha})| \geq 1-\varepsilon_{i_k}$ for every support vector $\mathbf{x}_{i_k}$.

It is difficult to determine what the value of the regularization parameter C should be at first glance; an optimal value can be obtained *via* a **tuning process**, which tries out various values and identifies the one that produces an optimal model.

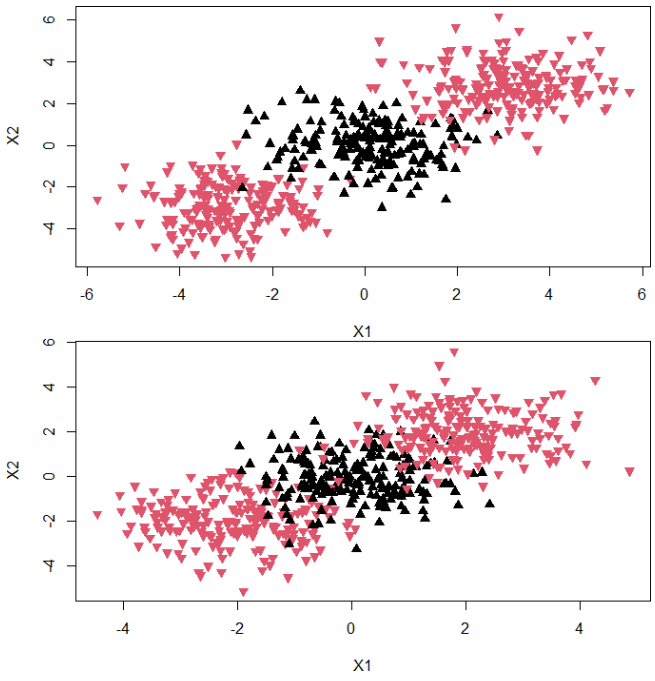
Example We train a SVM with $C = 0.1$ (obtained *via* a tuning procedure for C) for the 2011 Gapminder dataset to predict the life expectancy class in terms of the fertility rate and the logarithm of GDP per capita; $N = 116$ observations are used in the training set, the rest were set aside as the test set. The training plot and the SVM decision boundary are shown in Figure 25; the confusion matrix of the model on the test set is

LE		prediction	
		high	low
actual	high	17	1
	low	10	22

It is not a perfectly accurate model, but it is certainly acceptable given the class overlap in Tr.

7.2.1 Nonlinear Boundaries

If the boundaries between two classes is linear, the SVM classifier of the previous section is a natural way to attempt to separate the classes. In practice, however, the classes are rarely so separated.



In both the hard and the soft margin support vector classifiers, the function to optimize thakes the form

$$\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j\mathbf{x}_i^\top\mathbf{x}_j-\sum_{i=1}^{N}\alpha_i,$$

and the decision function, the form

$$T(\mathbf{x};\boldsymbol{\alpha})=\sum_{k=1}^{L}\alpha_{i_k}y_{i_k}\mathbf{x}_{i_k}^\top\mathbf{x}+\beta_0.$$

In practice, however, we do not actually need to know the support vectors $\mathbf{x}_{i_k}$ (or even the observations $\mathbf{x}_i$, for that matter) in order to compute the decision function values – it is sufficient to have access to the **inner products** $\mathbf{x}_i^\top\mathbf{x}_j$ or $\mathbf{x}_{i_k}^\top\mathbf{x}$, which are usually denoted by

$$\langle\mathbf{x}_{i_k},\mathbf{x}\rangle \quad \text{or} \quad \langle\mathbf{x}_i,\mathbf{x}_j\rangle.$$

The objective function and the decision function can thus be written as

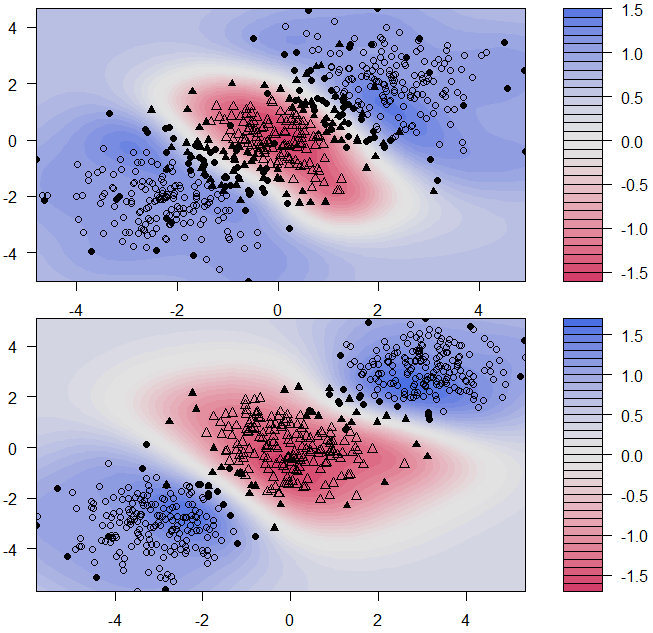
$$\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j\langle\mathbf{x}_i,\mathbf{x}_j\rangle-\sum_{i=1}^{N}\alpha_i,\quad T(\mathbf{x};\boldsymbol{\alpha})=\sum_{k=1}^{L}\alpha_{i_k}y_{i_k}\langle\mathbf{x}_{i_k},\mathbf{x}\rangle+\beta_0.$$

This seemingly innocuous remark opens the door to the **kernel approach**; we could conceivable replace the inner products $\langle\mathbf{x},\mathbf{w}\rangle$ by generalized inner products $K(\mathbf{x},\mathbf{w})$, which provide a measure of similarity between the observations $\mathbf{x}$ and $\mathbf{w}$.

Formally, a **kernel** is a symmetric positive (semi-)definite operator $K:\mathbb{R}^p\times\mathbb{R}^p\to\mathbb{R}_0^+$.[63] Common statistical learning kernels include:

- **linear** – $K(\mathbf{x},\mathbf{w})=\mathbf{x}^\top\mathbf{w}$;
- **polynomial of degree** d – $K_d(\mathbf{x},\mathbf{w})=(1+\mathbf{x}^\top\mathbf{w})^d$;
- **Gaussian** (or radial) – $K_\gamma(\mathbf{x},\mathbf{w})=\exp(-\gamma\|\mathbf{x}-\mathbf{w}\|_2^2)$, $\gamma>0$;
- **sigmoid** – $K_{\kappa,\delta}(\mathbf{x},\mathbf{w})=\tanh(\kappa\mathbf{x}^\top\mathbf{w}-\delta)$, for allowable κ,δ.

For instance, a radial kernel SVM with $\gamma=1$, $C=0.5$ yields the following classification on the datasets from the previous page.



[63] By analogy with positive definite square matrices, this means that $\sum_{i,j=1}^{N}c_ic_jK(\mathbf{x}_i,\mathbf{x}_j)\geq 0$ for $\mathbf{x}_i\in\mathbb{R}^p$, $c_j\in\mathbb{N}$.

The principle is the same: the objective function and the decision function are

$$\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_jK(\mathbf{x}_i,\mathbf{x}_j)-\sum_{i=1}^{N}\alpha_i,\quad T(\mathbf{x};\boldsymbol{\alpha})=\sum_{k=1}^{L}\alpha_{i_k}y_{i_k}K(\mathbf{x}_{i_k},\mathbf{x})+\beta_0.$$

For the radial kernel, for instance, if a test observation $\mathbf{x}$ is near a training observation $\mathbf{x}_i$, then $\|\mathbf{x}-\mathbf{x}_i\|_2^2$ is small and $K_\gamma(\mathbf{x},\mathbf{x}_i)\approx 1$; if they are far from one another, then $\|\mathbf{x}-\mathbf{x}_i\|_2^2$ is large and $K_\gamma(\mathbf{x},\mathbf{x}_i)\approx 0$.

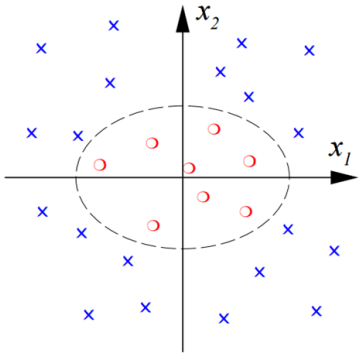
In other words, in the radial kernel framework, only those observations close to a test observation play a role in class prediction.

7.2.2 Kernel Trick

But why even use kernels in the first place?

The linear kernel is easier to interpret and implement, but as we have seen, not all data sets are linearly separable.

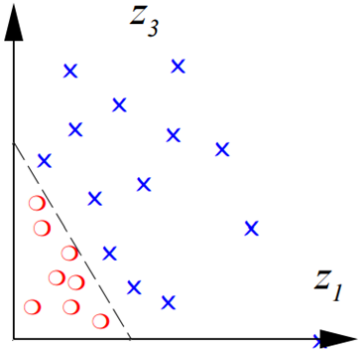
Consider the following classification problem (adapted from an unknown online source).



The optimal margin separating "strip" is obviously not linear.

One way out of this problem is to introduce a **transformation** Φ from the original X—feature space to a higher-dimensional (or at least, of the same dimension) Z—feature space in which the data is linearly separable, and to build a linear SVM on the transformed training observations $\mathbf{z}_i=\Phi(\mathbf{x})_i$.[64]

In this example, we have $\Phi:\mathbb{R}^2\to\mathbb{R}^3$; the projection of the transformation into the Z_1Z_3—plane could be as below.



[64] This might seem to go against reduction strategies used to counter the curse of dimensionality; the added dimensions are needed to "unfurl" the data, so to speak.

The objective function and the decision function take on the form

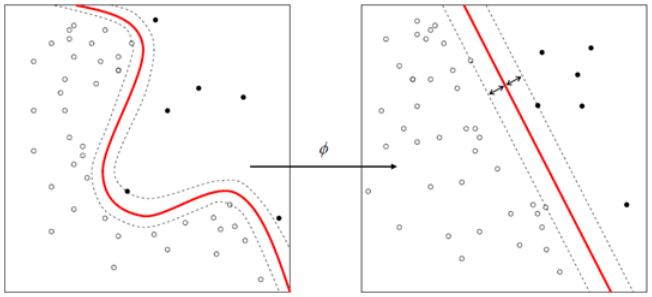
$$\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j\Phi(\mathbf{x}_i)^\top\Phi(\mathbf{x}_j)-\sum_{i=1}^{N}\alpha_i,\ T(\mathbf{x};\boldsymbol{\alpha})=\sum_{k=1}^{L}\alpha_{i_k}y_{i_k}\Phi(\mathbf{x}_{i_k})^\top\Phi(\mathbf{x})+\beta_0,$$

and the linear SVM is built as before (but in $Z-$space, not in $X-$space).

It sounds straightforward, but it takes a fair amount of experience to recognize that one way to separate the data is to use

$$\mathbf{z}=\Phi(\mathbf{x})=(x_1^2,\sqrt{2}x_1x_2,x_2^2).$$

And what transformation should be used for the case below?



The **kernel trick** simply states that Φ can remain unspecified if we replace $\Phi(\mathbf{x})^\top\Phi(\mathbf{w})$ by a "reasonable" (often radial) kernel $K(\mathbf{x},\mathbf{w})$.

General Classification What do we do if the response variable has $K>2$ classes?

In the **one-versus-all** (OVA) approach, we fit K different 2−class SVM decision functions $T_k(\mathbf{x};\boldsymbol{\alpha})$, $k=1,\ldots,K$; in each, one class versus the rest. The test observation $\mathbf{x}^*$ is assigned to the class for which $T_k(\mathbf{x}^*;\boldsymbol{\alpha})$ is largest.

In the **one-versus-one** (OVO) approach, we fit all $\binom{K}{2}$ pairwise 2−class SVM classifiers $\text{class}_{k,\ell}(\mathbf{x})$, for training observations with levels k,ℓ, where $k>\ell=1,\ldots,K-1$. The test observation $\mathbf{x}^*$ is assigned to the class that wins the most pairwise "competitions".

If K is large, $\binom{K}{2}$ might be too large to make OVO computationally efficient; when it is small enough, OVO is the recommended approach.

In practice, it is not always obvious whether one should use SVM, logistic regression, linear discriminant analysis (LDA), decision trees, etc:[65]

- if classes are (nearly) separable, SVM and LDA are usually preferable to logistic regression;
- otherwise, using logistic regression together with a ridge penalty is roughly equivalent to using SVM;
- if the aim is to estimate class membership probabilities, it is preferable to use logistic regression as SVM is not **calibrated**;[66]
- it is possible to use kernels in the logistic regression and LDA frameworks, but at the cost of increased computational complexity.

All in all, it remains crucial to understand that the No Free Lunch theorem remains in effect. There is no magical recipe.

[65] In Section 7.5, we will argue that it is usually preferable to train a variety of models, rather than just the one.

[66] The actual values of $T(\mathbf{x};\boldsymbol{\alpha})$ have no intrinsic meaning, other than their relative ordering.

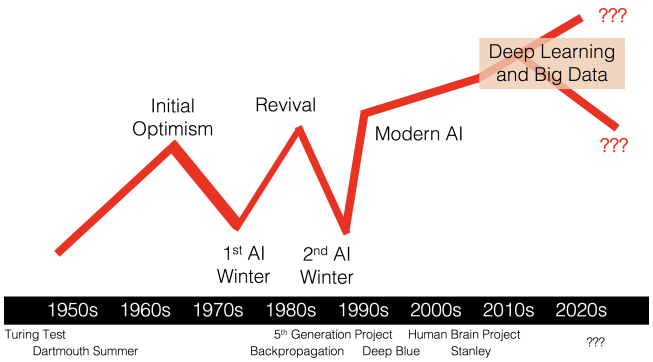


Figure 26. Conceptual timeline of the interest and optimism regarding artificial intelligence (A.I.); important milestones are indicated below the dates.

7.3 Artificial Neural Networks

When practitioners discuss using Artificial Intelligence techniques to solve a problem, the implicit assumption is often (but not always) that a neural (or some other variant of **deep learning**) network will be used, and for good reason: "neural networks blow all previous techniques out of the water in terms of performance" [7]. But there are some skeletons in the closet: "[...] given the existence of adversarial examples, it shows we really don't understand what's going on" [7].

At various times since Alan Turing's seminal 1950 paper (in which he proposed the celebrated **Imitation Game** [29]), complete artificial intelligence has been announced to be "just around the corner" (see A.I winters, Figure 26). With the advent of **deep learning** and Big Data processing, optimism is as high as it's ever been. At the very least, it seems to be the topic **du jour**. Case in point, consider the following four recent headlines:

- "AlphaGo vanquishes world's top Go player, marking A.I.'s superiority over human mind" [South China Morning Post, May 27, 2017]
- "A Japanese A.I. program just wrote a short novel, and it almost won a literary prize" [Digital Trends, March 23, 2016]
- "Elon Musk: Artificial intelligence may spark World War III" [CNET, September 4, 2017]
- "A.I. hype has peaked so what's next?" [TechCrunch, September 30, 2017]

Opinions on the topic are varied – to some commentators, A.I. is a brilliant success, while to others it is a spectacular failure. So what is really going on?

It is far from trivial to identify the **essential qualities and skills of an intelligence**. There have been multiple attempts to solve the problem by building on Turing's original effort. An early argument by Hofstadter [12] is that any intelligence ought to:

- provide flexible responses in various scenarios;
- take advantage of lucky circumstances;
- make sense out of contradictory messages;
- recognize the relative importance of a situation's elements;
- find similarities between different situations;
- draw distinctions between similar situations, and
- come up with new ideas from scratch or by re-arranging previous known concepts.

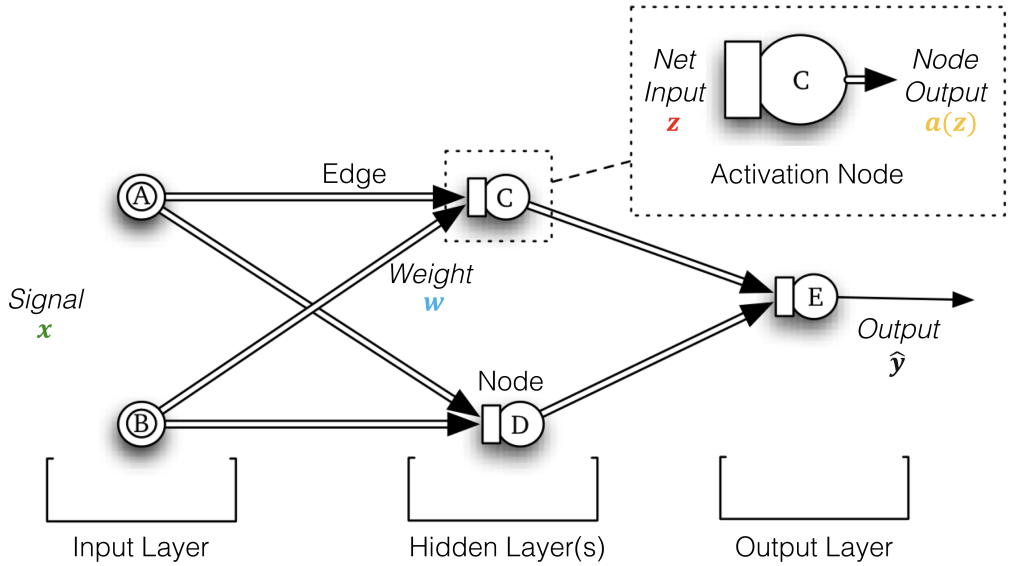


Figure 27. Artificial neural network topology – conceptual example. The number of hidden layers is arbitrary, as is the size of the signal and output vectors.

A.I. research is defined as the study of **intelligent agents** – any device that perceives its environment and takes actions to maximize its chance of success at some task/goal [30]. Examples include

- **expert systems** – TurboTax, WebMD, technical support, insurance claim processing, air traffic control, etc.;
- **decision-making** – Deep Blue, auto-pilot systems, "smart" meters, etc.;
- **natural Language Processing** – machine translation, Siri, named-entity recognition, etc.;
- **recommenders** – Google, Expedia, Facebook, LinkedIn, Netflix, Amazon, etc.;
- **content generators** – music composer, novel writer, animation creator, etc.;
- **classifiers** – facial recognition, object identification, fraud detection, etc.

A trained **artificial neural network** (ANN) is a function that maps inputs to outputs in a useful way: it uses a Swiss-army-knife approach to providing outputs – plenty of options are available in the **architecture**, but it's not always clear which ones should be used. One of the reasons that ANNs are so popular is that the user does not need to decide much about the function or know much about the problem space in advance – ANNs are **quiet models**.

Algorithms allow ANNs to **learn** (i.e. to generate the function and its internal values) automatically; technically, the only requirement is the user's ability to minimize a cost function (which is to say, to be able to solve optimization problems).

The presentation of material in the rest of this section follows [20], which borrows heavily from [5, 9].

Overview The simplest definition of an **artificial neural network** is provided by the inventor of one of the first neuro-computers, Robert Hecht-Nielsen, as:

> "[...] a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs. [4]"

An **artificial neural network** is an interconnected group of nodes, inspired by a simplification of neurons in a brain but on much smaller scales.

Neural networks are typically organized in **layers**. Layers are made up of a number of interconnected **nodes** which contain an **activation function**. A **pattern x** (input, signal) is presented to the network *via* the **input layer**, which communicates with one or more **hidden layers**, where the actual processing is done *via* a system of weighted **connections W** (edges). The hidden layers then link to an **output layer**, which outputs the **predicted response ŷ** (see Figure 27).

Neural Networks Architecture In order to train a neural network, we need the following objects [5]:

- some **input data**,
- a number of **layers**,
- a **model**, and
- a **learning process** (loss function and optimizer).

The object interactions is visualized in Figure 28: a network (model), which is composed of layers that are chained together, maps the input data into predictions.[67]

The loss function then compares these predictions to the targets, producing a loss value: a measure of how well the network's predictions match what was expected. The optimizer uses this loss value to update the network's weights.

Input Data Neural networks start with the **input training data** (and corresponding **targets**) in the form of a **tensor**. Generally speaking, most modern machine learning systems use tensors as their basic data structure. At its core, a tensor is a **container** for data – and it is almost always numerical.

A tensor is defined by three key attributes: its

1. **rank** (number of axes) – for instance, a 3D tensor has three axes, while a matrix (2D tensor) has two axes;

[67] In essence, a neural network is a **function**.

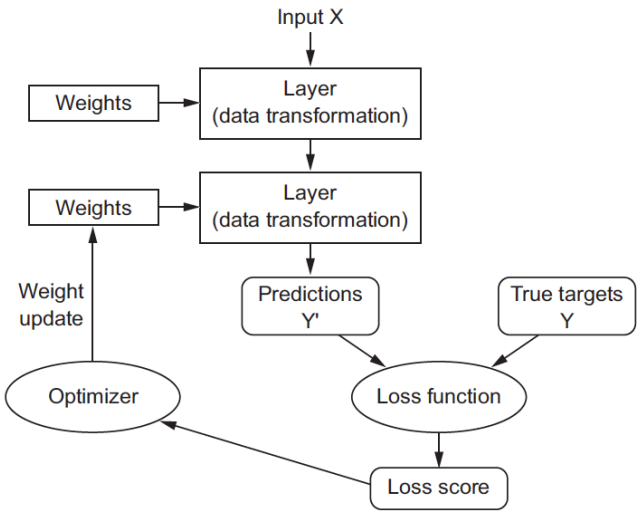


Figure 28. Relationship between the network, layers, loss function, and optimizer [5].

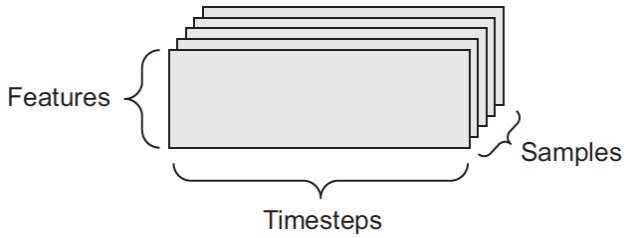


Figure 29. A 3D time series data tensor [5].

2. **shape**, a tuple of integers that describes how many dimensions the tensor has along each axis – for instance, a matrix's shape is described using two elements, such as (3, 5), a 3D tensor's shape has three elements, such as (3,5,5), a vector (1D tensor)'s shape is given by a single element, such as (5), whereas a scalar has an empty shape, ( );
3. its **data type** – for instance, a tensor's type could be `float32`, `uint8`, `float64`, etc.

Data tensors almost always fall into one of the following categories:

- the most common case is **vector data**; in such datasets, each single data point can be encoded as a vector, and a batch of data will be encoded as a matrix or 2D tensor of shape `(#samples, #features)`, or more simply, as an array of vectors where the first axis is the samples axis and the second axis is the features axis;
- **time series or sequence data**, whenever the passage of time is crucial to the observations in the dataset (or the notion of sequence order), can be stored in a 3D tensor with an explicit time axis; each sample can be encoded as a sequence of vectors (a 2D tensor), and a batch of data will be encoded as a 3D tensor of shape `(#samples, #timesteps, #features)`, as in Figure 29;
- **images** typically have three dimensions: height, width, and colour depth;[68] a batch of image data could thus be stored in a 4D tensor of shape `(#samples, #height, #width, #channels)`, as in Figure 30;
- **video data** is one of the few types of real-world data for which 5D tensors are needed – a video can be understood as a sequence of frames, each frame being a colour image; a sequence of frames can be stored in a 4D tensor `(#frames, #height, #width, #channels)`, and so a batch of different videos can be stored in a 5D tensor of shape `(#samples, #frames,#height,#width,#channels)`.

[68] Although grayscale images have only a single colour channel and could thus be stored in 2D tensors, by convention image tensors are always 3D, with a one-dimensional colour channel for grayscale images.

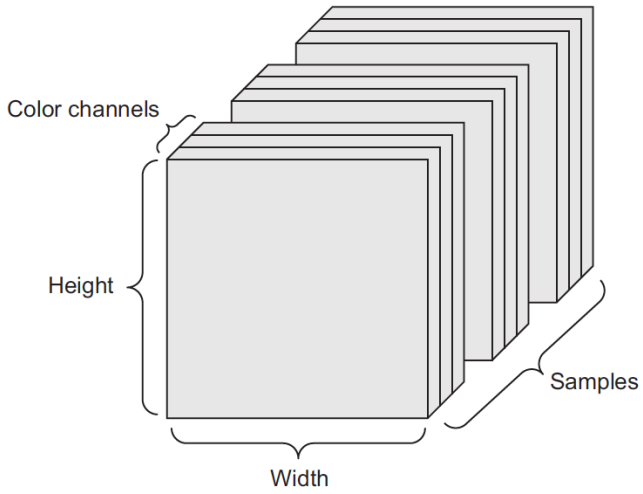


Figure 30. A 4D image data tensor [5].

Layers The core building block of neural networks is the **layer**, a data-processing module that is, in a sense, a **filter** for data: some data goes into the layer and comes out in a more useful form.

Specifically, layers extract **representations** out of the data fed into them – hopefully, representations that are **more meaningful** for the problem at hand. A layer takes as input 1+ tensors and outputs 1+ tensors.

Different layers are appropriate for different tensor formats and different types of data processing. For instance, simple vector data, stored in 2D tensors, is often processed by **densely connected** layers, also called **fully connected** or **dense** layers (the `Dense` class in Keras).

Sequence data, stored in 3D tensors, is typically processed by **recurrent** layers such as an `LSTM` layer. Image data, stored in 4D tensors, is usually processed by 2D **convolution** layers (`Conv2D`).

Most of deep learning consists of chaining together simple layers that will implement a form of **progressive data distillation**. However, to build deep learning models in tensor-based modules like Keras, it is important to clip together **compatible** layers to form useful data-transformation pipelines.

The notion of **layer compatibility** refers specifically to the fact that every layer can only accept input tensors of a certain shape and return output tensors of a certain shape.

Model: Networks of Layers A deep learning model is essentially a **data processing sieve**, made of a succession of increasingly refined data filters – the **layers**. The most common example of a model is a linear stack of layers, mapping a single input to a single output. Other network topologies include: **two-branch networks**, **multihead networks**, and **inception blocks**.

The topology of a network defines a **hypothesis space**. Since machine learning is basically

> "[...] searching for useful representations of some input data, within a predefined space of possibilities, using guidance from a feedback signal [5],"

by choosing a network topology, we constrain the space of possibilities (hypothesis space) to a specific series of tensor operations, mapping input data to output data. From a deep learning perspective, what we are searching for is a good set of values for the weight tensors involved in these tensor operations.

Picking the right network architecture is more an art than a science; and although there are some best practices and principles we can rely on, practical experience is the main factor in becoming a proper neural network architect.

Learning Process: Loss Function and Optimizer After a network architecture has been selected, two other objects need to be chosen:

- the **(objective) loss function** is the quantity that is minimized during training – it represents a measure of success for the task at hand, and
- the **optimizer** determines how the network is updated based on the loss function.

In this context, **learning** means finding a combination of model parameters that minimizes the **loss function** for a given set of training data observations and their corresponding targets. Learning happens by drawing random batches of data samples and their targets, and computing the **gradient** of the network parameters with respect to the **loss** on the batch. The network parameters are then updated by a small amount (the magnitude of the move is defined by the learning rate) in the opposite direction from the gradient.

The entire learning process is made possible by the fact that under a network disguise, neural networks are chains of differentiable tensor operations, to which it is possible to apply the chain rule of differentiation to find the gradient function mapping the current parameters and current batch of data to a gradient value.

Choosing the right objective function for a given problem is extremely important: the network is ruthless when it comes to lowering its loss function, and it will take any shortcut it can to achieve that objective. If the latter does not fully correlate with success for the task at hand, the network may face unintended side effects.

Simple guidelines exist to help analysts select an appropriate loss function for common problems such as classification, regression, and sequence prediction: use

- **binary cross entropy** for a binary classification;
- **categorical cross entropy** for a n–ary classification;
- **mean squared error** for a regression;
- **connectionist temporal classification** (CTC) for sequence-learning, etc.

In most cases, one of these will do the trick – only when analysts are working on truly new research problems do they have to develop their own objective functions.

Let us first illustrate the principles underlying ANNs with a simple example.

We have seen that ANNs are formed from an **input layer** from which the **signal vector x** is inputted, an **output layer** which produces an **output vector ŷ**, and any number of **hidden layers**; each layer consists of a number of **nodes** which are connected to the nodes of other layers *via* **directed edges** with associated **weights w** (see Figure 27). Nodes from the hidden and output layers are typically **activation nodes** – the output $\mathbf{a}(z)$ is some function of the input z. Signals propagate through the ANN using simple arithmetic, once a set of weights $\mathbf{w}$ and activation functions $\mathbf{a}(\cdot)$ have been selected (see Figure 31). In a nutshell, at each node, the neural net computes a weighted sum of inputs, applies an activation function, and sends a signal. This is repeated until the various signals reach the final output nodes.

That part is easy – given a signal, an ANN can produce an output, as long as the weights are specified. Matrix notation can simplify the expression for the output $\hat{y}$ in terms of the signal x, weights w, and activation function $a(\cdot)$.

For instance, consider the network of Figure 27; if $a(z) = (1 + \exp(-z))^{-1}$, the network topology can be re-written as:

- **input layer with p nodes**

$$\mathbf{X}_{N\times p} = \mathbf{X}_{n\times 2} = \begin{bmatrix} x_{A,1} & x_{B,1} \\ \vdots & \vdots \\ x_{A,N} & x_{B,N} \end{bmatrix};$$

- **weights from input layer to hidden layer with M nodes**

$$\mathbf{W}^{(1)}_{p\times M} = \mathbf{W}^{(1)}_{2\times 2} = \begin{bmatrix} w_{AC} & w_{AD} \\ w_{BC} & w_{BD} \end{bmatrix};$$

- **hidden layer with M nodes**

$$\mathbf{Z}^{(2)}_{N\times M} = \mathbf{Z}^{(2)}_{N\times 2} = \begin{bmatrix} z_{C,1} & z_{D,1} \\ \vdots & \vdots \\ z_{C,N} & z_{D,N} \end{bmatrix} = \mathbf{X}\mathbf{W}^{(1)};$$

- **activation function on hidden layer**

$$\mathbf{a}^{(2)} = \begin{bmatrix} (1+\exp(-z_{C,1}))^{-1} & (1+\exp(-z_{D,1}))^{-1} \\ \vdots & \vdots \\ (1+\exp(-z_{C,N}))^{-1} & (1+\exp(-z_{D,N}))^{-1} \end{bmatrix} = g(\mathbf{Z}^{(2)});$$

- **weights from hidden layer with M nodes to output layer with K nodes**

$$\mathbf{W}^{(2)}_{M\times K} = \mathbf{W}^{(2)}_{2\times 1} = \begin{bmatrix} w_{CE} \\ w_{DE} \end{bmatrix};$$

- **output layer with K nodes**

$$\mathbf{Z}^{(3)}_{N\times K} = \mathbf{Z}^{(3)}_{N\times 1} = \begin{bmatrix} z_{E,1} \\ \vdots \\ z_{E,N} \end{bmatrix} = \mathbf{a}^{(2)}\mathbf{W}^{(2)};$$

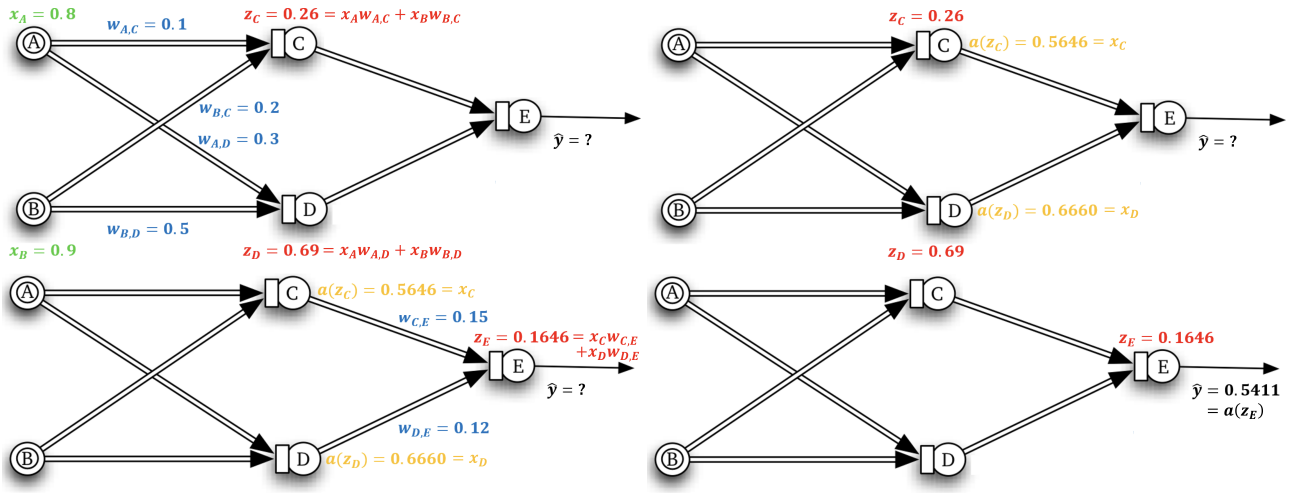


Figure 31. Signal propagating forward through an ANN; weights (in blue) and activation functions (in yellow) are given; inputs (in green), output (in black).

- **activation function on output layer**

$$\hat{\mathbf{y}} = \mathbf{a}^{(3)} = \begin{bmatrix} (1+\exp(-z_{E,1}))^{-1} \\ \vdots \\ (1+\exp(-z_{E,N}))^{-1} \end{bmatrix} = g(\mathbf{Z}^{(3)});$$

The problem is that unless the weights are judiciously selected, the output that is produced is unlikely to have anything to do with the desired output. For supervised learning tasks (i.e. when an ANN attempts to emulate the results of training examples), there has got to be some method to optimise the choice of the weights against an error function

$$R(\mathbf{W}) = \sum_{i=1}^{N}\sum_{k=1}^{K}(\hat{y}_{ik}(\mathbf{W}) - y_{ik})^2 \text{ or } R(W) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \ln \hat{y}_{ik}(\mathbf{W})$$

(for value estimation and classification, respectively), where N is the number of observations in the training set, K is the number of output nodes in the ANN, y_{ik} is the known value or class label for the k^{th} output of the i^{th} observation in the training set.

Enter **backpropagation**, which is simply an application of the chain rule to $R(\mathbf{W})$.Under reasonable regularity condition, the desired **minimizer** $\mathbf{W}^*$ satisfies $\nabla R(\mathbf{W}^*) = 0$ and is found using **numerical gradient descent**.

Gradient-Based Optimization Initially, the weight matrix $\mathbf{W}$ is filled with small random values (a step called **random initialization**). The weights are then gradually **trained** (or learned), based on a **feedback signal**. This occurs within a **training loop**, which works as follows:

1. draw a batch of training samples $\mathbf{x}$ and corresponding targets $\mathbf{y}$;
2. run the network on $\mathbf{x}$ (the **forward pass**) to obtain predictions $\hat{\mathbf{y}}$;
3. compute the loss of the network on the batch, a measure of the mismatch between $\hat{\mathbf{y}}$ and $\mathbf{y}$;
4. update all weights of the network in a way that slightly reduces the loss on this batch.

Repeat these steps in a loop, as often as necessary. Hopefully, the process will eventually converge on a network with a very low loss on the training data, which is to say that there will be a low mismatch between the predictions $\hat{y}$ and the target y. In the vernacular, we say that the ANN has **learned** to map its inputs to correct targets.

Step 1 is easy enough. Steps 2 and 3 are simply the application of a handful of tensor operations (or matrix multiplication, as above). Step 4 is more difficult: how do we update the network's weights? Given an individual weight coefficient in the network, how can we compute whether the coefficient should be increased or decreased, and by how much?

One solution would be to successively minimize the objective function along coordinate directions to find the minimum of a function. This algorithm is called **coordinate descent** and at each iteration determines a coordinate, then minimizes over the corresponding hyperplane while fixing all other coordinates [5].

It is based on the idea that minimization can be achieved by minimizing along one direction at a time. Coordinate descent is useful in situations where the objective function is not **differentiable**, as is the case for most regularized regression models, say.

But this approach would be inefficient in deep learning networks, where there is a large collection of individual weights to update. A smarter approach is use the fact that all operations used to propagate a signal in the network are differentiable, and compute the **gradient** of the objective function (loss) with regard to the network's coefficients. Following a long-standing principle of calculus, we can decrease the objective function by updating the coefficients in the **opposite direction to the gradient**.[69]

For an input vector $\mathbf{X}$, a weight matrix $\mathbf{W}$, a target $\mathbf{Y}$, and a loss function L, we predict a target candidate $\hat{\mathbf{Y}}(\mathbf{W})$, and compute the loss when approximating $\mathbf{Y}$ by $\hat{\mathbf{Y}}(\mathbf{W})$. If $\mathbf{X}$ and $\mathbf{Y}$ are fixed, the loss function maps weights $\mathbf{W}$ to loss values: $f(\mathbf{W}) = L(\hat{\mathbf{Y}}(\mathbf{W}), \mathbf{Y})$.

[69] The gradient is the derivative of a tensor operation; it generalizes the notion of the derivative to functions of multidimensional inputs.

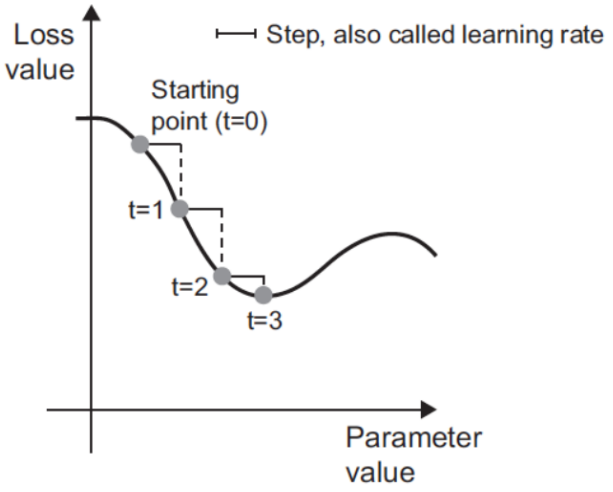


Figure 32. SGD with one parameter [5].

In much the same way that the derivative of a function $f(x)$ of a single variable at a point x_0 is the slope of the tangent at f at x_0, the gradient $\nabla f(\mathbf{W}_0)$ is the tensor describing the **curvature** of $f(\mathbf{W})$ around $\mathbf{W}_0$. As is the case with the derivative, we can reduce $f(\mathbf{W})$ by moving $\mathbf{W}_0$ to

$$\mathbf{W}_1 = \mathbf{W}_0 - s\nabla f(\mathbf{W}_0),$$

where s is the **learning rate**, a small scalar needed to approximate the curvature of the hypersurface close to $\mathbf{W}_0$.

Stochastic Gradient Descent When dealing with ANNs, we can take advantage of the differentiability of the gradient by finding its **critical points** $\nabla f(\mathbf{W}) = 0$ analytically.

If the neural network contains Q edges, this requires solving a polynomial equation in Q variables. However, real-world ANNs often have over a few thousand such connections (if not more), the analytical approach is not reasonable.

Instead, we modify the parameters slightly based on the current loss value on a random batch of data. Since we are dealing with a differentiable function, we can use a **mini-batch stochastic gradient descent** (minibatch SGD) to update the weights, simply by modifying Step 4 of the gradient descent algorithm as follows:

4a. compute the gradient of the loss with regard to the weights (the **backward pass**);

4b. update the weights "a little" in the direction opposite the gradient.

Figure 32 illustrates how SGD works when the network only has the one parameter to learn, with a single training sample. We automatically see why it is important to choose a reasonable learning rate (the step size); too small a value leads to either slow convergence or running the risk of staying stuck at some local minimum; too large a value may send the descent to essentially random locations on the curve and overshooting the global minimum altogether.

SGD Challenges The main issue with minibatch SGD is that "good" convergence rates are not guaranteed, but there are other challenges as well:

- selecting a reasonable learning rate can be difficult. Too small a rate leads to painfully slow convergence, too large a rate can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge [5];
- the same learning rate applies to all parameter updates, which might not be ideal when the data is sparse;
- a key challenge is in minimizing highly non-convex loss functions that commonly occur in ANNs and avoiding getting trapped in suboptimal local minima or saddle points. It is hard for SGD to escape these suboptimal local minima and even wors for the saddle points [6].

SGD Variants There are several SGD variants that are commonly used by the deep learning community to overcome the aforementioned challenges. They take into account the previous weight updates when computing the next weight update, rather than simply considering the current value of the gradients. Popular **optimzers** include SGD with momentum, Nesterov accelerated gradient, Adagrad, Adadelta, RMSProp, and many more [25,28].[70]

ANNs can be quite accurate when making predictions – more than other algorithms, if given a proper set up (but this can be hard to achieve). They degrade gracefully, and they often work when other things fail:

- when the relationship between attributes is **complex**;
- when there are a lot of dependencies/**nonlinear relationships**;
- when the inputs are **messy** and highly-connected (images, text and speech), and
- when dealing with non-linear classification.

But they are relatively slow and prone to overfitting (unless they have access to large and diverse training sets), they are notoriously hard to interpret due to their **blackbox** nature, and there is no algorithm in place to select the optimal network topology.

Finally, even when they do perform better than other options, ANNs may not perform that much better due to the **No Free-Lunch** theorems; and they always remain susceptible to various forms of **adversarial attacks**, so they should be used with caution.

7.4 Naïve Bayes Classification

In classical statistics, model parameters such as μ and σ are treated as constants; **Bayesian statistics**, on the other hand assume that **model parameters are random variables**.

Bayes' Theorem lies at the foundation of such statistics:

$$P(H \mid D) = \frac{P(D \mid H) \times P(H)}{P(D)},$$

where H represents the hypothesis and D denotes the observed data, which is sometimes written in shorthand as $P(H \mid D) \propto P(D \mid H) \times P(H)$; in other words, our **degree of belief in a hypothesis should be updated by the evidence provided by the data**.[71] More details are provided in [8].

[70] A beautiful animation (created by A. Radford) compares the performance of different optimization algorithms and shows that the methods usually take different paths to reach the minimum.

[71] Nobody disputes the validity of Bayes' Theorem, and it has proven to be a useful component in various models and algorithms, such as email spam filters, and the following example, but the **use** of Bayesian statistics is controversial in many quarters.

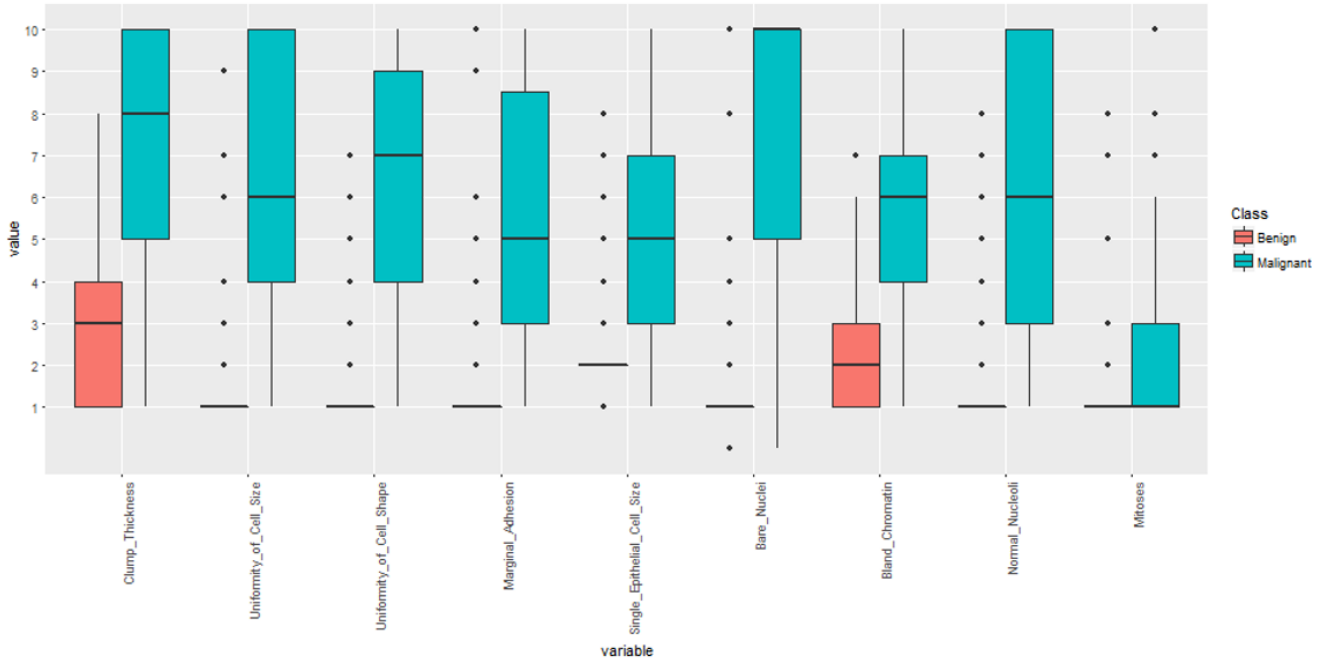


Figure 33. Boxplot visualisation of measurements for benign and malignant tumours.

Suppose we are interested in diagnosing whether a tumour is begin or malignant, based on several measurements obtained from video imaging. Bayes' Theorem can be recast as:

- **posterior:** $P(H \mid D) =$ based on collected data, how likely is a given tumour to be benign (or malignant)?
- **prior:** $P(H) =$ in what proportion are tumours benign (or malignant) in general?
- **likelihood:** $P(D \mid H) =$ knowing a tumour is benign (or malignant), how likely is it that these particular measurements would have been observed?
- **evidence:** $P(D) =$ regardless of a tumour being benign or malignant, what is the chance that a tumour has the observed characteristics?

To answer the above question (that is, to compute the posterior), we will use a **naïve Bayes classifier** (NBC).

7.4.1 Naïve Bayes Classification for Tumour Diagnoses

The NBC procedure is straightforward.

1. **Objective function:** a simple way to determine whether a tumour is benign or malignant is to compare **posterior probabilities** and choose the one with highest probability. That is, we diagnose a tumour as **malignant** if

$$\frac{P(\text{malignant} \mid D)}{P(\text{benign} \mid D)} = \frac{P(D \mid \text{malignant}) \times P(\text{malignant})}{P(D \mid \text{benign}) \times P(\text{benign})} > 1,$$

   and as **benign** otherwise.

2. **Dataset:** the classifier is built on a sample of $N = 458$ tumours with nine measurements, each scored on a scale of 1 to 10. The measurements include items such as *clump thickness* and *bare nuclei*; boxplots of these measurements are shown in Figure 33. We also have undiagnosed cases – an example of an **explanatory signature scores** is given in Figure 34; this is an observation for which a prediction is required.

3. **Assumptions:** we assume that the scores of the measurements in each class are independent of one another (hence the **naive** qualifier); this assumption reduces the likelihood to

$$\begin{aligned} P(D \mid H) &= P(x_1, x_2, \cdots, x_9 \mid H) \\ &= P(x_1 \mid H) \times \cdots P(x_9 \mid H). \end{aligned}$$

4. **Prior distribution:** we can ask subject matter experts to provide a rough estimate for the general ratio of benign to malignant tumours, or use the proportion of benign tumours in the sample as our prior. In situations where we have no knowledge about the distribution of priors, we may simply assume a **non-informative prior** (in this case, the prevalence rates would be the same for both responses).

5. **Computation of likelihoods:** under independence, each measurement is assumed to follow a multinomial distribution (since scores are on $1-10$ scale). Multiplying probabilities from each multinomial distribution (one each for both classes) provides the overall likelihoods for benign and malignant tumours, respectively. The likelihood of the undiagnosed case being a benign tumour is seen to be 9.06×10^{-4}, while the likelihood of being a malignant tumour is 5.85×10^{-11}, based on the multinomial probabilities given in Figure 35

6. **Computation of posterior:** Multiplying the prior probability and likelihood, we get a quantity that is proportional to the respective posterior probabilities. Looking at Figure 36, we conclude that the tumour in the undiagnosed case is **likely benign** (note that we have no measurement on how much more likely it is to be benign than to be malignant – the classifier is **not calibrated**).

In practice, various prior distributions or conditional distributions (for the features) can be used; domain matter expertise can come in handy during these steps.

Obs.	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
1	3	1	1	1	2	2	3	1	1

Figure 34. Scores for an undiagnosed tumour.

	Benign									Malignant							
Score	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
1	30.9%	83.2%	78.5%	81.9%	0.0%	85.4%	33.2%	88.0%	97.6%	0.0%	1.6%	1.1%	12.6%	0.0%	0.0%	0.0%	14.8%	50.8%
2	10.8%	0.0%	10.0%	8.0%	78.5%	4.3%	35.4%	0.0%	0.0%	1.6%	0.0%	0.0%	0.0%	9.8%	4.4%	0.0%	0.0%	43.1%
3	21.5%	0.0%	0.0%	0.0%	0.0%	0.0%	27.4%	0.0%	0.0%	4.9%	10.0%	0.0%	2.6%	16.4%	0.0%	16.4%	0.0%	0.0%
4	14.2%	1.0%	2.0%	1.1%	1.6%	1.5%	0.0%	0.0%	0.0%	0.0%	13.1%	13.7%	10.4%	14.2%	4.4%	12.6%	0.0%	0.0%
5	18.9%	0.0%	0.0%	0.0%	0.0%	2.4%	0.0%	0.0%	0.0%	17.5%	10.9%	12.0%	8.2%	16.9%	0.0%	14.2%	0.0%	0.0%
6	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	10.0%	0.0%	0.0%	15.3%	0.0%	0.0%	0.0%	0.0%
7	0.3%	0.3%	0.0%	0.0%	0.3%	0.0%	1.3%	0.3%	0.0%	0.0%	8.2%	12.0%	0.0%	0.0%	1.1%	25.7%	4.4%	1.1%
8	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	16.0%	12.0%	12.6%	10.4%	8.2%	0.0%	16.4%	0.0%	0.0%
9	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	1.1%	15.3%	0.0%	0.0%	0.0%	0.0%
10	0.0%	0.0%	0.0%	0.3%	0.0%	0.8%	0.0%	0.0%	0.0%	29.5%	26.2%	24.6%	24.0%	12.6%	53.0%	0.0%	26.8%	4.3%
	Likelihood								9.06E-04									5.85E-11

Figure 35. Multinomial probabilities for benign and malignant tumours.

Class	Prior	Likelihood	Posterior	Ratio
Malignant	0.327	5.85×10^{-11}	1.92×10^{-11}	3.15×10^{-8}
Benign	0.673	9.06×10^{-4}	6.09×10^{-4}	

Figure 36. Computation of posterior probabilities in the undiagnosed case of Figure 34.

We end this section with a few **notes and comments:**

- the naive assumption is made out of convenience, as it renders the computation of the likelihood much simpler;
- the variables in a dataset are not typically independent of one another, but NBC still works well with test data (usually) – the method seems to be robust against departure from the independence assumption;
- dependency among variables may change the true posterior values, but the class with maximum posterior probabilities is often unchanged;
- in the classification context, we typically get more insight from independent/correlated data than from correlated data;
- NBC works best for independent cases, but optimality can also be reached when dependency among variables inconsistently support one class over another;
- the choice of a prior may have a great effect on the classification predictions, as can the presence of outlying observations, especially when |Tr is small), and
- a final reminder that, like the SVM models, NBC is **not calibrated** and should not be used to estimate probabilities.

It is debatable whether NBC even counts as a Bayesian method, but it has shown success in various applications.

7.5 Ensemble Learning Methods

In practice, individual learners are often **weak** – they perform better than random guessing would, but not necessarily that much better, or sufficiently so for specific analytical purposes.

In the late 80's, Kearns and Valiant asked the following question: can a set of weak learners be used to create a strong learner? The answer, as it turn out, is yes – *via* **ensemble learning** methods.

As an example, scientists trained 16 pigeons (weak learners, one would assume) to identify pictures of magnified biopsies of possible breast cancers.

On average, each pigeon had an accuracy of about 85%, but when the most popular answer among the group was selected, the accuracy jumped to 99% (see [19] for more details).

7.5.1 Bagging

Bootstrap aggregation (also known as **bagging**) is an extension of bootstrapping. Originally, bootstrapping was used in situations where it is nearly impossible to compute the variance of a quantity of interest by exact means (see Section 4.2).

But it can also be used to improve the performance of various statistical learners, especially those that exhibit **high variance** (such as CART).[72]

Given a learning method, bagging can be used to **reduce the variance** of that method. If $Z_1, \dots, Z_B$ are independent observations with

$$\text{Cov}(Z_i, Z_j) = \begin{cases} \sigma^2 & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

the central limit theorem states that

$$\begin{aligned} \text{Var}(\overline{Z}) &= \text{Var}\left(\frac{Z_1 + \cdots + Z_B}{B}\right) = \frac{1}{B^2}\text{Var}(Z_1 + \cdots + Z_B) \\ &= \frac{1}{B^2}\sum_{i,j=1}^{B}\text{Cov}(Z_i, Z_j) = \frac{1}{B^2}\sum_{k=1}^{B}\text{Var}(Z_i) = \frac{\sigma^2}{B}. \end{aligned}$$

In other words, averaging a set of observations reduces the variance as $\sigma^2 \geq \frac{\sigma^2}{B}$ for all $B \in \mathbb{N}$.

[72] Low variance methods, in comparison, are those for which the results, structure, predictions, etc. remain roughly similar when using different training sets, such as OLS when $N/p \gg 1$, and are less likely to benefit from the use of ensemble learning.

In practice, this conclusion seems, at first, not to be as interesting as originally intended since we do not usually have access to multiple training. However, resampling methods can be used to generate multiple training sets from the original training set Tr.

Let $B > 1$ be an integer. We generate B bootstrapped training sets from Tr by sampling $N = |\text{Tr}|$ observations from Tr, with replacement, to yield

$$\text{Tr}_1, \dots, \text{Tr}_N,$$

and train a model $\hat{f}_i$ (for regression) or $\hat{C}_i$ (for classification) on each Tr_i, $i = 1, \dots, B$; for each $\mathbf{x}^* \in \text{Te}$, we then have B predictions

$$\hat{f}_1(\mathbf{x}^*), \dots, \hat{f}_B(\mathbf{x}^*) \quad \text{(for regression)}$$

or

$$\hat{C}_1(\mathbf{x}^*), \dots, \hat{C}_B(\mathbf{x}^*) \quad \text{(for classification)}.$$

The **bagging prediction** at $\mathbf{x}^* \in \text{Te}$ is the average of all predictions

$$\hat{f}_{\text{Bag}}(\mathbf{x}^*) = \frac{1}{B}\sum_{i=1}^{B} \hat{f}_i(\mathbf{x}^*) \quad \text{(for regression)}$$

or the most frequent prediction

$$\hat{C}_{\text{Bag}}(\mathbf{x}^*) = \text{Mode}\{\hat{C}_1(\mathbf{x}^*), \dots, \hat{C}_B(\mathbf{x}^*)\} \quad \text{(for classification)}.$$

Bagging is particularly helpful in the CART framework; to take full advantage of bagging, however, the trees should be grown **deep**, as their complexity will lead to high variance but low bias (thanks to the bias-variance trade-off).

In practice, the bagged tree predictions would also have low bias, but the variance will be reduced by the bagging process; bagging with 100s/1000s of trees typically produces greatly improved predictions (at the cost of interpretability, however).

Out-of-Bag Error Estimation As is usually the case with supervised models, we will need to estimate the test error for a bagged model. There is an easy way to provide the estimate without relying on cross-validation, which is computationally expensive when N is large.

The jth model is fit to the bootstrapped training set Tr_j, $j = 1, \dots, B$. We can show that, on average, each of the Tr_j contains $\approx 2/3$ distinct observations of Tr, which means that $\approx 1/3$ of the training observations are not used to build the model (we refer to those observations are **out-of-bag (OOB) observations**).

We can then predict the response y_i for the ith observation in Tr using only those models for which $\mathbf{x}_i$ was OOB; there should be about $B/3$ such predictions, and

$$\hat{y}_i = \text{Avg}\{\hat{f}_j(\mathbf{x}_i) \mid \mathbf{x}_i \in \text{OOB}(\text{Tr}_j) = \text{Tr} \setminus \text{Tr}_j\} \quad \text{(for regression)}$$

or

$$\hat{y}_i = \text{Mode}\{\hat{C}_j(\mathbf{x}_i) \mid \mathbf{x}_i \in \text{OOB}(\text{Tr}_j)\} \quad \text{(for classification)}.$$

The OOB MSE (or the OOB misclassification rate) are thus good Te error estimates since none of the predictions are given by models that used the test observations in their training.

Variable Importance Measure Bagging improves the accuracy of stand-alone models, but such an improvement comes at the cost of reduced interpretability, especially in the case of CART: the bagged tree predictions cannot, in general, be expressed with the help of a single tree.

In such a tree, the relative importance of the features is linked to the **hierarchy of splits** (namely, the most "important" variables appear in the earlier splits).

For bagged **regression trees**, a measure such as the total amount in decreased SSRes due to splits over a given predictor, in which we compare SSRes in trees with these splits against SSRes in trees without these splits, averaged over the B bagged trees provides a summary of the importance of each variable (large scores indicate important variables). For bagged **classification trees**, we would replace SSRes with the Gini index, instead.

Another approach might be to weigh the importance of a factor **inversely proportionally** to the level in which it appears (if at all) in each bagging tree and to average over all bagging trees.

For instance, if predictor X_1 appears in the 1st split level of bagged tree 1, the 4th split level of bagged tree 2, and the 3rd split level of bagged tree 5, whereas predictor X_2 appears in the 2nd, 2nd, 3rd, and 5th split levels of bagged trees 2, 3, 4, 5 (resp.), then the relative importance of each predictor over the 5 bagged trees is

$$X_1 : (1 + 1/4 + 0 + 0 + 1/3) \cdot \frac{1}{5} = 19/60 = 0.32$$
$$X_2 : (0 + 1/2 + 1/2 + 1/3 + 1/5) \cdot \frac{1}{5} = 23/75 = 0.31,$$

and the first variable would be nominally more important than the second.

7.5.2 Random Forests

In a bagging procedure, we fit models on various training sets, and we use the central limit theorem, assuming independence of the models, to reduce the variance.

In practice, however, the independence assumption is rarely met: if there are a few strong predictors in Tr, each of the bagged models (built on the bootstrapped training sets Tr_i) is likely to be similar to the others, and the various predictions are unlikely to be un-correlated, so that

$$\text{Var}(\hat{y}_i) \neq \frac{\sigma^2}{B};$$

averaging highly correlated quantities does not reduce the variance significantly (the central limit theorem assumption of independence of observations is necessary).

With a small tweak, however, we can **decorrelate the bagged models**, leading to variance reduction when we average the bagged predictions.

Random forests also build models on B bootstrapped training samples, but each model is built out of a **random subset** of $m \leq p$ predictors.

For decision trees, every time a split is considered, the set of allowable predictors is selected from a random subset of m predictors out of the full p predictors.

By selecting predictors randomly for each model, we lose out on building the best possible model on each training sample, but we also reduce the chance of them being correlated. For a test observations $\mathbf{x}^*$, the B predictions are combined as in bagging to yield the **random forest prediction**.

If $m = p$, random forests reduce to bagged models; in practice we use $m \approx \sqrt{p}$. When the predictors are highly correlated, however, smaller values of m are recommended.

7.5.3 Boosting

Another general approach to improving prediction results for statistical learners involves creating a sequence of models, each improving over the previous model in the series. **Boosting** does not involve bootstrap sampling; instead, it fits models on a hierarchical sequence of **residuals**, but it does so in a **slow manner**.

For regression problems, we proceed as follows:

1. set $\hat{f}(\mathbf{x}) = 0$ and $r_i = y_i$ for all $\mathbf{x}_i \in \text{Tr}$;
2. for $b = 1, 2, \ldots, B$:
   (a) fit a model $\hat{f}^b$ to the training $(\mathbf{X}, \mathbf{r})$;
   (b) update the regression function $\hat{f} := \hat{f} + \lambda \hat{f}^b$;
   (c) update the residuals $r_i := r_i - \lambda \hat{f}^b(\mathbf{x}_i)$ for all $\mathbf{x}_i \in \text{Tr}$;
3. output the boosted model $\hat{f}(\mathbf{x}) = \lambda(\hat{f}^1(\mathbf{x}) + \cdots + \hat{f}^B(\mathbf{x}))$.

In this version of the algorithm, boosting requires three tuning parameters:

- the **number of models** B, which can be selected through cross-validation (boosting can overfit if B is too large);
- the **shrinkage parameter** λ (typically, $0 < \lambda \ll 1$), which controls the **boosting learning rate** (a small λ needs a large B, in general); the optimal λ and B can be found *via* cross-validation, and
- although not explicitly stated, we also need the learning models to reach some **complexity threshold**.

Variants of the boosting algorithm allowing for classification and for varying weights depending on performance regions in predictor space also exist and are quite popular: details on **AdaBoost** and **Gradient Boosting** are available in [17], amongst others (classification boosting is a slightly more complicated affair).

Finally, note that while the No Free Lunch theorems guarantee that no supervised learning algorithm is always best regardless of context/data, the combination of AdaBoost with weak CART learners is seen by many as the best "out-of-the-box" classifier.

References

[1] C. C. Aggarwal and C. K. Reddy, editors. *Data Clustering: Algorithms and Applications*. CRC Press, 2014.

[2] P. Boily and J. Schellinck. Machine Learning 101. *Data Science Report Series*, 2021.

[3] G. E. P. Box. Use and abuse of regression. *Journal of Technometrics*, 8(4):625–629, Nov. 1966.

[4] M. Caudill. Neural networks primer, part 1. *AI Expert*, 2(12):46–52, Dec. 1987.

[5] F. Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017.

[6] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, 2014.

[7] D. Gershgorn. There's a glaring mistake in the way AI looks at the world . 2017.

[8] E. Ghashim and P. Boily. A Soft Introduction to Bayesian Data Analysis. *Data Science Report Series* , 2020.

[9] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press Cambridge, 2016.

[10] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd ed.* Springer, 2008.

[11] T. Hastie, R. Tibshirani, and M. Wainwright. *Statistical learning with sparsity : the lasso and generalizations*. Monographs on statistics and applied probability, no. 143. CRC Press, 2015.

[12] D. Hoftsadter. *Gödel, Escher, Bach: an Eternal Golden Braid*. Basic Books, 1979.

[13] R. Hogg and E. Tanis. *Probability and Statistical Inference*. Pearson/Prentice Hall, 7th edition, 2006.

[14] Interactive visualization to teach about the curse of dimensionality .

[15] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer, 2014.

[16] M. Kutner, C. Nachtsheim, J. Neter, and W. Li. *Applied Linear Statistical Models*. McGraw Hill Irwin, 2004.

[17] O. Leduc and P. Boily. Boosting with AdaBoost and Gradient Boosting . *Data Action Lab Blog*, 2019.

[18] O. Leduc, A. Macfie, A. Maheshwari, M. Pelletier, and P. Boily. Feature Selection and Dimension Reduction . *Data Science Report Series* , 2020.

[19] R. M. Levenson, E. A. Krupinski, V. M. Navarro, and E. A. Wasserman. Pigeons (columba livia) as trainable observers of pathology and radiology breast cancer images. *PLOS ONE*, 10(11):1–21, 11 2015.

[20] S. Patel, R. Pourhasan, and P. Boily. Artificial Neural Networks and Deep Learning with Applications. *Data Science Report Series* , in progress.

[21] F. Provost and T. Fawcett. *Data Science for Business*. O'Reilly, 2015.

[22] D. Robinson. What's the difference between data science, machine learning, and artificial intelligence? *Variance Explained*, Jan 2018.

[23] H. Rosling. *The Health and Wealth of Nations* . Gapminder Foundation, 2012.

[24] H. Rosling, O. Rosling, and A. Rönnlund. *Factfulness: Ten Reasons We're Wrong About The World - And Why Things Are Better Than You Think*. Hodder & Stoughton, 2018.

[25] S. Ruder. An overview of gradient descent optimization algorithms, 2016.

[26] H. Sahai and M. Ageel. *The Analysis of Variance: Fixed, Random and Mixed Models*. Birkhäuser, 2000.

[27] J. Schellinck and P. Boily. Spotlight on Clustering. *Data Science Report Series*, 2022.

[28] R. S. Sutton. Two problems with backpropagation and other steepest-descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum, 1986.

[29] A. Turing. Computing machinery and intelligence. *Mind*, 1950.

[30] Wikipedia. Artificial Intelligence . 2020.

[31] Wikipedia. Binary classification . 2021.

[32] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1997.

[33] D. Wolpert and W. Macready. Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation*, 9(6):721–735, 2005.