Applied Statistics - Notes

260236

March 2024

Preface

Every theory section in these notes has been taken from two sources:

- An Introduction to Statistical Learning [1]
- Applied Multivariate Statistical Analysis (sixth edition). [2]

About:

GitHub repository

Contents

1 Sample Geometry

1.1 The Geometry of the Sample

A single **multivariate observation** is the **collection of measurements on p different variables taken on the same item or trial**. If n **observations** have been obtained, the entire data set can be placed in an $n \times p$ array (or matrix), also called **data frame**:

$$\underset{(n\times p)}{\mathbf{X}} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} \tag{1}$$

Each **row** of $\mathbf{X}$ represents a **multivariate observation**. Since the entire data frame is often one particular realization of what might have been observed, we say that the data frame are a **sample of size n from a p-variate "population"**. The sample then consists of n measurements, each of which has p components.

Look at the matrix, n measurements (rows), each of which has p components (columns). In mathematics, each n row contains p columns and vice versa.

The data frame can be plotted in two different ways:

1. p-dimensional scatter plot, where the rows represent n points in p-dimensional space;

2. Geometrical representation, p vectors in n-dimensional space.

1.1.1 Scatter plot

For the **p-dimensional scatter plot**, the rows of $\mathbf{X}$ represent n points in p-dimensional space:

$$\underset{(n\times p)}{\mathbf{X}} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} = \begin{bmatrix} \mathbf{x}'_1 \\ \mathbf{x}'_2 \\ \vdots \\ \mathbf{x}'_n \end{bmatrix} \begin{array}{l} \leftarrow \text{1st (multivariate) observation} \\ \\ \\ \leftarrow n\text{th (multivariate) observation} \end{array} \tag{2}$$

The row vector $\mathbf{x}'_j$, representing the jth observation, contains the coordinates of a point. The **scatter plot** of n points in p-dimensional space **provides information** on the **locations and variability of the points**.

Note: when p (dimensional space) is greater than 3, the **scatter plot** representation cannot actually be graphed. Yet the consideration of the data as n points in p dimensions provides **insights that are not readily available from algebraic expressions**.

1.1.2 Geometrical representation

The alternative geometrical representation is constructed by considering the data as p **vectors in n-dimensional space**. Here we take the elements of the columns of the data frame to be the coordinates of the vectors:

$$\underset{(n\times p)}{\mathbf{X}} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} = [\mathbf{y}_1 \,|\, \mathbf{y}_2 \,|\, \cdots \,|\, \mathbf{y}_p] \tag{3}$$

Then the **coordinates** of the first point $\mathbf{y}_1 = [x_{11}, x_{21}, \ldots, x_{n1}]$ **are the n measurements** on the first variable.

In general, the ith point $\mathbf{y}_i = [x_{11}, x_{21}, \ldots, x_{n1}]$ is determined by the n-tuple of all measurements on the ith variable.

Geometrical representations usually **facilitate understanding** and lead to further insights. The ability to **relate algebraic expressions to the geometric concepts** of length, angle and volume is therefore **very important**.

1.1.3 Geometrical interpretation of the process of finding a sample mean

Before starting the explanation, you need to understand a few things.

- The length of a vector $\mathbf{x}' = [x_1, x_2, \ldots, x_n]$ with n components is defined by:

  $$L_x = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \tag{4}$$

  Multiplication of a vector $\mathbf{x}$ by a scalar c changes the length:

  $$\begin{aligned} L_{cx} &= \sqrt{c^2 \cdot x_1^2 + c^2 \cdot x_2^2 + \cdots + c^2 \cdot x_n^2} \\ &= |c| \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \\ &= |c|\, L_x \end{aligned}$$

  So, for example, in $n = 2$ dimensions, the vector:

  $$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

  The length of $\mathbf{x}$, written L_x, is defined to be:

  $$L_x = \sqrt{x_1^2 + x_2^2}$$

- Another important concept is **angle**. Consider two vectors in a plane and the angle θ between them: The value θ can be represented as the difference between the angles θ_1 and θ_2 formed by the two vectors and the first coordinate axis. Since, by definition:

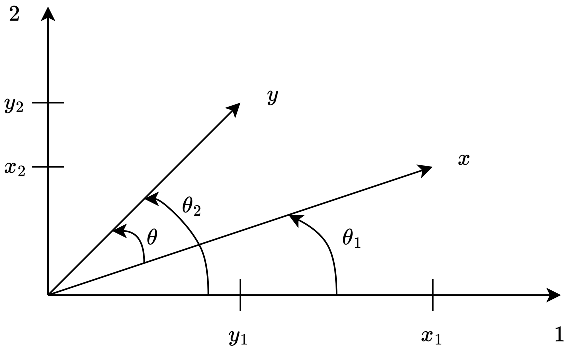


Figure 1: The angle θ between $\mathbf{x}' = [x_1, x_2]$ and $\mathbf{y}' = [y_1, y_2]$.

$$\cos(\theta_1) = \frac{x_1}{L_x} \quad \cos(\theta_2) = \frac{y_1}{L_y}$$
$$\sin(\theta_1) = \frac{x_2}{L_x} \quad \sin(\theta_2) = \frac{y_2}{L_y}$$
$$\cos(\theta) = \cos(\theta_2 - \theta_1) = \cos(\theta_2)\cos(\theta_1) + \sin(\theta_2)\sin(\theta_1)$$

The angle θ between the two vectors $\mathbf{x}' = [x_1, x_2]$ and $\mathbf{y}' = [y_1, y_2]$ is specified by:

$$\cos(\theta) = \cos(\theta_2 - \theta_1) = \left(\frac{y_1}{L_y}\right)\left(\frac{x_1}{L_x}\right) + \left(\frac{y_2}{L_y}\right)\left(\frac{x^2}{L_x}\right) = \frac{x_1y_1 + x_2y_2}{L_xL_y} \tag{5}$$

- With the angle equation 5, it's convenient to introduce the **inner product** of two vectors:

$$\mathbf{x}\mathbf{y}' = x_1y_1 + x_2y_2$$

So let us rewrite:

  - The **length** equation 4:

$$\mathbf{x}'\mathbf{x} = x_1x_1 + x_1x_1 = x_1^2 + x_2^2 \longrightarrow L_x = \sqrt{x_1^2 + x_2^2} \Longrightarrow L_x = \sqrt{\mathbf{x}'\mathbf{x}} \tag{6}$$

  - The **angle** equation 5:

$$\cos(\theta) = \frac{x_1y_1 + x_2y_2}{L_xL_y} \Longrightarrow \cos(\theta) = \frac{\mathbf{x}'\mathbf{y}}{L_xL_y}$$

    And using the rewritten length equation:

$$\cos(\theta) = \frac{\mathbf{x}'\mathbf{y}}{L_xL_y} \Longrightarrow \cos(\theta) = \frac{\mathbf{x}'\mathbf{y}}{\sqrt{\mathbf{x}'\mathbf{x}} \cdot \sqrt{\mathbf{y}'\mathbf{y}}}$$

- The **projection** (or shadow) of a vector $\mathbf{x}$ on a vector $\mathbf{y}$ is:

$$\frac{(\mathbf{x}'\mathbf{y})}{\mathbf{y}'\mathbf{y}}\mathbf{y} = \frac{(\mathbf{x}'\mathbf{y})}{L_y}\frac{1}{L_y}\mathbf{y} \tag{7}$$

  Where the vector $\frac{1}{L_y}\mathbf{y}$ has unit length. The **length of the projection** is:

$$\frac{|\mathbf{x}'\mathbf{y}|}{L_y} = L_x\left|\frac{\mathbf{x}'\mathbf{y}}{L_x L_y}\right| = L_x\left|\cos(\theta)\right| \tag{8}$$

  Where θ is the angle between $\mathbf{x}$ and $\mathbf{y}$:

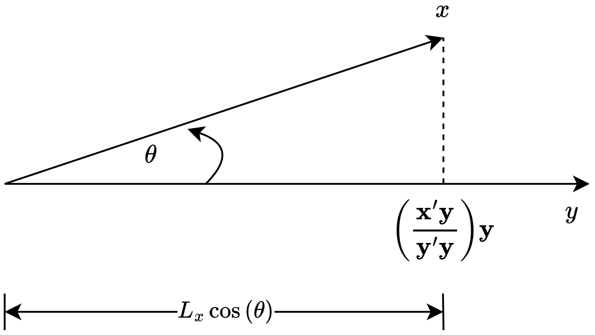

x
θ
y
$\left(\frac{\mathbf{x}'\mathbf{y}}{\mathbf{y}'\mathbf{y}}\right)\mathbf{y}$
$L_x\cos(\theta)$

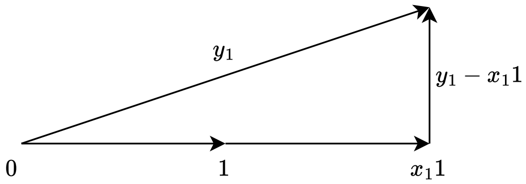
Figure 2: The projection of $\mathbf{x}$ on $\mathbf{y}$.

Start by defining the $n\times 1$ vector $\mathbf{1}_n' = [1,1,\ldots,1]$. The vector $\mathbf{1}$ forms equal angles with each of the n coordinates axes, so the vector $\left(\frac{1}{\sqrt{n}}\right)\mathbf{1}$ has unit length in the equal-angle direction. Consider the vector $\mathbf{y}_i' = [x_{1i}, x_{2i}, \ldots, x_{ni}]$. The projection of $\mathbf{y}_i$ on the unit vector $\left(\frac{1}{\sqrt{n}}\right)\mathbf{1}$ is:

$$\mathbf{y}_i'\left(\frac{1}{\sqrt{n}}\mathbf{1}\right)\frac{1}{\sqrt{n}}\mathbf{1} = \frac{x_{1i}+x_{2i}+\cdots+x_{ni}}{n}\mathbf{1} = \overline{x}_i\mathbf{1} \tag{9}$$

Although it may seem like a complex equation at first glance, it is nothing more than the mean! In fact, the **sample mean** $\overline{\mathbf{x}}_i = \frac{(x_{1i}+x_{2i}+\cdots+x_{ni})}{n} = \frac{\mathbf{y}_i'\mathbf{1}}{n}$ corresponds to the multiple of $\mathbf{1}$ required to give the projection of $\mathbf{y}_i$ onto the line determined by $\mathbf{1}$.

Furthermore, using the projection, you can obtain the **deviation** (**mean corrected**). For each $\mathbf{y}_i$ we have the decomposition:



Where $\overline{x}_i \mathbf{1}$ is perpendicular to $y_i - \overline{x}_i \mathbf{1}$. The **deviation**, or **mean corrected**, vector is:

$$\mathbf{d}_i = \mathbf{y}_i - \overline{x}_i \mathbf{1} = \begin{bmatrix} x_{1i} - \overline{x}_i \\ x_{2i} - \overline{x}_i \\ \vdots \\ x_{ni} - \overline{x}_i \end{bmatrix} \tag{10}$$

The **elements** of $\mathbf{d}_i$ are the **deviations of the measurements on the** $\boldsymbol{i}$**th variable from their sample mean**.

Using the length rewritten with inner product (equation 6) and the deviation (equation 10), we obtain:

$$L_{\mathbf{d}_i}^2 = \mathbf{d}_i' \mathbf{d}_i = \sum_{j=1}^{n} \left(x_{ji} - \overline{x}_i\right)^2 \tag{11}$$

$$\text{(Length of deviation vector)}^2 = \text{sum of squared deviations}$$

From the sample standard deviation, we see that the **squared length is proportional to the variance** of the measurements on the ith variable. Equivalently, the **length is proportional to the standard deviation**. So longer vectors represent more variability than shorter vectors.

Furthermore, for any two deviation vectors $\mathbf{d}_i$ and $\mathbf{d}_k$:

$$\mathbf{d}_i' \mathbf{d}_k = \sum_{j=1}^{n} \left(x_{ji} - \overline{x}_i\right) \left(x_{jk} - \overline{x}_k\right) \tag{12}$$

And with a few mathematical operations, we can get it:

$$r_{ik} = \frac{s_{ik}}{\sqrt{s_{ii}}\sqrt{s_{kk}}} = \cos\left(\theta_{ik}\right) \tag{13}$$

Where the **cosine** of the angle is the **sample correlation coefficient**. Note: s_{ik} is the **sample covariance**:

$$s_{ik} = \frac{1}{n} \sum_{j=1}^{n} \left(x_{ji} - \overline{x}_i\right) \left(x_{jk} - \overline{x}_k\right) \qquad i = 1, 2, \dots, p, \quad k = 1, 2, \dots, p \tag{14}$$

Thus:

- If the two deviation vectors have **nearly the same orientation**, the sample correlation will be close to 1;

- If the two vectors are **nearly perpendicular**, the sample correlation will be approximately zero;
- If the two vectors are oriented in **nearly opposite directions**, the sample correlation will be close to -1.

1.2 Generalized Variance

Before starting the explanation, you need to understand what is a sample variance.

A **sample variance** is defined as:

$$s_k^2 = s_{kk} = \frac{1}{n-1}\sum_{j=1}^{n}(x_{jk}-\overline{x}_k)^2 \qquad k = 1, 2, \dots, p \tag{15}$$

With a single variable, the **sample variance is often used to describe the amount of variation in the measurements on that variable**. When p variables are observed on each unit, the variation is described by the **sample variance-covariance matrix**:

$$\mathbf{S} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1p} \\ s_{21} & s_{22} & \cdots & s_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ s_{p1} & s_{p2} & \cdots & s_{pp} \end{bmatrix} = \left\{ s_{ik} = \frac{1}{n-1}\sum_{j=1}^{n}(x_{ji}-\overline{x}_i)(x_{jk}-\overline{x}_k) \right\} \tag{16}$$

The sample covariance matrix contains p variances and $\frac{1}{2}p(p-1)$ potentially different covariances. Sometimes it's desirable to **assign a single numerical value for the variation expressed by S**. One choice for a value is the determinant of $\mathbf{S}$, which reduces to the usual sample variance of a single characteristic when $p = 1$. This determinant is called the **generalized sample variance**:

$$\text{Generalized sample variance} = \det(\mathbf{S}) = |\mathbf{S}| \tag{17}$$

2 Statistical Learning

2.1 Introduction

Suppose that we observe a quantitative response Y and p different predictors, $X_1, X_2, \ldots, X_p$. We assume that there is some relationship between Y and $X = (X_1, X_2, \ldots, X_p)$, which can be written in the general form:

$$Y = f(X) + \varepsilon \tag{18}$$

Where ε is an error term, which is **independent** of X and has **mean zero**. The function f represents the systematic information that X provides about Y. The **function** f that connects the input variables to the output variable **is in general unknown**.

Example 1

For example, on the left-hand panel of figure 3, a plot `income` versus `years of education` for 30 individuals in the Income data set.

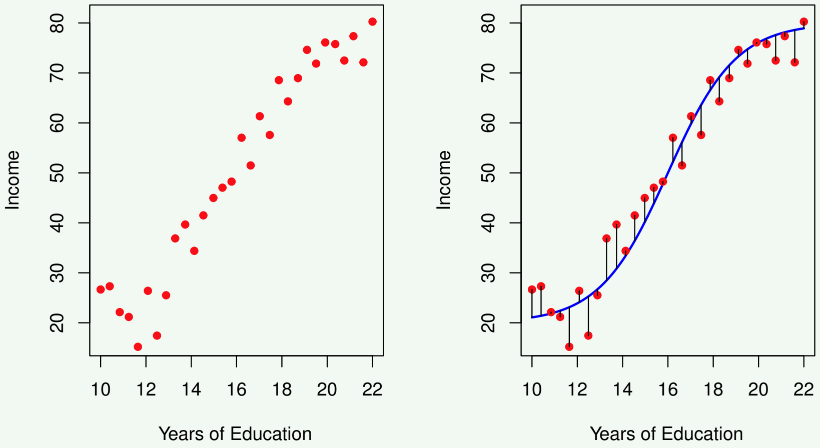


Figure 3: The `Income` data set. [1]

As you can see, the plot suggests that one might be able to predict `income` using `years of education`. Since `Income` is a simulated data set, the function f is known and is shown by the blue curve in the right-hand panel. The **vertical lines** represent the **error terms** ε. We note that some of the 30 observations lie above the blue curve and some lie below it; overall, the **errors have approximately mean zero**.

In essence, **statistical learning refers to a set of approaches for estimating** f. In this chapter we outline some of the key theoretical concepts that arise in estimating f.

2.2 Why Estimate f (systematic information provided by a predictor about a quantitative response)?

There are two main reasons that we may wish to estimate f: **prediction** and **inference**.

2.2.1 Prediction

In many situations, a set of inputs X are readily available, but the output Y cannot be easily obtained. In this setting, since the error term ε averages to zero, we can predict Y using:

$$\hat{Y} = \hat{f}\left(X\right) \tag{19}$$

- $\hat{f}$ represents our **estimate for f**
- $\hat{Y}$ represents **prediction** for Y

The function $\hat{f}$ is often treated as a **black box**, in the sense that one is not typically concerned with the exact form of $\hat{f}$, provided that **it yields accurate predictions for Y**.

Example 2

As an **example**, suppose that:

- $X_1, \dots, X_p$ are **characteristics of a patient's blood sample** that can be easily measured in a lab.
- Y is a variable encoding the **patient's risk for a severe adverse reaction to a particular drug**.

It is natural to seek to predict Y using X, since we can then avoid giving the drug in question to patients who are at high risk of an adverse reaction. That is, patients for whom the estimate of Y is high.

The accuracy of $\hat{Y}$ as a prediction for Y depends on two quantities: **reducible error** and **irreducible error**.

- In general, $\hat{f}$ will not be a perfect estimate for f, and this **inaccuracy** will introduce some error. This is a **reducible error** because we can potentially **improve the accuracy of $\hat{f}$ by using the most appropriate statistical learning technique to estimate f**.
- Even if it were possible to form a perfect estimate for f, so that our estimated response took the form $\hat{Y} = f\left(X\right)$, our prediction would still have some error in it! This is because Y is also a function of ε (error term), which, by definition, cannot be predicted using X. Therefore, variability associated with ε also affects the accuracy of our predictions. This is the **irreducible error**, because **no matter how well we estimate f, we cannot reduce the error introduced by ε**.

The real question is: *why is the irreducible error larger than zero?* Well, the quantity ε may contain unmeasured variables that are useful in predicting Y: since we don't measure them, f cannot use them for its prediction. The quantity ε may also contain unmeasurable variation.

Example 3

For **example**, the risk of an adverse reaction might vary for a given patient on a given day, depending on manufacturing variation in the drug itself or the patient's general feeling of well-being on that day.

Consider a given estimate $\hat{f}$ and a set of predictors X, which yields the prediction $\hat{Y} = \hat{f}(X)$. Assume for a moment that both $\hat{f}$ and X are fixed, so that the only variability comes from ε (error term). Then, it's easy to show that:

$$\begin{aligned} E\left(Y - \hat{Y}\right)^2 &= E\left[f(X) + \varepsilon - \hat{f}(X)\right]^2 \\ &= \underbrace{\left[f(X) - \hat{f}(X)\right]^2}_{\text{Reducible}} + \underbrace{\operatorname{Var}(\varepsilon)}_{\text{Irreducible}} \end{aligned} \tag{20}$$

- $\left[f(X) - \hat{f}(X)\right]^2$ represents the **squared difference between the predicted and actual value of Y**
- $E\left(Y - \hat{Y}\right)^2$ represents the **average**, or **exprected value**
- $\operatorname{Var}(\varepsilon)$ represents the **variance associated with the error term ε**

The focus of this course is on *techniques* for estimating f with the aim of **minimizing the reducible error**. It is important to keep in mind that the irreducible error will always provide an upper bound on the accuracy of our prediction for Y. Unfortunately, this bound is almost always unknown in practice.

Example 4

Consider a company that is interested in conducting a direct-marketing campaign.
The *goal* is to identify individuals who are likely to respond positively to a mailing, based on observations of demographic variables measured on each individual.
In this case:

- The demographic variables serve as *predictors*;
- Response to the marketing campaign (either positive or negative) serves as the *outcome*.

The company is not interested in obtaining a deep understanding of the relationships between each individual predictor and the response; instead, the company simply **wants to accurately predict the response using the predictors**.
This is an example of **modeling for prediction**.

2.2.2 Inference

We are often interested in understanding the association between Y (quantitative response) and $X_1, \ldots, X_p$ (p-predictors). In this situation we wish to estimate f (systematic information), but our goal is not necessarily to make predictions for Y. Now it's obviously that $\hat{f}$ cannot be treated as a black box, because we need to know its exact form. In this setting, one may be interested in **answering the following questions**:

- ***Which predictors are associated with the response?*** It is often the case that only a small fraction of the available predictors are substantially associated with Y. So, **identifying** the few **important predictors among a large set of possible variables can be extremely useful**.
- ***What is the relationship between the response and each predictor?*** Larger values of the predictor are associated with larger values of Y. Other predictors may have the opposite relationship. The relationship between the response and the given predictor may **depend** on:
  - The **complexity** of f;
  - The **values of the other predictors**.
- ***Can the relationship between Y and each predictor be adequately summarized using a linear equation, or is the relationship more complicated?*** Historically, **most methods** for estimating f **have** taken **linear form**. But often the true relationship is more complicated, in which case a **linear model may not provide an accurate representation** of the relationship between the input and the output variables.

Example 5

Modeling the brand of a product that a customer might purchased based on variables such as:

- Price
- Store
- Location
- Discount levels
- Competition price

And so forth. In this situation one might really be most interested in the **association between each variable and the probability of purchase**. For instance, *to what extent is the product's price associated with sales?*
This is an example of **modeling for inference**.

Example 6

Consider the following figure:

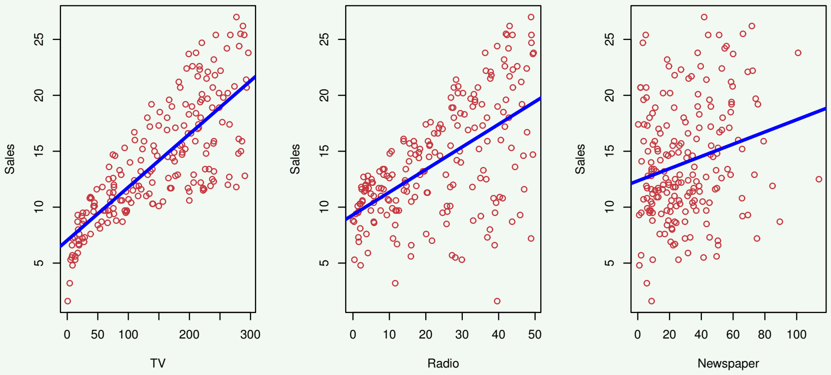


Figure 4: The `Advertising` data set. The plot displays `sales`, in thousands of units, as a function of `TV`, `radio`, and `newspaper` budgets, in thousands of dollars, for 200 different markets. In each plot we show the simple least squares fit of `sales` to that variable. In other words, each blue line represents a simple model that can be used to predict `sales` using `TV`, `radio`, and `newspaper`, respectively.

One may be interested in answering questions such as:

- *Which media are associated with sales?*
- *Which media generate the biggest boost in sales?*
- *How large of an increase in sales is associated with a given increase in TV advertising?*

This situation falls into the **inference model**.

2.2.3 Difference between prediction and inference

Example 7

In a real estate setting, one may seek to relate values of homes to inputs such as:

- Crime rate
- Zoning
- Distance from a river
- Air quality
- Schools
- Income level of community
- Size of houses

And so forth. In this case one might be interested in the association between each individual input variable and housing price. For instance, *how much extra will a house be worth if it has a view of the river?* This is an **inference problem**.

But attention! Alternatively, one may simply be interested in predicting the value of a home given its characteristics: *is this house under or over valued?* And this is a **prediction problem**.

So, as you can see from the example, the difference between a prediction problem and an inference problem is so small. A problem can change its nature because the ultimate goal is also changing.

2.3 How do we estimate f?

We will always assume that we have observed a set of n different data points. For example, in figure 3 at page 10 we observed $n = 30$ data points. These observations are called **training data** because we will **use these observations to train, or teach, our method how to estimate $\boldsymbol{f}$**.

Let:

- x_{ij} represent the value of the jth predictor, or input, for observation i, where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, p$
- y_i represent the response variable for the ith observation.

Then, our training data consist of:

$$\left\{ \left(x_1, y_1\right), \left(x_2, y_2\right), \ldots, \left(x_n, y_n\right) \right\}$$

Where $x_i = \left(x_{i1}, x_{i2}, \ldots, x_{ip}\right)^T$.

Our goal is to **apply a statistical learning method to the training data in order to estimate the unknown function $\boldsymbol{f}$**. In other words, we want to find a function $\hat{f}$ such that $Y \approx \hat{f}\left(X\right)$ for any observations $\left(X, Y\right)$. Most statistical learning methods for this task can be characterized as either **parametric** or **non-parametric**.

2.3.1 Parametric Methods

The **parametric methods** involve a two-step model-based approach:

1. Select a model.

   (a) **Make an assumption about the functional form**, or shape, of f. For **example**, one very simple assumption is that f is linear in X:
   $$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p \tag{21}$$
   This is a **linear model** (that will be discussed in the future). Once we have assumed that f is linear, **the problem of estimating f is greatly simplified**. Instead of having to estimate an entirely arbitrary p-dimensional function $f(X)$, one only needs to **estimate the $p+1$ coefficients** $\beta_0, \beta_1, \dots, \beta_p$.

2. Use training data to fit/train the model.

   (b) After a model has been selected, we need a **procedure that uses the training data to fit the model** or **train the model**. In the case of the linear method, we need to estimate the parameters $\beta_0, \beta_1, \dots, \beta_p$. So, we want to find values of these parameters such that:
   $$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$
   The most **common approach to fitting** the (linear) model is referred to as (**ordinary**) **least squares** (that will be discussed in the future). However, the least squares is one of many possible ways to fit the linear model.

The parametric model-based reduces the problem of estimating f down to one of **estimating a set of parameters**. In fact, assuming a parametric form for f simplifies the problem of estimating f because it is generally much easier to estimate a set of parameters in the linear model, than it is to fit an entirely arbitrary function f.

⚠ Potential disadvantage

The **model** we choose will **usually not match the true unknown form of f**. If the chosen model is **too far** from the true f, then our **estimate will be poor**.

✓ Possible (partial) solution

We can try to address this problem by **choosing flexible models** that can **fit many different possible functional forms for f**. But fitting a more flexible model **requires estimating a greater number of parameters**.

These more complex models (**flexible models**) can lead to a phenomenon known as **overfitting** the data, which essentially means **they follow the errors**, or **noise**, **too closely** (these issues are discussed throughout this course).

2.3.2 Non-Parametric Methods

The **non-parametric** methods do not make explicit assumptions about the functional form of f. Instead they seek an **estimate of f that gets as close to the data points as possible without being too rough or wiggly**.

✓ Major advantage over parametric approaches

By avoiding the assumption of a particular functional form for f, non-parametric approaches have the **potential to accurately fit a wider range of possible shapes** for f. Any parametric approach brings with it the possibility that the functional form used to estimate f is very different from the true f, in which case the resulting model will not fit the data well.

⚠ Disadvantage

Since non-parametric approaches do not reduce the problem of estimating f to a small number of parameters, **a very large number of observations** (far more than is typically needed for a parametric approach) **is required in order to obtain an accurate estimate** for f.

2.4 Supervised and Unsupervised Learning

Most statistical learning problems fall into one of two categories: **supervised learning** or **unsupervised learning**.

Supervised learning

The examples that we have discussed in this chapter all fall into the **supervised learning** domain. For each observation of the predictor measurement(s) $x_i, i = 1, \ldots, n$ there is an associated response measurement y_i.

We wish to **fit a model that relates the response to the predictors**, with the aim of:

- **Accurately predicting the response for future observations** (prediction, section 2.2.1)
- **Better understanding the relationship between the response and the predictors** (inference, section 2.2.2)

Unsupervised learning

The **unsupervised learning** describes the somewhat more challenging situation in which **for every observation** $i = 1, \ldots, n$, **we observe a vector of measurements** x_i **but no associated response** y_i.

In this setting, we are in some sense *working blind*; the situation is referred to as **unsupervised** because **we lack a response variable that can supervise our analysis**. We can **seek to understand the relationships between the variables or between the observations**.

2.5 Assessing Model Accuracy

The aim of this section is to decide which method will give the best results for a given set of data.

2.5.1 Measuring the Quality of Fit (MSE)

In order to evaluate the performance of a statistical learning method on a given data set, we need some way to measure how well its predictions actually match the observed data. We need to **quantify the extent to which the predicted response value for a given observation is close to the true response value for that observation**. The most commonly-used measure is the **mean squared error (MSE)**:

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{f}\left(x_i\right) \right)^2 \tag{22}$$

- $\hat{f}\left(x_i\right)$ is the prediction that $\hat{f}$ gives for the ith observation
- y_i the ith true response

Obviously,the MSE will be:

- **Small** if the predicted responses are very close to the true responses;
- **Large** if for some of the observations, the predicted and true responses differ substantially.

In general, we do not really care how well the method works on the training data. Rather, **we are interested in the accuracy of the predictions that we obtain when we apply our method to previously unseen test data**.

Example 8

Suppose that we are interested in developing an algorithm to predict a stock's price based on previous stock returns.

We can train the method using stock returns from the past 6 months. But we don't really care how well our method predicts last week's stock price.
We instead **care about how well it predict tomorrow's price or next month's price**.

Example 9

Suppose that we have clinical measurements (e.g. weight, blood pressure, height, age, family history of disease) for a number of patients, as well as information about whether each patient has diabetes.

We can use these patients to train a statistical learning method to predict risk of diabetes based on clinical measurements.

In practice, **we want this method to accurately predict diabetes risk for *future patients* based on their clinical measurements**. Again, we are not very interested in whether or not the method accurately predicts diabetes risk for patients used to train the mode, since we already know which of those patients have diabetes!

In mathematical terms, suppose that we fit our statistical learning method on our training observations:

$$\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

And we obtain the estimate $\hat{f}$. We can then compute:

$$\hat{f}(x_1), \hat{f}(x_2), \dots, \hat{f}(x_n)$$

If these are approximately equal to:

$$y_1, y_2, \dots, y_n$$

Then **the training MSE is small**.

However, we are really not interested in whether $\hat{f}(x_i) \approx y_i$; instead, we want to know whether $\hat{f}(x_0)$ is approximately equal to y_0, where (x_0, y_0) is a **previously unseen test observation not used to train the statistical learning method**.

We **want to choose the method that gives the lowest test mean squared error (MSE)**, as opposed to the lowest training MSE. In other words, if we had a large number of test observations, we could compute:

$$\text{Ave}\left(y_0 - \hat{f}(x_0)\right)^2 \tag{23}$$

The **average squared prediction error for these test observations** (x_0, y_0).

⚠ Problem to find the lowest training MSE

There is no guarantee that the method with the lowest training MSE will also have the lowest test MSE.

The problem is that **many statistical methods specifically estimate coefficients so as to minimize the training set MSE**. For these methods, the **training set MSE can be quite small**, **but the test MSE is often much larger**.

Example 10

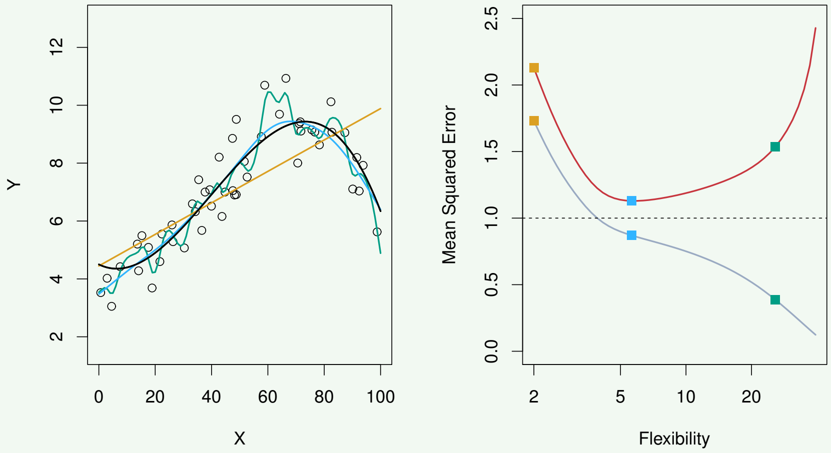


Figure 5: On the left: *data simulated from f, shown in black. Three estimates of f are shown: the linear regression (orange curve), and two smoothing spline fits (blue and green curves).* Right: *Training MSE (grey curve), test MSE (red curve), and minimum possible test MSE over all methods (dashed line). Squares represent the training and test MSEs for the three fits shown in the left-hand panel.* [1]

In the left-hand panel we have generated observations from the (error term) equation 18 with the true f given by the black curve.
The orange, blue and green curves illustrate three possible estimates for f obtained using methods with increasing levels of flexibility.

It is clear that as the **level of flexibility increases**, the **curves fit the observed data more closely**.
The *green curve* is the most flexible and matches the data very well; however, we observe that it fits the true f (shown in black) poorly because it is too wiggly.
By **adjusting the level of flexibility** of the smoothing spline fit, we can **produce many different fits to this data**.

Example 10

Referring to Figure 5

We now move on to the right-hand panel. The grey curve displays the average training MSE as a function of flexibility, or more formally the **degrees of freedom**[a], for a number of smoothing splines.
The orange, blue and green squares indicate the MSEs associated with the corresponding curve in the left-hand panel.
A more restricted and hence smoother curve has fewer degrees of freedom than a wiggly curve. The *linear regression* is at the most restrictive end, with two degrees of freedom.

The **training MSE declines monotonically as flexibility increases**. In this example, the true f is non-linear, and so the orange linear fit is not flexible enough to estimate f well.
The *green curve* has the lowest training MSE of all three methods, since it corresponds to the most flexible of the three curves fit in the left-hand panel.

The test MSE is displayed using the red curve. As with the training MSE, the test MSE initially declines as the level of flexibility increases. At some point, the test MSE levels off and then starts to increase again. Consequently, the orange and green curves both have high test MSE. The blue curve minimizes the test MSE, which should not be surprising given that visually it appears to estimate f the best in the left-hand panel.
The horizontal dashed line indicates $\mathrm{Var}(\varepsilon)$, the **irreducible error** (eq. 20), which **corresponds to the lost achievable test MSE among all possible methods**. Hence, the smoothing spline represented by **the blue curve is close to optimal**.

[a] The degrees of freedom is a **quantity that summarizes the flexibility of a curve**.

In the right-hand panel of figure 5, as the flexibility of the Statistical learning method increases, we observe a **monotone decrease in the training MSE and a U-shape** in the test MSE. This is a **fundamental property** of statistical learning that holds regardless of the particular data set at hand and regardless of the Statistical method being used.

As model flexibility increases, the training MSE will decrease, but the test MSE may not. **When a given method yields a small training MSE but a large test MSE**, we are said to be **overfitting** the data.

❓ Why does this phenomenon happen?

This happens because our **statistical learning procedure** is working too hard to find patterns in the training data, and **may be picking up some patterns that are just caused by random chance** rather than by true properties of the unknown function f.

So when we *overfit* the training data, the **test MSE** will be **very large because the supposed patterns that the method found in the training data simply don't exist in the test data**.

We almost always expect the **training MSE to be smaller than the test MSE** because most **statistical learning methods** either directly or indirectly seek to **minimize the training MSE**. *Overfitting* refers specifically to the test case in which a **less flexible model would have yielded a smaller test MSE**.

Example 11

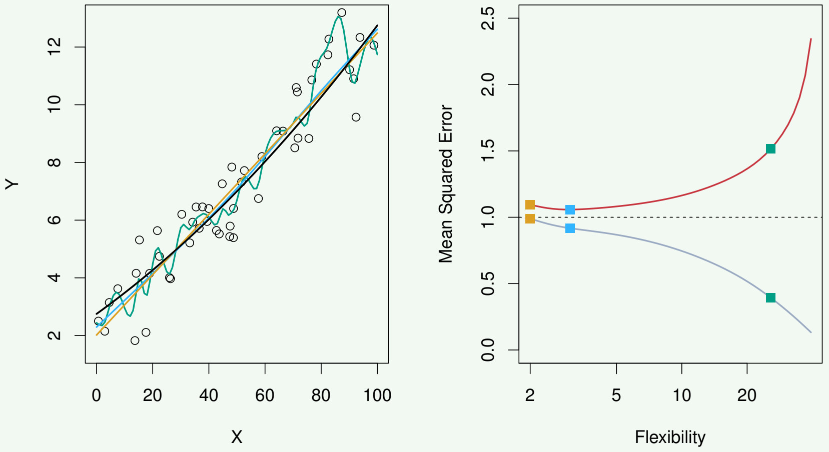


Figure 6: Details are as in Figure 5, using a different true f that is much closer to linear. In this setting, linear regression provides a very good fit to the data. [1]

This figure provides another **example** in which the true f is approximately linear. Again we observe that the training MSE decreases monotonically as the model flexibility increases, and that there is a *U-shape* in the test MSE.

However, because the truth is close to linear, the **test MSE only decreases slightly before increasing again**, so that the **orange least squares fit is substantially better than the highly flexible green curve**.

Example 12

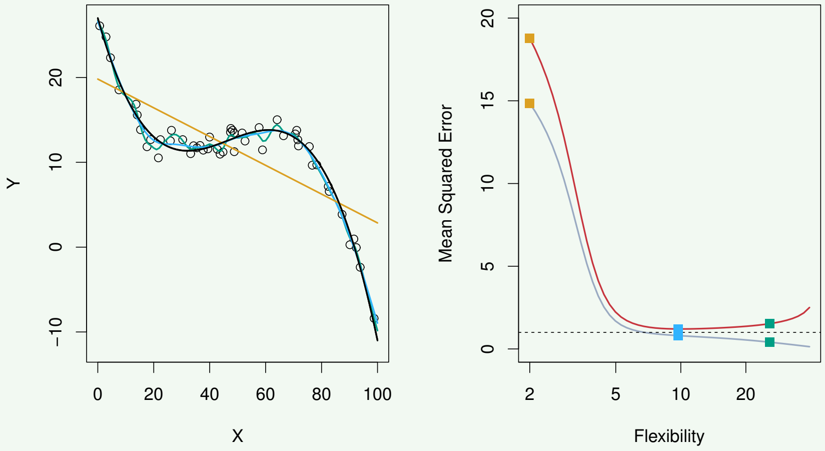


Figure 7: Details are as in Figure 5, using a different true f that is far from linear. In this setting, linear regression provides a very poor fit to the data. [1]

Finally, this figure displays an **example** in which f is highly non-linear.

The training and test MSE curves still exhibit the same general patterns, but now there is a rapid decrease in both curves before the test MSE start to increase slowly.

2.5.2 The Bias-Variance Trade-Off

The U-shape observed in the test MSE curves (Figures: 5, 6, 7) turns out to be the result of two competing properties of statistical learning methods.

The expected test MSE, for a given value x_0, can always be decomposed into the sum of three fundamental quantities:

- The **variance** of $\hat{f}\left(x_0\right)$
- The squared **bias** of $\hat{f}\left(x_0\right)$
- The **variance of the error terms** $\boldsymbol{\varepsilon}$

$$E\left(y_0 - \hat{f}\left(x_0\right)\right)^2 = \operatorname{Var}\left(\hat{f}\left(x_0\right)\right) + \left[\operatorname{Bias}\left(\hat{f}\left(x_0\right)\right)\right]^2 + \operatorname{Var}\left(\varepsilon\right) \tag{24}$$

Where $E\left(y_0 - \hat{f}\left(x_0\right)\right)^2$ defines the **expected test MSE** at x_0 and refers to the **average test MSE** that we would obtain if we **repeatedly estimated** $\boldsymbol{f}$ **using a large number of training sets, and tested each at** x_0.

The equation 24 tell us that in order to minimize the expected test error, we need to **simultaneously select a statistical learning method that** achieves **<u>low</u> variance and <u>low</u> bias**. Note that variance is inherently a nonnegative quantity, and squared bias is also nonnegative. Hence, we see that the expected test MSE can never lie below $\operatorname{Var}\left(\varepsilon\right)$, the irreducible error (equation 20).

☆ Meaning of the variance

The **variance** refers to the **amount by which** $\boldsymbol{\hat{f}}$ **would change if we estimated it using a different training data set**. So different training data sets will result in a different $\hat{f}$. But ideally the estimate for f should not vary too much between training sets. However, **if a method has high variance then small changes in the training data can result in large changes in** $\boldsymbol{\hat{f}}$.
In general, **more flexible statistical methods have higher variance**.

Example 13

Consider the green and the orange curves in Figure 5 at page 22.

The flexible green curve is following the observations very closely. It has high variance because changing any one of these data points may cause the estimate $\hat{f}$ to change considerably.

In contrast, the orange least squares line is relatively inflexible and has low variance, because moving any single observations will likely cause only a small shift in the position of the line.

☆ Meaning of the bias

The **bias** refers to the **error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model**.

Example 14

For **example**, linear regression assumes that there is a linear relationship between Y and $X_1, X_2, \ldots, X_p$. It is unlikely that any real-life problem truly has such a simple linear relationship, and so performing linear regression will undoubtedly result in some bias in the estimate of f.

In the Figure 7 on page 25, the true f is substantially non-linear, so no matter how many training observations we are given, it will not be possible to produce an accurate estimate using linear regression. In other words, linear regression results in high bias in this example.

However, in Figure 6 on page 24 the true f is very close to linear, and so given enough data, it should be possible for linear regression to produce an accurate estimate.

Generally, as we use **more flexible methods**, the **variance will increase** and the **bias will decrease**.

As we increase the flexibility of a class of methods, the bias tends to initially decrease faster than the variance increases. Consequently, the expected test MSE declines. However, at some point increasing flexibility has little impact on the bias but starts to significantly increase the variance. When this happens the test MSE increases. Note that we observed this pattern of decreasing test MSE followed by increasing test MSE in the right-hand panels of Figures 5, 6, 7. In summary:

1. We increase the flexibility of a class of methods;
2. The bias tends to initially decrease faster than the variance increases;
3. The expected test MSE declines;
4. At some point increasing flexibility has little impact on the bias but starts to significantly increase the variance;
5. The test MSE increases.

Example 15

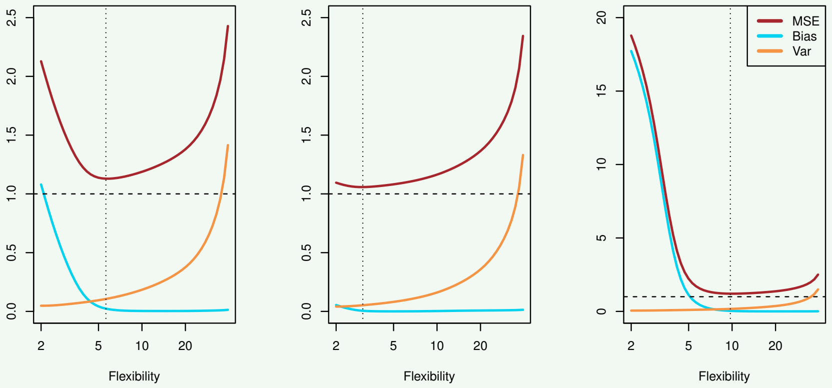


Figure 8: Squared bias (blue curve), variance (orange curve), Var (ε) (dashed line), and test MSE (red curve) for the three data sets in Figures 5, 6, 7. The vertical dotted line indicates the flexibility level corresponding to the smallest test MSE. [1]

Three plots illustrate equation 24 on page 26 for the examples in Figure 5, 6, 7.
In each case the blue solid curve represents the squared bias, for different levels of flexibility, while the orange curve corresponds to the variance. The horizontal dashed line represents Var (ε), the irreducible error. Finally, the red curve, corresponding to the test set MSE, is the sum of these three quantities.

In all three cases, the variance increases and the bias decreases as the method's flexibility increases. However, the flexibility level corresponding to the optimal test MSE differs considerably among the three data sets, because the squared bias and variance change at different rates in each of the data sets.

In the left-hand panel of this Figure, the bias initially decreases rapidly, resulting in an initial sharp decrease in the expected test MSE.

On the other hand, in the center panel of this Figure the true f is close to linear, so there is only a small decrease in bias as flexibility increases, and the test MSE only declines slightly before increasing rapidly as the variance increases.

Finally, in the right-hand panel of this Figure, as flexibility increases, there is a dramatic decline in bias because the true f is very non-linear. There is also very little increase in variance as flexibility increases. Consequently, the test MSE declines substantially before experiencing a small increase as model flexibility increases.

☆ Meaning of the bias-variance trade-off

The relationship between bias, variance, and test set MSE given in equation 24 on page 26 and displayed in the Figure 8 (previous example) is referred to as the **bias-variance trade-off**.

Good test set performance of a statistical learning method requires low variance as well as low squared bias. This is referred to as a **trade-off** because it is **easy to obtain a method with extremely low bias but high variance**[1] or **a method with very low variance but high bias** (by fitting a horizontal line to the data).

The **challenge lies in finding a method for which both the variance and the squared bias are low**. This trade-off is one of the most important recurring themes in this course.

[1] For **instance**, by drawing a curve that passes through every single training observation

2.6 Algorithm: K-Nearest Neighbors (KNN)

Many approaches attempt to **estimate the conditional distribution of *Y* given *X***, and **then classify a given observation to the class** with **highest estimated probability**. One such method is the **K-nearest neighbors (KNN)** classifier.

In mathematical terms, given a positive integer K and a test observation x_0 the KNN classifier:

1. **Identifies** the K points in the training data that are closest to x_0, represented by $\mathcal{N}_0$.
2. It then **estimates** the conditional probability for class j as the fraction of points in $\mathcal{N}_0$ whose response values equal j:

$$\Pr\left(Y = J \mid X = x_0\right) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I\left(y_i = j\right) \tag{25}$$

3. Finally, KNN **classifies** the test observation x_0 to the class with the largest probability from the previous equation.

Example 16

Suppose that we choose $K = 3$. Then KNN algorithm:

1. Identify the three observations that are closet to the cross. As you can see in the Figure 9 on page 31, this neighborhood is shown as a circle. It consists of two blue points and one orange point, resulting in estimated probabilities of $\frac{2}{3}$ for the blue class and $\frac{1}{3}$ for the orange class.
2. Hence, KNN will predict that the black cross belongs to the blue class.

Example 17

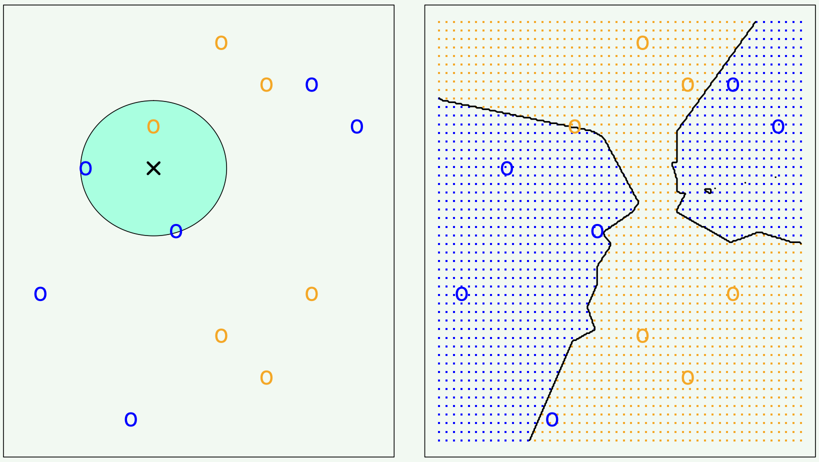


Figure 9: The KNN approach, using $K = 3$, is illustrated in a simple situation with six blue observations and six orange observations. Left: *a test observation at which a predicted class label is desired is shown as a black cross. The three closest points to the test observation are identified, and it is predicted that the test observation belongs to the most commonly-occurring class, in this case blue.* Right: *the KNN decision boundary for this example is shown in black. The blue grid indicates the region in which a test observation will be assigned to the blue class, and the orange grid indicates the region in which it will be assigned to the orange class.*

This figure provides an illustrative example of the KNN approach. In the left-hand panel, we have plotted a small training data set consisting of six blue and six orange observations. Our goal is to make a prediction for the point labeled by the black cross.

In the right-hand panel, we have applied the KNN approach with $K = 3$ at all of the possible values for X_1 and X_2, and have drawn in the corresponding KNN decision boundary.

Example 18

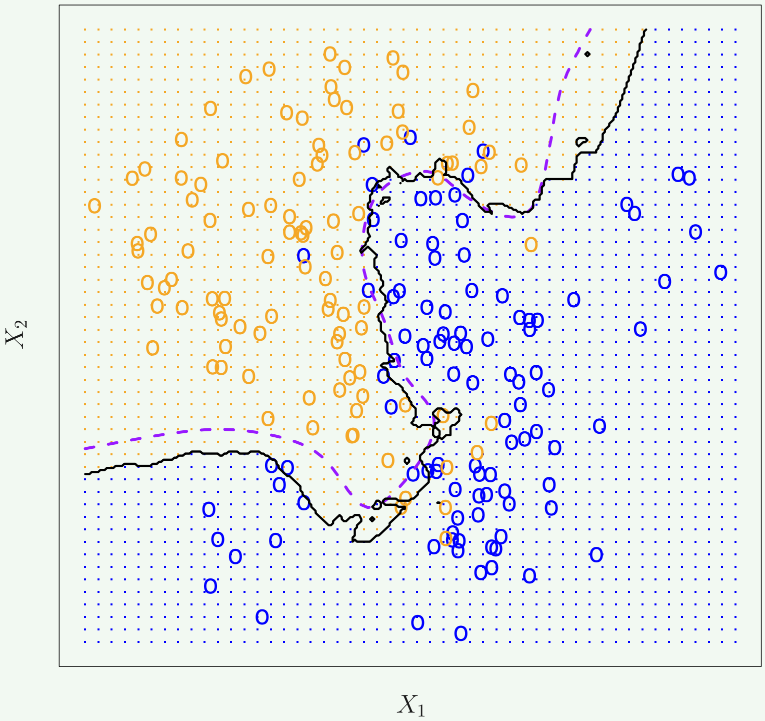


Figure 10: The black curve indicates the KNN decision boundary on the data, using $K = 10$. The Bayes decision boundary is shown as a purple dashed line. The KNN and Bayes decision boundaries are very similar.

This Figure displays the KNN decision boundary, using $K = 10$, when applied to the larger simulated data set.

3 R language programming

3.1 Introduction to R

There is no introduction to RStudio in these notes. But you can find a detailed guide here.

`R` uses functions to perform operations. To run a function called `funcname`, we type `funcname(input1, input2)`, where the inputs (or arguments) `input1` and `input2` tell `R` how to run the function.

3.1.1 Scalars, vectors and matrices

- Create a **scalar**

```
a <- 1  # classic R syntax for assignment
a = 1   # equivalent assignment using "="
a       # print the value
```

- Create a **vector**

```
v <- c(2, 3, 4, 5)
v

u <- seq(2, 5, len=4)
u

u <- seq(2, 5, by=1)
u

z <- 2:5
z
```

And the result is always the same:

```
[1] 2 3 4 5
```

  - `c` function (documentation) takes n arguments to create a vector of length n.
  - `seq` function (doc) takes two arguments to create a vector with these two values as its lower and upper bound. In the example code, the values are passed implicitly, but we can make them explicit with `from` and `to`:

```
u <- seq(from=2, to=5)
u
```

And the result is always the same:

```
[1] 2 3 4 5
```

The `len` parameter specifies the length of the vector. For example:

```
u <- seq(2, 5, len=10)
u
```

And the result is:

```
[1] 2.000000 2.333333 2.666667 3.000000 3.333333 3.666667
    4.000000 4.333333 4.666667 5.000000
```

With the `by` argument, we can increment the sequence:

```
u <- seq(2, 5, by=0.5)
u
```

And the result is:

```
[1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

  - The `:` (colon operator, doc) generates a regular sequence. It's very easy to use: `from:to`.

- Create a **matrix**

```
W <- rbind(c(11, 13, 15), c(12, 14, 16))
W

W <- cbind(c(11, 12), c(13, 14), c(15, 16))
W

W <- matrix(data = c(11, 12, 13, 14, 15, 16),
            nrow = 2, ncol = 3, byrow = F)
W

W <- matrix(c(11, 12, 13, 14, 15, 16), 2, 3)
W
```

And the result is always the same:

```
     [,1] [,2] [,3]
[1,]   11   13   15
[2,]   12   14   16
```

  – The `rbind` and `cbind` (doc) functions are very similar. Both take a sequence of vector, matrix or data-frame arguments and combine by rows or columns, respectively. So if we use `rbind`, we need to specify the lines. If we use `cbind` instead, we need to specify each column.

    A useful piece of advice when using `rbind` or `cbind` is the code style. The following code is easier to read:

```
W <- rbind(
    c(11, 13, 15),
    c(12, 14, 16)
    )

W <- cbind(
    c(11, 12),
    c(13, 14),
    c(15, 16)
    )
```

    We can also convert a vector into a row vector with `rbind` or a column vector with `cbind`.

  – The `matrix` (doc) function creates a matrix from the given set of values. The arguments:

    * `data` is an optional data vector.
    * `nrow` is the desired number of rows.
    * `ncol` is the desired number of columns.
    * `byrow` is a logical argument. If `FALSE` (or `F`, the default) the matrix is filled by columns, otherwise the matrix is filled by rows.

```
W <- matrix(c(11, 12, 13,
              14, 15, 16), nrow=2, ncol=3, byrow=TRUE)
W
#      [,1] [,2] [,3]
# [1,]   11   12   13
# [2,]   14   15   16
```

3.1.2 Access elements

We can access to an **element of a vector** using the square brackets.

- By explicitly inserting the index

```
v <- c(2, 3, 4, 5)
v[2]
# Output: [1] 3
```

- By explicitly inserting a vector as an index to access multiple elements

```
v[c(2, 3)]
# Output: [1] 3 4
```

- By explicitly inserting a negative value as an index to access all values except the specified index value (is the opposite of a positive index value)

```
v[-1]
# [1] 3 4 5
v[-2]
# [1] 2 4 5
v[-3]
# [1] 2 3 5
v[-4]
# [1] 2 3 4
```

- By explicitly inserting a negative vector as an index to access all values except the specified vector value (is the opposite of a positive vector value)

```
v[-c(1, 4)]
# [1] 3 4
```

We can access to an **element of a matrix** using the square brackets.

- By explicitly inserting the index

```
W <- matrix(data = c(11, 12, 13, 14, 15, 16), nrow = 2, ncol = 3, byrow = F)
W
#       [,1] [,2] [,3]
# [1,]    11   13   15
# [2,]    12   14   16
W[2, 3]
# [1] 16
```

- By explicitly inserting a vector as an index (column or row) to access multiple elements

```
W[2, c(2, 3)]
# [1] 14 16
```

- By explicitly inserting a blank to access all values of the row/column

```
W[2, ]
# [1] 12 14 16
W[, c(2, 3)]
#       [,1] [,2]
# [1,]    13   15
# [2,]    14   16
```

3.1.3 Algebraic operations

By default, operations in R are performed on a component-by-component basis. For example, given the following data:

```
a <- 1
b <- 2

f <- c(2, 3, 4)
d <- c(10, 10, 10)

Z <- matrix(c(1, 10, 3, 10, 5, 10), nrow = 2, ncol = 3, byrow = F)
```

Operations with the **vectors**:

- **Sum** between two scalars

```
a + b # scalar + scalar
# [1] 3
```

- **Sum** between two vectors

```
f + d # vector + vector
# [1] 12 13 14
```

- **Sum** between a vector and a scalar

```
f + a # vector + scalar
# [1] 3 4 5
```

- **Sums** the components of a vector

```
sum(f) # sums the components of f
# [1] 9
```

- **Multiply** between two scalars

```
a * b # scalar * scalar
# [1] 2
```

- **Multiply** between two vectors

```
f * d # vector * vector (component-wise)
# [1] 20 30 40
```

- **Multiplies** between the components of a vector

```
prod(f) # returns the product of the components of f
# [1] 24
```

- **Exponential** of a vector

```
f^2
# [1]  4  9 16
```

- **Exponential function** of a vector

```
exp(f)
# [1]  7.389056 20.085537 54.598150
```

Operations with the **matrices**. Given the following data:

```
a <- 1
f <- c(2, 3, 4)
Z <- matrix(c(1, 10, 3, 10, 5, 10), nrow=2, ncol=3, byrow=F)
W <- matrix(c(11, 12, 13, 14, 15, 16), nrow=2, ncol=3, byrow=F)
```

- **Transpose** of a matrix

```
V <- t(W) # transpose of a matrix
#       [,1] [,2]
# [1,]    11   12
# [2,]    13   14
# [3,]    15   16
```

- **Inverse** of a matrix

```
A <- matrix(c(11, 13, 12, 14), ncol=2, nrow=2, byrow=TRUE)
#       [,1] [,2]
# [1,]    11   13
# [2,]    12   14
```

- **Determinant** of a matrix

```
det(A)
# [1] -2
```

- The generic function **solves** the equation `a %*% x = b` for `x`, where `b` can be either a vector or a matrix. If `b` is missing, it's taken to be an identity matrix and `solve` will return the inverse of `a`.

```
solve(A)
#       [,1] [,2]
# [1,]    -7  6.5
# [2,]     6 -5.5

# Solution of a linear system Ax=b
b <- c(1, 1)
solve(A, b)
# [1] -0.5  0.5
```

- **Sum** between matrices

```
Z + W # matrix + matrix (component-wise)
#       [,1] [,2] [,3]
# [1,]    12   16   20
# [2,]    22   24   26
```

- **Sum** between matrices and scalars

```
W + a # matrix + scalar
#       [,1] [,2] [,3]
# [1,]    12   14   16
# [2,]    13   15   17
```

- **Sum** between matrices and vectors

```
W + f # matrix + vector
#       [,1] [,2] [,3]
# [1,]    13   17   18
# [2,]    15   16   20
```

- **Multiplication** (component-by-component) between matrices

```
Z * W # matrix * matrix (component-wise)
#       [,1] [,2] [,3]
# [1,]    11    39    75
# [2,]   120   140   160
```

- **Multiplication** (classic) between matrices

```
V * W # matrix * matrix (component-wise) (error!)
# Error in V * W : non-conformable arrays

V %*% W # Matrix multiplication
#       [,1] [,2] [,3]
# [1,]   265   311   357
# [2,]   311   365   419
# [3,]   357   419   481

W %*% V
#       [,1] [,2]
# [1,]   515   554
# [2,]   554   596
```

3.1.4 Categorical data

The function `factor` is used to encode a vector as a factor. Arguments:

- `x`: a vector of data, usually taking a small number of distinct values.
- `levels`: an optional vector of the unique values (as character strings) that `x` might have taken.

For example:

```r
district <- c(’MI’, ’MI’, ’VA’, ’BG’, ’LO’, ’LO’, ’CR’, ’Alt’,
              ’CR’, ’MI’, ’Alt’, ’CR’, ’LO’, ’VA’, ’MI’, ’Alt’,
              ’LO’, ’MI’)
district <- factor(district,
                   levels=c(’MI’, ’LO’, ’BG’, ’CR’, ’VA’, ’Alt’))
district
#  [1] MI  MI  VA  BG  LO  LO  CR  Alt CR  MI  Alt CR  LO  VA  MI  Alt LO  MI
# Levels: MI LO BG CR VA Alt
```

The function `table` uses cross-classifying factors to build a contingency table of the counts at each combination of factor levels.

```r
# table of absolute frequencies
resass <- table(district)
resass
# district
#  MI  LO  BG  CR  VA Alt
#   5   4   1   3   2   3

# table of relative frequencies
resrel <- table(district) / length(district)
resrel
# district
#         MI         LO         BG         CR         VA        Alt
# 0.27777778 0.22222222 0.05555556 0.16666667 0.11111111 0.16666667
```

3.1.5 Lists

A `list` in `R` is a generic object consisting of an **ordered collection of objects**. Lists are one-dimensional, heterogeneous data structures. The list can be a list of vectors, a list of matrices, a list of characters and a list of functions, and so on.

For example, here is a `list` containing the results of an exam:

```
exam <- list (
    course = 'Applied Statistics',
    date = '27/09/2022',
    enrolled = 7,
    corrected = 6,
    student_id = as.character(c(45020, 45679,
                                46789, 43126,
                                42345, 47568, 45674)),
    evaluation = c(30, 29, 30, NA, 25, 26, 27)
)
exam
# $course
# [1] "Applied Statistics"
#
# $date
# [1] "27/09/2022"
#
# $enrolled
# [1] 7
#
# $corrected
# [1] 6
#
# $student_id
# [1] "45020" "45679" "46789" "43126" "42345" "47568" "45674"
#
# $evaluation
# [1] 30 29 30 NA 25 26 27
```

To access a property of the object, we can use the $ symbol or the square brackets:

```
exam$evaluation
# [1] 30 29 30 NA 25 26 27

exam[[6]]
# [1] 30 29 30 NA 25 26 27
```

3.1.6 Data Frames

The function `data.frame()` creates data frames, tightly coupled collections of variables which share many of the properties of matrices and of lists, used as the fundamental data structure by most of `R`'s modeling software.

For example:

```
exam <- data.frame(
  student_id = factor(as.character(c(45020, 45679,
                                     46789, 43126,
                                     42345, 47568,
                                     45674))),
  # evaluation aka: evaluate
  evaluate_W = c(30, 29, 30, NA, 25, 26, 17),
  evaluate_O = c(28, 30, 30, NA, 28, 27, NA),
  evaluate_P = c(30, 30, 30, 30, 28, 28, 28),
  outcome  = factor(c('Passed', 'Passed', 'Passed',
                      'To be repeated', 'Passed',
                      'Passed', 'To be repeated')))
exam
#   student_id evaluate_W evaluate_O evaluate_P        outcome
# 1      45020         30         28         30         Passed
# 2      45679         29         30         30         Passed
# 3      46789         30         30         30         Passed
# 4      43126         NA         NA         30 To be repeated
# 5      42345         25         28         28         Passed
# 6      47568         26         27         28         Passed
# 7      45674         17         NA         28 To be repeated
```

Like the lists, to access a property of the data frame, we can use $, or the square brackets:

```
# a data.frame is a particular kind of list!
exam$evaluate_W
# [1] 30 29 30 NA 25 26 17
exam[[2]]
# [1] 30 29 30 NA 25 26 17
exam[2, ]
#   student_id evaluate_W evaluate_O evaluate_P outcome
# 2      45679         29         30         30  Passed
```

The data frame has two important and frequently used functions: `attach` and `detach`:

- In R, `attach()` is a function that allows us to attach a database (usually a `data frame`) to the `R` search path. This function makes it easier to interact with objects within `data frames` by eliminating the need to repeatedly reference the data frame itself.

  In simpler terms, the `attach()` function takes a data frame and places it in the search path of R's environment. Once a data frame is attached, we can **call its variables directly, without the need to use the $ operator or square brackets**.

  ```
  attach(exam)
  # Note: This variable has not been declared before!
  #       It's a property of exam!
  evaluate_W
  # [1] 30 29 30 NA 25 26 17
  ```

- The `detach()` function detaches a database. Usually this is either a `data.frame` which has been attached or a package which was attached by library.

```
detach(exam)
evaluate_W
# Error: object ’evaluate_W’ not found
```

3.1.7 Reading and writing data

The `read.table` function reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file. It can also read a csv file. It has the following (not all) parameters:

- `file`: **the name of the file which the data are to be read from**. Each row of the table appears as one line of the file. If it does not contain an absolute path, the file name is relative to the current working directory, `getwd()`.

  For example, if our working directory is

  `C:\Users\Applied-Statistics`

  and we have a folder `1-lesson` within the working directory, to access a file within the folder we can write the absolute path:

  `C:\Users\Applied-Statistics\1-lesson\file-name.txt`

  or the relative path `1-lesson\file-name.txt`.

- `header`: a logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: **header is set to `TRUE` if and only if the first row contains one fewer field than the number of columns**.

For example, given the following raw (txt) file:

```
"m100" "m200" "m400" "m800" "m1500" "m3000" "Marathon"
"argentin" 11.61 22.94 54.5 2.15 4.43 9.79 178.52
"australi" 11.2 22.35 51.08 1.98 4.13 9.08 152.37
"austria" 11.43 23.09 50.62 1.99 4.22 9.34 159.37
"belgium" 11.41 23.04 52 2 4.14 8.88 157.85
"bermuda" 11.46 23.05 53.3 2.16 4.58 9.81 169.98
"brazil" 11.31 23.17 52.8 2.1 4.49 9.77 168.75
"burma" 12.14 24.47 55 2.18 4.45 9.51 191.02
"canada" 11 22.25 50.06 2 4.06 8.81 149.45
"chile" 12 24.52 54.9 2.05 4.23 9.37 171.38
"china" 11.95 24.41 54.97 2.08 4.33 9.31 168.48
"columbia" 11.6 24 53.26 2.11 4.35 9.46 165.42
"cookis" 12.9 27.1 60.4 2.3 4.84 11.1 233.22
"costa" 11.96 24.6 58.25 2.21 4.68 10.43 171.8
"czech" 11.09 21.97 47.99 1.89 4.14 8.92 158.85
"denmark" 11.42 23.52 53.6 2.03 4.18 8.71 151.75
"domrep" 11.79 24.05 56.05 2.24 4.74 9.89 203.88
"finland" 11.13 22.39 50.14 2.03 4.1 8.92 154.23
"france" 11.15 22.59 51.73 2 4.14 8.98 155.27
"gdr" 10.81 21.71 48.16 1.93 3.96 8.75 157.68
"frg" 11.01 22.39 49.75 1.95 4.03 8.59 148.53
"gbni" 11 22.13 50.46 1.98 4.03 8.62 149.72
"greece" 11.79 24.08 54.93 2.07 4.35 9.87 182.2
"guatemal" 11.84 24.54 56.09 2.28 4.86 10.54 215.08
"hungary" 11.45 23.06 51.5 2.01 4.14 8.98 156.37
"india" 11.95 24.28 53.6 2.1 4.32 9.98 188.03
"indonesi" 11.85 24.24 55.34 2.22 4.61 10.02 201.28
"ireland" 11.43 23.51 53.24 2.05 4.11 8.89 149.38
"israel" 11.45 23.57 54.9 2.1 4.25 9.37 160.48
"italy" 11.29 23 52.01 1.96 3.98 8.63 151.82
```

```
"japan" 11.73 24 53.73 2.09 4.35 9.2 150.5
"kenya" 11.73 23.88 52.7 2 4.15 9.2 181.05
"korea" 11.96 24.49 55.7 2.15 4.42 9.62 164.65
"dprkorea" 12.25 25.78 51.2 1.97 4.25 9.35 179.17
"luxembou" 12.03 24.96 56.1 2.07 4.38 9.64 174.68
"malaysia" 12.23 24.21 55.09 2.19 4.69 10.46 182.17
"mauritiu" 11.76 25.08 58.1 2.27 4.79 10.9 261.13
"mexico" 11.89 23.62 53.76 2.04 4.25 9.59 158.53
"netherla" 11.25 22.81 52.38 1.99 4.06 9.01 152.48
"nz" 11.55 23.13 51.6 2.02 4.18 8.76 145.48
"norway" 11.58 23.31 53.12 2.03 4.01 8.53 145.48
"png" 12.25 25.07 56.96 2.24 4.84 10.69 233
"philippi" 11.76 23.54 54.6 2.19 4.6 10.16 200.37
"poland" 11.13 22.21 49.29 1.95 3.99 8.97 160.82
"portugal" 11.81 24.22 54.3 2.09 4.16 8.84 151.2
"rumania" 11.44 23.46 51.2 1.92 3.96 8.53 165.45
"singapor" 12.3 25 55.08 2.12 4.52 9.94 182.77
"spain" 11.8 23.98 53.59 2.05 4.14 9.02 162.6
"sweden" 11.16 22.82 51.79 2.02 4.12 8.84 154.48
"switzerl" 11.45 23.31 53.11 2.02 4.07 8.77 153.42
"taipei" 11.22 22.62 52.5 2.1 4.38 9.63 177.87
"thailand" 11.75 24.46 55.8 2.2 4.72 10.28 168.45
"turkey" 11.98 24.44 56.45 2.15 4.37 9.38 201.08
"usa" 10.79 21.83 50.62 1.96 3.95 8.5 142.72
"ussr" 11.06 22.19 49.19 1.89 3.87 8.45 151.22
"wsamoa" 12.74 25.85 58.73 2.33 5.81 13.04 306
```

The `R` code to read it is:

```
record <- read.table('1_IntroR/record.txt', header=TRUE)
```

Some useful functions:

- `head` or `tail`: Returns the first or last parts of a vector, matrix, table, data frame or function.

  ```
  head(record)
  #            m100  m200  m400 m800 m1500 m3000 Marathon
  # argentin 11.61 22.94 54.50 2.15  4.43  9.79   178.52
  # australi 11.20 22.35 51.08 1.98  4.13  9.08   152.37
  # austria  11.43 23.09 50.62 1.99  4.22  9.34   159.37
  # belgium  11.41 23.04 52.00 2.00  4.14  8.88   157.85
  # bermuda  11.46 23.05 53.30 2.16  4.58  9.81   169.98
  # brazil   11.31 23.17 52.80 2.10  4.49  9.77   168.75
  ```

- `dim`: Retrieve or set the dimension of an object.

  - For `data.frame` returns the lengths of the `row.names` attribute of `x` and of `x` (as the numbers of rows and columns respectively).
  - For an `array` (and hence in particular, for a `matrix`) `dim` retrieves the `dim` attribute of the object.

  In the previous example, the number of rows, excluding the header, is 55 and the number of columns is 7:

  ```
  dim(record)
  # [1] 55  7
  ```

- `dimnames`: Retrieve or set the `dimnames` of an object. The `dimnames` of a `data frame` are its `row.names` and its `names`.

```
dimnames(record)
# [[1]]
#  [1] "argentin" "australi" "austria"  "belgium"  "bermuda"  "brazil"   "burma"    "canada"   "chile"    "china"
# [11] "columbia" "cookis"   "costa"    "czech"    "denmark"  "domrep"   "finland"  "france"   "gdr"      "frg"
# [21] "gbni"     "greece"   "guatemal" "hungary"  "india"    "indonesi" "ireland"  "israel"   "italy"    "japan"
# [31] "kenya"    "korea"    "dprkorea" "luxembou" "malaysia" "mauritiu" "mexico"   "netherla" "nz"       "norway"
# [41] "png"      "philippi" "poland"   "portugal" "rumania"  "singapor" "spain"    "sweden"   "switzerl" "taipei"
# [51] "thailand" "turkey"   "usa"      "ussr"     "wsamoa"
#
# [[2]]
# [1] "m100"     "m200"     "m400"     "m800"     "m1500"    "m3000"    "Marathon"
```

To serialize (“save”) a specific object or an entire workspace, we can use:

- `save` writes an external representation of `R` objects to the specified file. The objects can be read back from the file at a later date by using the function `load` or `attach` (or `data` in some cases).

- `save.image` is short-cut for “save my current workspace”.

- `load`: Reload datasets written with the function `save`.

Note: the extension of the serialization file must be `.RData`

```
# to save several objects in the workspace
W <- matrix(data = c(11, 12, 13, 14, 15, 16), nrow = 2, ncol = 3, byrow = F)
V <- t(W)
a <- 1

save(W, V, a, file = ’variousobjects.RData’)

# to save the entire workspace: save.image(’FILENAME.RData’)
save.image("myworkspace.RData")

# to load a workspace (i.e., .RData)
load("variousobjects.RData")
```

We can also use the `ls` and `rm` combination to clean up the entire workspace:

- `ls` returns a vector of character strings giving the names of the objects in the specified environment. When invoked with no argument at the top level prompt, `ls` shows what data sets and functions a user has defined. When invoked with no argument inside a function, `ls` returns the names of the function’s local variables.

- `rm` can be used to remove objects.

```
# this command remove all the variable of the workspace
ls()
rm(list=ls())
```

3.1.8 Example: analysis of quantitative data

Given the raw file on page 44, we convert the values of the last 4 columns into seconds. Then we load it:

```r
record <- read.table('record.txt', header=T)

# Transform times in seconds
record[, 4:7] <- record[, 4:7] * 60

write.table(record, file = '1_IntroR/record_mod.txt')

record <- read.table('record_mod.txt', header=T)
record
#           m100  m200  m400  m800 m1500 m3000 Marathon
# argentin 11.61 22.94 54.50 129.0 265.8 587.4  10711.2
# australi 11.20 22.35 51.08 118.8 247.8 544.8   9142.2
# austria  11.43 23.09 50.62 119.4 253.2 560.4   9562.2
# belgium  11.41 23.04 52.00 120.0 248.4 532.8   9471.0
# bermuda  11.46 23.05 53.30 129.6 274.8 588.6  10198.8
# brazil   11.31 23.17 52.80 126.0 269.4 586.2  10125.0
# burma    12.14 24.47 55.00 130.8 267.0 570.6  11461.2
# canada   11.00 22.25 50.06 120.0 243.6 528.6   8967.0
# chile    12.00 24.52 54.90 123.0 253.8 562.2  10282.8
# china    11.95 24.41 54.97 124.8 259.8 558.6  10108.8
# columbia 11.60 24.00 53.26 126.6 261.0 567.6   9925.2
# cookis   12.90 27.10 60.40 138.0 290.4 666.0  13993.2
# costa    11.96 24.60 58.25 132.6 280.8 625.8  10308.0
# czech    11.09 21.97 47.99 113.4 248.4 535.2   9531.0
# denmark  11.42 23.52 53.60 121.8 250.8 522.6   9105.0
# domrep   11.79 24.05 56.05 134.4 284.4 593.4  12232.8
# finland  11.13 22.39 50.14 121.8 246.0 535.2   9253.8
# france   11.15 22.59 51.73 120.0 248.4 538.8   9316.2
# gdr      10.81 21.71 48.16 115.8 237.6 525.0   9460.8
# frg      11.01 22.39 49.75 117.0 241.8 515.4   8911.8
# gbni     11.00 22.13 50.46 118.8 241.8 517.2   8983.2
# greece   11.79 24.08 54.93 124.2 261.0 592.2  10932.0
# guatemal 11.84 24.54 56.09 136.8 291.6 632.4  12904.8
# hungary  11.45 23.06 51.50 120.6 248.4 538.8   9382.2
# india    11.95 24.28 53.60 126.0 259.2 598.8  11281.8
# indonesi 11.85 24.24 55.34 133.2 276.6 601.2  12076.8
# ireland  11.43 23.51 53.24 123.0 246.6 533.4   8962.8
# israel   11.45 23.57 54.90 126.0 255.0 562.2   9628.8
# italy    11.29 23.00 52.01 117.6 238.8 517.8   9109.2
# japan    11.73 24.00 53.73 125.4 261.0 552.0   9030.0
# kenya    11.73 23.88 52.70 120.0 249.0 552.0  10863.0
# korea    11.96 24.49 55.70 129.0 265.2 577.2   9879.0
# dprkorea 12.25 25.78 51.20 118.2 255.0 561.0  10750.2
# luxembou 12.03 24.96 56.10 124.2 262.8 578.4  10480.8
# malaysia 12.23 24.21 55.09 131.4 281.4 627.6  10930.2
# mauritiu 11.76 25.08 58.10 136.2 287.4 654.0  15667.8
# mexico   11.89 23.62 53.76 122.4 255.0 575.4   9511.8
# netherla 11.25 22.81 52.38 119.4 243.6 540.6   9148.8
# nz       11.55 23.13 51.60 121.2 250.8 525.6   8728.8
# norway   11.58 23.31 53.12 121.8 240.6 511.8   8728.8
# png      12.25 25.07 56.96 134.4 290.4 641.4  13980.0
# philippi 11.76 23.54 54.60 131.4 276.0 609.6  12022.2
# poland   11.13 22.21 49.29 117.0 239.4 538.2   9649.2
# portugal 11.81 24.22 54.30 125.4 249.6 530.4   9072.0
# rumania  11.44 23.46 51.20 115.2 237.6 511.8   9927.0
# singapor 12.30 25.00 55.08 127.2 271.2 596.4  10966.2
# spain    11.80 23.98 53.59 123.0 248.4 541.2   9756.0
# sweden   11.16 22.82 51.79 121.2 247.2 530.4   9268.8
# switzerl 11.45 23.31 53.11 121.2 244.2 526.2   9205.2
# taipei   11.22 22.62 52.50 126.0 262.8 577.8  10672.2
# thailand 11.75 24.46 55.80 132.0 283.2 616.8  10107.0
# turkey   11.98 24.44 56.45 129.0 262.2 562.8  12064.8
# usa      10.79 21.83 50.62 117.6 237.0 510.0   8563.2
# ussr     11.06 22.19 49.19 113.4 232.2 507.0   9073.2
# wsamoa   12.74 25.85 58.73 139.8 348.6 782.4  18360.0
```

Now we want to calculate: the means of each column (we will show two methods); the standard deviation; the variance; the covariance.

- `colMeans`: Form row and column sums and means for numeric arrays (or data frames).
- `apply`: Returns a list of the same length as `X` (input parameter), each element of which is the result of applying `FUN` (function, e.g. `mean`, `var`, etc.) to the corresponding element of `X`.
- `mean`: Arithmetic mean.
- `sd`: This function computes the standard deviation of the values in `x` (input parameter). If `na.rm` is `TRUE` then missing values are removed before computation proceeds.
- `var`: Computes the variance.
- `cov`: Computes the covariance.
- `cor`: Computes the correlation.

```
# some synthetic indices
colMeans(record)
sapply(record, mean)
sapply(record, sd)
sapply(record, var)
cov(record)
cor(record)
```

And the result is on the following page.

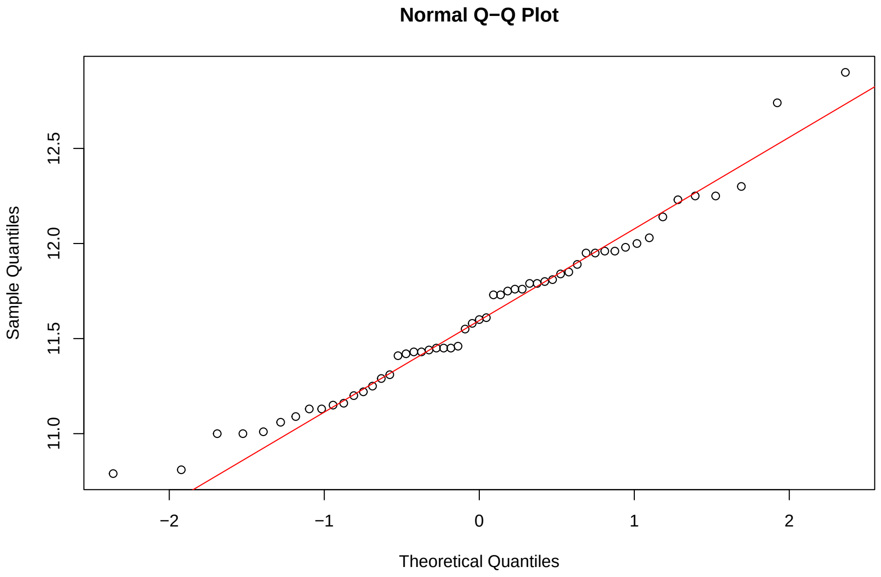
```
> # some synthetic indices
> colMeans(record)
       m100        m200        m400        m800       m1500       m3000    Marathon 
   11.61855    23.64164    53.40582   124.58182   259.52727   566.85818 10395.19636 

> sapply(record, mean)
       m100        m200        m400        m800       m1500       m3000    Marathon 
   11.61855    23.64164    53.40582   124.58182   259.52727   566.85818 10395.19636 

> sapply(record, sd)
        m100         m200         m400         m800        m1500        m3000     Marathon 
   0.4522103    1.1110602    2.6783367    6.4934466   19.9455319   49.4601474 1825.7726951 

> sapply(record, var)
        m100         m200         m400         m800        m1500        m3000     Marathon 
2.044941e-01 1.234455e+00 7.173488e+00 4.216485e+01 3.978242e+02 2.446306e+03 3.333446e+06 

> cov(record)
                m100         m200        m400        m800        m1500       m3000     Marathon
m100       0.2044941    0.4787135    1.010955    2.136788     6.569596    16.58912     566.6616
m200       0.4787135    1.2344547    2.550142    5.223808    15.476232    39.00968    1390.7176
m400       1.0109549    2.5501422    7.173488   15.624737    42.087172   103.01428    3449.5477
m800       2.1367879    5.2238081   15.624737   42.164848   116.772727   277.34848    9238.9436
m1500      6.5695960   15.4762323   42.087172  116.772727   397.824242   956.09394   31970.8279
m3000     16.5891232   39.0096808  103.014285  277.348485   956.093939  2446.30618   81258.0017
Marathon 566.6616242 1390.7175616 3449.547725 9238.943636 31970.827879 81258.00170 3333445.9341

> cor(record)
              m100      m200      m400      m800     m1500     m3000  Marathon
m100     1.0000000 0.9527911 0.8346918 0.7276888 0.7283709 0.7416988 0.6863358
m200     0.9527911 1.0000000 0.8569621 0.7240597 0.6983643 0.7098710 0.6855745
m400     0.8346918 0.8569621 1.0000000 0.8984052 0.7878417 0.7776369 0.7054241
m800     0.7276888 0.7240597 0.8984052 1.0000000 0.9016138 0.8635652 0.7792922
m1500    0.7283709 0.6983643 0.7878417 0.9016138 1.0000000 0.9691690 0.8779334
m3000    0.7416988 0.7098710 0.7776369 0.8635652 0.9691690 1.0000000 0.8998374
Marathon 0.6863358 0.6855745 0.7054241 0.7792922 0.8779334 0.8998374 1.0000000
```

Univariate t-test for the mean value of the quantity

We print a quantile-quantile plot[2] (`qqplot`) to verify (qualitatively) the Gaussian assumption on the distribution generating sample.

- `qqnorm` is a generic function the default method of which produces a normal QQ plot of the values in `y` (input parameter).
- `qqline` adds a line to a “theoretical”, by default normal, quantile-quantile plot which passes through the probs quantiles, by default the first and third quartiles.

```r
qqnorm(record$m100) # quantile-quantile plot
qqline(record$m100, col=’red’) # theoretical line
```



Furthermore, we use Shapiro-Wilk test[3] to verify (quantitatively) the Gaussian assumption on the distribution generating sample.

```r
shapiro.test(record$m100)
#   Shapiro-Wilk normality test
#
# data:  record$m100
# W = 0.97326, p-value = 0.2569
```

- `W`: the value of the Shapiro-Wilk statistic.
- `p.value`: an approximate p-value for the test. This is said in Royston (1995) to be adequate for `p.value < 0.1`.
- `data.name`: a character string giving the name(s) of the data.

[2] In statistics, a Q-Q plot (quantile-quantile plot) is a probability plot, a graphical method for comparing two probability distributions by plotting their quantiles against each other. [3]

[3] The Shapiro-Wilk test is a test of normality. In statistics, normality tests are used to determine if a data set is well-modeled by a normal distribution and to compute how likely it is for a random variable underlying the data set to be normally distributed (source).

The null-hypothesis of this test is that the population is normally distributed. Thus, if:

- The `p.value` is **less than the chosen `alpha`** level, then the null hypothesis is **rejected** and there is evidence that the data tested are not normally distributed.
- The `p.value` is **greater than the chosen `alpha`** level, then the null hypothesis (that the data came from a normally distributed population) can not be rejected.

So in our case the data is normally distributed because the alpha level is 0.05.

Now, we perform the Student's t-Test[4]. The arguments:

- `x` a non-empty numeric vector of data values.
- `alternative` a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`
- `mu` a number indicating the true value of the mean (or difference in means if you are performing a two sample test).
- `conf.level` confidence level of the interval.

The output values:

- `t`: the value of the t-test.
- `df`: the degrees of freedom for the t-test.
- `p-value`: the p-value for the test.
- `confidence interval`: a confidence interval for the mean appropriate to the specified alternative hypothesis.
- `estimate`: the estimated mean or difference in means depending on whether it was a one-sample test or a two-sample test.

See the code in the following page.

[4] Student's t-Test is a statistical test used to test whether the difference between the response of two groups is statistically significant or not. A **one-sample** Student's t-test is a location test of whether the mean of a population has a value specified in a null hypothesis. In testing the null hypothesis that the population mean is equal to a specified value μ_0, one uses the statistic:

$$t = \frac{\overline{x} - \mu_0}{s \div \sqrt{n}}$$

Where $\overline{x}$ is the sample mean, s is the sample standard deviation and n is the sample size. The degrees of freedom used in this test are $n - 1$. Although the parent population does not need to be normally distributed, the distribution of the population of sample means $\overline{x}$ is assumed to be normal.

```r
alpha <- .05
mean.H0 <- 11.5

# automatically
t.test(record$m100, mu = mean.H0, alternative = 'two.sided', conf.level = 1-alpha)
#   One Sample t-test
#
# data:  record$m100
# t = 1.9441, df = 54, p-value = 0.0571
# alternative hypothesis: true mean is not equal to 11.5
# 95 percent confidence interval:
#  11.4963 11.7408
# sample estimates:
# mean of x
#  11.61855
```

We can also run the Student's t-test manually:

```r
sample.mean <- mean(record$m100)
sample.sd <- sd(record$m100)
n <- length(record$m100)
tstat <- (sample.mean - mean.H0) / (sample.sd / sqrt(n))
cfr.t <- qt(1 - alpha/2, n-1)
abs(tstat) < cfr.t  # cannot reject H0 (accept H0)
# [1] TRUE

pval  <- ifelse(tstat >= 0, (1 - pt(tstat, n-1))*2, pt(tstat, n-1)*2)
pval
# [1] 0.05709702

IC <- c(inf      = sample.mean - sample.sd / sqrt(n) * qt(1 - alpha/2, n-1),
        center   = sample.mean,
        sup      = sample.mean + sample.sd / sqrt(n) * qt(1 - alpha/2, n-1))
IC
#       inf    center       sup
# 11.49630 11.61855 11.74080
```

Simple linear regression

We want to calculate the simple linear regression[5] of the property `m100` and of the property `m200`. We then want to create a scatter plot to see the graphical relationship between these two sets of data. We will also create a linear model to calculate the true linear regression.

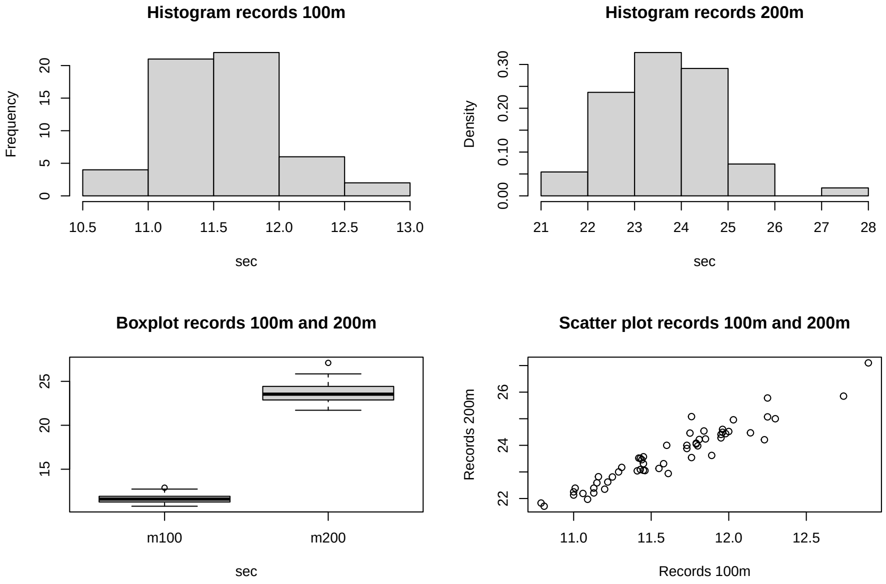
In this section, `m100` e `m200` are properties of the `record` object created on page 47.

First, we plot `m100` and `m200` using:

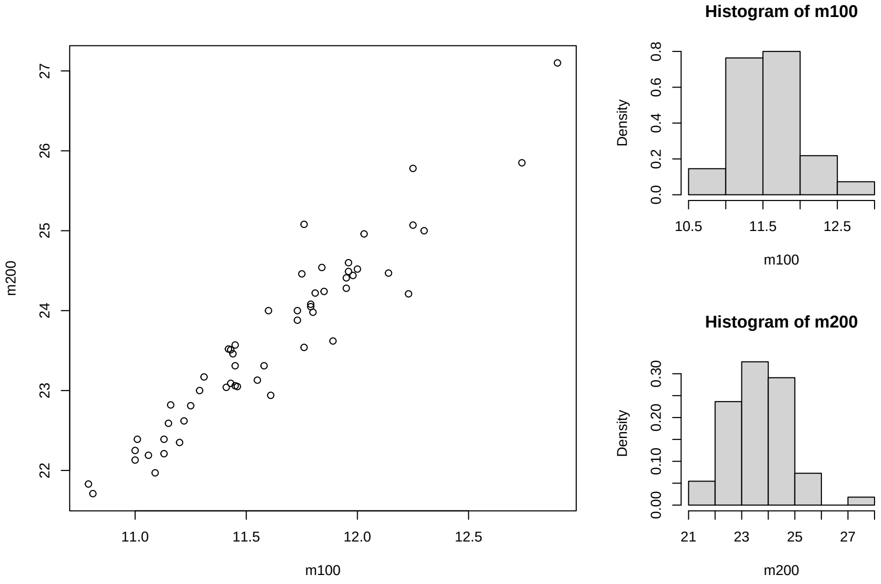
- `par`: prepares the graphical environment. It is a powerful tool and in this case it is used to create a 2 × 2 matrix so that each graph can be placed in just one figure. It's also possible to use the `layout` function.
- `hist`: The generic function `hist` computes a histogram of the given data values. With the arguments:
  - `main`, `xlab`, `ylab`: main title and axis labels.
  - `par`: is for `S` compatibility (not required)
- `boxplot`: produce box-and-whisker plot(s) of the given (grouped) values.
- `plot`: generic function for plotting of `R` objects.

```r
# More than one plot in a unique device (commands par or layout)
# (command par)
par(mfrow=c(2, 2))
hist(m100, main="Histogram records 100m", xlab="sec")
hist(m200, prob=T, main="Histogram records 200m", xlab="sec")
boxplot(record[,1:2], main="Boxplot records 100m and 200m",
        xlab="sec")
plot(m100, m200, main='Scatter plot records 100m and 200m',
     xlab="Records 100m", ylab="Records 200m")

dev.off() # Clear plot

# (command layout)
layout(cbind(c(1, 1), c(2, 3)), widths=c(2, 1), heights=c(1, 1))
plot(m100, m200)
hist(m100, prob=T)
hist(m200, prob=T)
```

[5] In statistics, simple linear regression (SLR) is a linear regression model with a single explanatory variable. That is, it concerns two-dimensional sample points with one independent variable and one dependent variable (conventionally, the x and y coordinates in a Cartesian coordinate system) and finds a linear function (a non-vertical straight line) that, as accurately as possible, predicts the dependent variable values as a function of the independent variable. The adjective simple refers to the fact that the outcome variable is related to a single predictor.



Using `par` command



Using `layout` command

Now we calculate the regression using the function:

- `lm` is used **to fit linear models**, including multivariate ones. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance. Arguments:
  - `formula`: a symbolic description of the model to be fitted.

Models for `lm` are specified symbolically. A typical model has the form `response` ∼ `terms` where

- `response` is the (numeric) response vector.
- `terms` is a series of terms which specifies a linear predictor for response.

A terms specification of the form:

- `first + second` indicates all the terms in `first` together with all the terms in `second` with duplicates removed.
- `first:second` indicates the set of terms obtained by taking the interactions of all terms in `first` with all terms in `second`.
- `first*second` indicates the *cross* of `first` and `second`. This is the same as `first + second + first:second`.

Note 1: tilde ∼ is used to separate the left- and right-hand sides in a model formula.
Note 2: `summary` is a generic function used to produce result summaries of the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.

```
# Fit of the linear model (command lm)
# Model: m200 = beta0 + beta1 * m100 + eps, eps ~ N(0, sigma^2)
regression <- lm(m200 ~ m100)
regression
# Call:
# lm(formula = m200 ~ m100)
#
# Coefficients:
# (Intercept)         m100
#      -3.557        2.341

summary(regression)
# Call:
# lm(formula = m200 ~ m100)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -0.86303 -0.16559 -0.00756  0.16599  1.10722
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)  -3.5570     1.1914  -2.985  0.00428 **
# m100          2.3410     0.1025  22.845  < 2e-16 ***
# ---
# Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
#
# Residual standard error: 0.3405 on 53 degrees of freedom
# Multiple R-squared:  0.9078,  Adjusted R-squared:  0.9061
# F-statistic: 521.9 on 1 and 53 DF,  p-value: < 2.2e-16
```

With the regression it is possible to calculate some interesting values:

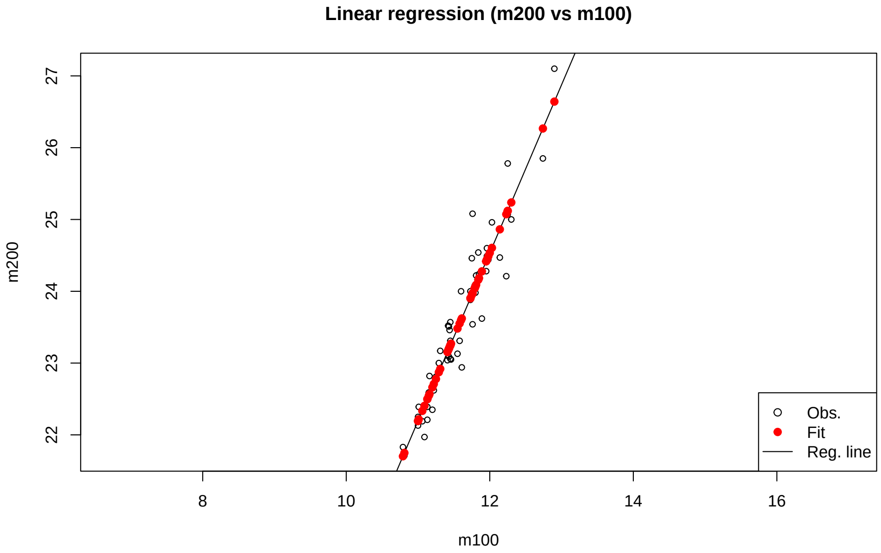
- `coef` is a generic function which **extracts model coefficients** from objects returned by modeling functions. `coefficients` is an alias for it.
- `vcov` returns the **variance-covariance** matrix of the main parameters of a fitted model object. The “main” parameters of model correspond to those returned by `coef`, and typically do not contain a nuisance scale parameter (sigma).
- `residuals` is a generic function which **extracts model residuals** from objects returned by modeling functions.
- `fitted` is a generic function which **extracts fitted values** from objects returned by modeling functions. `fitted.values` is an alias for it.

In the following code we use:

- `abline` adds one or more straight lines through the current plot.
- `points` is a generic function to draw a sequence of points at the specified coordinates. The specified character(s) are plotted, centered at the coordinates.
- `legend` used to add legends to plots. Arguments:
  - `x` (first argument), `y` (second argument): the x and y co-ordinates to be used to position the legend.
  - `col`: the color of points or lines appearing in the legend.
  - `lty`, `lwd`: the line types and widths for lines appearing in the legend. One of these two *must* be specified for line drawing.
  - `pch`: the plotting symbols appearing in the legend, as numeric vector or a vector of 1-character strings (see `points`). Unlike `points`, this can all be specified as a single multi-character string. *Must* be specified for symbol drawing.

```
coef(regression)
# (Intercept)        m100
#   -3.556967    2.340965

vcov(regression)
#             (Intercept)        m100
# (Intercept)   1.4195378 -0.12199717
# m100         -0.1219972  0.01050021

residuals(regression)
#            1            2            3            4            5
               6            7            8            9
# -0.681631757 -0.311836293 -0.110258139 -0.113438848 -0.220487075
     0.250657607 -0.392342968  0.056356617 -0.014607931
#           10           11           12           13           14
              15           16           17           18
# -0.007559704  0.401777888  0.458523975  0.159030651 -0.434330192
     0.343151507  0.006994624 -0.107968774  0.045211935
#           19           20           21           22           23
              24           25           26           27
```

```
# -0.038860119  0.172946971 -0.063643383  0.036994624  0.379946396
    -0.187077430 -0.137559704  0.056536751  0.309741861
#           28           29           30           31           32
              33           34           35           36
#  0.322922570  0.127476898  0.097452497 -0.022547503  0.049030651
     0.660150932  0.355163132 -0.863029777  1.107223560
#           37           38           39           40           41
              42           43           44           45
# -0.657101831  0.031115480 -0.351173885 -0.241402821 -0.049849068
    -0.432776440 -0.287968774  0.130175333  0.236332216
#           46           47           48           49           50
              51           52           53           54
# -0.236897296 -0.086415022  0.251802289  0.062922570 -0.088655584
     0.510633206 -0.047788640  0.127959172 -0.144101256
#           55
# -0.416921697

fitted(regression)
#        1        2        3        4        5        6        7
         8        9       10       11       12       13
# 23.62163 22.66184 23.20026 23.15344 23.27049 22.91934 24.86234
    22.19364 24.53461 24.41756 23.59822 26.64148 24.44097
#       14       15       16       17       18       19       20
        21       22       23       24       25       26
# 22.40433 23.17685 24.04301 22.49797 22.54479 21.74886 22.21705
    22.19364 24.04301 24.16005 23.24708 24.41756 24.18346
#       27       28       29       30       31       32       33
        34       35       36       37       38       39
# 23.20026 23.24708 22.87252 23.90255 23.90255 24.44097 25.11985
    24.60484 25.07303 23.97278 24.27710 22.77888 23.48117
#       40       41       42       43       44       45       46
        47       48       49       50       51       52
# 23.55140 25.11985 23.97278 22.49797 24.08982 23.22367 25.23690
    24.06642 22.56820 23.24708 22.70866 23.94937 24.48779
#       53       54       55
# 21.70204 22.33410 26.26692

# print the linear regression
plot(m100, m200, asp=1, cex=0.75)
abline(coef(regression))
points(m100, fitted(regression), col='red', pch=19)

legend(
  'bottomright',
  c('Obs.', 'Fit', 'Reg. line'),
  col = c('black', 'red', 'black'),
  lwd = c(1, 1, 1),
  lty = c(-1, -1, 1),
  pch = c(c(1, 19, -1))
)

title(main='Linear regression (m200 vs m100)')
```



We can do the F-test[6] by hand:

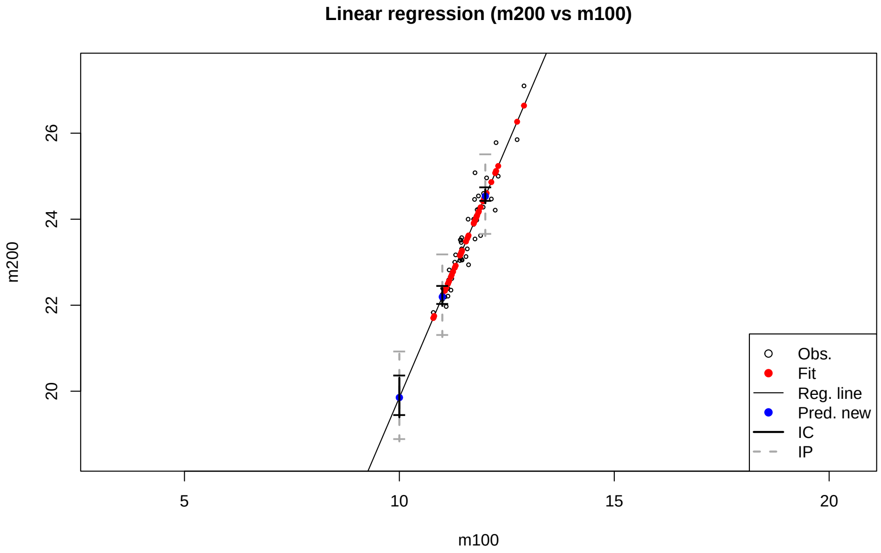
```
# Test F "by hand" (H0: beta1=0 vs H1: beta1!=0)
SSreg <- sum((fitted(regression) - mean(m200))^2)
SSres <- sum(residuals(regression)^2)
SStot <- sum((m200 - mean(m200))^2)

n <- length(m200)
Fstat <- (SSreg/1) / (SSres/(n-2))
P <- 1 - pf(Fstat, 1, n-2)
P # reject H0
# [1] 0
```

Furthermore, we can calculate confidence and prediction interval using `predict` command:

- `predict` is a generic function for predictions from the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.

[6] An F-test is any statistical test used to compare the variances of two samples or the ratio of variances between multiple samples. The test statistic, random variable F, is used to determine if the tested data has an F-distribution under the true null hypothesis, and true customary assumptions about the error term (ε). It is most often used when comparing statistical models that have been fitted to a data set, in order to identify the model that best fits the population from which the data were sampled.

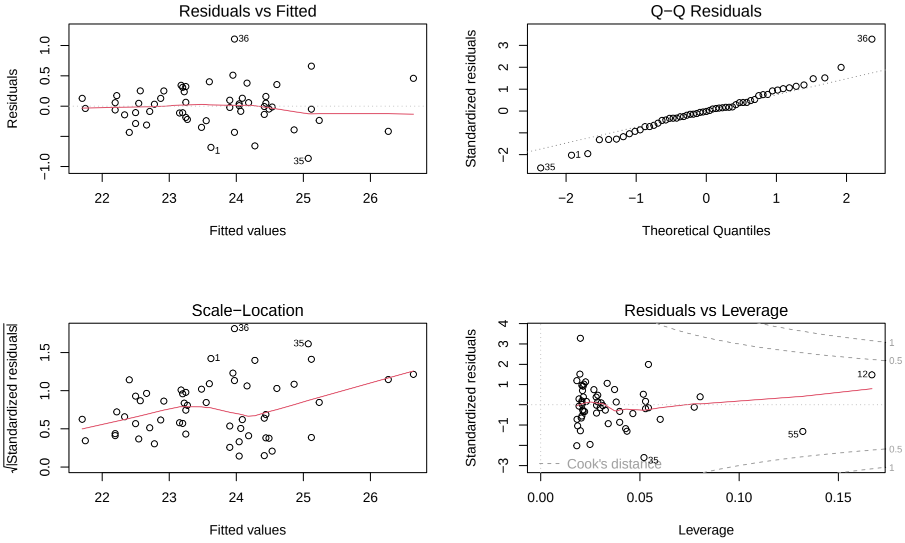
```r
newdata <- data.frame(m100=c(10, 11, 12))
pred_nd <- predict(regression, newdata)
pred_nd
#         1        2        3
# 19.85268 22.19364 24.53461

IC_nd <- predict(regression, newdata, interval='confidence', level=.99)
IC_nd
#         fit      lwr      upr
# 1 19.85268 19.39288 20.31248
# 2 22.19364 21.98453 22.40276
# 3 24.53461 24.37350 24.69572

IP_nd <- predict(regression, newdata, interval='prediction', level=.99)
IP_nd
#         fit      lwr      upr
# 1 19.85268 18.83330 20.87206
# 2 22.19364 21.26013 23.12716
# 3 24.53461 23.61066 25.45856

plot(m100, m200, asp=1, ylim=c(18.5, 27.5), cex=0.5)
abline(coef(regression))
points(m100, fitted(regression), col='red', pch=20)
points(c(10, 11, 12), pred_nd, col='blue', pch=16)

matlines(rbind(c(10, 11, 12), c(10, 11, 12)), t(IP_nd[, -1]), type="l", lty=2,
         col='dark grey', lwd=2)
matpoints(rbind(c(10, 11, 12), c(10, 11, 12)), t(IP_nd[, -1]), pch="-", lty=2,
          col='dark grey', lwd=2, cex=1.5)
matlines(rbind(c(10, 11, 12), c(10, 11, 12)), t(IC_nd[, -1]), type="l", lty=1,
         col='black', lwd=2)
matpoints(rbind(c(10, 11, 12), c(10, 11, 12)), t(IC_nd[, -1]), pch="-", lty=1,
          col='black', lwd=2, cex=1.5)

legend(
  'bottomright',
  c('Obs.', 'Fit', 'Reg. line', 'Pred. new', 'IC', 'IP'),
  col = c('black', 'red', 'black', 'blue', 'black', 'dark grey'),
  lwd = c(1, 1, 1, 1, 2, 2),
  lty = c(-1, -1, 1, -1, 1, 2),
  pch = c(c(1, 19, -1, 19, -1, -1))
)

title(main='Linear regression (m200 vs m100)')
```



Linear regression (m200 vs m100)
m200
20
22
24
26
m100
5
10
15
20
Obs.
Fit
Reg. line
Pred. new
IC
IP

Once we’ve fitted a regression model, it’s a good idea to also produce **diagnostic plots** to analyze the residuals of the model and make sure that a linear model is appropriate for the particular data we’re working with.

```
# diagnostic of residuals
par(mfrow=c(2, 2))
boxplot(residuals(regression), main=’Boxplot of residuals’)
qqnorm(residuals(regression))
plot(m100, residuals(regression), main=’Residuals vs m100’)
abline(h=0, lwd=2)
plot(fitted(regression), residuals(regression), main=’Residuals vs
    fitted m200’)
abline(h=0, lwd=2)

par(mfrow=c(2, 2))
plot(regression)
```



- `Residuals vs Fitted`. This plot is used to determine if the **residuals exhibit non-linear patterns**. If the red line across the center of the plot is roughly horizontal then we can assume that the residuals follow a linear pattern.

  In our example, we can see that the red line is almost the same as the perfect horizontal line. We declare that the residuals follow a roughly linear pattern and that a linear regression model is appropriate for this dataset.

- `Q-Q residuals`. This plot is used to determine if the **residuals of the regression model are normally distributed**. If the points in this plot fall roughly along a straight diagonal line, then we can assume the residuals are normally distributed.

  In our example, we can see that the points fall roughly along the straight diagonal line. We can then assume that the residuals are normally distributed.

- `Scale-Location`. This plot is used to **check the assumption of equal variance** (also called homoscedasticity) **among the residuals in our regression model**. If the red line is roughly horizontal across the plot, then the assumption of equal variance is likely met.

  In our example, we can see that the red line isn't exactly horizontal across the plot, but it doesn't deviate too much at any point. We would probably say that the assumption of equal variance is not violated in this case.

- `Residuals vs Leverage`. This plot is used to **identify influential observations**. If any points in this plot fall outside of Cook's distance[7] (the dashed lines) then it is an influential observation.

  In our example we can see that observations 12, 35 and 55 are closest to the boundary of Cook's distance, but they don't fall outside the dashed line. This means that there aren't any overly influential points in our data set.

A useful guide to follow when creating diagnostic plots: How to Interpret Diagnostic Plots in R.

[7] In statistics, Cook's distance or Cook's D is a commonly used estimate of the influence of a data point when performing a least-squares regression analysis. In a practical ordinary least squares analysis, Cook's distance can be used in several ways: to indicate influential data points that are particularly worth checking for validity; or to indicate regions of the design space where it would be good to be able to obtain more data points.

References

[1] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer New York, 2013.

[2] R.A. Johnson and D.W. Wichern. *Applied Multivariate Statistical Analysis*. Applied Multivariate Statistical Analysis. Pearson Prentice Hall, 2007.

[3] M. B. WILK and R. GNANADESIKAN. Probability plotting methods for the analysis for the analysis of data. *Biometrika*, 55(1):1–17, 03 1968.

Index