Subject CS2

CMP Upgrade 2021/22

CMP Upgrade

This CMP Upgrade lists the changes to the Syllabus objectives, Core Reading and the ActEd material since last year that might realistically affect your chance of success in the exam. It is produced so that you can manually amend your 2021 CMP to make it suitable for study for the 2022 exams. It includes replacement pages and additional pages where appropriate.

Alternatively, you can buy a full set of up-to-date Course Notes / CMP at a significantly reduced price if you have previously bought the full-price Course Notes / CMP in this subject. Please see our 2022 *Student Brochure* for more details.

This upgrade does not contain amendments to the assignments. We only accept the current version of assignments for marking, *ie* those published for the sessions leading to the 2022 exams. If you wish to submit your script for marking but have only an old version, then you can order the current assignments free of charge if you have purchased the same assignments in the same subject in a previous year, and have purchased marking for the 2022 session.

Alternatively, if you wish to purchase the 2022 assignments, then you can do so at a significantly reduced rate. Again, please see our 2022 *Student Brochure* for more details.

This CMP Upgrade contains:

- all significant changes to the Syllabus objectives and Core Reading
- additional changes to the accompanying ActEd Course Notes that will make them suitable for study for the 2022 exams.

1 Changes to the Syllabus

This section contains all the ***non-trivial*** changes to the syllabus objectives.

The objectives for machine learning have been changed to the following:

Machine learning (10%)

5.1 Explain and apply elementary principles of machine learning. (Chapter 21)

5.1.1 Explain the bias/variance trade-off and its relationship with model complexity.

5.1.2 Use cross-validation to evaluate models on unseen data, and to estimate hyperparameters.

5.1.3 Explain how regularisation can be used to reduce overfitting in highly parameterised models.

5.1.4 Use software to apply supervised learning techniques to solve regression and classification problems.

5.1.5 Use metrics such as precision, recall, F_1 score and diagnostics such as the ROC curve and confusion matrix to evaluate the performance of a binary classifier.

5.1.6 Apply unsupervised learning techniques (principal component analysis and K-means clustering) to reduce data dimensionality, identify latent substructure and detect anomalies.

2 Changes to the Core Reading

This section contains all the ***non-trivial*** changes to the Core Reading.

Chapter 6

Section 4, pages 21-22

The R boxes on these pages have been updated. Replacement pages are included at the end of this document.

Chapter 10

Section 7.5, page 38

The Core Reading describing the probability of obtaining exactly t positive groups has been updated to reference the hypergeometric distribution. The updated Core Reading and following ActEd paragraph now reads:

(c) There are $\binom{m}{n_1}$ ways to arrange n_1 positive and n_2 negative *signs*,

since, by definition, $m = n_1 + n_2$.

Hence, the probability of exactly t positive groups can be obtained from the hypergeometric distribution:

$$P(G=t) = \frac{\binom{n_1-1}{t-1}\binom{n_2+1}{t}}{\binom{n_1+n_2}{n_1}}$$

This formula is given on page 34 of the *Tables*. G follows a hypergeometric distribution and can be thought of as the number of 'successes' drawn from a finite population with n_2+1 total 'successes' and n_1-1 total 'failures' when n_1 total items are drawn.

Chapter 12

Section 2.5, page 31

The expression for the penalised log-likelihood has been updated to remove the factor of $\frac{1}{2}$ (this also impacts ActEd text on page 43 of the summary and page 54 of the practice questions). The Core Reading now reads:

- **Estimate the parameters of the model, including the number of splines, by maximising the penalised log-likelihood**

$$l_p(\theta) = l(\theta) - \lambda P(\theta)$$

where $l(\theta)$ is the log-likelihood from model (4).

Section 2.5, page 33

The reference to the `MortalitySmooth` package has been removed. The following R box is no longer in the Core Reading:

***p*-spline forecasting for single years of age or for many ages simultaneously can be carried out using the `MortalitySmooth` package in R.**

Chapter 13

Section 3.3, page 27

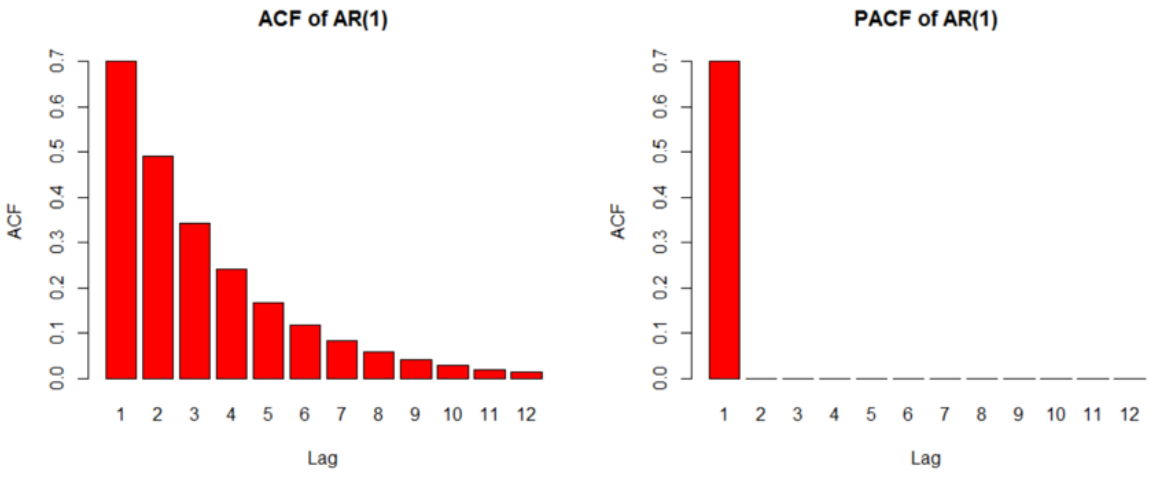
The R box on this page has been updated to the following:

The following lines in R calculate the ACF and PACF functions up to lag 12 for an $AR(1)$ model with $\alpha = 0.7$:

```
    ARMAacf(ar = 0.7, lag.max = 12)

          0           1           2           3           4
1.00000000 0.70000000 0.49000000 0.34300000 0.24010000
          5           6           7           8           9
0.16807000 0.11764900 0.08235430 0.05764801 0.04035361
         10          11          12
0.02824752 0.01977327 0.01384129

    ARMAacf(ar = 0.7, lag.max = 12, pacf = TRUE)

[1] 0.7 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

The following lines in R plot ACF and PACF functions up to lag 12 as bar plots:

```
      par(mfrow = c(1, 2))

      barplot(ARMAacf(ar = 0.7, lag.max = 12)[-1],
              xlab = "Lag", ylab = "ACF",
              main = "ACF of AR(1)", col = "red")

      barplot(ARMAacf(ar = 0.7, lag.max = 12, pacf = TRUE),
              xlab = "Lag", ylab = "ACF",
              main = "PACF of AR(1)", col = "red", names.arg = 1:12)
```



Note that the value of ρ_0 (= 1) has been excluded from the ACF bar plot using negative indexing, *ie* using the code `[-1]`.

Figure 13.2: ACF and PACF of $AR(1)$ with $\alpha = 0.7$.

Section 3.5, page 39

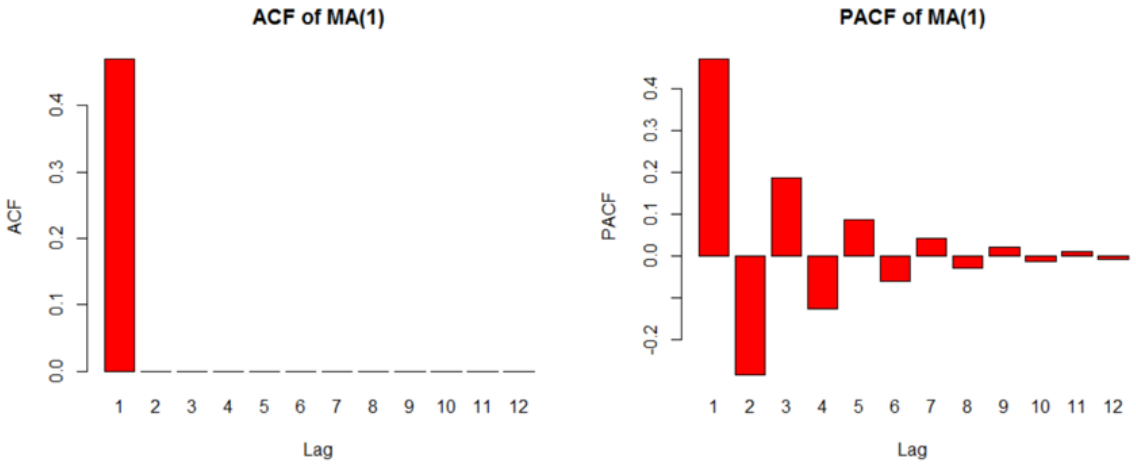
The R box on this page has been updated to the following:

The following lines in R generate the ACF and PACF functions up to lag 12 for an *MA*(1) model with $\beta = 0.7$:

```
      ARMAacf(ma = 0.7, lag.max = 12)

        0         1         2         3         4
1.0000000 0.4697987 0.0000000 0.0000000 0.0000000
        5         6         7         8         9
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
       10        11        12
0.0000000 0.0000000 0.0000000

      ARMAacf(ma = 0.7, lag.max = 12, pacf = TRUE)
[1]  0.469798658 -0.283220623  0.185631274
[4] -0.126010484  0.086918768 -0.060410709
[7]  0.042140739 -0.029448439  0.020596774
[10] -0.014411872  0.010086299 -0.007059719
```

The following lines in R plot the ACF and PACF functions up to lag 12 as bar plots:

```
    par(mfrow = c(1,2))

    barplot(ARMAacf(ma=0.7,lag.max = 12)[-1],
            xlab = "Lag", ylab = "ACF", main="ACF of MA(1)",
            col = "red")

    barplot(ARMAacf(ma=0.7,lag.max = 12,pacf = TRUE),
            xlab = "Lag",   ylab = "PACF", main = "PACF of MA(1)",
            col = "red", names.arg = 1:12)
```



Note that the value of ρ_0 (= 1) has been excluded from the ACF bar plot using negative indexing, *ie* using the code `[-1]`.

Figure 13.3: ACF and PACF of *MA*(1) with $\beta = 0.7$

Chapter 14

Section 4.2, page 37

The R box on this page has been updated to the following:

Predicting three steps ahead in R using the *ARIMA*(1,0,1) model fitted to the data generated in Section 2.1:

```
    predict(fit, n.ahead = 3)

$pred
Time Series:
Start = 301
End = 303
Frequency = 1
[1] 1.1164950 0.7184494 0.4749197

$se
Time Series:
Start = 301
End = 303
Frequency = 1
[1] 0.9495467 1.4808587 1.6359396
```

The `$pred` component of the output contains the predicted values and the `$se` component contains estimated standard errors.

The following code outputs predictions and estimated standard errors for 15 to 20 steps ahead:

```
    predict(fit, n.ahead = 20)$pred[15:20]

[1] 0.09215216 0.09174233 0.09149159 0.09133819 0.09124433 0.09118691

    predict(fit, n.ahead = 20)$se[15:20]

[1] 1.722052 1.722053 1.722053 1.722053 1.722053 1.722053
```

As indicated in Section 4.1, the predicted values and standard errors are converging.

The predicted values are converging to the estimated mean of the process, which is 0.0911 to 4 decimal places.

The standard errors are converging to 1.722053, which is the square root of γ_0 of the fitted model.

Chapter 15

Section 1.1, pages 7-10

The R boxes on these pages have been updated. Replacement pages are included at the end of this document.

Section 2.1, page 12

The R box on this page has been updated to the following:

The built in R functions use σ as the second parameter instead of σ^2.

Example R code for simulating a random sample of 100 values (with a seed of 3) **from the lognormal distribution with mean and standard deviation (on the logarithmic scale) of 0 and 1, respectively, is:**

```
set.seed(3)
log.norm.sample = rlnorm(100, 0, 1)
```

Sample moments can then be calculated as well as the sample histogram plotted with the `hist()` **function. For example, the mean of this sample is:**

```
mean(log.norm.sample)
```

```
[1] 1.414397
```

Similarly, the PDF, CDF and quantiles can be obtained using the R functions `dlnorm()`**,** `plnorm()` **and** `qlnorm()`**.**

Section 2.2, page 14

The R box on this page has been updated to the following:

There is no built in R code for the Pareto distribution (in the basic R installation) so we have to define the functions `rpareto()`, `dpareto()`, `ppareto()` **and** `qpareto()` **from first principles as follows:**

```
rpareto <- function(n, a, lambda){
      lambda * ((1 - runif(n))^(-1/a)-1)
}

dpareto <- function(x, a, lambda){
      a * lambda^(a)/((lambda + x)^(a+1))
}

ppareto <- function(q, a, lambda){
      1 - (lambda/(lambda + q))^a
}

qpareto <- function(p, a, lambda){
      lambda * ((1 - p)^(-1/a)-1)
}
```

Example R code for simulating a random sample of 100 values (with a seed of 4) **from the Pareto distribution with parameters $\alpha = 3$ and $\lambda = 200$ is:**

```
set.seed(4)
pareto.sample = rpareto(100, 3, 200)
```

Sample moments can then be calculated as well as the sample histogram plotted with the `hist()` **function. For example, the mean of this sample is:**

```
mean(pareto.sample)
```

```
[1] 120.12
```

Section 3, pages 21-30

The R boxes on these pages have been updated. Replacement pages are included at the end of this document.

Chapter 16

Section 3.2, page 22

The first paragraph has been updated and a line of ActEd text added underneath. It now reads:

More generally we find that, for a large class of underlying distributions of the data, the distribution of the threshold exceedances more closely resembles a generalised Pareto distribution as the threshold u increases towards x_F.

Recall that x_F is the maximum possible value of the underlying distribution.

Chapter 18

Section 1.1, page 8

The R box on this page has been updated to the following:

Suppose claims (in £'s) have an exponential distribution with parameter $\lambda = 0.0005$. The R code for simulating 10,000 claims, `x`, is given by:

```
x = rexp(10000, 0.0005)
```

We can obtain the simulated amounts paid by the insurer, `y`, using the vectorised minimum function `pmin()` to compare each claim value with the retention limit, `M`:

```
y = pmin(x,M)
```

Similar code can be used to obtain the simulated reinsurer payments, `z`, using the vectorised maximum function `pmax()`:

```
z = pmax(0, x - M)
```

For example, suppose that the retention limit is 3000:

```
set.seed(1)
x = rexp(10000, 0.0005)
M = 3000

y = pmin(x, M)
z = pmax(0, x - M)
```

We can then obtain the simulated means and variances using the R functions `mean()` and `var()`. For example:

```
mean(y)

[1] 1543.317

var(y)

[1] 1125872
```

```
    mean(z)
[1] 453.4059
    var(z)
[1] 1679221
```

We can use these vectors to estimate probabilities. For example, we can use the following code to estimate the probability that the insurer pays less than £1,000 on any given claim:

```
    length(y[y < 1000]) / length(y)
[1] 0.3949
```

The estimated probability is 39.49%.

Section 4, page 24

The R box on this page has been updated. Replacement pages for pages 23-26 are included at the end of this document.

Chapter 19

Section 3.3, page 12

The Core Reading paragraph describing formula 19.5 has been updated to the following:

Formula (19.5) is the law of total variance, which can be seen as decomposing variability in S into two distinct components. The first term represents the variability in S due to variability between individual claims, and the second term is the variability in S attributable to variability in the number of claims.

Section 3.4, page 15

The equation near the bottom of the page showing that $\frac{d^3}{dt^3}\log M_S(t)\Big|_{t=0}$ is equal to λm_3 is incorrect. It has been updated to:

$$\frac{d^3}{dt^3}\log M_S(t)\Big|_{t=0} = \lambda\left[\frac{d^3}{dt^3}\left(M_X(t)-1\right)\right]_{t=0} = \lambda m_3$$

Section 3.3, pages 16

The R box on this page has been updated. Replacement pages are included for pages 15-18 at the end of this document.

Section 3.6, page 22

The following Core Reading paragraph has been added under the probability function for the negative binomial distribution:

In this form, N counts the number failures before k successes are observed in a sequence of independent Bernoulli trials, where the probability of an individual success is p.

Chapter 20

Section 1.2, page 9

The R box on this page has been updated to the following:

To simulate the collective risk model with *individual* reinsurance we can combine the R code from Chapters 15, 18 and 19.

For example, to simulate 10,000 values for a reinsurer where claims have a compound Poisson distribution with parameter 1,000 and a gamma claims distribution with $\alpha = 750$ and $\lambda = 0.25$ under *individual* excess of loss with retention 2,500 we could use:

```
set.seed(123)

sims = 10000
M  = 2500

n = rpois(sims, 1000)
sR = rep(NA, sims)
for(i in 1:sims){
      x = rgamma(n[i], shape = 750, rate = 0.25)
      z = pmax(0, x - M)
      sR[i] = sum(z)
}
```

We can now estimate moments, the coefficient of skewness, probabilities and quantiles as before. For example, the sample mean and variance are:

```
mean(sR)

[1] 499588.8

var(sR)

[1] 257183130
```

Section 1.3, page 14

The R box on this page has been updated to the following:

Suppose aggregate claims have a compound Poisson distribution with parameter 1,000 and a gamma claims distribution with $\alpha = 750$ and $\lambda = 0.25$.

To simulate 10,000 values for a reinsurer under *aggregate* excess of loss with retention limit 3,000,000, we could take the simulations of the aggregate underlying claims, S, from Section 3.4 of Chapter 19 and use the `pmax()` function:

```
      sR.agg = pmax(0, s - 3000000)
```

We can now estimate moments, the coefficient of skewness, probabilities and quantiles as before. For example, the sample mean and variance are:

```
      mean(sR.agg)
[1] 36153.85
      var(sR.agg)
[1] 2932084069
```

Chapter 21

There were significant changes to the Core Reading and accompanying ActEd text throughout this chapter. The full new chapter is included at the end of this document.

3 Changes to the ActEd material

This section contains all the ***non-trivial*** changes to the ActEd text.

Chapter 10

Section 7.5, page 38

A question has been added to show the relationship between the number of positive groups and the hypergeometric distribution. The question is as follows:

Question

Consider a bag containing n_2+1 blue balls (successes) and n_1-1 red balls (failures). Let B be the random variable representing the number of blue balls drawn when drawing a total of n_1 balls without replacement.

Show that B has the same distribution as G by considering the probability of drawing exactly t blue balls in the sample, *ie* $P(B=t)$.

Solution

Using notation from Subject CS1, B follows the hypergeometric distribution with:

$k=n_2+1$ (the number of successes)

$N=n_2+1+n_1-1=n_1+n_2$ (the total population size)

$N-k=n_1-1$ (the number of failures)

$n=n_1$ (the number of draws)

So, using the probability function for a hypergeometric random variable:

$$P(B=t)=\binom{k}{t}\binom{N-k}{n-t}\Big/\binom{N}{n}$$

$$=\binom{n_2+1}{t}\binom{n_1-1}{n_1-t}\Big/\binom{n_1+n_2}{n_1}$$

However:

$$\binom{n_1-1}{n_1-t}=\binom{n_1-1}{(n_1-1)-(n_1-t)}=\binom{n_1-1}{t-1}$$

Also:

$$n_1 + n_2 = \text{m}$$

So:

$$P(B=t) = \binom{n_2+1}{t}\binom{n_1-1}{t-1} \Big/ \binom{n_1+n_2}{n_1} = P(G=t)$$

Therefore, B has the same distribution as G.

Chapter 12

Summary, page 43

The expression for the penalised log-likelihood in the *p*-splines section has been updated to remove the factor of $\frac{1}{2}$ in the penalty term.

Practice questions, page 54

The expression for the penalised log-likelihood in the fourth bullet point has been updated to remove the factor of $\frac{1}{2}$ in the penalty term.

Chapter 16

Summary, page 37

The Generalised Pareto distribution summary has been updated to the following:

Let losses X_i be IID with cumulative distribution $F(x_i)$. The distribution of conditional losses above a threshold u, $X-u \mid X>u$, is given by:

$$F_u(x) = \frac{F(x+u)-F(u)}{1-F(u)}$$

For a large class of underlying distributions for the data, this distribution will more closely resemble a generalised Pareto distribution (GPD) as u increases towards x_F.

The CDF of the GPD is of the form:

$$G(x) = \begin{cases} 1-\left(1+\dfrac{x}{\gamma\beta}\right)^{-\gamma} & \gamma \neq 0 \\ 1-\exp\left(-\dfrac{x}{\beta}\right) & \gamma = 0 \end{cases}$$

The two parameters of the GPD family are:

- a scale parameter, $\beta > 0$
- a shape parameter, γ.

Chapter 17

Section 7.1, page 38 and Summary, page 47

The summary of upper and lower tail dependence results has been updated to include the case when $\rho = -1$ for the Student's t copula. The summary for this copula now reads:

$$\begin{aligned} &1 \text{ for all } \gamma \text{ when } \rho = 1 \\ &0 \text{ for all } \gamma \text{ when } \rho = -1 \\ &> 0 \text{ if } \gamma < \infty \text{ and } \rho \in (-1,1), \text{ decreasing as } \gamma \text{ increases} \\ &0 \text{ if } \gamma = \infty \text{ and } \rho \neq 1 \end{aligned}$$

Chapter 18

Section 1.3, page 11

The distribution in the question on this page is the three-parameter Pareto distribution, not the generalised Pareto distribution. The question has been updated to read:

Question

Claims from a particular portfolio have a three-parameter Pareto distribution with parameters $\alpha = 6$, $\lambda = 200$ and $k = 4$. A proportional reinsurance arrangement is in force with a retained proportion of 80%.

Calculate the mean and variance of the amount paid by the insurer and the reinsurer in respect of a single claim.

Section 4, page 25

The solution on this page has been updated to include the general form of the estimate of the parameter. Replacement pages for pages 23-26 are included at the end of this document.

Chapter 21

There were significant changes to the Core Reading throughout this chapter. The full new chapter is included at the end of this document.

Other tuition services

In addition to the CMP you might find the following services helpful with your study.

3.1 Study material

We also offer the following study material in Subject CS2:

- Flashcards
- Revision Notes
- ASET (ActEd Solutions with Exam Technique) and Mini-ASET
- Mock Exam and AMP (Additional Mock Pack).

For further details on ActEd's study materials, please refer to the 2022 *Student Brochure*, which is available from the ActEd website at **www.ActEd.co.uk**.

3.2 Tutorials

We offer the following (face-to-face and/or online) tutorials in Subject CS2:

- a set of Regular Tutorials (lasting a total of five days)
- a Block (or Split Block) Tutorial (lasting five full days)
- an Online Classroom.

For further details on ActEd's tutorials, please refer to our latest *Tuition Bulletin*, which is available from the ActEd website at **www.ActEd.co.uk**.

3.3 Marking

You can have your attempts at any of our assignments or mock exams marked by ActEd. When marking your scripts, we aim to provide specific advice to improve your chances of success in the exam and to return your scripts as quickly as possible.

For further details on ActEd's marking services, please refer to the 2022 *Student Brochure*, which is available from the ActEd website at **www.ActEd.co.uk**.

3.4 Feedback on the study material

ActEd is always pleased to receive feedback from students about any aspect of our study programmes. Please let us know if you have any specific comments (*eg* about certain sections of the notes or particular questions) or general suggestions about how we can improve the study material. We will incorporate as many of your suggestions as we can when we update the course material each year.

If you have any comments on this course, please send them by email to **CS2@bpp.com**.

4 Simple parametric survival models

Several survival models are in common use in which the random variable denoting future lifetime has a distribution expressed in terms of a small number of parameters. Perhaps the simplest is the exponential model, in which the hazard is constant:

$$\mu_x = \mu$$

It follows from (6.2) above that in the exponential model:

$$_tp_x = S_x(t) = \exp\left\{-\int_0^t \mu\, ds\right\} = \exp\left\{-\left[\mu s\right]_0^t\right\} = \exp(-\mu t)$$

and hence that:

$$_tq_x = 1 - {}_tp_x = 1 - \exp(-\mu t)$$

For example, if μ_x takes the constant value 0.001 between ages 25 and 35, then the probability that a life aged exactly 25 will survive to age 35 is:

$$_{10}p_{25} = \exp\left(-\int_0^{10} 0.001\, dt\right) = e^{-0.01} = 0.99005$$

We can use R to simulate values from an exponential distribution, plot its PDF, and calculate probabilities and percentiles.

Suppose we have an exponential distribution with parameter $\lambda = 0.5$. The PDF evaluated at x, $f(x)$, can be calculated using the function `dexp()` as follows:

```
dexp(x, 0.5)
```

More generally, the PDF evaluated at a vector of values `<values>` for the exponential distribution with parameter `<rate>` can be obtained with the following code structure:

```
dexp(<values>, <rate>)
```

For example, calculating $f(2)$ and $f(3)$ for the exponential distribution with parameter $\lambda = 0.5$:

```
dexp(c(2,3), 0.5)
```

```
[1] 0.1839397 0.1115651
```

A vector of rates can also be provided as the second argument. In this case, R cycles through the inputted rates as it cycles through the inputted values.

The PDF can be plotted using the `plot()` function with general structure:

```
plot(<x values>, <y values> type = "l")
```

This plots a series of points with *x*-coordinates given by the values in `<x values>` and *y*-coordinates given by the values in `<y values>`. Specifying `type = "l"` (lower case 'L') outputs a line graph joining the points.

Suppose we want to plot the PDF over the interval 0 to 10. Appropriate *x*-coordinates can be obtained using the `seq()` function with the structure:

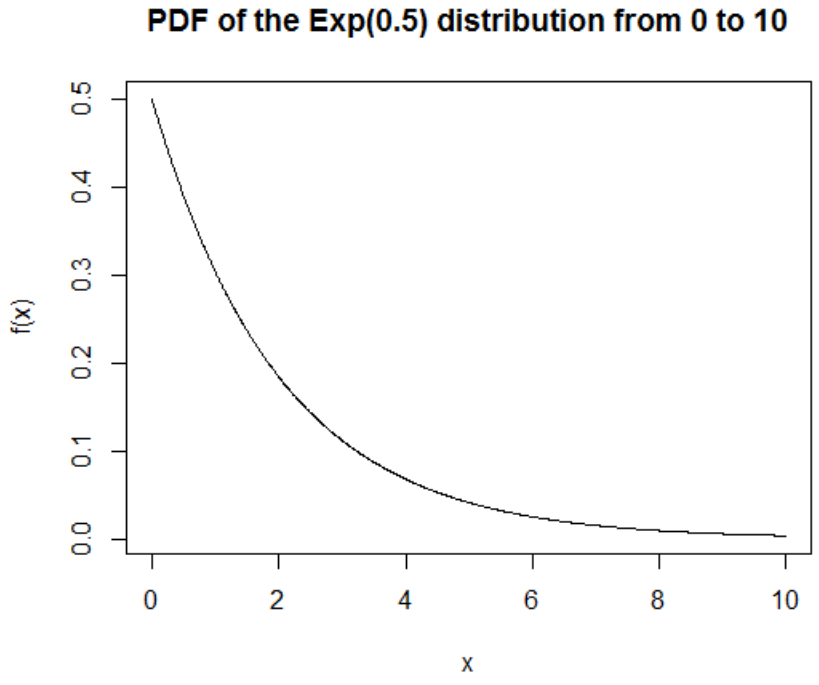
```
seq(<start>, <end>, by = <increment>)
```

This produces a vector of values from `<start>` to `<end>` in increments of `<increment>`. As a line graph is the desired output, the increment needs to be small enough to give the appearance of a smooth curve.

Example code for calculating the *x*-coordinates, *y*-coordinates and plotting the PDF is:

```
xvalues = seq(0, 10, by = 0.01)
yvalues = dexp(xvalues, 0.5)

plot(xvalues, yvalues, type = "l",
 xlab = "x", ylab = "f(x)",
 main = "PDF of the Exp(0.5) distribution from 0 to 10")
```

The `xlab`, `ylab` arguments are used to label the axes. The `main` argument is used to give the graph a title.



Alternatively, the `curve()` function can be used with the following structure:

```
curve(<function>, <start>, <end>)
```

Example code for plotting the PDF of the exponential distribution with parameter $\lambda = 0.5$ is:

```
curve(dexp(x, 0.5), 0, 10, xlab = "x", ylab = "f(x)",
      main = "PDF of the Exp(0.5) distribution from 0 to 10")
```

To calculate probabilities for a continuous distribution we use the CDF, $F(x)$, which is obtained by `pexp(<value>, <rate>)`**. For example, if X is a random variable that follows the exponential distribution with parameter $\lambda = 0.5$, to calculate $P(X \leq 2) = 0.6321206$ we can use the R code:**

```
pexp(2,0.5)
```

```
[1] 0.6321206
```

Similarly, the quantiles can be calculated with `qexp(<probability>, <rate>)`**. For example, the lower quartile of the exponential distribution with parameter $\lambda = 0.5$ is 0.5753641 and can be calculated using the following R code:**

```
qexp(0.25, 0.5)
```

```
[1] 0.5753641
```

The function `rexp(<n>, <rate>)` **can be used to generate a random sample of size** `<n>` **from the exponential distribution with parameter** `<rate>`**.**

The `set.seed(<integer>)` **function can be used prior to simulation to ensure reproducibility.**

For example, the following R code simulates 100 values from the exponential distribution with parameter $\lambda = 0.5$ using the seed 1 and stores them in the object `exp.sample`**:**

```
set.seed(1)
exp.sample = rexp(100, 0.5)
```

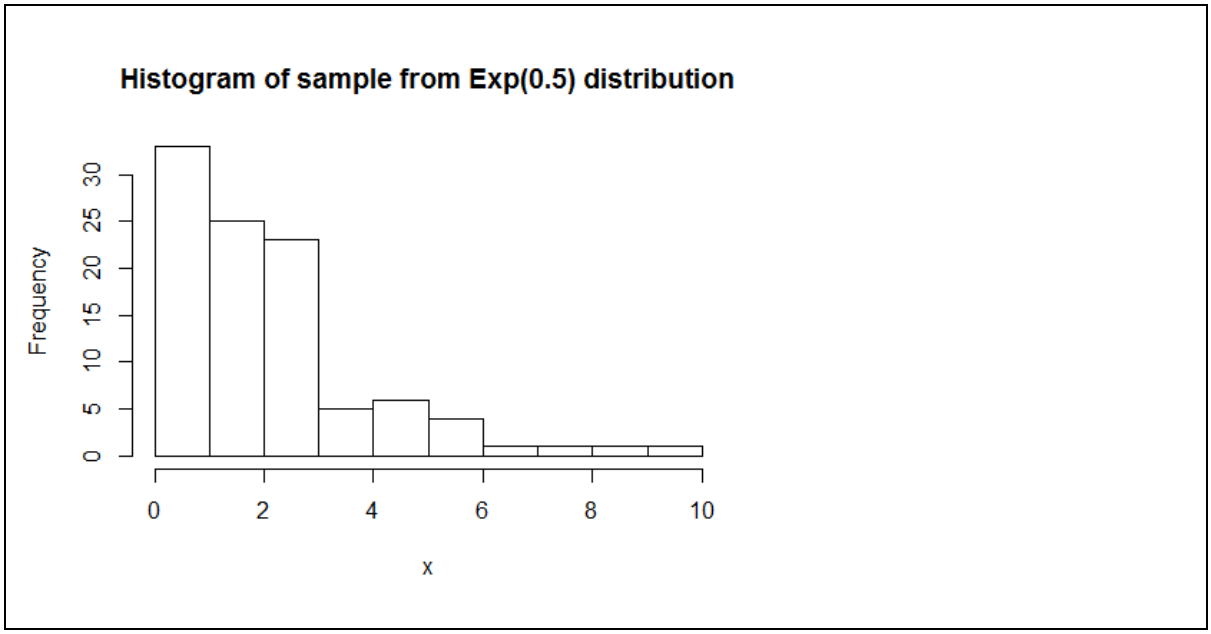
Sample moments can then be calculated using the `mean()` **and** `var()` **functions. For instance, the mean of this sample can be calculated as follows:**

```
mean(exp.sample)
```

```
[1] 2.061353
```

A histogram of the sample can be generated with the `hist()` **function:**

```
hist(exp.sample, xlab = "x",
main = "Histogram of sample from Exp(0.5) distribution")
```



A simple extension to the exponential model is the Weibull model, in which the survival function $S_x(t)$ is given by the two-parameter formula:

$$S_x(t) = \exp\left[-\alpha t^{\beta}\right] \qquad \textbf{(6.3)}$$

Recall that $S_x(t) = 1 - F_x(t)$, where $F_x(t) = P(T_x \le t)$. The CDF of the Weibull distribution is given on page 15 of the *Tables.*

Since:

$$\mu_{x+t} = -\frac{\partial}{\partial t}\log[S_x(t)]$$

we see that:

$$\mu_{x+t} = -\frac{\partial}{\partial t}[-\alpha t^{\beta}] = -[-\alpha\beta t^{\beta-1}] = \alpha\beta t^{\beta-1}$$

Different values of the parameter β can give rise to a hazard that is monotonically increasing or monotonically decreasing as t increases, or in the specific case where $\beta = 1$, a hazard that is constant, since if $\beta = 1$:

$$\alpha\beta t^{\beta-1} = \alpha.1.t^{0} = \alpha$$

This can be seen also from the expression for $S_x(t)$ (6.3), from which it is clear that, when $\beta = 1$, the Weibull model is the same as the exponential model.

We can adjust the R code given above for an exponential distribution to calculate corresponding quantities for a Weibull distribution.

Example R code for simulating a random sample of 100 values from the Weibull distribution with $c = 2$ and $\gamma = 0.25$ is:

```
set.seed(5)
weibull.sample = rweibull(100, 0.25, 2^(-1/0.25))
```

Note that R uses a different parameterisation for the scale parameter, *c*, from that in the *Formulae and Tables for Actuarial Examinations* and presented here.

Sample moments can then be calculated as well as the sample histogram plotted with the `hist()` function. For example, the mean of this sample is:

```
mean(weibull.sample)
```

```
[1] 0.7669583
```

Similarly, the PDF, CDF and quantiles can be obtained using the R functions `dweibull()`, `pweibull()` and `qweibull()`.

Alternatively, we could redefine them from first principles using the parameterisation presented here and in the *Formulae and Tables for Actuarial Examinations* as follows:

```
rweibull2 <- function(n, c, g){
    (-log(1 - runif(n))/c)^(1/g)
}

dweibull2 <- function(x, c, g){
    c * g * x^(g - 1)*exp(-c * x^g)
}

pweibull2 <- function(q, c, g){
    1 - exp(-c * q^g)
}

qweibull2 <- function(p, c, g){
    (-log(1 - p)/c)^(1/g)
}
```

This page has been left blank so that you can easily put in the replacement pages.

$E(e^{tX} \mid \text{Type I})$ is the MGF of the exponential distribution with mean 500, *ie*:

$$E(e^{tX} \mid \text{Type I}) = (1-500t)^{-1}$$

Similarly, $E(e^{tX} \mid \text{Type II})$ is the MGF of the exponential distribution with mean 1,000, *ie*:

$$E(e^{tX} \mid \text{Type II}) = (1-1{,}000t)^{-1}$$

So:

$$M_X(t) = 0.7(1-500t)^{-1} + 0.3(1-1{,}000t)^{-1}$$

We can use R to simulate values from statistical distributions, plot their PDFs, and calculate probabilities and percentiles. An example involving the exponential distribution is given below.

Suppose we have an exponential distribution with parameter $\lambda = 0.5$. The PDF evaluated at x, $f(x)$, can be calculated using the function `dexp()` as follows:

```
dexp(x, 0.5)
```

More generally, the PDF evaluated at a vector of values `<values>` for the exponential distribution with parameter `<rate>` can be obtained with the following code structure:

```
dexp(<values>, <rate>)
```

For example, calculating $f(2)$ and $f(3)$ for the exponential distribution with parameter $\lambda = 0.5$:

```
dexp(c(2,3), 0.5)
```

```
[1] 0.1839397 0.1115651
```

A vector of rates can also be provided as the second argument. In this case, R cycles through the inputted rates as it cycles through the inputted values.

The PDF can be plotted using the `plot()` function with general structure:

```
plot(<x values>, <y values> type = "l")
```

This plots a series of points with x-coordinates given by the values in `<x values>` and y-coordinates given by the values in `<y values>`. Specifying `type = "l"` (lower case 'L') outputs a line graph joining the points.

Suppose we want to plot the PDF over the interval 0 to 10. Appropriate x-coordinates can be obtained using the `seq()` function with the structure:

```
seq(<start>, <end>, by = <increment>)
```

This produces a vector of values from `<start>` to `<end>` in increments of `<increment>`. As a line graph is the desired output, the increment needs to be small enough to give the appearance of a smooth curve.

Example code for calculating the x-coordinates, y-coordinates and plotting the PDF is:

```
xvalues = seq(0, 10, by = 0.01)
yvalues = dexp(xvalues, 0.5)

plot(xvalues, yvalues, type = "l",
 xlab = "x", ylab = "f(x)",
 main = "PDF of the Exp(0.5) distribution from 0 to 10")
```

The `xlab`, `ylab` arguments are used to label the axes. The `main` argument is used to give the graph a title.

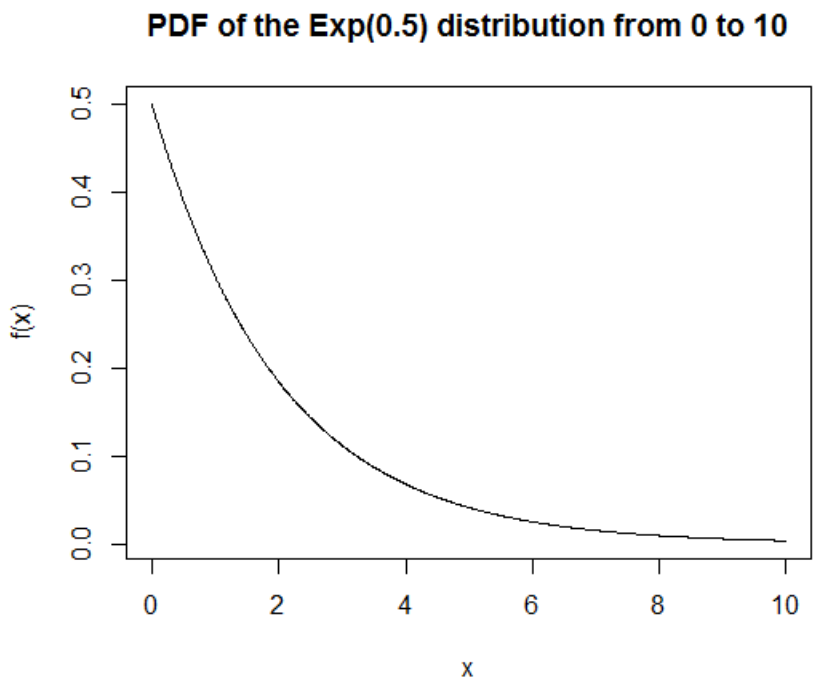


Figure 15.1

Alternatively, the `curve()` function can be used with the following structure:

```
curve(<function>, <start>, <end>)
```

Example code for plotting the PDF of the exponential distribution with parameter $\lambda = 0.5$ is:

```
curve(dexp(x, 0.5), 0, 10, xlab = "x", ylab = "f(x)",
      main = "PDF of the Exp(0.5) distribution from 0 to 10")
```

To calculate probabilities for a continuous distribution we use the CDF, $F(x)$, which is obtained by `pexp(<value>, <rate>)`. For example, if X is a random variable that follows the exponential distribution with parameter $\lambda = 0.5$, to calculate $P(X \leq 2) = 0.6321206$ we can use the R code:

```
pexp(2, 0.5)
```

```
[1] 0.6321206
```

Similarly, the quantiles can be calculated with `qexp(<probability>, <rate>)`. For example, the lower quartile of the exponential distribution with parameter $\lambda = 0.5$ is 0.5753641 and can be calculated using the following R code:

```
qexp(0.25, 0.5)
```

```
[1] 0.5753641
```

The function `rexp(<n>, <rate>)` can be used to generate a random sample of size `<n>` from the exponential distribution with parameter `<rate>`.

The `set.seed(<integer>)` function can be used prior to simulation to ensure reproducibility.

For example, the following R code simulates 100 values from the exponential distribution with parameter $\lambda = 0.5$ using the seed 1 and stores them in the object `exp.sample`:

```
set.seed(1)
exp.sample = rexp(100, 0.5)
```

Sample moments can then be calculated using the `mean()` and `var()` functions. For instance, the mean of this sample can be calculated as follows:

```
mean(exp.sample)
```

```
[1] 2.061353
```

A histogram of the sample can be generated with the `hist()` function:

```
hist(exp.sample, xlab = "x",
main = "Histogram of sample from Exp(0.5) distribution")
```

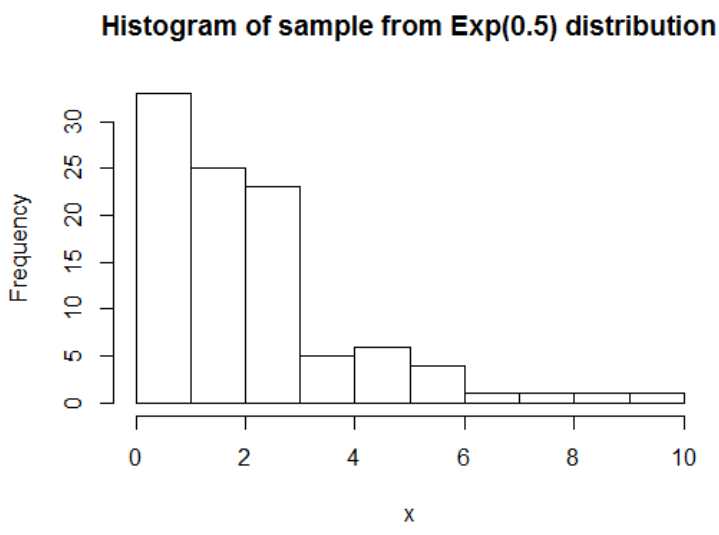


Figure 15.2

This code can be adapted to deal with other statistical distributions.

1.2 The gamma distribution

The random variable X has a gamma distribution with parameters $\alpha > 0$ and $\lambda > 0$ if it has PDF:

$$f(x) = \frac{\lambda^{\alpha}}{\Gamma(\alpha)} x^{\alpha-1} \exp(-\lambda x), \quad x > 0$$

In that case we write $X \sim Ga(\alpha, \lambda)$.

This may also be written as $Gamma(\alpha, \lambda)$.

The gamma function, $\Gamma(\alpha)$, appears in the denominator of this PDF. The definition and properties of this function are given on page 5 of the *Tables*.

The mean and variance of X are:

$$E(X) = \frac{\alpha}{\lambda}$$

$$\text{var}(X) = \frac{\alpha}{\lambda^2}$$

Question

If $X \sim Gamma(\alpha, \lambda)$, show that the MGF of X is:

$$M_X(t) = \left(1 - \frac{t}{\lambda}\right)^{-\alpha}$$

Solution

Using the definition of the MGF, we have:

$$M_X(t) = E(e^{tX}) = \int_0^{\infty} e^{tx} \frac{1}{\Gamma(\alpha)} \lambda^{\alpha} x^{\alpha-1} e^{-\lambda x} \, dx = \int_0^{\infty} \frac{1}{\Gamma(\alpha)} \lambda^{\alpha} x^{\alpha-1} e^{-(\lambda-t)x} \, dx$$

We can make the integrand look like the PDF of the $Gamma(\alpha, \lambda - t)$ distribution by writing:

$$M_X(t) = \left(\frac{\lambda}{\lambda - t}\right)^{\alpha} \int_0^{\infty} \frac{1}{\Gamma(\alpha)} (\lambda - t)^{\alpha} x^{\alpha-1} e^{-(\lambda-t)x} dx$$

This integral is equal to 1 provided $\lambda - t > 0$, so:

$$M_X(t) = \left(\frac{\lambda}{\lambda - t}\right)^{\alpha} = \left(\frac{\lambda - t}{\lambda}\right)^{-\alpha} = \left(1 - \frac{t}{\lambda}\right)^{-\alpha}, \quad \text{for } t < \lambda$$

By differentiating the MGF, we can obtain the non-central moments, $E(X^k)$, $k = 1,2,3,\ldots$:

$$E(X) = M_X'(0), \quad E(X^2) = M_X''(0), \quad etc$$

The variance and skewness can be obtained more quickly using the cumulant generating function (CGF). Recall that:

$$C_X(t) = \ln M_X(t)$$

Question

Suppose that $X \sim Gamma(\alpha, \lambda)$.

Derive formulae for the skewness and coefficient of skewness of X.

Solution

The skewness of X is its third central moment, $E\left[(X - E(X))^3\right]$. It can be obtained by differentiating the CGF three times and evaluating the third derivative when $t = 0$.

Since $X \sim Gamma(\alpha, \lambda)$:

$$C_X(t) = \ln\left(1 - \frac{t}{\lambda}\right)^{-\alpha} = -\alpha \ln\left(1 - \frac{t}{\lambda}\right)$$

Differentiating using the chain rule:

$$C_X'(t) = -\alpha\left(-\frac{1}{\lambda}\right)\left(1 - \frac{t}{\lambda}\right)^{-1} = \frac{\alpha}{\lambda}\left(1 - \frac{t}{\lambda}\right)^{-1}$$

$$C_X''(t) = \frac{\alpha}{\lambda}\left(-\frac{1}{\lambda}\right)(-1)\left(1 - \frac{t}{\lambda}\right)^{-2} = \frac{\alpha}{\lambda^2}\left(1 - \frac{t}{\lambda}\right)^{-2}$$

$$C_X'''(t) = \frac{\alpha}{\lambda^2}\left(-\frac{1}{\lambda}\right)(-2)\left(1 - \frac{t}{\lambda}\right)^{-3} = \frac{2\alpha}{\lambda^3}\left(1 - \frac{t}{\lambda}\right)^{-3}$$

So:

$$\text{skew}(X) = C_X'''(0) = \frac{2\alpha}{\lambda^3}\left(1 - \frac{0}{\lambda}\right)^{-3} = \frac{2\alpha}{\lambda^3}$$

The coefficient of skewness of X is:

$$\text{coeff of skew}(X) = \frac{\text{skew}(X)}{\left[\text{var}(X)\right]^{3/2}}$$

The variance can also be obtained from the CGF:

$$\text{var}(X) = C_X''(0) = \frac{\alpha}{\lambda^2}\left(1 - \frac{0}{\lambda}\right)^{-2} = \frac{\alpha}{\lambda^2}$$

So:

$$\text{coeff of skew}(X) = \frac{\text{skew}(X)}{\left[\text{var}(X)\right]^{3/2}} = \frac{2\alpha / \lambda^3}{(\alpha / \lambda^2)^{3/2}} = \frac{2}{\alpha^{1/2}} = \frac{2}{\sqrt{\alpha}}$$

Formulae for the PDF, MGF, mean, variance, non-central moments and coefficient of skewness of the gamma distribution are all given on page 12 of the *Tables*.

There is no closed form (*ie* no simple formula) for the CDF of a gamma random variable, which means that it is not easy to find gamma probabilities directly without using a computer package such as R. However, these probabilities can be obtained using the relationship between the gamma and chi-squared distributions.

Relationship between gamma and chi-squared distributions

If $X \sim Gamma(\alpha, \lambda)$ and 2α is an integer, then:

$$2\lambda X \sim \chi^2_{2\alpha}$$

This result is also given on page 12 of the *Tables*.

As an illustration of how this relationship can be used, suppose that $X \sim Gamma(10, 4)$ and we want to calculate $P(X > 4.375)$. Using the result above, we know that $8X \sim \chi^2_{20}$, so:

$$P(X > 4.375) = P(8X > 8 \times 4.375) = P(\chi^2_{20} > 35)$$

From page 166 of the *Tables*, we see that:

$$P(\chi^2_{20} \leq 35) = 0.9799$$

So:

$$P(X > 4.375) = 1 - 0.9799 = 0.0201$$

Example R code for simulating a random sample of 100 values (with a seed of 2) **from the gamma distribution with $\alpha = 2$ and $\lambda = 0.25$ is:**

```
set.seed(2)
gamma.sample = rgamma(100, 2, 0.25)
```

Sample moments can then be calculated using the `mean()` and `var()` functions as before, as well as the sample histogram plotted with the `hist()` function.

For example, the mean of this sample is:

```
mean(gamma.sample)
```

```
[1] 8.732949
```

The PDF, CDF and quantiles of gamma distributions can be obtained using the R functions `dgamma()`, `pgamma()` **and** `qgamma()` **respectively.**

1.3 The normal distribution

The normal distribution arises in a variety of contexts. It is of limited use for modelling loss distributions because of its symmetry (as loss distributions tend to be positively skewed).

Question

Derive the formula for the MGF of a standard normal random variable.

This page has been left blank so that you can easily put in the replacement pages.

3 Estimation

The methods of maximum likelihood, moments and percentiles can be used to fit distributions to sets of data.

The method of percentiles is outlined in Section 3.3; the other methods have been covered in Subject CS1, Actuarial Statistics.

We will now give a summary of the method of moments and maximum likelihood estimation and introduce the method of percentiles. We then discuss how these methods may be applied to the distributions covered in this chapter. In the next Section we will give a brief reminder of how the chi-squared test can be used to check the fit of a statistical distribution to a data set.

3.1 The method of moments

For a distribution with r parameters, the parameter estimates are obtained by solving the following equations:

$$m_j = \frac{1}{n}\sum_{i=1}^{n} x_i^j \qquad j = 1, 2 \ldots r$$

where:

$m_j = E(X^j \mid \theta)$ **is the** j **th population moment (around the origin), a function of the unknown parameter(s),** θ **, being estimated**

$n =$ **the sample size**

$x_i =$ **the** i ***th*** **value in the sample**

So, for example, if we are trying to estimate the value of a single parameter, and we have a sample of n claims whose sizes are $x_1, x_2, \ldots, x_n$, we would solve the equation:

$$E(X) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

ie we would equate the first non-central moments for the population and the sample.

If we are trying to find estimates for two parameters (for example if we are fitting a gamma distribution and need to obtain estimates for both parameters), we would solve the simultaneous equations:

$$E(X) = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{and} \quad E(X^2) = \frac{1}{n}\sum_{i=1}^{n} x_i^2$$

In fact, in the two-parameter case, estimates are often obtained by equating sample and population means and variances.

If we use the *n*-denominator sample variance:

$$s_n^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \frac{1}{n}\left[\sum_{i=1}^{n} x_i^2 - n\bar{x}^2\right]$$

this will give the same estimates as would be obtained by equating the first two non-central moments.

More generally, we use as many equations of the form $E(X^k) = \frac{1}{n}\sum_{i=1}^{n} x_i^k$, $k = 1, 2, \ldots$ as are needed to determine estimates of the relevant parameters.

3.2 Maximum likelihood estimation

For a distribution with one parameter, the likelihood function of a random variable, X, is the probability (or PDF) of observing what was observed given a hypothetical value of the parameter, θ. The maximum likelihood estimate (MLE) is the one that yields the highest probability (or PDF), *ie* that maximises the likelihood function.

The likelihood function $L(\theta)$ can be expressed as:

$$L(\theta) = \prod_{i=1}^{n} P\left(X = x_i \mid \theta\right) \text{ for a discrete random variable, } X$$

or:

$$L(\theta) = \prod_{i=1}^{n} f\left(x_i \mid \theta\right) \text{ for a continuous random variable, } X$$

To determine the MLE, the likelihood function needs to be maximised. Often it is practical to consider the log-likelihood function:

$$l(\theta) = \log L(\theta) = \sum_{i=1}^{n} \log P\left(X = x_i \mid \theta\right) \text{ for a discrete random variable, } X$$

or:

$$l(\theta) = \log L(\theta) = \sum_{i=1}^{n} \log f\left(x_i \mid \theta\right) \text{ for a continuous random variable, } X$$

If $l(\theta)$ can be differentiated with respect to θ, the MLE, expressed as $\hat{\theta}$, satisfies the expression:

$$\frac{d}{d\theta} l(\hat{\theta}) = 0$$

It is necessary to check, either formally or through simple logic, that the turning point is a maximum. Generally, the likelihood starts at zero, finishes at or tends to zero, and is nonnegative. Therefore, if there is one turning point it must be a maximum.

Where there is more than one parameter, the MLEs for each parameter can be determined by taking partial derivatives of the log-likelihood function and setting each to zero.

Checking that the turning point is a maximum is more complicated when there is more than one parameter. This is beyond the scope of the syllabus.

The determination of MLEs when the data are incomplete is covered in Chapter 18.

We will now look at the distributions described earlier in this chapter and consider how the parameters can be estimated in each case.

The exponential distribution

It is possible to use the method of maximum likelihood (ML) or the method of moments to estimate the parameter of the exponential distribution.

For example, suppose that an insurance company uses an exponential distribution to model the cost of repairing insured vehicles that are involved in accidents, and the average cost of repairing a random sample of 1,000 vehicles is £2,200.

We can calculate the maximum likelihood estimate of the exponential parameter as follows.

Let $x_1, x_2, \ldots, x_{1,000}$ denote the individual repair costs.

The likelihood of obtaining these values for the costs, if they come from an exponential distribution with parameter λ, is:

$$L = \prod_{i=1}^{1,000} \lambda e^{-\lambda x_i} = \lambda^{1,000} e^{-\lambda \Sigma x_i} = \lambda^{1,000} e^{-1,000\lambda \bar{x}}$$

(where $\bar{x} = \frac{1}{1,000} \sum_{i=1}^{1,000} x_i$ denotes the average claim amount).

We want to determine the value of λ that maximises the likelihood, or equivalently the value that maximises the log-likelihood:

$$\ln L = 1,000 \ln \lambda - 1,000 \lambda \bar{x}$$

Differentiating with respect to λ:

$$\frac{\partial}{\partial \lambda} \ln L = \frac{1,000}{\lambda} - 1,000 \bar{x}$$

This is equal to 0 when:

$$\lambda = \frac{1}{\bar{x}}$$

The second derivative is:

$$\frac{\partial^2}{\partial\lambda^2}\log L = -\frac{1{,}000}{\lambda^2}$$

Since the second derivative is negative when $\lambda = \frac{1}{\bar{x}}$, the stationary point is a maximum. So, $\hat{\lambda}$, the maximum likelihood estimate of λ is $\frac{1}{\bar{x}}$, or $\frac{1}{2{,}200}$.

Alternatively, we could argue that the likelihood function is continuous and is always positive (by necessity) and that $\lambda^n e^{-\lambda n\bar{x}} \to 0$ as $\lambda \to 0$ or $\lambda \to \infty$. So any stationary point that we find must be a maximum.

The ML estimate for the exponential distribution is $1/\bar{x}$. We can use R to calculate the ML estimate for the sample generated earlier:

```
      1 / mean(exp.sample)
[1] 0.4851183
```

The estimated value of the parameter is 0.4851183.

We could also use the `fitdistr()` function in the `MASS` package as follows:

```
      fitdistr(<data vector>, "exponential")
```

The `fitdistr()` function calculates ML estimates analytically when they exist in closed form (*ie* can be expressed in terms of elementary functions) and uses numerical methods when they do not. For example, the function returns $1/\bar{x}$ for the exponential distribution.

Using the function to calculate the ML estimate based on the data in `exp.sample`:

```
      fitdistr(exp.sample, "exponential")

      rate
  0.48511830
 (0.04851183)
```

The second number outputted in brackets is an estimate of the standard error of the estimator of the rate parameter.

Another approach to ML estimation in R is to define a function that calculates the negative log-likelihood and use the command `nlm()` (non-linear minimisation) to carry out a minimisation of it. This is equivalent to maximising the log-likelihood. This function uses a numerical algorithm to try to converge on a solution and requires a starting value for the parameters. The general command structure is:

```
      nlm(<negative log-likelihood function>,
          <parameter(s) starting value(s)>, <other arguments>)
```

Example R code to construct a function to calculate the negative log-likelihood for the exponential distribution is:

```
neg.log.lik <- function(param, data.vector){
      -sum(log(dexp(data.vector, param)))
}
```

So, to fit an exponential distribution to the vector `exp.sample` **with initial estimate of** $\lambda = 0.5$ **we could use:**

```
lambda <- 0.5

(MLE = nlm(neg.log.lik, lambda, data.vector = exp.sample))
```

```
$minimum
[1] 172.3363

$estimate
[1] 0.4851181

$gradient
[1] 0.0001084857

$code
[1] 1

$iterations
[1] 2

Warning messages:
1: In dexp(data.vector, param) : NaNs produced
2: In nlm(nfMLE, lambda, data.vector = exp.sample) :
  NA/Inf replaced by maximum positive value
3: In dexp(data.vector, param) : NaNs produced
4: In nlm(nfMLE, lambda, data.vector = exp.sample) :
  NA/Inf replaced by maximum positive value
```

There are various components outputted by the `nlm()` **function.**

`$minimum` **is the value of the negative log-likelihood for the estimated parameter value.**

`$estimate` **is the estimate of the parameter. The estimated value of 0.4851181 is close to the true value of 0.5. It is also close to the analytical estimate of 0.4851183 obtained by using** $1/\bar{x}$ **or the** `fitdistr()` **function.**

In general, the estimates obtained with this method may differ to those obtained when using the `fitdistr()` **function, even when it also uses numerical methods.**

`$gradient` **is the estimated gradient at the estimated minimum of the negative log-likelihood function.**

`$code` **indicates why the numerical algorithm terminated. It is important to check whether this is because a possible solution has been found or for some other reason. An output of 1 indicates that the relative gradient is close to 0, *ie* it is likely that a maximum has been identified. More detail on the possible output codes is provided in the help page for this function. This can be accessed with the command** `?nlm`**.**

`$iterations` **indicates how many iterations the numerical algorithm performed before terminating.**

The output from the `nlm()` **function has been stored in the R object** `MLE`**. These components can be extracted from this object. For example, extracting the estimate:**

```
MLE$estimate
```

```
[1] 0.4851181
```

The gamma distribution

The moments have a simple form and so the method of moments is very easy to apply. The MLEs for the gamma distribution cannot be obtained in closed form but the method of moment estimators can be used as initial estimators in a numerical search for the MLEs.

To obtain ML estimates in R, we can use the `fitdistr()` **function in the MASS package as follows:**

```
fitdistr(<data vector>, "gamma")
```

For example, fitting a gamma distribution to the sample generated earlier:

```
fitdistr(gamma.sample, "gamma")
```

```
     shape          rate
  1.66867434    0.19107663
 (0.21650487) (0.02886839)
```

As closed form expressions do not exist for the ML estimates, the `fitdistr()` **function uses a numerical algorithm and this requires starting values. The default starting values used by the** `fitdistr()` **function are the method of moments estimates. Alternative starting values** `<alpha>` **and** `<lambda>` **can be specified using the argument** `start` **as follows:**

```
fitdistr(gamma.sample, "gamma",
start = list(shape = <alpha>, rate = <lambda>))
```

As the parameters of a gamma distribution need to be positive, it is also a good idea to include a lower limit of 0. For example:

```
fitdistr(<data vector>, "gamma", lower = 0)
```

Specifying a lower (or upper) limit changes the numerical algorithm used and may lead to different estimates.

Alternatively, we could define a function to calculate the negative log-likelihood and use the `nlm()` function on it as before:

```
neg.log.lik <- function(params, data.vector){
   -sum(log(dgamma(data.vector, params[1], params[2])))
}
```

As the `nlm()` function can only minimise with respect to one input, the `params` input to the `neg.log.lik()` function is a vector of length two containing values for the parameters, α and λ.

So, to fit a gamma distribution to the vector `gamma.sample` with initial estimates of $\alpha = 1$ and $\lambda = 0.5$, say, we could use:

```
params <- c(1, 0.5)

(MLE = nlm(neg.log.lik , params, data.vector = gamma.sample))
```

```
$minimum
[1] 309.9164

$estimate
[1] 1.6686885 0.1910796

$gradient
[1] -1.440461e-05 6.125589e-05

$code
[1] 1

$iterations
[1] 18
```

The normal distribution

The method of moments and maximum likelihood estimation are both straightforward to apply in this case. Both give the following estimates:

$$\hat{\mu} = \bar{x} \qquad \text{and} \qquad \hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

The estimate for the population variance is $\frac{n-1}{n}\times$ the usual sample variance. Of course, provided the sample size is large, there will be little difference between estimates calculated using the two different sample variance formulae.

The lognormal distribution

Estimation for the lognormal distribution is straightforward since μ and σ^2 may be estimated using the log-transformed data. Let $x_1, x_2, \ldots, x_n$ be the observed values and let $y_i = \log x_i$. The MLEs of μ and σ^2 are $\bar{y}$ and s_y^2, where the subscript y signifies a sample variance (n-denominator) computed on the y values.

In other words, the maximum likelihood estimates are:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} (y_i - \bar{y})^2$$

The ML estimates for the lognormal distribution exist in closed form. We can use R to calculate the ML estimates of μ and σ for the sample generated earlier:

```
n = length(log.norm.sample)

(m = mean(log(log.norm.sample)))
[1] 0.01103557

(s = sqrt((n-1) / n * var(log(log.norm.sample))))
[1] 0.8517857
```

Recall that R uses σ rather than σ^2 as the second parameter.

We could also use the `fitdistr()` function in the `MASS` package as follows, which calculates the estimates in the same way:

```
fitdistr(<data vector>, "log-normal")
```

For example, fitting a lognormal distribution to the sample generated earlier:

```
fitdistr(log.norm.sample, "log-normal")

    meanlog       sdlog
  0.01103557   0.85178567
 (0.08517857) (0.06023034)
```

Alternatively, we can define a function to calculate the negative log-likelihood and use the `nlm()` function on it as before (even though, in practice, a numerical approach is not required here):

```
neg.log.lik <- function(params, data.vector){
      -sum(log(dlnorm(data.vector, params[1], params[2])))
}
```

The input `params` is a vector of length two, corresponding to the parameters μ and σ, the mean and standard deviation (on the logarithmic scale).

For example, to fit a lognormal distribution to the vector `log.norm.sample` **with the sample mean and the sample standard deviation of the logged data as initial estimates (** $\mu = 0.01$ **and** $\sigma = 0.86$ **to** 2 decimal places**) we could use:**

```
      (m = mean(log(log.norm.sample)))

[1] 0.01103557

      (s = sd(log(log.norm.sample)))

[1] 0.8560768

      params <- c(m, s)
      (MLE = nlm(neg.log.lik, params, data.vector = log.norm.sample))

$minimum
[1] 126.9554

$estimate
[1] 0.01103519 0.85178553

$gradient
[1] 1.698197e-05 9.808332e-05

$code
[1] 1

$iterations
[1] 2
```

Alternatively, the method of moments can be used to estimate the parameters.

As an example, suppose that based on an analysis of past claims, an insurance company believes that individual claims in a particular category for the coming year will be lognormally distributed with a mean size of £5,000 and a standard deviation of £7,500. The company wants to estimate the proportion of claims that will exceed £25,000.

To do this, it needs to estimate the parameters, μ and σ^2, of the lognormal distribution. Equating the formulae for the mean and standard deviation of the lognormal distribution to the values given gives:

$$e^{\mu+\frac{1}{2}\sigma^2} = 5{,}000 \text{ and } e^{\mu+\frac{1}{2}\sigma^2}\sqrt{e^{\sigma^2}-1} = 7{,}500$$

Dividing the second equation by the first gives:

$$\sqrt{e^{\sigma^2}-1} = \frac{7{,}500}{5{,}000} = 1.5$$

$$\Rightarrow \quad \sigma^2 = 1.179$$

We can now solve for μ:

$$\mu = \log 5{,}000 - ½(1.179) = 7.928$$

So the proportion of claims expected to exceed £25,000 is:

$$\begin{aligned} P(X > 25{,}000) &= P(\ln X > \ln 25{,}000) \\ &= P(N(7.928, 1.179) > \ln 25{,}000) \\ &= P\left(N(0,1) > \frac{\ln 25{,}000 - 7.928}{\sqrt{1.179}} \right) \\ &= 1 - \Phi(2.025) = 0.021 \end{aligned}$$

ie 2.1% of claims are expected to exceed £25,000.

The method of moments estimates were also calculated in the previous R example, which we then used as initial values when determining the maximum likelihood estimates.

The two-parameter Pareto distribution

Since the CDF exists in closed form, it may be possible to fit the Pareto distribution to data by using the method of percentiles. The method of moments is also very easy to apply in the case of the two-parameter Pareto distribution, but the estimators obtained in this way will tend to have rather large standard errors, mainly because S^2, the sample variance, has a very large variance. However, the method does provide initial estimates for more efficient methods of estimation that may not be so simple to apply, like maximum likelihood, where numerical methods may need to be used.

Question

Claims arising from a particular group of policies are believed to follow a Pareto distribution with parameters α and λ. A random sample of 20 claims gives values such that $\sum x = 1{,}508$ and $\sum x^2 = 257{,}212$. Estimate α and λ using the method of moments.

Solution

Suppose that X is the claim amount random variable. Then:

$$E(X) = \frac{\lambda}{\alpha - 1}$$

Rearranging the variance formula to find $E(X^2)$, we have:

$$E(X^2) = \text{var}(X) + [E(X)^2] = \frac{2\lambda^2}{(\alpha-1)(\alpha-2)}$$

So we set:

$$\frac{\lambda}{\alpha-1}=\frac{1,508}{20}=75.4 \quad \text{and} \quad \frac{2\lambda^2}{(\alpha-1)(\alpha-2)}=\frac{257,212}{20}=12,860.6$$

Squaring the first of these equations and substituting into the second, we see that:

$$\frac{2\times 75.4^2(\alpha-1)}{\alpha-2}=12,860.6$$

Solving this equation, we find that the method of moments estimates of α and λ are 9.630 and 650.7, respectively.

To carry out numerical maximum likelihood estimation, we can define a function to calculate the negative log-likelihood and use the `nlm()` function on it as before:

```
neg.log.lik = function(params, data.vector){
   -sum(log(dpareto(data.vector, params[1], params[2])))
}
```

Here we are using the `dpareto()` function that we defined in Section **Error! Reference source not found.**. **The input `params` is a vector of length two, corresponding to the parameters α and λ.**

For example, to fit a Pareto distribution to the vector `pareto.sample` with initial estimates of $\alpha = 4.53$ and $\lambda = 424$ (the method of moments estimates using the $n-1$ denominator sample variance**) we could use:**

```
m = mean(pareto.sample)
v = var(pareto.sample)

(a = (-2*v/m^2) / (1 - v/m^2))
```

```
[1] 4.530004
```

```
(lam = m * (a - 1))
```

```
[1] 424.024
```

```
params <- c(a, lam)
(MLE = nlm(neg.log.lik, params, data.vector = pareto.sample))
```

```
$minimum
[1] 575.0624

$estimate
[1]   3.540334 308.705072

$gradient
[1] -3.692868e-06  4.934819e-08

$code
[1] 1

$iterations
[1] 19
```

The three-parameter Pareto distribution

Things are not quite so easy for the three-parameter Pareto distribution.

As for estimation, the CDF does not exist in closed form, so the method of percentiles is not available.

The method of percentiles is described in Section 3.3.

ML can be used, but again suitable computer software is required; the method of moments can provide initial estimates for any iterative scheme.

We will need to define a function to calculate the negative log-likelihood and use the function `nlm()` on it as before.

The Weibull distribution

Neither the method of moments nor maximum likelihood is elementary to apply if both c and γ are unknown (although if a computer is available, as would be the case in practice, the equations are simple enough).

To obtain ML estimates in R, we could use the `fitdistr()` function in the `MASS` package. As the parameters need both be positive, we can specify a lower limit as follows:

```
fitdistr(<data vector>, "weibull", lower = 0)
```

For example, fitting a Weibull distribution to the sample generated earlier:

```
(MLE = fitdistr(weibull.sample, "weibull", lower = 0))
```

```
     shape         scale
  0.24117205    0.04588711
 (0.01930196) (0.02000550)
```

These estimates are for the parameterisation used in R. We can transform these into estimates for the parameterisation used here in the notes and in the *Tables* **(using the invariance property of ML estimates) as follows:**

```
(c = MLE$estimate["scale"] ^ (-MLE$estimate["shape"]))
```

```
 scale
2.10263
```

```
(g = MLE$estimate["shape"])
```

```
 shape
0.241172
```

Recall that the shape and scale parameters used by R are γ and $c^{-1/\gamma}$ respectively. So:

$$\gamma = shape \qquad c = \left(c^{-1/\gamma}\right)^{-\gamma} = scale^{-shape}$$

The `fitdistr()` function uses a numerical algorithm for the Weibull distribution, which requires starting values. If no values are provided, then the function automatically calculates a starting point.

Estimates obtained from the method of percentiles could also be used as the initial values.

The method of percentiles is covered in Section 3.3.

For example, using the first and third sample quartiles:

```
      Q1 = quantile(weibull.sample, 0.25)
      Q3 = quantile(weibull.sample, 0.75)

      (g = (log(log(0.75) / log(0.25)) / log(Q1 / Q3)))

      25%
0.2035839

      (c = log(0.75) / -Q1^g)

     25%
1.868239
```

Next applying the `fitdistr()` function with these starting values:

```
      fitdistr(weibull.sample, "weibull",
               start = list(shape = g, scale = c^(-1/g)),
               lower = 0)

   shape         scale
 0.24117530   0.04587929
(0.01930205) (0.01999973)
```

Converting these into the parameterisation presented here and in the *Tables* after first saving the fit in an object:

```
      fit = fitdistr(weibull.sample, "weibull",
                     start = list(shape = g, scale = c^(-1/g)),
                     lower = 0)

      (c = fit$estimate["scale"] ^ (-fit$estimate["shape"]))

scale
2.102737

      (g = fit$estimate["shape"])

shape
0.2411753
```

These estimates are very similar to those calculated using the starting values calculated by the function.

Alternatively, we could define a function to calculate the negative log-likelihood and use the function `nlm()` on it as before:

```
neg.log.lik = function(params, data.vector){
-sum(log(dweibull2(data.vector, params[1], params[2])))
}

params <- c(c, g)

(MLE = nlm(neg.log.lik, params, data.vector = weibull.sample))
```

```
$minimum
[1] -251.7054

$estimate
[1] 2.1027447 0.2411672

$gradient
[1] -6.758241e-07 1.164750e-03

$code
[1] 2

$iterations

[1] 13
```

Here `dweibull2()` has been used to construct the negative likelihood function. So, these estimates are for the parameterisation presented here and in the *Formulae and Tables for Actuarial Examinations.*

Recall that `$code` indicates why the numerical algorithm terminated. An output of 2 indicates that it is likely that the maximum likelihood estimate has been identified. More detail on the possible output codes is provided in the help page for this function. This can be accessed with the command `?nlm`.

In the case where γ has the known value γ^*, maximum likelihood is easy enough.

To do this, we let $y_i = x_i^{\gamma}$. If the original distribution is Weibull, the y values now have an exponential distribution. The MLE of c can now be determined in the usual way.

The Burr distribution

Since the CDF exists in closed form, it may be possible to fit the Burr distribution to data by using the method of percentiles; ML will certainly require the use of computer software that allows non-linear optimisation.

To carry out numerical maximum likelihood estimation, we need to define a function to calculate the negative log-likelihood and use the function `nlm()` on it as before.

In the case where γ has the known value γ^*, then $y_i = x_i^{\gamma}$ has a Pareto distribution if the original distribution is Burr. This can be seen by comparing the CDFs. A Pareto distribution can then be fitted to the y_i's.

3.3 The method of percentiles

The distribution function of the $W(c,\gamma)$ distribution is an elementary function, and a simple method of estimation of both c and γ is based on this. The method involves equating selected sample percentiles to the distribution function; for example, equate the sample quartiles, the 25th and 75th sample percentiles, to the population quartiles. This corresponds to the way in which sample moments are equated to population moments in the method of moments. This method will be referred to as the method of percentiles.

In the method of moments, the first two moments are used if there are two unknown parameters, and this seems intuitively reasonable (although the theoretical basis for this is not so clear). In a similar fashion, when using the method of percentiles, the median would be used if there were one parameter to estimate. With two parameters, the best procedure is less clear, but the lower and upper quartiles seem a sensible choice.

Example

Estimate c and γ in the Weibull distribution using the method of percentiles, where the first sample quartile is 401 and the third sample quartile is 2,836.75.

Solution

The two equations for c and γ are:

$$F(401) = 1 - \exp(-c \times 401^{\gamma}) = 0.25$$

$$F(2{,}836.75) = 1 - \exp(-c \times 2{,}836.75^{\gamma}) = 0.75$$

which can be rewritten as:

$$-c \times 401^{\gamma} = \ln 0.75$$

and: $$-c \times 2{,}836.75^{\gamma} = \ln 0.25$$

Dividing, it is found that $\tilde{\gamma}$ = 0.8038, and hence $\tilde{c}$ = 0.002326, where ~ denotes the percentile estimate. Note that $\tilde{\gamma}$ is less than 1, indicating a fatter tail than the exponential distribution gives.

The CDF of the Weibull distribution is:

$$F(x) = 1 - e^{-cx^{\gamma}}$$

So, when using the method of percentiles with the lower and upper quartiles, Q_1 and Q_3, we have:

$$F(Q_1) = 1 - e^{-cQ_1^{\gamma}} = 0.25$$

$$F(Q_3) = 1 - e^{-cQ_3^{\gamma}} = 0.75$$

Rearranging and taking logs gives:

$$-cQ_1^{\gamma} = \ln 0.75$$

$$-cQ_3^{\gamma} = \ln 0.25$$

Dividing one by the other and taking logs again gives:

$$\gamma = \frac{\ln\left(\frac{\ln 0.75}{\ln 0.25}\right)}{\ln\left(\frac{Q_1}{Q_3}\right)}$$

The estimate of γ can then be substituted into either of the starting equations to find the estimate for c.

We can apply the method of percentiles to any distribution for which it is possible to calculate a closed form for the cumulative distribution function, although the resulting algebra can be messy.

Question

Claims arising from a particular group of policies are believed to follow a Pareto distribution with parameters α and λ. A random sample of 20 claims has a lower quartile of 11 and an upper quartile of 85. Estimate the values of α and λ using the method of percentiles.

Solution

The cumulative distribution function of the Pareto distribution is $F(x) = 1 - \left(\frac{\lambda}{\lambda + x}\right)^{\alpha}$.

Q_1, the lower quartile of the distribution, satisfies the equation:

$$F(Q_1) = 1 - \left(\frac{\lambda}{\lambda + Q_1}\right)^{\alpha} = 0.25$$

Q_3, the upper quartile of the distribution, satisfies the equation:

$$F(Q_3) = 1 - \left(\frac{\lambda}{\lambda + Q_3}\right)^{\alpha} = 0.75$$

So:

$$Q_1 = \lambda\left[(3/4)^{-1/\alpha} - 1\right] \qquad \text{and} \qquad Q_3 = \lambda\left[(1/4)^{-1/\alpha} - 1\right]$$

The method of percentiles estimates $\tilde{\alpha}$ and $\tilde{\lambda}$ are obtained by setting $Q_1 = 11$ and $Q_3 = 85$:

$$11 = \tilde{\lambda}\left[(3/4)^{-1/\tilde{\alpha}} - 1\right]$$

$$85 = \tilde{\lambda}\left[(1/4)^{-1/\tilde{\alpha}} - 1\right]$$

We can eliminate $\tilde{\lambda}$ by dividing these equations. This gives:

$$\frac{11}{85} = \frac{(3/4)^{-1/\tilde{\alpha}} - 1}{(1/4)^{-1/\tilde{\alpha}} - 1}$$

We cannot solve this algebraically but it can easily be done on a computer, *eg* using the Goal Seek function in Excel. Doing this, we find that $\tilde{\alpha} = 1.284$ and hence $\tilde{\lambda} = 43.790$.

These estimates are very different from those obtained using the method of moments. (In an earlier question, we calculated the method of moments estimates of α and λ to be 9.630 and 650.7, respectively.)

The method of percentiles is very unreliable for estimating the parameters of a Pareto distribution unless we use extremely large samples. In this particular case, the method of percentiles is unlikely to give us reasonable estimates unless we use samples of, say, 1,000 or more.

3.4 Goodness of fit

We can check the fit in R by plotting a histogram of the data and superimposing the density function of the fitted distribution.

For example, plotting a histogram of the data in `exp.sample` and overlaying the fitted distribution:

```
hist(exp.sample, xlab = "x",
     main = "Histogram of sample from Exp(0.5)
    distribution", freq = FALSE)

lambda = 1 / mean(exp.sample)

curve(dexp(x, lambda), add = TRUE, col = "red")
```

In the `hist()` function, the argument `freq` has been set to `FALSE` to plot the data on a density scale rather than a frequency scale. This makes it comparable with the scale of a PDF. Alternatively, the `prob` argument can be set to `TRUE`.

In the `curve()` function, the add argument has been set to `TRUE` to add the curve to the existing histogram.

A legend can also be added to label the different information in the plot:

```
legend("topright",
legend = c("histogram of sample from Exp(0.5) distribution",
"Fitted exponential distribution PDF"),
col = c("black", "red"),
lty = 1)
```

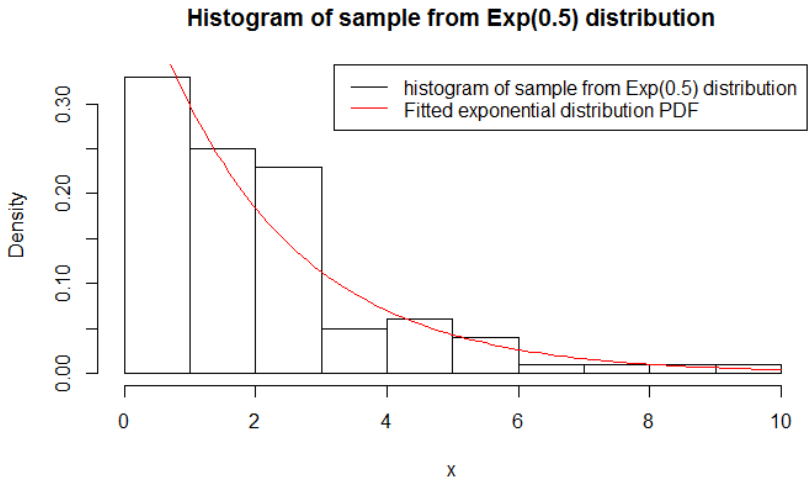


Figure 15.3

Better yet, we can plot an empirical density function from the data, using the function `density()`, and compare to the true density function of the fitted distribution. We can add the empirical density with the `lines()` function:

```
lines(density(exp.sample), col = "blue")

legend("topright", legend = c("histogram of sample from
       Exp(0.5) distribution", "Fitted exponential
       distribution PDF", "Empirical density"),
       col = c("black", "red", "blue"), lty = 1)
```

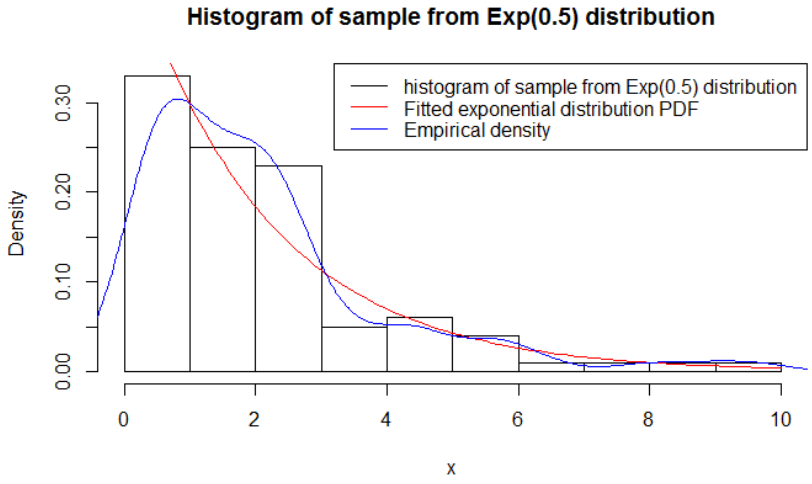


Figure 15.4

An even better way is to use the `qqplot()` function to compare the sample data to theoretical values from the fitted model distribution:

```
qqplot(<theoretical values>, <sample values>)
```

A straight diagonal line indicates perfect fit. A comparison line can be added with the `abline(<intercept>, <slope>)` function:

```
abline(0, 1)
```

The relevant theoretical values can be calculated using the quantile function of the relevant distribution, evaluated at appropriate percentage points. These can be calculated using the `ppoints(<n>)` function, where `<n>` is the sample size.

For example:

```
n = length(exp.sample)

qqplot(qexp(ppoints(n), lambda), exp.sample,
 xlab = "Theoretical quantiles",
 ylab = "Sample quantiles",
 main = "Q-Q plot comparing sample from the Exp(0.5) distribution
 to the fitted exponential distribution")

abline(0, 1)
```

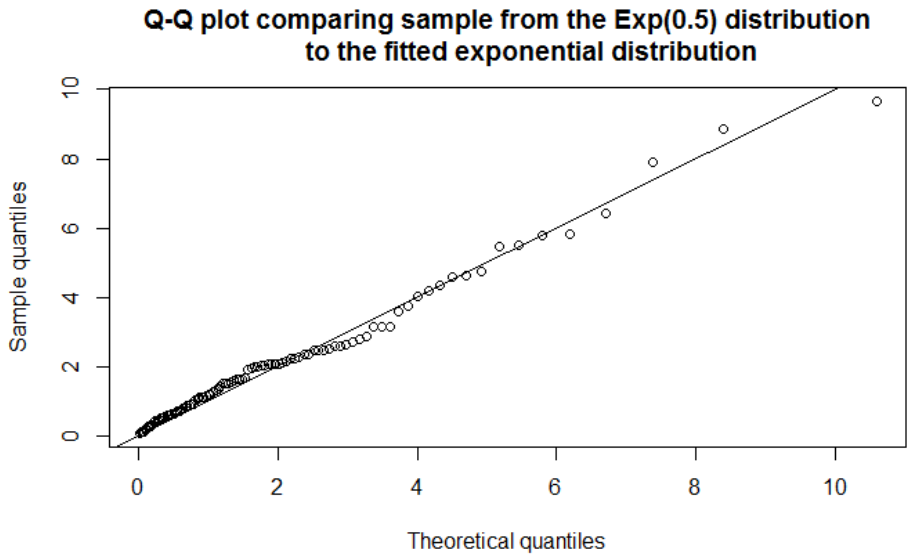


Figure 15.5

The fit of the distribution can also be tested formally by using a χ^2 test. The χ^2 test has been covered in Subject CS1, Actuarial Statistics.

We recap the chi-squared test in the next Section.

This page has been left blank so that you can easily put in the replacement pages.

Given a sample of claims in the vector `x`, a set of sample payments of the insurer, `y`, and the reinsurer, `z`, with retention `M` after applying the inflation factor `k` would be:

```
y = pmin(k * x, M)
z = pmax(0, k * x - M)
```

We can then estimate moments, probabilities and quantiles for the distributions of the insurer's and reinsurer's payments as before.

4 Estimation

Consider the problem of estimation in the presence of excess of loss reinsurance. Suppose that the claims record shows only the net claims paid by the insurer. A typical claims record might be:

$$x_1, x_2, M, x_3, M, x_4, x_5, \ldots \qquad \textbf{(18.6)}$$

and an estimate of the underlying gross claims distribution is required.

As before, we wish to estimate the parameters for the distribution we have assumed for the claims.

The method of moments is not available since even the mean claim amount cannot be computed. On the other hand, it may be possible to use the method of percentiles without alteration; this would happen if the retention level M is high and only the higher sample percentiles were affected by the (few) reinsurance claims.

The statistical terminology for a sample of the form (18.6) is censored. In general, a censored sample occurs when some values are recorded exactly and the remaining values are known only to exceed a particular value, here the retention level M.

Maximum likelihood can be applied to censored samples. The likelihood function is made up of two parts. If the values of $x_1, x_2, \ldots, x_n$ are recorded exactly these contribute a factor of:

$$L_1(\underline{\theta}) = \prod_{i=1}^{n} f(x_i; \underline{\theta})$$

If a further m claims are referred to the reinsurer, then the insurer records a payment of M for each of these claims. These censored values then contribute a factor:

$$L_2(\underline{\theta}) = \prod_{j=1}^{m} P(X > M) \quad \textit{ie} \ \left[P(X > M)\right]^m$$

The complete likelihood function is:

$$L(\underline{\theta}) = \prod_{i=1}^{n} f(x_i; \underline{\theta}) \times \left[1 - F(M; \underline{\theta})\right]^m$$

where $F(.;\underline{\theta})$ is the CDF of the claims distribution.

The reason for multiplying is that the likelihood reflects the probability of getting the n claims with known values *and* m claims exceeding M. Also, we are assuming that the claims are independent.

In R, we can define a function to calculate the negative censored log-likelihood and use the function `nlm()` on it as in Chapter 15.

Consider the vector `exp.sample` generated in Chapter 15.

The code to generate this sample is:

```
set.seed(1)
exp.sample = rexp(100, 0.5)
```

Say that the values in this vector represent full claim amounts and that the insurer has excess of loss reinsurance with $M = 3$. We can create a vector of the censored data (the claim amounts paid by the insurer) as follows:

```
M = 3
cens.sample = exp.sample
cens.sample[cens.sample > M] = M
```

The final line selects all the claim values greater than the retention limit and sets them equal to this limit, which is the amount paid by the insurer on these claims.

For example, counting how many claims were above the retention limit:

```
length(cens.sample[cens.sample == M])
```

```
[1] 19
```

We can then create a function to calculate the negative log-likelihood assuming that only the censored data is available (and not the underlying claim amounts):

```
neg.log.lik = function(lambda, cens.vector, M){
    n = length(cens.vector[cens.vector == M])

    cont.1 = -sum(log(dexp(cens.vector[cens.vector < M], lambda)))
    cont.2 = -n * log(pexp(M, lambda, lower = FALSE))

    cont.1 + cont.2
}
```

As we are working with continuous data, we assume that all values in the censored data exactly equal to the retention limit correspond to underlying claim values over the limit.

We can then minimise it with the `nlm()` function using the reciprocal of the sample mean of the censored data, say, as the starting value (which would be the MLE for an uncensored sample)**:**

```
(MLE = nlm(neg.log.lik, p = 1 / mean(cens.sample),
       cens.vector = cens.sample, M = 3))
```

```
$minimum
[1] 138.3461

$estimate
[1] 0.4926395

$gradient
[1] 7.048584e-06

$code
[1] 1

$iterations
[1] 5
```

Note that for the exponential distribution, the ML estimate can be calculated analytically even with censored data. So numerical methods are not actually required. Obtaining the estimate exactly:

```
      (n1 = length(cens.sample[cens.sample < M]))
[1] 81
      (n2 = length(cens.sample[cens.sample == M]))
[1] 19
      n1 / (n2 * M + sum(cens.sample[cens.sample < M]))
[1] 0.49264
```

This formula is derived for a particular scenario in the question below.

Question

Claims from a portfolio are believed to follow an $Exp(\lambda)$ distribution. The insurer has effected individual excess of loss reinsurance with a retention limit of 1,000.

The insurer observes a random sample of 100 claims, and finds that the average amount of the 90 claims that do not exceed 1,000 is 82.9. There are 10 claims that do exceed the retention limit.

Calculate the maximum likelihood estimate of the parameter λ.

Solution

Here $X \sim Exp(\lambda)$ and $P(X > 1{,}000) = e^{-1{,}000\lambda}$. So the likelihood function is:

$$L(\lambda) = \lambda e^{-\lambda x_1} \lambda e^{-\lambda x_2} \ldots \lambda e^{-\lambda x_{90}} \times (e^{-1{,}000\lambda})^{10} = \lambda^{90} e^{-(10{,}000 + \sum x_i)\lambda}$$

Taking logs:

$$\ln L = 90 \ln \lambda - (10{,}000 + \sum x_i)\lambda$$

Differentiating with respect to λ:

$$\frac{\partial}{\partial \lambda} \ln L = \frac{90}{\lambda} - (10{,}000 + \sum x_i)$$

This is equal to 0 when:

$$\lambda = \frac{90}{10{,}000 + \sum x_i} = \frac{90}{10{,}000 + (90 \times 82.9)} = 0.005154$$

This is of the form $\frac{n_1}{n_2 \times M + \sum x_i}$ where, using the notation in the R example above:

- M represents the retention limit
- n_1 represents the number of claims lower than the retention limit
- n_2 represents the number of claims above the retention limit
- $\sum x_i$ represents the sum of claims lower than the retention limit

Differentiating again:

$$\frac{\partial^2}{\partial \lambda^2} \ln L = -\frac{90}{\lambda^2}$$

This is negative when $\lambda = 0.005154$. (In fact it is always negative.) So we have a maximum turning point and hence $\hat{\lambda} = 0.005154$.

5 Policy excess

Insurance policies with an excess are common in motor insurance and many other kinds of property and accident insurance. Under this kind of policy, the insured agrees to carry the full burden of the loss up to a limit, L, called the excess. If the loss is an amount X, greater than L, then the policyholder will claim only $X - L$. If Y is the amount actually paid by the insurer, then:

$$Y = 0 \qquad \text{if } X \le L$$

$$Y = X - L \qquad \text{if } X > L$$

Clearly, the premium due on any policy with an excess will be less than that on a policy without an excess.

This assumes that some of the saving is actually passed on to the policyholder. A policy excess may also be referred to as a *deductible*.

The position of the insurer for a policy with an excess is exactly the same as that of the reinsurer under excess of loss reinsurance. The position of the policyholder as far as losses are concerned is exactly the same as that of an insurer with an excess of loss reinsurance contract.

In practice, expenses form a significant part of the insurance cost. So the presence of an excess might not affect the premium as much as might be expected. A premium calculated ignoring expenses is called a 'risk premium'.

Question

An insurer believes that claims from a particular type of policy follow a Pareto distribution with parameters $\alpha = 2$ and $\lambda = 900$. The insurer wishes to introduce a policy excess so that 20% of losses result in no claim to the insurer.

Calculate the size of the excess.

Solution

Let L be the size of the excess. The insurer wants to set L so that $P(X < L) = 0.2$. Using the given loss distribution, we have:

$$P(X < L) = 1 - \left(\frac{900}{900 + L}\right)^2$$

So we require:

$$1 - \left(\frac{900}{900 + L}\right)^2 = 0.2$$

Note also that the formula for the skewness of S has a simple form when S is a compound Poisson random variable:

$$\boldsymbol{skew[S] = \lambda m_3} \qquad \textbf{(19.12)}$$

***ie*:**

$$skew(S) = \lambda E(X^3)$$

The easiest way to show that the third central moment of S is λm_3 is to use the cumulant generating function:

$$\boldsymbol{C_S(t) = \log M_S(t)}$$

To determine the skewness, we differentiate it three times with respect to t and set $t = 0$, *ie*:

$$\boldsymbol{skew[S] = \frac{d^3}{dt^3}\log M_S(t)\Big|_{t=0}}$$

In other words:

$$skew(S) = C_S'''(0)$$

Recall also that:

$$E(S) = C_S'(0)$$

$$\text{var}(S) = C_S''(0)$$

Since $M_S(t) = \exp\left[\lambda(M_X(t) - 1)\right]$, it follows that:

$$\boldsymbol{\log M_S(t) = \lambda\left(M_X(t) - 1\right)}$$

So:

$$\boldsymbol{\frac{d^3}{dt^3}\log M_S(t)\Big|_{t=0} = \lambda\left[\frac{d^3}{dt^3}\left(M_X(t) - 1\right)\right]_{t=0} = \lambda m_3}$$

ie $\boldsymbol{skew[S] = \lambda m_3}$

This is because $M_X'''(0) = E(X^3)$ for any random variable.

The coefficient of skewness of S is given by:

$$\frac{skew(S)}{[\text{var}(S)]^{3/2}}$$

Hence the coefficient of skewness $= \lambda m_3 / (\lambda m_2)^{3/2}$.

This result shows that the distribution of S is positively skewed, since m_3 is the third moment about zero of X_i and hence is greater than zero because X_i is a non-negative valued random variable. Note that the distribution of S is positively skewed even if the distribution of X_i is negatively skewed. The coefficient of skewness of S is $\lambda m_3 / (\lambda m_2)^{3/2}$, and hence goes to 0 as $\lambda \to \infty$. Thus for large values of λ, the distribution of S is almost symmetric.

Mean, variance and skewness of a compound Poisson random variable

If $N \sim Poisson(\lambda)$, then S is a compound Poisson random variable and:

$$E(S) = \lambda E(X) = \lambda m_1$$

$$\text{var}(S) = \lambda E(X^2) = \lambda m_2$$

$$skew(S) = \lambda E(X^3) = \lambda m_3$$

These results are all given on page 16 of the *Tables*.

Despite being able to calculate the moments of a compound Poisson distribution we are not able to calculate its probabilities as it forms no standard distribution. On a piece of paper, we could use the Central Limit Theorem to approximate it using a normal distribution. However, with computer packages, such as R, we can simulate an empirical compound distribution from which we can estimate probabilities with ease.

The following R code simulates 10,000 values from a compound Poisson distribution with parameter 1,000 and a gamma claims distribution with $\alpha = 750$ and $\lambda = 0.25$:

```
set.seed(123)

sims = 10000

n = rpois(sims, 1000)
s = rep(NA, sims)

for(i in 1:sims){
 x = rgamma(n[i], shape = 750, rate = 0.25)
 s[i] = sum(x)
}
```

Note we have used `set.seed(123)` so you can obtain the same values as this example.

We can obtain the sample mean of 2,997,651, the sample standard deviation of 93,719.71 and the sample coefficient of skewness of 0.02655921 as follows:

```
mean(s)

[1] 2997651

sd(s)

[1] 93719.71
```

```
      n = length(s)

      skewness = sum((s - mean(s))^3) / n

      skewness / var(s)^(3/2)

[1] 0.02655921
```

We can estimate P(S>3,000,000) to be 0.4881 as follows:

```
      length(s[s > 3000000]) / n

[1] 0.4881
```

Finally, we could estimate the 90th percentile to be 3,115,719 as follows:

```
      quantile(s, 0.9)

    90%

3115719
```

We can plot a histogram of the sample from the compound distribution using the `hist()` function:

```
      hist(s, prob = TRUE, main= "Histogram of simulated
      values from compound distribution")
```

We set `prob = TRUE` so that the histogram can be compared to probability density functions. Alternatively, setting `freq = FALSE` produces the same output.

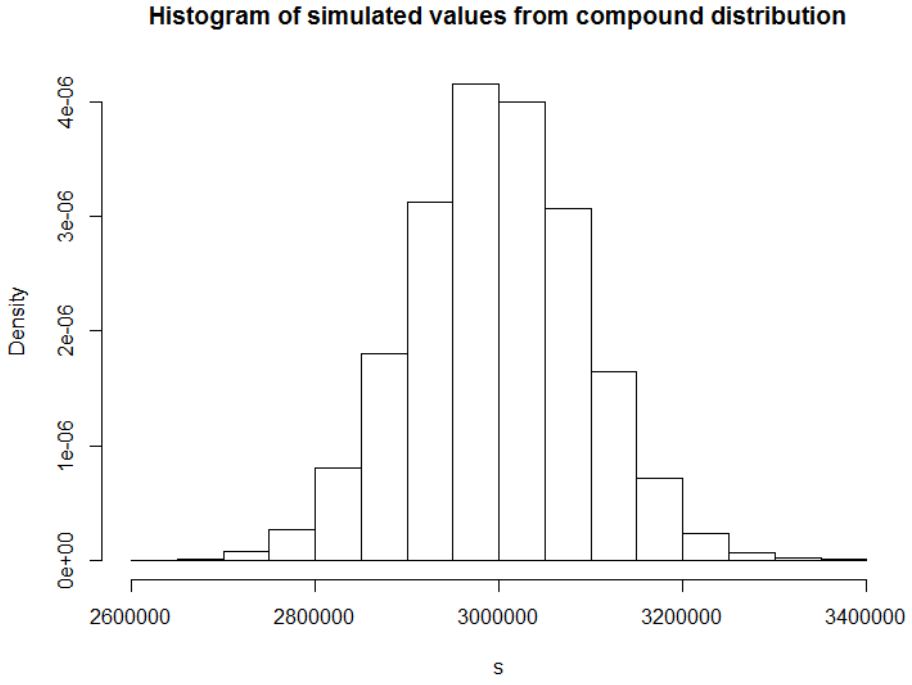


Figure 19.1

We can calculate an empirical density function using the `density()` function and overlay this onto the histogram using the `lines()` function:

```
lines(density(s), col = "blue", lwd = 3)
```

We can then also superimpose a normal or other distribution to see if it provides a good approximation as well as a legend to identify the curves:

```
curve(dnorm(x, mean(s), sd(s)), col = "red", add = TRUE,
      lwd = 3, lty = 2)

legend("topleft",
       legend = c("Empirical density", "Normal approximation"),
       col = c("blue", "red"), lty = c(1,2), lwd = 3)
```

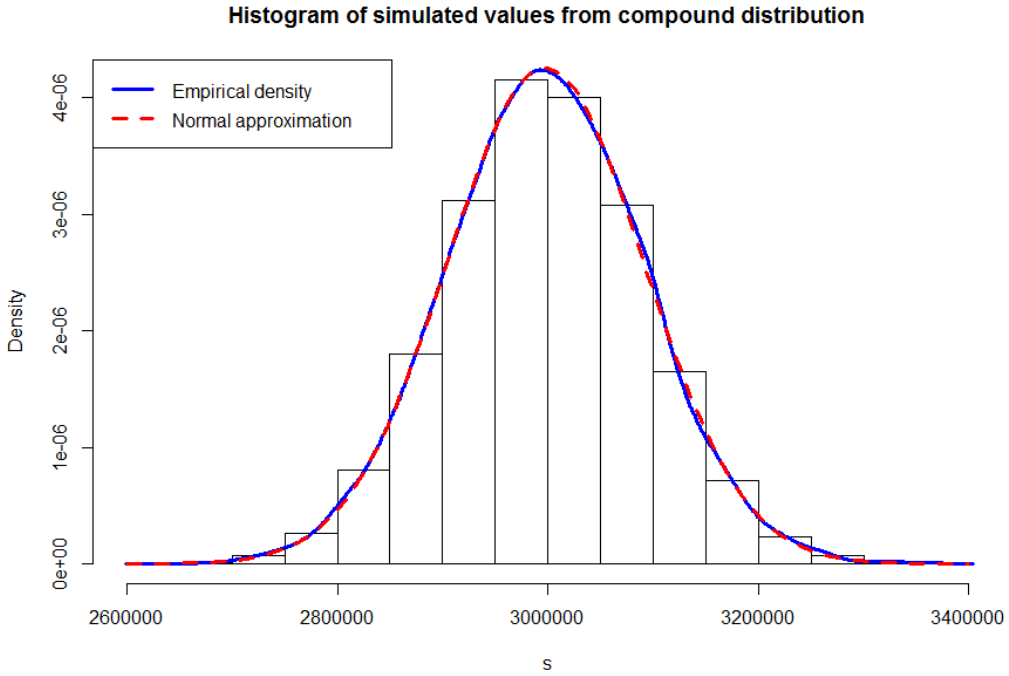


Figure 19.2

The plot indicates that the normal distribution appears to be a reasonable approximation.

However, a better way to check the fit with a normal distribution is to examine a Q-Q plot using the `qqnorm()` function:

```
qqnorm(<simulated values>)
```

If the normal distribution is a good approximation, then the points should be close to the straight diagonal line plotted with `qqline(<simulated values>)`. For example:

```
qqnorm(s, main = "Q-Q plot comparing quantiles of simulated
compound distribution values against normal quantiles")
qqline(s, col = "red")
```

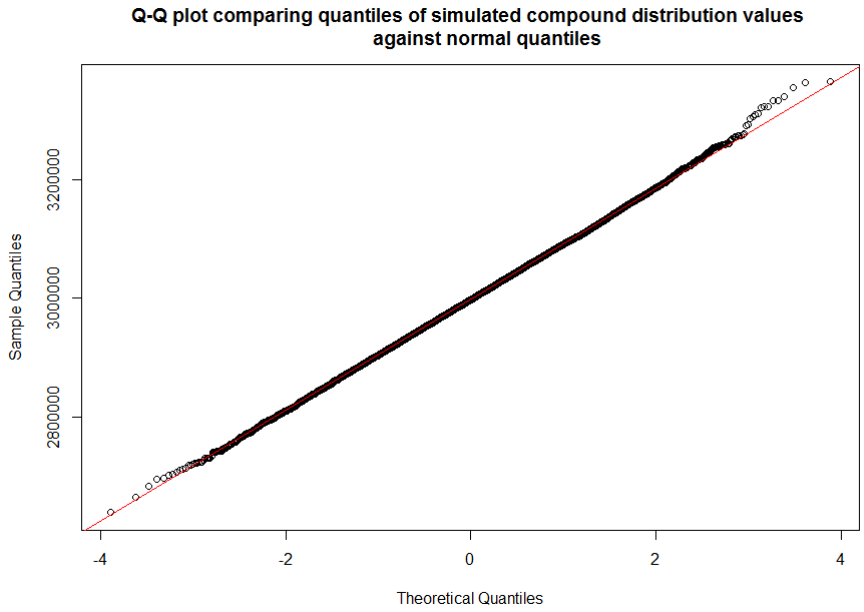


Figure 19.3

Despite showing some deviation to the line in the tails, the Q-Q plot also indicates that the normal distribution appears to be a reasonable approximation.

To check the fit more generally with any model distribution, we can use the `qqplot()` function to compare the sample data to the theoretical values of the fitted distribution as described in Chapter 15.

Sums of independent compound Poisson random variables

A very useful property of the compound Poisson distribution is that the sum of independent compound Poisson random variables is itself a compound Poisson random variable. A formal statement of this property is given below.

Let $S_1, S_2, \ldots, S_n$ be independent random variables. Suppose that each S_i has a compound Poisson distribution with parameter λ_i, and that the CDF of the individual claim amount random variable for each S_i is $F_i(x)$.

Define $A = S_1 + S_2 + \cdots + S_n$. Then A has a compound Poisson distribution with parameter Λ, and $F(x)$ is the CDF of the individual claim amount random variable for A, where:

$$\Lambda = \sum_{i=1}^{n} \lambda_i \quad \text{and} \quad F(x) = \frac{1}{\Lambda} \sum_{i=1}^{n} \lambda_i F_i(x)$$

Recall that Λ is the capital form of the Greek letter λ.

This is a very important result.

To prove the result, first note that $F(x)$ is a weighted average of distribution functions and that these weights are all positive and sum to one. This means that $F(x)$ is a distribution function and this distribution has MGF:

$$M(t) = \int_0^\infty e^{tx} f(x)\,dx$$

where $f(x) = F'(x)$ is the PDF of the individual claim amount random variable for A.

So:

$$M(t) = \int_0^\infty \exp(tx)\,\frac{1}{\Lambda}\sum_{i=1}^{n}\lambda_i f_i(x)\,dx$$

where $f_i(x)$ is the density of $F_i(x)$. Hence:

$$M(t) = \frac{1}{\Lambda}\sum_{i=1}^{n}\lambda_i \int_0^\infty \exp\{tx\} f_i(x)\,dx = \frac{1}{\Lambda}\sum_{i=1}^{n}\lambda_i\, M_i(t) \qquad \textbf{(19.13)}$$

where $M_i(t)$ is the MGF for the distribution with CDF $F_i(x)$.

Let $M_A(t)$ denote the MGF of A. Then:

$$M_A(t) = E[\exp(tA)] = E[\exp(tS_1 + tS_2 + \cdots + tS_n)]$$

By independence of $\{S_i\}_{i=1}^n$:

$$M_A(t) = \prod_{i=1}^{n} E(\exp(tS_i))$$

As S_i is a compound Poisson random variable, its MGF is of the form given by formula (19.11), so:

$$E[\exp(tS_i)] = \exp\left[\lambda_i(M_i(t) - 1)\right]$$

Thus:

$$M_A(t) = \prod_{i=1}^{n}\exp\{\lambda_i(M_i(t)-1)\} = \exp\left\{\sum_{i=1}^{n}\lambda_i(M_i(t)-1)\right\}$$

***ie*:**

$$M_A(t) = \exp\{\Lambda(M(t)-1)\} \qquad \textbf{(19.14)}$$

where:

$$\Lambda = \sum_{i=1}^{n}\lambda_i \text{ and } M(t) = \frac{1}{\Lambda}\sum_{i=1}^{n}\lambda_i M_i(t)$$

By the one-to-one relationship between distributions and MGFs, formula (19.14) shows that A has a compound Poisson distribution with Poisson parameter Λ. By (19.13), the individual claim amount distribution has CDF $F(x)$.

21

Machine learning

Syllabus objectives

5.1 Explain and apply elementary principles of machine learning.

- 5.1.1 Explain the bias/variance trade-off and its relationship with model complexity.
- 5.1.2 Use cross-validation to evaluate models on unseen data, and to estimate hyperparameters.
- 5.1.3 Explain how regularisation can be used to reduce overfitting in highly parameterised models.
- 5.1.4 Use software to apply supervised learning techniques to solve regression and classification problems.
- 5.1.5 Use metrics such as precision, recall, F_1 score and diagnostics such as the ROC curve and confusion matrix to evaluate the performance of a binary classifier.
- 5.1.6 Apply unsupervised learning techniques (principal component analysis and K-means clustering) to reduce data dimensionality, identify latent substructure and detect anomalies.

0 Introduction

In this chapter we introduce the concept of machine learning. This is an extremely broad topic with a wide range of applications. We focus on getting to grips with some fundamentals of two types of machine learning, supervised and unsupervised.

We start with a description of machine learning as well as some examples from everyday life that may be familiar.

In Section 2 we look at supervised learning in more detail. First, we introduce the aim of a supervised learning algorithm, which is to approximate the functional relationship between input variables (*eg* age, sex and smoking status) and an output variable (*eg* life expectancy). We then consider how we might evaluate the model output and some of the issues that can lead to poor performance. We also look at some common supervised learning problems and the typical workflow involved in approaching such problems.

In Section 3 we turn to applications of supervised learning and introduce some commonly used techniques with some of examples of them in action.

Finally, we give an overview of unsupervised learning and consider the K-means algorithm and principal components analysis as example applications.

1 What is machine learning?

Machine learning is a collection of methods for the automatic detection and exploitation of patterns in data. The field has arisen in response to vast increases in the volume of available data, and in the speed and capacity of computers to analyse it.

When describing the new age of 'big data' researchers often talk about the three V's: volume, velocity and variety. There are now very large *volumes* of data in use, which computers can process very rapidly (*velocity*) and the data can take many different forms (*variety*).

While there is a large overlap between the fields of machine learning and statistical modelling, in machine learning, there is typically more emphasis on *prediction* than on *inference*.

For example, using a fitted model to predict future claims rather than carrying out tests on the model's parameters or constructing confidence intervals.

Machine learning uses highly flexible models, which can capture complex and subtle patterns in data. The downside of this flexibility is the tendency of machine learning models to *overfit* to the data used to train them: they mistakenly capture incidental properties of the data used to train them.

The idea of overfitting to the training data is explored further in Section 2.4.

Broadly, machine learning problems can be divided into *supervised learning* problems and *unsupervised learning* problems. This chapter will mostly focus on supervised learning.

Examples of problems which are commonly solved using machine learning include:

- targeting of advertising at consumers using web sites
- location of stock within supermarkets to maximise turnover
- forecasting of election results
- prediction of which borrowers are most likely to default on a loan.

Question

Give some other examples where machine learning is used:

(a) in everyday life

(b) in an actuarial / insurance / financial context.

Solution

(a) Other everyday examples include:

- using an internet search engine such as Google to find relevant web pages
- using a spam filter to identify and remove unwanted emails
- using face recognition to identify known criminals on CCTV footage
- identifying criminals using fingerprints or DNA
- recommending items to purchase on online shopping sites
- matching job applicants to available positions
- suggesting suitable matches on a dating app
- automatically identifying people in photographs
- recognising voice commands, *eg* in software such as Siri, Cortana and Alexa
- converting handwriting to text
- translating text from one language to another.

(b) Other examples in an actuarial / insurance / financial context include:

- classifying the risk for motor insurance policyholders using in-car monitoring devices
- identifying marker genes that are associated with particular medical conditions
- identifying insurance claims that might be fraudulent
- identifying fraudulent benefit claims
- identifying fraudulent tax declarations
- predicting life expectancy based on characteristics such as age, sex and smoking status
- predicting claim values using policyholder characteristics
- predicting whether applicants will default on mortgage payments based on salary, debts and credit history.

2 Supervised learning

In a typical supervised learning problem, the aim is to determine the relationship between a collection of d input variables, $x_1,...,x_d$, and an output variable, y, based on a sample of training data $\left(\underline{x}_i, y_i\right)$ $i = 1,2,...,n$. For each observation, the values of the **input variables, $\underline{x}_i = \left(x_{i1}, x_{i2},..., x_{id}\right)$, are structured as a vector, so that** the entirety of the input data, $\boldsymbol{x}$, **can be considered as an $n \times d$ matrix, *ie* with observations in rows and input variables in columns.**

With supervised learning, the algorithm has a target output, y_i, for each point in the training data. It aims to predict this target based on the values of the input variables by identifying a relationship between input and output.

Once a model has been created using the training data, it can be used to predict the output in cases where it has not yet been observed, based on the values of the input variables.

Examples of supervised learning tasks commonly encountered in actuarial work include:

- **determining the relationship between expected survival time (the output) and background covariates such as age, sex and smoking behaviour (the inputs)**
- **predicting whether or not mortgage applicants are likely to default over the mortgage term (the output), given background information on salary, debts and prior credit behaviour (the inputs).**

In the first example, a supervised learning model would be trained on a data set where:

- y_i is the observed survival time for the i th life in the training data set
- $\underline{x}_i = \left(x_{i1}, x_{i2},..., x_{id}\right)$ is the corresponding vector of covariate values for this life, *ie* age, sex, smoker status *etc*.

Once trained, it can be used to predict survival times for lives still alive based on their characteristics.

In the second example, the model would be trained on a data set where:

- y_i represents whether or not the i th applicant defaulted on their mortgage
- $\underline{x}_i = \left(x_{i1}, x_{i2},..., x_{id}\right)$ is the corresponding vector of background information for this applicant.

Once trained, we can predict whether a new applicant will default on their mortgage.

Another actuarial example is using a generalised linear model in insurance pricing to predict claim values based on policyholder characteristics.

2.1 The functional relationship

Suppose we write the relationship between input and output as

$$y_i = f\left(\underline{x}_i\right) + \varepsilon_i$$

where f is the functional relationship to be learnt and ε is an error term.

f represents the true unknown relationship between the input and the output. When training a supervised learning model, we are trying to approximate this relationship as best as possible based on the available data.

ε is taken to be a random variable with mean zero, representing variability in y that is unrelated to x. For each observation i, the input $\underline{x}_i$ is a vector $\left(x_{i1}, x_{i2}, ..., x_{id}\right)$ of d input variables.

One familiar setting is where f is a linear function of the inputs and the errors ε_i are taken to be normally distributed random variables: this is the problem of linear regression, covered in Subject CS1.

In the (multiple) linear regression model, the assumed relationship between input and output is as follows:

$$Y_i = \alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_d x_{id} + \varepsilon_i \qquad i = 1, \ldots, n$$

So, in this case, it is assumed that f takes the form:

$$f(x_i) = \alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_d x_{id}$$

Supervised learning methods seek good approximations to f in settings where the functional form of the relationship between inputs and outputs is unknown, and when d, the number of input variables, is potentially large.

In linear regression we assume that the functional form of f is linear. Not all machine learning methods assume a particular form of f.

The direct output of a supervised learning method is a function $\hat{f}$, thought of as a best approximation to f, which can then be used in *prediction* and *inference* tasks.

2.2 Prediction vs inference

For many practical problems, inputs x are readily available, but the output is unknown. For example, x might be the background characteristics of a loan applicant, and y the binary variable indicating whether or not the applicant (if given a loan) will default. Supposing that the error term averages to zero, we predict y as $\hat{y} = \hat{f}(x)$.

Here $\hat{f}(x)$ is the approximation of f outputted by the supervised learning algorithm, which would have been trained on a data set for which the outputs are known.

Where prediction is the primary concern, machine learning methods often regard $\hat{f}$ as a black box – its functional form is unimportant, so long as it produces good predictions.

If prediction is the main aim, then we may not be that interested in the underlying structure of the relationship between input and output, only how accurate our model is.

However, we are often also interested in *inference*: determining which inputs, or combinations of inputs, are most closely associated with the output; discriminating between different hypotheses about the data, and making statements about the uncertainty in predictions.

While the most practical use of machine learning methods is accurate prediction, the question of inference is also important. For example, where machine learning methods are used in making credit decisions, there may be strong regulatory or ethical pressures to understand the factors that contribute to a decision.

2.3 Typical supervised learning problems

Problems where the output variable to be predicted is a numerical variable, such as an amount of revenue, an individual's survival time or the number of employees in a firm are said to be *regression* problems.

Problems where the output variable is qualitative (categorical), such as when predicting credit default (yes/no), are said to be *classification* problems. The most common classification tasks are binary classification tasks, where the outcome variable takes two distinct values.

Question

Give examples of problems that would come under the headings of classification and regression.

Solution

An example of a classification problem is a spam filter that classifies emails into the two categories 'Safe' or 'Suspicious'.

An example of a regression problem is a health awareness app that predicts the user's life expectancy.

Commonly used statistical models, such as linear regression for continuous output variables, or logistic regression for binary output variables, are examples of *parametric* machine learning methods. They make assumptions about the functional form of f, which simplify the learning problem. Moreover, these assumptions give the regression parameters an explicit interpretation. For example, in the linear model:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

β_1 is the expected change in the output variable y for a unit change in x.

By contrast, many machine learning methods make no (or few) assumptions about the explicit functional form of f. Instead, they aim to produce an estimate, $\hat{f}$, that performs well on the data used to train the model, without being too badly behaved. These more flexible methods are often called non-parametric methods. This does not mean that they don't have parameters – they typically have many – rather, it means that their parameters do not have simple interpretations in the context of the problem, and they are not usually of direct interest.

A wealth of different non-parametric **machine learning methods exists, including:**

- **smoothing splines**
- **support vector machines**
- **neural networks.**

These methods will not be discussed in this chapter, but they are well-covered in the references at the end of the chapter. In Section 3.3, we will demonstrate bagged decision trees and the random forest. These are flexible non-parametric models with comparable performance to the methods above.

2.4 Evaluating performance for regression problems

To evaluate the performance of a supervised learning **model, we compare the model's predictions with the true output values. For a continuous output variable, one common measure is the *mean square error* (MSE):**

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

More precisely, this might be called the *training MSE*, because it is evaluated on the data used to train the model. Given that we want to use the model to predict *unseen* data, this measure does not quite match up with our requirements. Given enough parameters, we could easily make a model that matches the training data with zero training error, but such a model would likely perform badly when used to predict new observations.

The aim is to capture the underlying relationship between input and output, not just capture the idiosyncrasies of the training data. With enough parameters, a model can be built so that the predictions will exactly match the observed output for the training data. However, this reflects the specifics of that particular data set and is therefore not likely to produce accurate predictions for other, unseen, data points.

Figure 21.1 shows three different models fitted to a sample of size $n = 50$. It illustrates the two different modes of failure when fitting statistical models to data, and their relationship with model complexity.

The true function f is a polynomial of degree 5 in a single variable x, and the three different models are polynomials of different degrees, although the behaviour exhibited here is seen for supervised learning models more generally.

The Core Reading here means that the behaviour demonstrated in his section (the two modes of failure) is not specific to polynomials.

In a polynomial regression model, the approximation to f takes the form:

$$\hat{f}(x) = \sum_{k=0}^{d} \hat{\beta}_k x^k$$

where the parameter estimates, $\hat{\beta}_k$, are the values that minimise the training MSE.

Figure 21.1 shows $\hat{f}$ for $d = 1, 5, 15$.

Three models have been fitted to the training data to compare their behaviour:

- a linear model, which has far fewer parameters than the true function
- a polynomial of degree 5, which has the same number of parameters as the true function
- a polynomial of degree 15, which has many more parameters than the true function.

Figure 21.1 below shows a plot of the training data, the three fitted models and the true model:

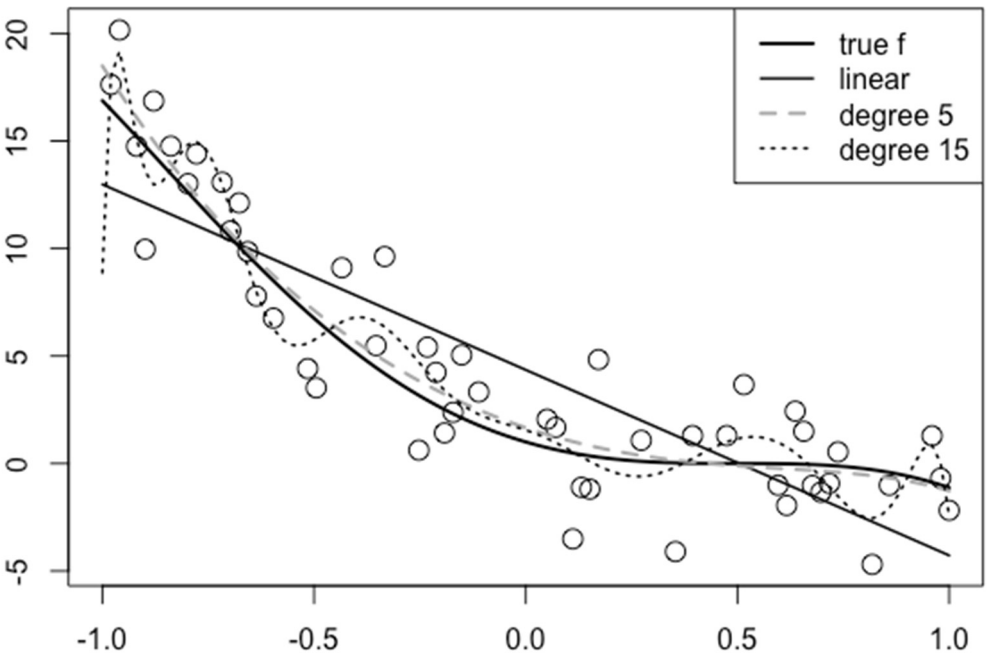


Figure 21.1

For $d = 1$, the model is just a straight line, and so it is too rigid to accommodate the curvature in f. This leads to a bias in the predictions of $\hat{f}$: systematically too large in the centre, and too small at the edges.

This model doesn't have enough parameters to capture the changes in shape that we can see exhibited by the true f. We saw this same issue when we considered the graduation of mortality rates by parametric formula in Chapter 11. If we use a formula with too few parameters, the graduated rates will be overgraduated. They will be smoothed too much and will not follow the underlying pattern of the rates closely enough, *eg* 'smoothing over' genuine features such as the accident hump.

In contrast, the model with $d = 15$ is too flexible: $\hat{f}$ bends to accommodate noise in the training data. The more flexible the model, the better it will fit the training data. However, in bending to accommodate the training data, $\hat{f}$ departs substantially from f so that the error in predicting unseen observations will be large.

The more parameters in the model, the more flexible it is. If we fitted a polynomial with high enough degree to this data, then it would exactly go through each of the observed data points (*ie* it would have a training MSE of 0). This would perfectly capture the specifics of the training data but would not be a good approximation to the true relationship shown on the graph. So, it would likely not produce accurate predictions for unseen data.

We also saw this issue with graduation by parametric formula. If we use a formula with too many parameters, the graduated rates will be undergraduated. They will follow the crude rates too closely, reflecting a lot of the random 'noise' present in the data, rather than just capturing the underlying pattern of the rates.

The model with $d = 5$ approximates the true f well, unsurprisingly, since it has exactly the right form. While the illustration here uses a simple class of models – polynomials, the relationship between the flexibility of a model, as measured by the number of free parameters, and prediction performance on unseen data, is very general. Evaluating and controlling model flexibility is an important part of designing good machine learning methods.

2.5 The bias-variance decomposition

For a given (unseen) value x_0, the expected mean square error for the corresponding output value y_0 can be decomposed into three interpretable terms:

$$E\left[\left(y_0 - \hat{f}(x_0)\right)^2\right] = \text{var}\left(\varepsilon_0\right) + \text{var}\left[\hat{f}(x_0)\right] + \left\{f(x_0) - E\left[\hat{f}(x_0)\right]\right\}^2$$

The previous MSE formula was for the training data. This is now the expected MSE for an unseen data point based on our approximation, $\hat{f}$. There are two elements of randomness over which this expectation is being taken. Firstly, $\hat{f}$ is being treated as a random variable (*ie* we are considering the estimator, rather than the estimate). This is because different training data sets lead to different approximations of the functional relationship. Secondly, the value of y_0 given x_0 is random due to the presence of the error term, ε_0.

The derivation of this decomposition is shown in the appendix in Section 6.

The *expected MSE* on the left can be thought of as the average of the (squared) error on an unseen observation with input value x_0, taken over the distribution of random errors in the training sample, $\varepsilon_1,...,\varepsilon_n$, and the random error of the unseen observation, ε_0. Note that here $\hat{f}$ is considered a random variable, as it depends on the training observations $\left(\underline{x}_i, y_i\right)$, $i = 1,2,...,n$.

The Core Reading is describing the two elements of randomness outlined previously. The distribution of random errors in the training sample drives different approximations of the functional relationship for different training data sets. The distribution of the random error for an unseen observation drives different values of y_0 for a given x_0.

The *bias-variance decomposition* allows us to understand the contributions to the average squared error that come from the random error ε_0, the randomness in the training sample, which leads to randomness in $\hat{f}$, and the structural differences between the data and the model.

$\text{var}(\varepsilon_0)$ is the *error variance*, sometimes called the *irreducible error*. It represents a fundamental limitation on the precision of an individual prediction, as a result of the random error term present in each observation.

$\text{var}\left[\hat{f}(x_0)\right]$, the *variance* of $\hat{f}$, is the contribution to the expected error that comes from having used only a finite sample of size n: a different sample of size n would give a different estimate. High variance is a sign that the model is *too flexible* – this additional flexibility soaks up noise in the training data, leading to *overfitting*. This mode of failure can be seen in the left panel of Figure 21.2 below.

If the variance of $\hat{f}$ is large, then this means that the estimate varies a lot between training samples. This is a sign that it is overfitting, capturing the idiosyncratic trends of the training data.

$\left\{f(x_0) - E\left[\hat{f}(x_0)\right]\right\}^2$ is the square of the *bias* of $\hat{f}$. It represents the systematic incompatibility between the model and the data.

Recall from Subject CS1 that the bias of an estimator is the difference between its expectation and the quantity being estimated. Here the bias of $\hat{f}$ is given by:

$$E[\hat{f}] - f$$

So, the square of the bias is:

$$\left(E[\hat{f}] - f\right)^2 = \left(f - E[\hat{f}]\right)^2$$

This is a measure of the fundamental difference between the structure of the model being fitted and the true functional relationship.

For example, if simple linear regression is used for a problem where the relationship between x and y is genuinely non-linear, then even in the limit of an infinite sample, and small error variance, an error will be incurred when using $\hat{f}$ in place of f: this is bias. This mode of failure can be seen in the right panel of Figure 21.2 below. High bias is a sign that the model is not flexible enough – it is not able to capture all of the signal in the data, leading to *underfitting*.

Again from Subject CS1, we know that the MSE of an estimator can be written as:

$$MSE[\hat{f}] = \text{var}(\hat{f}) + bias(\hat{f})^2$$

So, for a given (unseen) value x_0, the expected mean square error for the corresponding output value y_0 can be thought of as the irreducible error (*ie* $\text{var}(\varepsilon_0)$) plus the MSE of the estimator, $MSE[\hat{f}]$. Although we have no control over $\text{var}(\varepsilon_0)$, we can try to find an approach of estimating the functional form that minimises $MSE[\hat{f}]$ and hence minimises the expected MSE for an unseen output. It is possible for an estimator to be biased but have a lower overall MSE, in which case it may be the preferred approach. Even though it is wrong on average (biased), it is more likely to give something closer to the true functional relationship (low MSE).

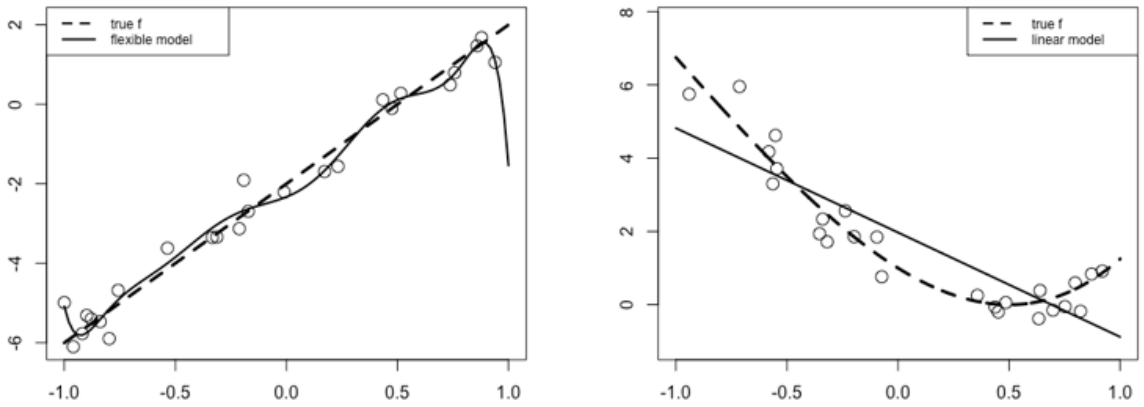


Figure 21.2

Machine learning models typically contain many parameters, and so are capable of capturing faint patterns, which simpler methods might not be able to distinguish from noise. However, in being so flexible, their primary mode of failure is in overfitting, which results in high-variance predictions. For this reason, it is important to control the flexibility of our models (see Section 3.1 on penalized regression models) or find ways of reducing the variance of predictions (see Section 3.3 on decision trees and random forests).

2.6 Evaluating binary classifiers

When evaluating a binary classifier, it is important to note that there are two different kinds of classification error: classifying an observation with $y = 1$ to the class $y = 0$, and classifying an observation with $y = 0$ to the class $y = 1$. In principle, these two errors may have very different costs: one example is if the binary classifier is a test for a disease. Incorrectly diagnosing a healthy patient may incur moderate costs but failing to diagnose a sick patient could have serious consequences.

The output of a binary classifier can be represented as shown in the table below. When populated with values from data, this table is known as a *confusion matrix*.

The terminology is based on the idea that the matrix of possible outcomes quantifies the extent to which, in the context of disease diagnosis, the test 'confuses' patients who do / do not have the condition.

		Test result classified / predicts patient as having condition	
		YES	**NO**
Patient actually has condition	**YES**	**True positive (TP)**	**False negative (FN)**
	NO	**False positive (FP)**	**True negative (TN)**

Question

A country is introducing a new screening programme for early identification of people with a particular type of cancer.

(i) Explain what 'false positive' and 'false negative' results would be in this context.

(ii) Discuss the impact of false positives and false negatives from the point of view of a patient.

(iii) State an additional concern regarding false negatives if this had been a test for an infectious disease.

Solution

(i) A false positive is a patient that the test flags as having the disease, but in fact does not.

A false negative is a patient that the test indicates as not having the disease, when in fact they do have it.

(ii) A false positive outcome is undesirable because the patient may be caused unnecessary worry or required to undergo further tests or treatment before it is established that they do not actually have the disease.

A false negative outcome is also undesirable because the patient may not now be identified early enough to receive effective treatment for the disease.

(iii) With an infectious disease, there is the additional concern that false negative patients may unknowingly spread the disease to other people.

There are several useful measures we can calculate from the confusion matrix to gauge the effectiveness of the test.

Aspects of the performance of a binary classifier are summarised in single-number measures, such as:

***Accuracy*. This is the proportion of correct predictions. Note that raw accuracy scores can be misleading, particularly for imbalanced data sets, where most of the observations come from a single class. For a problem where 95% of the observations are from one class, even the *naïve classifier* that assigns all observations to the more common class has 95% accuracy. Hence it is common to report the accuracy alongside the accuracy of the naïve classifier for comparison.**

Using the abbreviations in the table, we have:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

This measure can also be used for classification problems where the outcome variable can take more than two values.

***Precision and recall*. Consider a diagnostic test for a medical condition. The patients who take the test either have the condition or they do not. The test will classify (predict) patients as having the condition or not according to whether the outcome of the test fulfils certain criteria.**

***Precision* is the proportion of cases correctly classified as positive. Using the abbreviations in the table, this is:**

$$\textbf{precision} = \frac{\textbf{TP}}{\textbf{TP+FP}}$$

Ideally, we would like to have $FP = 0$, which would result in a precision of 1, *ie* 100%.

***Recall* is the proportion of positive cases that were identified:**

$$\textbf{recall} = \frac{\textbf{TP}}{\textbf{TP+FN}}$$

In Subject CS1, this measure is called the *sensitivity* and is equivalent to $1 - P(\text{type II error})$ or the power of a test where the null hypothesis is that the patient is healthy. It is also called the true positive rate. If $FN = 0$, *ie* if the test has not missed anyone who has the condition, it equals 100%.

These can be combined in a single measure known as the *F1 score*:

$$\textbf{F1 score} = \frac{\textbf{2 x precision x recall}}{\textbf{precision + recall}}$$

The '*F*' here arose historically and doesn't actually stand for anything. The '1' subscript just identifies this measure out of several similar measures that have also been proposed and could be used instead.

Question

(i) Explain why the precision and recall are not necessarily useful stand-alone measures, by considering example confusion matrices.

(ii) Derive and simplify a formula for the harmonic mean of the precision and the recall.

Hint: The harmonic mean of a set of values is the reciprocal of the arithmetic mean of their reciprocals.

(iii) Comment on the answer in (ii).

Solution

(i) Consider a diagnostic test that predicts every individual as having the condition. In this case, we have FN = 0 (the other values in the confusion matrix are not directly relevant for this example, though we would also have TN = 0). So, the recall is:

$$\text{recall} = \frac{\text{TP}}{\text{TP+FN}} = \frac{\text{TP}}{\text{TP}} = 1$$

This is the best possible recall score. However, this test is likely not very useful, it does not differentiate between patients at all.

Next consider a diagnostic test that is always correct when it predicts a patient as having a condition but rarely makes such predictions across the affected population. In this case we have that FP = 0 (the other values in the confusion matrix are not directly relevant for this example as long as we assume that $\text{TP} > 0$). So, the precision is:

$$\text{precision} = \frac{\text{TP}}{\text{TP+FP}} = \frac{\text{TP}}{\text{TP}} = 1$$

This is the best possible precision score. However, this test on its own may not be that useful as it identifies few patients with the condition.

For example, imagine there is a particular compound that, when present in the blood, means that the patient definitely has the condition but the lack of that compound is inconclusive. If the presence of that compound is extremely rare within the population with the condition, then this test may not be that useful as the sole diagnostic tool. However, it could of course be used as part of a more comprehensive diagnostic strategy.

(ii) Using the hint given, we can see that the harmonic mean H of the precision and the recall can be found from the equation:

$$\frac{1}{H} = \frac{1}{2}\left(\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}\right)$$

So:

$$\frac{1}{H} = \frac{1}{2}\left(\frac{\text{Recall}+\text{Precision}}{\text{Precision}\times\text{Recall}}\right) \Rightarrow H = \frac{2\times\text{Precision}\times\text{Recall}}{\text{Recall}+\text{Precision}} = F_1$$

(iii) F_1 is the harmonic mean of the precision and the recall. This is different from the more familiar *arithmetic* mean, but it also gives an average value of the two measures taken together and results in a value in the same range, *ie* 0 to 1.

Another measure is the *false positive rate*.

There is a trade-off between *recall* (the *true positive rate*, also known as the *sensitivity*) and the false positive rate (the proportion of cases that are incorrectly classified as positives). The false positive rate is:

$$\textbf{false positive rate} = \frac{\textbf{FP}}{\textbf{TN+FP}}$$

This is not the same as (1 – the true positive rate), *ie* it is not the same as 1 – the recall rate.

In the context of this trade-off, the sensitivity is often compared with the *specificity*, which is 1 – false positive rate.

Recall from Subject CS1, that the *specificity* is defined to be the true negative rate (or $1-P(\text{type I error})$ for a test where the null hypothesis is that the patient is healthy).

The trade-off between recall and the false positive rate can be illustrated using a *receiver operating characteristic (ROC) curve*. An example is shown below, taken from Alan Chalk and Conan McMurtrie 'A practical introduction to Machine Learning concepts for actuaries' *Casualty Actuarial Society E-forum,* Spring 2016.

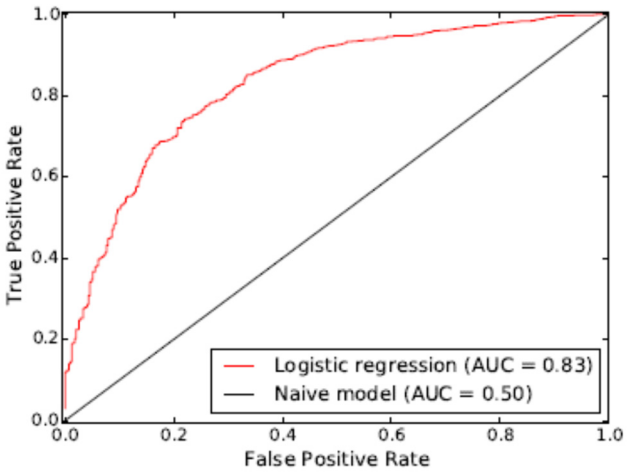
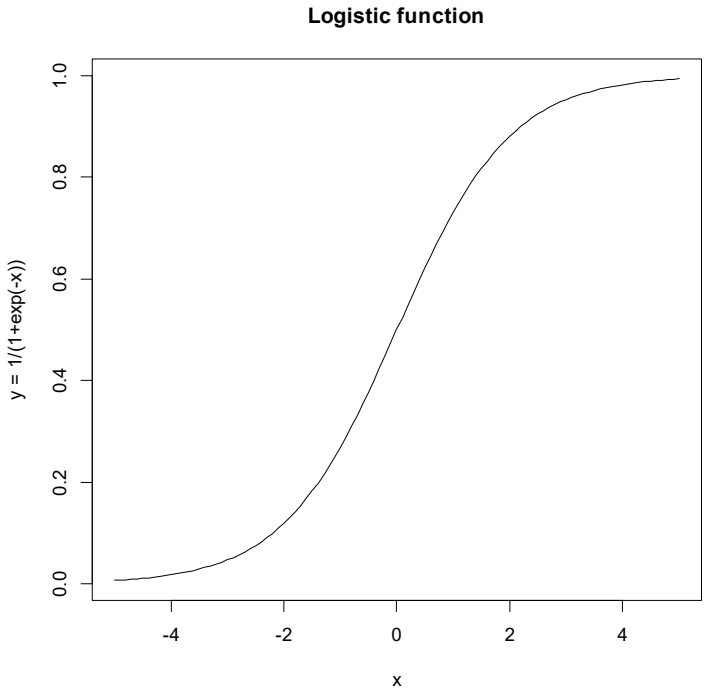


Figure 21.3

This figure shows the ROC curve for a logistic regression classifier fitted to the cause codes for aircraft accidents.

Logistic regression is a type of generalised linear model where the response variable is a binary outcome. It is based on the logistic function $f(x)=\frac{1}{1+e^{-x}}$, shown in the graph below. This function converts an input value, which can be anywhere in the entire range $-\infty < x < \infty$, to an output value on a continuous scale between 0 and 1. If we interpret the output value as a probability, we can convert it to a categorical output by saying that values exceeding a specified value p (*eg* $p=0.5$) correspond to Yes, while smaller values correspond to No.



This graph was plotted using the R code:

```
logistic <- function(x) 1/(1 + exp(-x))
plot(logistic, xlim = c(-5,5), main = "Logistic function",
ylab = "y = 1/(1+exp(-x))")
```

The ROC curve is used when the test involves a threshold of some kind. In the aircraft accidents example, the aim of the logistic regression model is to identify accidents that were caused by the aircraft (outcome 1) or for some other reason (outcome 2) based on key words in descriptions of the incidents. The output of the logistic regression model is interpreted as the probability of outcome 1. A threshold is then chosen such that accidents with an output above that threshold are classified as outcome 1 and outcome 2 otherwise. The false positive rate and true positive rate can then be calculated for that particular threshold based on the classifications. This is repeated for different thresholds and the points plotted on a graph.

Points near the top left of the graph correspond to a good test where the true positive rate is high and the false positive rate is low.

The diagonal line represents a naïve model that classifies at random.

The diagonal line corresponds to a neutral 'zero-sum' test where there is a simple trade-off with any improvement in the true positive rate being matched by an equal deterioration in the false positive rate. As the diagonal represents naïve models that randomly guess the outcome for each individual, it is therefore a lower bound for models with any predictive power.

The area under the ROC curve (AUC) is a commonly reported single-number measure of model performance. However, as the AUC aggregates performance over the entire range of false positives, it is not a reliable guide to performance in realistic problem domains, where only classifiers with well-controlled error rates would be deployed: ordinarily, only classifiers with small false positive rates would be useful in practice.

In the previous figure, the AUC for the naïve model is the area of the lower right triangle, which is 0.5. For the logistic regression model, the AUC is 0.83. The area of the whole rectangle is 1 (representing the best possible AUC for an ROC curve). So, a model that has some predictive power over just random guessing will have an AUC between 0.5 and 1.

However, the AUC can sometimes be misleading. It is a single metric that describes model performance for a range of thresholds, which may not always be an appropriate measure. For example, say that the treatment for a particular condition is incredibly invasive, whilst the condition itself is not particularly serious. When diagnosing patients, it may be desirable to keep the false positive rate fairly low (to avoid giving the treatment to people that don't need it). Suppose that a false positive rate below 5% is desired. In this case, a model that has strictly higher true positive rates for false positive rates below 5% should be preferred, even if it has a lower overall AUC.

Question

Two different tests have been applied to 100 individuals to identify whether or not a particular feature is present. These resulted in the following confusion matrices:

(a) **Test 1**

PREDICTED / ACTUAL	YES	NO	TOTAL
YES	TP = 76	FN = 4	80
NO	FP = 4	TN = 16	20
TOTAL	80	20	100

(b) **Test 2**

PREDICTED / ACTUAL	YES	NO	TOTAL
YES	TP = 70	FN = 10	80
NO	FP = 1	TN = 19	20
TOTAL	71	29	100

Calculate the precision, recall, F_1 score, false positive rate and accuracy for each matrix and comment on the answers.

Solution

(a) $\text{precision} = \frac{\text{TP}}{\text{TP}+\text{FP}} = \frac{76}{76+4} = 95\%$, $\text{recall} = \frac{\text{TP}}{\text{TP}+\text{FN}} = \frac{76}{76+4} = 95\%$

$$F_1 \text{ score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{2 \times 95\% \times 95\%}{95\% + 95\%} = 95\%$$

$$\text{false positive rate} = \frac{\text{FP}}{\text{TN}+\text{FP}} = \frac{4}{16+4} = 20\%$$

$$\text{accuracy} = \frac{\text{TP+TN}}{\text{TP+TN}+\text{FP}+\text{FN}} = \frac{76+16}{76+16+4+4} = 92\%$$

(b) $\text{precision} = \frac{\text{TP}}{\text{TP}+\text{FP}} = \frac{70}{70+1} = 98.6\%$, $\text{recall} = \frac{\text{TP}}{\text{TP}+\text{FN}} = \frac{70}{70+10} = 87.5\%$

$$F_1 \text{ score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{2 \times 98.6\% \times 87.5\%}{98.6\% + 87.5\%} = 92.7\%$$

$$\text{false positive rate} = \frac{\text{FP}}{\text{TN}+\text{FP}} = \frac{1}{1+19} = 5\%$$

$$\text{accuracy} = \frac{\text{TP+TN}}{\text{TP+TN}+\text{FP}+\text{FN}} = \frac{70+19}{70+19+1+10} = 89\%$$

The precision values show that Test 2 is more effective at correctly identifying individuals who do have the feature, *ie* out of those predicted as having the condition, Test 2 gets a higher proportion correct.

However, the recall values show that Test 1 is much better at identifying individuals who do have the feature. In other words, Test 1 correctly identifies a higher proportion out of the population of those with the condition.

Given that the tests each outperform the other for one of these metrics, which of them may be more useful depends on the misclassification costs.

According to the F_1 scores, the overall performance of Test 1 is (just) better than Test 2.

However, Test 1 has a much higher false positive rate, indicating that it flags a higher proportion of individuals who do not have the feature as having it.

The accuracy shows that Test 1 is a better predictor overall (although they are very similar). A naïve predictor that predicts everyone as having the feature has an accuracy of 80%. Both tests have a higher accuracy, indicating they have some predictive power over and above this naïve classifier.

2.7 Train-validation-test

The most important test of a machine learning method is an evaluation of its out-of-sample performance: how does it perform on unseen observations?

Train/test split

It is common to withhold a fraction of a dataset to evaluate out-of-sample performance. This is called a *training/test* split. The split should be taken at random, or a suitably stratified sample taken where there are distinct subpopulations. The training sample is used to fit the model. The input variables of the test observations are then passed to the trained model to determine predicted outputs for the test observations. We can then compute the average prediction error on the test observations.

Train/validation/test split

Some machine learning methods have tuneable parameters, which we might call hyperparameters, that control aspects of the model-fitting process, or the penalty for overfitting. One example is the regularisation parameter in penalised regression, discussed in Section 3.1.

These hyperparameters can only be determined by evaluating model performance on out-of-sample data. This suggests a variation on the train/test split above in which the dataset is split into training, validation and test subsets. The model is fit using the training data, for a range of hyperparameter values, and the best hyperparameter values are chosen by evaluating model performance on the validation subset. The out-of-sample performance of this best model can then be evaluated on the test set.

We can say that the *parameters* of a model are variables internal to the model whose values are estimated from the data and are used to calculate predictions using the model.

Hyperparameters are variables external to the model whose values are set in advance by the user. They are chosen based on the user's knowledge and experience in order to produce a model that works well. The chosen hyperparameters are then evaluated and tuned using the validation data set.

Question

Give some other examples of hyperparameters from other models.

Solution

Graduation

If we are graduating mortality data using a Gompertz-Makeham formula $\mu_x = p_1(t) + \exp[p_2(t)]$, where $p_1(t) = \sum_{k=0}^{r-1} a_k t^k$ and $p_2(t) = \sum_{k=0}^{s-1} b_k t^k$ are polynomial functions, we need to decide on the values to use for r and s, which determine the order of the two polynomials ($r-1$ and $s-1$).

This form of the Gompertz-Makeham family of curves is the one used most widely. However, it does not match the form given on page 32 of the *Tables*.

Time series

If we are fitting a linear time series using an ARIMA model, we need to decide on the values of d, p and q, which determine the number of levels of differencing to apply and the number of moving average and autoregressive terms to include.

K-fold cross-validation

An alternative to the train/validation/test split considered above is to split the data into K subsets of roughly equal size. K different instances of the model are then trained, with instance i using all data except the ith subset. Subset i is then used to evaluate the out-of-sample error of model instance i. The average of the K different out-of-sample errors can then be used as a proxy for the true out-of-sample error.

An advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once.

2.8 Typical supervised learning workflow

Supervised learning tasks can be broken down into a sequence of steps.

Collecting data

Data may come from a variety of sources, including sample surveys, population censuses, company administration systems and databases constructed for specific purposes (such as the Human Mortality Database, www.mortality.org).

During the last 20–30 years, the size of datasets available for analysis by actuaries and other researchers has increased enormously. Datasets, such as those on purchasing behaviour collected by supermarkets, relate to millions of transactions.

Types of data

There are many different types of data we might need to deal with. The table below illustrates the 'traditional' types of data that have been used by actuaries and statisticians.

DATA TYPES				
NUMERICAL (*ie* numbers)		**CATEGORICAL** (*ie* not numbers)		
DISCRETE	**CONTINUOUS**	**ATTRIBUTE (DICHOTOMOUS)**	**NOMINAL**	**ORDINAL**
↓	↓	↓	↓	↓
Age last birthday	Exact age	Alive / Dead	Customer name	Month (Jan, Feb, Mar, …)
Number of children	Salary	Male / Female	Type of claim	Exam grade (A, B, C, …)
Number of claims	Claim amount	Claim / No claim	Occupation	Size (S, M, L, XL)
		Pass / Fail	Marital status	Agree/Don't know/Disagree
			Country	
			Colour of car	

Attribute (or *dichotomous*) data refers to variables whose values consist of just two categories.

Ordinal variables take values that can be ordered in a natural way, whereas the values for *nominal* variables cannot.

Some variables can be classed in several ways, *eg* the number of claims could be treated as a continuous variable, rather than discrete, if the values were large. We've seen this idea before when we used a normal approximation to a binomial or Poisson distribution in Subject CS1.

Similarly, colour of car could be recorded as red, orange, yellow, *etc* (*ie* nominal data) or (if it came from a video image) it could be measured on the RGB scale (*ie* a vector of three discrete numerical values). It's also quite common to record attribute data as 1's and 0's (*ie* with discrete numerical values) to make it easier to count subsets and to calculate proportions and averages.

Nowadays, however, there are many other types of data that don't directly correspond to these familiar types. For example, a motor insurer might be provided with a memory card containing footage of an accident recorded on a vehicle's dash cam (dashboard camera). This will typically be a very large file containing a mixture of video and audio information, as well as structural information (*eg* markers for the start and end of each frame of the video), header information (*eg* the date it was captured, the serial number of the camera and the software version) and other embedded information such as time markers for the footage and satellite coordinates.

Exploring and preparing data

Some forms of 'raw' data or complex data structures first need to be converted to one of the types in the table. For example, a Word .docx file, although actually just a long sequence of 1's and 0's, has a specific structure from which the page layouts, the fonts and the text itself would need to be removed before we could analyse the content. Similarly, we would need to prepare a raw audio or video file before we could use the data it contains for speech recognition or face recognition.

A data set must first be prepared in such a way that a computer is able to access it and apply a range of algorithms. If the data are already in a regular format, this may be a simple matter of importing the data into whatever computer package is being used to develop the algorithms. If the data are stored in complex file formats, it will be useful to convert the data to a format suitable for analysis. See Tidy Data (Wickham) for a more systematic approach to data preparation beyond the CS2 syllabus.

Important checks include:

- **All variables should be assigned to the correct type (nominal/categorical, ordinal, continuous).**
- **Missing values need to be identified and the mechanism for missingness understood. If appropriate, missing values can be imputed statistically using other information.**

  It may be possible to estimate or derive missing values from other available information.
- **Look for obvious errors in variables, and in combinations of variables.**

Exploratory data analysis **(EDA) should be used to check for errors and to understand high-level properties of the data. When performing extensive exploratory analysis, it is important to be careful of 'data dredging', which could result in mistaking chance relationships for signals. It is a good idea to state the relationships that would be expected between variables *before* carrying out exploratory data analysis.**

These first few steps are also covered in the data analysis chapter of Subject CS1.

Feature scaling

Some machine learning techniques will only work effectively if the variables are of similar scale. We can see this by recalling that, in a linear regression model (Subject CS1, Chapter 12) the parameter β_j, associated with variable x_j, measures the impact on y of a one-unit change in x_j. If x_j is measured in, say, metres, the value of β_j will be 100 times larger than it would be with the same data if x_j were measured in centimetres.

In Section 3.1, we consider expressions for regression penalties, such as ridge regression. The ridge penalty for model complexity has the form $\lambda\sum_{j=1}^{d}\beta_j^2$, which is only a meaningful expression if the parameters β_j are on the same scale. One common approach is to centre input variables, by subtracting the sample mean, and then to rescale each variable to have standard deviation 1.

Although this section is focused on supervised learning, the scaling of the variables is particularly important with the unsupervised K-means algorithm, which we will discuss later.

Splitting the data into training, validation and test sets

The training/validation/test approach is covered in Section 2.7.

A typical split as described in Section 2.7 would use 60% of the data for training, 20% for validation and 20% for testing. A guide might be to select enough data for the validation data set and the testing data set so that the validation and testing processes can function, and to allocate the rest of the data to the training data set. In practice, this often leads to around a 60%/20%/20% split.

Training the model

This involves fitting a suitable Machine Learning model to the training subset of the data. The model will typically have free parameters, which need to be estimated from the data. This stage is essentially the same as the process of fitting a model to data described in Subject CS1 chapters 12 and 13 on regression and generalised linear models.

Evaluation

Once the model has been trained on a set of data, its performance should be evaluated. How this is done may depend on the purpose of the analysis. If the aim is prediction, then one obvious approach is to test the model on a set of data different from the one used for development. If the aim is to identify hidden patterns in the data, other measures of performance may be needed.

2.9 Reproducibility of research

It is important that data analysis be *reproducible.* This means that someone else can take the same data, analyse it in the same way, and obtain the same results. In order that an analysis be reproducible, the following criteria are necessary:

- **The data used should be fully described and available to other researchers.**
- **Any modifications to the data (*eg* recoding or transformation of variables, or computation of new variables) should be clearly described, ideally with the computer code used. In Machine Learning this is often called *feature engineering*.**
- **The selection of the algorithm and the development of the model should be described, again with computer code being made available. This should include the parameters of the model and how and why they were chosen.**

There is an inherent problem with reproducing stochastic models, as they depend on random elements, which in turn depend on a random seed. Of course, details of the random number generator seeds chosen, and the precise command and package used to generate any randomness, could be presented. However, since stochastic models are typically run many times to produce a distribution of results, it normally suffices that the distribution of the results is reproducible.

To ensure reproducibility in stochastic models in R, use the same numerical seed in the function `set.seed()`**.**

3 Applications: supervised learning

In this section, we describe some commonly used supervised Machine Learning techniques. Some of these methods are direct extensions of methods covered elsewhere in the syllabus, notably in Subject CS1 chapters 12 and 13 (linear regression and generalised linear models) and Chapter 12 of this subject, to which reference should be made. A fuller account can be found in the book 'Introduction to Statistical Learning in R', given in the references.

3.1 Penalised generalised linear models

Suppose we have a dataset consisting of n observations of d input variables $\boldsymbol{x} = (\underline{x}_1, ..., \underline{x}_n)$, where each $\underline{x}_i = (x_{i1}, x_{i2}, ..., x_{id})$, and corresponding n observations of an output variable $y = (y_1, ..., y_n)$. To fit a generalised linear model, we normally specify the link function and a linear predictor depending on the parameters of the model corresponding to the covariates, $\beta_1, ..., \beta_d$, and the intercept parameter, β_0. We then maximise the log-likelihood $\ell(\beta_0, \beta_1, ..., \beta_d)$, or equivalently minimise the deviance. However, in the setting where d is a reasonable fraction of n, so that there are relatively few observations per parameter, the variances of the parameter estimators can be large, leading to large errors when predicting unseen outcomes.

This is the problem of overfitting, the estimate too closely resembles the specifics of the training data so large variation is observed between training samples.

One approach is to seek a model that uses fewer parameters by imposing a penalty on model complexity. Commonly used criteria include the *Akaike Information Criterion* (AIC) and the *Bayesian Information Criterion* (BIC):

$$\text{AIC} = \text{deviance} + 2d$$

$$\text{BIC} = \text{deviance} + d[\ln(n)]$$

Here d represents the number of parameters in the model and n is the number of observations.

The deviance here is a measure of the average error of the model's outputs and measures the goodness of fit. The extra terms reflect the number of parameters in the model. If we aim to minimise the AIC or the BIC, this will allow us to find a good trade-off between the two objectives of obtaining a good fit to the data and minimising the number of parameters in the model.

These criteria attempt to allow for the fact that more complex models will necessarily fit the training data better. Instead of seeking models with the lowest deviance, models with the lowest AIC or BIC are preferred. Automatic model selection methods such as *stepwise selection* can systematically explore the space of all possible models to find reasonable candidate models according to these criteria. See Chapter 13 of Subject CS1. However, this approach will not work well when the number of input variables is large: it is computationally expensive to explore a large model space.

The more input variables there are, the more possible models there are and so the larger the model space. The more models we want to fit and evaluate, the more computing power is required.

***Regularization* extends the idea of penalizing more complex models. Instead of maximising the log-likelihood function, we maximise a penalized log-likelihood:**

$$\ell(\beta_0,\beta_1,\ldots,\beta_d \mid \boldsymbol{x},\boldsymbol{y}) - \lambda g(\beta_0,\beta_1,\ldots,\beta_d)$$

where g represents a penalty function and the regularization parameter $\lambda \geq 0$ controls how strongly the penalty should be applied. As $\lambda \to 0$ we recover the unpenalized model.

By maximising this expression, we are aiming for a trade-off where we try to maximise the log-likelihood but, at the same time, try to minimise the penalty applied (since this is subtracted).

The two most common form of the penalty function are:

***Ridge regression*:** $g(\beta_0,\beta_1,\ldots,\beta_d) = \sum_{j=1}^{d} \beta_j^2$

***Lasso regression*:** $g(\beta_0,\beta_1,\ldots,\beta_d) = \sum_{j=1}^{d} |\beta_j|$

The names are based on the geometrical interpretations of these measures, which you are not expected to know.

In either case, for large values of λ, small absolute values of the parameters are encouraged – hence, these are commonly known as *shrinkage* methods. A good value of λ is typically chosen by cross-validation.

The parameter λ is an example of a hyperparameter.

Question

A random sample $x_1,\ldots,x_n$ of n values has been taken from a Poisson distribution with unknown mean μ. The value of μ is to be estimated using the penalty function $\lambda(\mu-5)^2$.

(i) Explain why this particular penalty function might have been chosen.

(ii) Write down an expression for the penalised log-likelihood function.

(iii) Show that $\hat{\mu}$, the estimated value of μ, satisfies the equation:

$$n(\hat{\mu}-\bar{x})+2\lambda\hat{\mu}(\hat{\mu}-5)=0$$

(iv) Calculate the value of $\hat{\mu}$ based on the sample of values 5.7, 5.4, 4.6, 5.0 and 4.9 when $\lambda = 0.2$.

(v) Use the equation in part (iii) to show algebraically what happens to the value of $\hat{\mu}$:

(a) when λ is equal to zero and

(b) when λ is very large.

(vi) Comment on your answers in part (v).

Solution

In this question we have just one parameter (*ie* $d=1$), which is called μ (corresponding to β_1).

(i) We might choose this penalty function if we believe the true value of μ is close to 5, as the penalty when $\mu=5$ would be zero.

(ii) The likelihood function for the sample is:

$$L=\prod_{i=1}^{n}\frac{e^{-\mu}\mu^{x_i}}{x_i!}=e^{-n\mu}\mu^{\sum x_i}\times\text{constant}=e^{-n\mu}\mu^{n\bar{x}}\times\text{constant}$$

So the log-likelihood is:

$$\log L=-n\mu+n\bar{x}\log\mu+\text{constant}$$

The penalised log-likelihood is:

$$(\log L)^*=-n\mu+n\bar{x}\log\mu+\text{constant}-\lambda(\mu-5)^2$$

(iii) To maximise this, we equate the derivative with respect to the parameter μ to zero:

$$\frac{\partial(\log L)^*}{\partial\mu}=-n+\frac{n\bar{x}}{\mu}-2\lambda(\mu-5)=0$$

Multiplying by μ and rearranging gives:

$$n(\mu-\bar{x})+2\lambda\mu(\mu-5)=0$$

So $\hat{\mu}$ satisfies the equation:

$$n(\hat{\mu}-\bar{x})+2\lambda\hat{\mu}(\hat{\mu}-5)=0$$

(iv) For this sample, we have:

$$n=5 \quad\text{and}\quad \bar{x}=\frac{1}{5}(5.7+5.4+4.6+5.0+4.9)=\frac{25.6}{5}=5.12$$

So with $\lambda=0.2$, the equation in part (iii) becomes:

$$5(\hat{\mu}-5.12)+2(0.2)\hat{\mu}(\hat{\mu}-5)=0$$

This can be rearranged to give:

$$0.4\hat{\mu}^2+3\hat{\mu}-25.6=0$$

Using the quadratic formula:

$$\hat{\mu} = \frac{-3 \pm \sqrt{3^2 - 4(0.4)(-25.6)}}{2(0.4)} = \frac{-3 \pm \sqrt{49.96}}{0.8} = -12.585 \text{ or } 5.085$$

Since μ must be positive, the required estimate is $\hat{\mu} = 5.085$.

(v)(a) If $\lambda = 0$, the equation in part (iii) becomes:

$$n(\hat{\mu} - \bar{x}) = 0 \quad \Rightarrow \hat{\mu} = \bar{x}$$

(v)(b) If we divide the equation in part (iii) by λ, we get:

$$\frac{n(\hat{\mu} - \bar{x})}{\lambda} + 2\hat{\mu}(\hat{\mu} - 5) = 0$$

As $\lambda \to \infty$, this becomes:

$$2\hat{\mu}(\hat{\mu} - 5) = 0 \quad \Rightarrow \hat{\mu} = 0 \text{ or } \hat{\mu} = 5$$

Since the value of μ must be strictly positive, the required estimate would be $\hat{\mu} = 5$.

(vi) If we apply no penalty, as in part (v)(a), the method reduces to maximum likelihood estimation and we get the usual estimate for μ, *ie* the sample mean of 5.12.

If we apply a high penalty to values that are not close to the target value of 5, as in part (v)(b), the method will produce a value close to 5, irrespective of the actual values in the sample.

As expected, the estimated value of 5.085 lies between these two values.

3.2 Naïve Bayes classification

The naïve Bayes model is used for classification problems, *ie* when the output is categorical.

Recall from Subject CS1 Chapter 14 that if $B_1, B_2, ..., B_R$ constitute a partition of a sample space S and $P(B_i) \neq 0$ for $i = 1, 2, ..., R$ then for any event A in S such that $P(A) \neq 0$:

$$P(B_r \mid A) = \frac{P(A \mid B_r)P(B_r)}{P(A)}$$

where

$$P(A) = \sum_{i=1}^{R} P(A \mid B_i)P(B_i)$$

for $r = 1, 2, ..., R$.

This is *Bayes' Theorem*, which allows us to 'invert' conditional probabilities, *ie* to work out the values of the probabilities $P(B_r \mid A)$ when we know the probabilities $P(A \mid B_r)$.

Question

Derive Bayes' Theorem.

Solution

The proof uses the definition of conditional probabilities, $P(X \mid Y) = \frac{P(X,Y)}{P(Y)}$, and the equivalent identity $P(X,Y) = P(X \mid Y)P(Y)$.

Using the definition of conditional probabilities, the probability we want to find is:

$$P(B_r \mid A) = \frac{P(B_r, A)}{P(A)}$$

Using the identity above, the numerator on the right-hand side can be written as:

$$P(B_r, A) = P(A \mid B_r)P(B_r)$$

If we condition the denominator on the different possible values of B_r, we can write it as:

$$P(A) = P(A \mid B_1)P(B_1) + P(A \mid B_2)P(B_2) + \cdots + P(A \mid B_R)P(B_R) = \sum_{r=1}^{R} P(A \mid B_r)P(B_r)$$

If we substitute these in, we then get Bayes' formula:

$$P(B_r \mid A) = \frac{P(A \mid B_r)P(B_r)}{P(A)} = \frac{P(A \mid B_r)P(B_r)}{\sum_{r=1}^{R} P(A \mid B_r)P(B_r)}$$

Naïve Bayes classification uses this formula to classify cases into mutually exclusive categories on some outcome y, on the basis of a set of covariates $x_1, \ldots, x_d$. The events A are equivalent to the covariates taking some set of values, and the partition $B_1, B_2, \ldots, B_R$ is the set of values that the outcome can take.

Suppose the outcome is whether a person will survive for ten years. Let $y_i = 1$ denote the outcome that person i survives, and $y_i = 0$ denote the outcome that person i dies. Then if we have d covariates, we can write the posterior probability that $y_i = 1$, given the covariate information as:

$$P(y_i = 1 \mid x_{i1}, \ldots, x_{id}) = \frac{P(x_{i1}, \ldots, x_{id} \mid y_i = 1)P(y_i = 1)}{P(x_{i1}, \ldots, x_{id})}$$

It is difficult to estimate the joint distribution of the variables $x_1, \ldots, x_d$, particularly in the high-dimensional setting.

In practice many of the combinations of values will not be present in the sample, so the corresponding probabilities, $P(x_{i1},...,x_{id})$, cannot be estimated. For example, for a motor insurer that uses several rating factors (*eg* age of policyholder, occupation, make of car, age of car, postcode area), many of the subsets will be empty.

The naïve Bayes algorithm assumes that the values of the covariates x_{ij}, $j=1,...,d$, are conditionally independent given the value of y_i.

So we are assuming that:

$$P(x_{i1},x_{i2},\dots,x_{id}\mid y_i=1)=P(x_{i1}\mid y_i=1)\times P(x_{i2}\mid y_i=1)\times\cdots\times P(x_{id}\mid y_i=1)$$

Question

Illustrate why this assumption might not be accurate for a motor insurer using the rating factors age of policyholder, occupation, make of car, age of car, and postcode area to build a model to predict whether or not a claim will be made.

Solution

As an example, say that out of the policyholders in the training data without a claim, 25% are under 25, 20% drive high performance cars and 40% drive cars less than 3 years old.

However, it is unlikely that 2% (*ie* $25\%\times20\%\times40\%$) of policyholders without a claim would be under 25 with high performance cars under 3 years old, as young drivers are unlikely to be able to afford such vehicles – or to be able to pay to insure them.

This allows the formula to be re-written:

$$P(y_i=1\mid x_{i1},...,x_{id})=\frac{P(x_{i1}\mid y_i=1)P(x_{i2}\mid y_i=1)...P(x_{id}\mid y_i=1)P(y_i=1)}{P(x_{i1},...,x_{id})}$$

So that:

$$P(y_i=1\mid x_{i1},...,x_{id})\propto P(y_i=1)\prod_{j=1}^{d}P(x_{ij}\mid y_i=1)$$

Assuming that the costs of misclassification are equal, we assign an observation to the class with the highest posterior probability.

For a given set of values of $x_{i1},...,x_{id}$, the denominator $P(x_{i1},...,x_{id})$ doesn't change. So, we can treat this as a constant in the calculations and just look at the relative values – hence the proportional sign. To find the actual values of the individual probabilities, we can just divide by the total, to produce a set of probabilities that add up to 1.

Question

A motor insurer has analysed a sample of 1,000 claims for three different geographical regions split by the size of claim (Small / Medium / Large). It has then classified them according to whether they proved to be fraudulent or genuine claims. The results are shown in the tables below:

FRAUDULENT	Region 1	Region 2	Region 3	Total
Small	3	0	6	9
Medium	10	5	20	35
Large	3	1	2	6
Total	16	6	28	50

GENUINE	Region 1	Region 2	Region 3	Total
Small	57	38	70	165
Medium	250	95	180	525
Large	176	46	38	260
Total	483	179	288	950

(i) Give a formula that could be used to estimate the probability that a new claim from Region 3 for a Medium amount will prove to be fraudulent.

(ii) Estimate the probability that each of the following types of new claim will be fraudulent:

(a) a claim from Region 3 for a Medium amount

(b) a claim from Region 1 for a Large amount

(c) a claim from Region 2 for a Small amount.

The insurer decides to classify all claims with a probability of being fraudulent greater than 5% as fraudulent (pending further investigation).

(iii) Determine which, if any, of the claim types in part (ii) would be investigated.

Solution

(i) Using Bayes' Theorem (and obvious abbreviations for the events), we have:

$$P(F \mid R3, M) = \frac{P(F, R3, M)}{P(R3, M)} = \frac{P(R3, M \mid F)P(F)}{P(R3, M \mid F)P(F) + P(R3, M \mid G)P(G)}$$

(ii)(a) From the tables above, we have:

$$P(R3, M \mid F) = \frac{20}{50} = 0.4, \quad P(R3, M \mid G) = \frac{180}{950},$$

$$P(F) = \frac{50}{50 + 950} = 0.05, \quad P(G) = \frac{950}{50 + 950} = 0.95$$

So: $$P(F \mid R3, M) = \frac{0.4 \times 0.05}{0.4 \times 0.05 + 180/950 \times 0.95} = 0.1$$

So the estimated probability that a claim from Region 3 for a Medium amount is fraudulent is 10%.

In fact, we can do this calculation directly from the table. For Region 3 and Medium amounts there were 20 fraudulent claims and 180 genuine claims. So the estimated probability that a claim from Region 3 for a Medium amount is fraudulent is $\frac{20}{20 + 180} = 0.1$, *ie* 10%.

(ii)(b) Similarly, the estimated probability that a claim from Region 1 for a Large amount is fraudulent is $\frac{3}{3 + 176} = 0.168$, *ie* 1.68%.

(ii)(c) Since there were no fraudulent claims from Region 2 for a Small amount in the sample, the estimated probability that a claim from Region 2 for a Small amount is fraudulent is 0.

(iii) The insurer would investigate claims for a Medium amount from Region 3 (from (ii)(a)) but neither of the other two claim types.

The Core Reading stated earlier that, for equal misclassification costs, we allocate points to the category with the highest posterior probability for classification. This is unlikely to be the case here as misclassifying a fraudulent claim as genuine is likely more costly than misclassifying a genuine claim as fraudulent.

In addition, if the approach is taken to only classify claims as fraudulent if the posterior probability is greater than 0.5, then no claims would ever be classified as fraudulent based on this model. This is due to the low overall prevalence of fraud in the three regions. However, a more complex model that takes into account more attributes of the claims, as opposed to just region, may lead to higher possible posterior probabilities of fraud.

3.3 Decision trees

A Classification and Regression Tree (CART) is a simple, flexible machine learning method that can be used for classification or regression. The model takes the form of a simple, binary decision tree as shown in Figure 21.4.

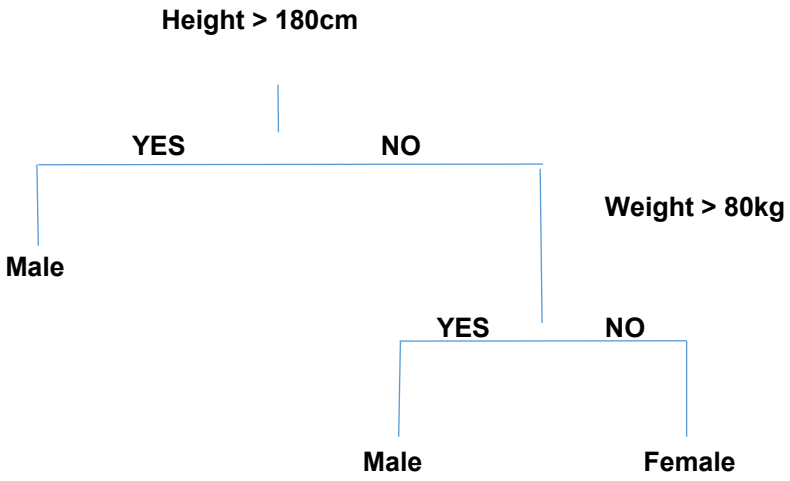


Figure 21.4

This figure shows a classification tree for predicting gender based on height and weight. Given the height and weight of a new individual, it is easy to predict the individual's gender. To do so, start at the top of the tree and, at each level of the tree, choose the appropriate path. If the individual's height is greater than 180 cm, predict male. Otherwise, consider their weight: if it is greater than 80kg, predict male; if not, predict female.

The rationale here may be that tall people (Height > 180cm) tend to be male, so we can separate them out at the first stage. Of the remaining people, males tend to be heavier than females, so we can split these at a level that is likely to a reasonable boundary between males and females (80kg, say).

We now consider how this decision tree may have been constructed (learned) using data.

Learning a decision tree

A tree is built by recursively partitioning the *d*-dimensional space of inputs along the coordinate axes, as shown in Figure 21.5:

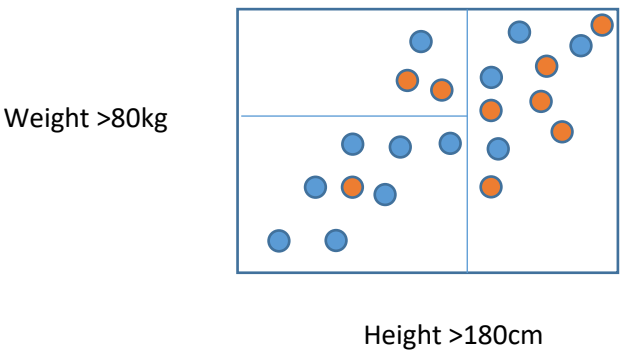


Figure 21.5

At each stage, the best partition is chosen, to minimise a loss function appropriate to the problem.

In order to build a decision tree, we need to choose partitions at each stage, *ie* how to divide up the input space. To do this we need to choose the input variable to use, here height or weight, and the value at which to split the data.

In this algorithm, at each stage we choose the split that appears to be the most effective at separating the remaining elements, without thinking ahead to the consequences that this might have on the later splits. So, we just 'bite off as much as we can' at each stage. The effectiveness of splits is measured by a loss function, which is discussed later.

The partition of the sample space corresponds to the tree in Figure 21.4, with each of the three sub-rectangles corresponding to a terminal node in the tree.

Figure 21.5 shows how the example decision tree in Figure 21.4 partitions the input space. First the space is divided into two rectangles at the point where height is 180cm. The rectangle corresponding to height of no more than 180cm is then further split into two rectangles based on weight.

An unseen observation would then be placed in the most common class within its sub-rectangle.

For a classification tree, we assign a prediction category to each of the terminal nodes (sub-rectangles in the input space). One way to do this is to predict the category associated with the most individuals from the training data in that node. This is then the predicted category for new, unseen, data points if they lie within that sub-rectangle of the input space.

For a problem with asymmetric misclassification costs, a different threshold could be applied. For example, if the classification instead represents the positive or negative outcome of the test for a serious disease, it would be prudent to declare an observation as positive if the proportion of training observations in the sub-rectangle crosses some low threshold.

In this example, incorrectly classifying a sick individual as healthy has a much higher cost than incorrectly classifying a healthy individual as sick. These asymmetric misclassification costs mean that it may be better to predict 'sick' for terminal nodes with a certain threshold of sick individuals, rather than requiring the majority of individuals in that node to be sick.

Loss functions

For a classification problem with K classes (here, for binary classification $K = 2$), one common loss function is the *Gini index*. This is a measure of how 'pure' the leaf nodes are (*ie* how mixed the training data assigned to each node is). For a tree with external nodes $j = 1,\ldots,J$ this is:

$$\sum_{j=1}^{J} n_j \sum_{k=1}^{K} p_{jk}(1 - p_{jk})$$

where p_{jk} is the proportion of training observations under node j of type k and n_j is the number of observations under node j.

This formula can also be written as:

$$\sum_{j=1}^{J} n_j \sum_{k=1}^{K} p_{jk}(1-p_{jk}) = \sum_{j=1}^{J} n_j \sum_{k=1}^{K} \left(p_{jk} - p_{jk}^2 \right)$$

$$= \sum_{j=1}^{J} n_j \left[\sum_{k=1}^{K} p_{jk} - \sum_{k=1}^{K} p_{jk}^2 \right]$$

$$= \sum_{j=1}^{J} n_j \left[1 - \sum_{k=1}^{K} p_{jk}^2 \right]$$

The measure of purity for the jth external (terminal) node (or resulting sub-rectangle) is $\sum_{k=1}^{K} p_{jk}(1-p_{jk})$ or $1-\sum_{k=1}^{K} p_{jk}^2$. The overall Gini index is then a weighted sum of these values over all external nodes. The weights are the number of data points in each node, the n_j.

If all the training observations under node j are of the same type (perfect class purity), then:

$$\sum_{k=1}^{K} p_{jk}(1-p_{jk}) = 0$$

This formula gives the probability that two items are selected at random (with replacement) are of different types (*ie* if the first item is of type k, the second item will not be of type k). So, if all the items are of the same type, the probability will be 0. If all observations are of the same type, say k', then $p_{jk'} = 1$ and $p_{jk} = 0$ for all other types, k.

For a node that has an equal split of classes for a binary classification problem (worst purity), this quantity takes its maximum value of 0.5.

For a classification problem where the data points are divided into m distinct categories, the Gini index must take a value between 0 and $1-\frac{1}{m}$. As $m \to \infty$, the upper limit of the Gini index tends to 1.

For a regression problem, the predicted output value of a new observation under node j, $\hat{y}_j$, is the mean output value of all training observations under node j. The corresponding loss function is the squared error loss. If the jth node contains n_j observations, the squared error is:

$$\sum_{j=1}^{J} \sum_{i=1}^{n_j} (y_i - \hat{y}_j)^2$$

For each node, the predicted value that minimises the sum of squared errors for that node is the mean output value for the training observations in that node. To see this, we have that the squared error for node j, SE_j, given some predicted value for this node, $\hat{y}_j$, is:

$$SE_j = \sum_{i=1}^{n_j} (y_i - \hat{y}_j)^2$$

Taking the derivative with respect to $\hat{y}_j$ we get:

$$\frac{dSE_j}{d\hat{y}_j} = -2\sum_{i=1}^{n_j} (y_i - \hat{y}_j)$$

Setting this to 0 and rearranging, we have:

$$-2\sum_{i=1}^{n_j} (y_i - \hat{y}_j) = 0$$

$$\Leftrightarrow n_j \hat{y}_j = \sum_{i=1}^{n_j} y_i$$

$$\Leftrightarrow \hat{y}_j = \frac{\sum_{i=1}^{n_j} y_i}{n_j} = \overline{y}_j$$

The second derivative is:

$$\frac{d^2 SE_j}{d\hat{y}_j^{\,2}} = 2n_j > 0$$

So, $\hat{y}_j = \overline{y}_j$ minimises the residual sum of squares for each node.

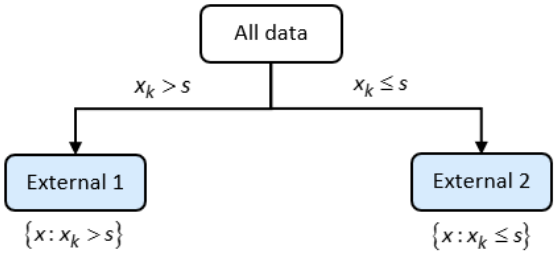
The tree-building procedure is top-down: initially, all training observations belong to a single region. The partitioning procedure then chooses the input variable x_k and the threshold value s such that dividing up the space into the two regions $\{x : x_k > s\}$ and $\{x : x_k \leq s\}$ gives the largest possible reduction in the loss function. Subsequent steps repeat this procedure to determine the best possible partition of an existing region. Note that this is a *greedy algorithm*: it finds the best partition *at each stage* – this approach may not find the decision tree with the lowest possible value of the loss function.

At the start of the procedure for a classification problem, when all observations belong to the same region, the value of the Gini impurity score is:

$$n\sum_{k=1}^{K} p_k(1 - p_k)$$

Here p_k is the proportion of training observations of type k and n is the total number of data points.

When deciding on the first partition, we need to choose the input variable and the threshold. Say we are evaluating the partition using some variable x_k and some threshold s. This partition creates two 'child' nodes from the parent node that contains all the data. After this split, these two child nodes are now the external (terminal) nodes of the tree, which looks something like the following:



The Gini impurity score for the tree after this split is:

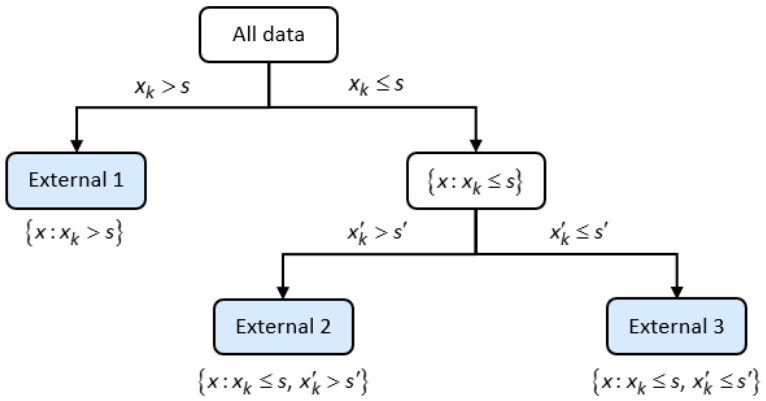
$$\sum_{j=1}^{2} n_j \sum_{k=1}^{K} p_{jk}(1-p_{jk})$$

where p_{jk} and n_j are as defined previously. The reduction in the loss function is then given by:

$$n\sum_{k=1}^{K} p_k(1-p_k) - \sum_{j=1}^{2} n_j \sum_{k=1}^{K} p_{jk}(1-p_{jk})$$

We want to identify the partition that maximises the above reduction in the loss function. However, the first part of the expression does not change for different partitions. So, this is equivalent to identifying the partition that minimises the score for the tree after the split, *ie* that minimises $\sum_{j=1}^{2} n_j \sum_{k=1}^{K} p_k(1-p_k)$.

Say that using variable x_k and threshold s maximises the reduction in the loss function for the first partition. Consider, for example, further splitting the data for which $x_k \leq s$. When we split this data using some variable x'_k and some threshold s' the tree looks something like the following:



The Gini impurity score for the tree after this split is now:

$$\sum_{j=1}^{3} n'_j \sum_{k=1}^{K} p'_{jk}(1-p'_{jk})$$

where p'_{jk} and n'_j are as defined previously but for the revised tree. The reduction in the loss function is then given by:

$$\sum_{j=1}^{2} n_j \sum_{k=1}^{K} p_{jk}(1-p_{jk}) - \sum_{j=1}^{3} n'_j \sum_{k=1}^{K} p'_{jk}(1-p'_{jk})$$

Once again, we want to identify the partition that maximises the above reduction in the loss function. However, as before, the first part of the expression does not change for different partitions on the data for which $x_k \leq s$. So, this is equivalent to obtaining the partition that minimises the score for the tree after the split, *ie* that minimises $\sum_{j=1}^{3} n'_j \sum_{k=1}^{K} p'_{jk}(1-p'_{jk})$.

Also, the impurity score for the first external node does not change when considering these different partitions, *ie* $n'_1 = n_1$ and $p'_{1k} = p_{1k}$ for all k. So, this is equivalent to minimising the following (where the sum starts from 2 instead of 1 as we are ignoring the first external node):

$$\sum_{j=2}^{3} n'_j \sum_{k=1}^{K} p'_{jk}(1-p'_{jk})$$

This can be generalised to the result stated below.

Greedy splitting in classification problems using the Gini impurity score

For any given split point, the partition that maximises the reduction in the loss function also minimises the weighted sum of the Gini impurity scores for the child nodes resulting from the split. In other words, it minimises:

$$\sum_{c=1}^{2} n_c \sum_{k=1}^{K} p_{ck}(1-p_{ck})$$

where $c=\{1,2\}$ indexes the two child nodes of the split, p_{ck} is the proportion of training observations under node c of type k and n_c is the number of data points in the cth node.

It is generally easier to calculate $\sum_{c=1}^{2} n_c \sum_{k=1}^{K} p_{ck}(1-p_{ck})$ instead of the full reduction in the loss function.

A similar result can be stated for regression problems:

Greedy splitting in regression problems using the sum of squared errors

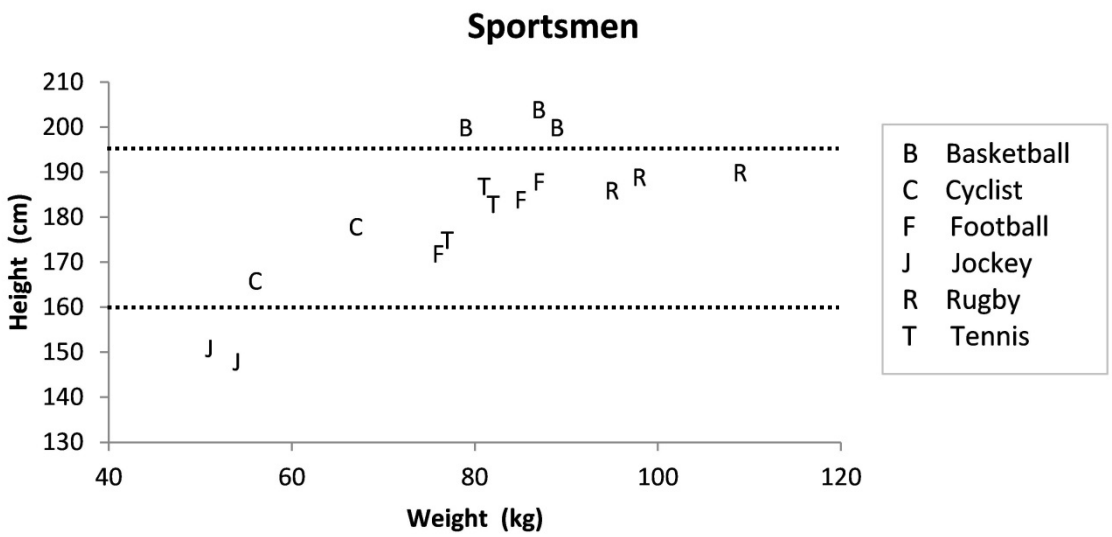
For any given split point, the partition that maximises the reduction in the loss function also minimises the sum of squared errors for the child nodes resulting from the split. In other words, it minimises:

$$\sum_{c=1}^{2} \sum_{i=1}^{n_c} (y_i - \hat{y}_c)^2$$

where $c=\{1,2\}$ indexes the two child nodes of the split, y_i is the observed value of the i th data point in the c th child node, $\hat{y}_c$ is the average value of the data points in the c th child node and n_c is the total number of data points in the c th child node.

Question

The graph below shows the heights and weights of the 16 sportsmen.



(i) Calculate the Gini index after each of the following proposed splits for the first split of a decision tree for the sportsmen (these heights are shown with the dotted lines in the graph above):

(a) Height > 195cm

(b) Height < 160cm.

(ii) Explain which of the two proposed splits is preferred when using the greedy approach with the Gini index.

Solution

(i)(a) ***Height > 195cm***

Let the first child node of the proposed split contain those sportsmen that are taller than 195cm. This node only contains basketball players (*ie* it is perfectly pure). The Gini impurity score for this child node can be calculated as follows:

$$G_1 = 1 - \left(p_{1B}^2 + p_{1C}^2 + p_{1F}^2 + p_{1J}^2 + p_{1R}^2 + p_{1T}^2 \right)$$

$$= 1 - \left[\left(\frac{3}{3}\right)^2 + \left(\frac{0}{3}\right)^2 + \left(\frac{0}{3}\right)^2 + \left(\frac{0}{3}\right)^2 + \left(\frac{0}{3}\right)^2 + \left(\frac{0}{3}\right)^2 \right]$$

$$= 0$$

Here, p_{1B} is the proportion of basketball players in the first child node, p_{1C} the proportion of cyclists *etc*. As expected, the score for this node is 0, as it only contains one type of athlete.

The second child node of the proposed split contains all those sportsmen that are shorter than 195cm. The Gini impurity score for this child node is:

$$\begin{aligned} G_2 &= 1-\left(p_{2B}^2+p_{2C}^2+p_{2F}^2+p_{2J}^2+p_{2R}^2+p_{2T}^2\right) \\ &= 1-\left[\left(\tfrac{0}{13}\right)^2+\left(\tfrac{2}{13}\right)^2+\left(\tfrac{3}{13}\right)^2+\left(\tfrac{2}{13}\right)^2+\left(\tfrac{3}{13}\right)^2+\left(\tfrac{3}{13}\right)^2\right] \\ &= \frac{134}{169}=0.793 \end{aligned}$$

The overall Gini index after this proposed split is a weighted sum of G_1 and G_2. The weights are the number of data points in each of the child nodes. So:

$$\begin{aligned} G &= 3G_1+13G_2 \\ &= 3\times 0+13\times\frac{134}{169}=\frac{134}{13}=10.307 \end{aligned}$$

(i)(a) ***Height < 160cm***

Let the first child node of the proposed split contain those sportsmen that are shorter than 160cm. This node only contains jockeys (*ie* it is perfectly pure). The Gini impurity score for this child node can be calculated as follows:

$$\begin{aligned} G_1 &= 1-\left(p_{1B}^2+p_{1C}^2+p_{1F}^2+p_{1J}^2+p_{1R}^2+p_{1T}^2\right) \\ &= 1-\left[\left(\tfrac{0}{2}\right)^2+\left(\tfrac{0}{2}\right)^2+\left(\tfrac{0}{2}\right)^2+\left(\tfrac{2}{2}\right)^2+\left(\tfrac{0}{2}\right)^2+\left(\tfrac{0}{2}\right)^2\right] \\ &= 0 \end{aligned}$$

Here, p_{1B} is the proportion of basketball players in the first child node, p_{1C} the proportion of cyclists *etc*. As expected, the score for this node is 0, as it is pure.

The second child node of the proposed split contains all those sportsmen that are taller than 160cm. The Gini impurity score for this child node is:

$$\begin{aligned} G_2 &= 1-\left(p_{2B}^2+p_{2C}^2+p_{2F}^2+p_{2J}^2+p_{2R}^2+p_{2T}^2\right) \\ &= 1-\left[\left(\tfrac{3}{14}\right)^2+\left(\tfrac{2}{14}\right)^2+\left(\tfrac{3}{14}\right)^2+\left(\tfrac{0}{14}\right)^2+\left(\tfrac{3}{14}\right)^2+\left(\tfrac{3}{14}\right)^2\right] \\ &= \frac{39}{49}=0.796 \end{aligned}$$

The overall Gini index after this proposed split is a weighted sum of G_1 and G_2, where the weights are based on the number of data points in each of the child nodes:

$$G = 2G_1 + 14G_2$$

$$= 2 \times 0 + 14 \times \frac{39}{49} = \frac{78}{7} = 11.143$$

(ii) ***Preferred split***

When using the greedy approach, we want to maximise the reduction in the Gini impurity score. This is the same as minimising the Gini impurity score after the split. The first split has a lower score of 10.307 compared to 11.143 for the second split. So, the first split is preferred.

To calculate the actual reduction in the loss function we can first calculate the Gini impurity score for the data before any splits:

$$G_0 = 16 \times \left[1 - \left(p_B^2 + p_C^2 + p_F^2 + p_J^2 + p_R^2 + p_T^2 \right) \right]$$

$$= 16 \times \left\{ 1 - \left(\left(\tfrac{3}{16}\right)^2 + \left(\tfrac{2}{16}\right)^2 + \left(\tfrac{3}{16}\right)^2 + \left(\tfrac{2}{16}\right)^2 + \left(\tfrac{3}{16}\right)^2 + \left(\tfrac{3}{16}\right)^2 \right) \right\}$$

$$= \frac{53}{4} = 13.25$$

The reduction in the loss function for the first split is:

$$13.25 - 10.307 = 2.942$$

For the second it is:

$$13.25 - 11.143 = 2.107$$

So, the first split has a larger reduction in the loss function.

Stopping and pruning

The recursive partitioning procedure stops according to a pre-specified criterion, *eg* when no region contains more than 5 observations. However, stopping criteria of this form tend to produce very deep, complex trees. This is analogous to a linear regression model using many predictor variables and is similarly prone to overfitting. It is often better to have a tree with fewer splits – these will typically have much smaller variance, at the expense of introducing a small amount of bias (for a lower out-of-sample MSE overall). One way to do this is to grow a deep tree and then prune it back, using *cross-validation* to determine optimal tree depth. See Chapter 8.1 of Introduction to Statistical Learning in R for a more detailed account that goes beyond the CS2 syllabus.

Bagged decision trees

A decision tree is typically a *high variance* model: slightly different training data can lead to a very different decision tree. It can be made much more robust by using *bootstrap aggregation* (bagging). The idea is to make a large number of decision trees, B, using the procedure above, each on a slightly perturbed version of the training data. Predictions for an unseen observation are then made by aggregating the B individual predictions: for a regression problem, we would take the average; for classifications, we would choose the most frequently predicted category.

For each unseen observation we have B predictions, one from each of the trees. For regression problems we average each of these for an overall prediction. For classification problems, we predict the category with the most occurrences amongst the individual predictions.

The process of perturbing the training dataset is *bootstrapping*: sampling with replacement from the original data.

The idea of bootstrapping is introduced in Chapter 8 of Subject CS1.

For example, if the training dataset were $\{1,2,3,4,5\}$, three bootstrap samples might be:

$$\{1,2,3,4,4\},\ \{1,2,2,3,3\},\ \{1,1,2,2,2\}$$

While sampling with replacement induces slight biases into individual bootstrap samples, as some observations are over- or under-represented, across the collection of B bootstraps samples, sampling should be representative.

Averaging the predictions from B decisions trees built on bootstrap samples typically leads to much better predictions than would be obtained from a single decision tree in isolation. In particular, bagging reduces the variance of the predictions.

One nice feature of this approach is that it naturally suggests a way to estimate the performance of the method on unseen data. A typical bootstrap sample will contain around two thirds $\left(\approx 1-\frac{1}{e}\right)$ of the original observations. To see this, note that since we sample with replacement, the number of times an individual observation occurs in any given bootstrap sample is binomially distributed on n trials, with success probability $\frac{1}{n}$. This distribution is roughly Poisson with mean 1. It follows that a particular observation is excluded from a bootstrap sample (zero successes) with probability close to $\frac{1}{e}$. These excluded observations are said to be out-of-bag, and the average out-of-bag error can be used as an estimate of the error on unseen data.

For example, if the bootstrap sample from the set of observations $\{1,2,3,4,5\}$ was $\{1,2,2,3,3\}$, then this sample doesn't contain the original observations 4 and 5. These are the out-of-bag observations for this sample. We can evaluate the performance of the tree fitted on the sample $\{1,2,2,3,3\}$ by using observations 4 and 5. This can be repeated for each of the trees built using the different bootstrapped samples to give an overall estimate of the performance on unseen data.

Random forest

This is an improvement to the bagged decision tree method, which aims to reduce the correlations between the predictions from different trees. As in the previous section, we build B different trees, each on a bootstrapped sample. However, at each stage in the building of a decision tree, we now consider only a random sample of the d different input variables. One common choice for classification problems is to take $\approx\sqrt{d}$ of the input variables at each stage and for regression problems to choose $\approx\frac{d}{3}$ input variables.

At each stage of building a decision tree, we need to consider how to partition the input space, *ie* which variable to use and what the split point should be. In a random forest, we only consider a subset of the available input variables at each point. This subset of 'candidate' variables is selected at random for each split point.

While it seems counterintuitive to disregard information, this approach has good statistical motivation. In a setting with a single strong relevant input variable, and a number of weaker variables, in each bootstrapped sample, the strong predictor would always determine the first split.

At each stage in a greedy algorithm, we choose the split that appears to be the most effective at separating the remaining elements. If there was one very strong predictor, then this is likely to be chosen as the first split in every bootstrapped sample.

This means that predictions from different trees would be *highly correlated*, leading to a smaller variance reduction for a given number of bootstrap samples. Selecting the best split from a randomly chosen subset of the predictor variables leads to trees that are uncorrelated and maximises the variance reduction from bagging.

Equivalently, sampling input variables compensates for the greedy nature of the tree-building algorithm, and so allows for better exploration of the predictor space.

By only considering a subset of variables at each split point, we should get a wider variety of partitions across the trees that may have better overall performance than trees constructed using a greedy approach on the entire input parameter set.

Example: Fitting Decision Tree and Random Forest models in R

To illustrate bagged decision trees and Random Forest, we work with the Boston housing dataset from the MASS package in R. The dataset aims to predict median house prices using background information.

```
library(MASS)

library(tree)

library(randomForest)
```

Split the data into a subset used for training, and a withheld test subset.

```
set.seed(1)

train <- sample(1:nrow(Boston), nrow(Boston)/2)
```

This creates a set of row numbers indicating the data points to be used as the training data.

First, we make a decision tree using all input variables

```
boston_tree <- tree(medv~., data = Boston[train,])
```

The first argument in the tree function is a formula object, similar to the input required in the `lm()`, `glm()` or `survfit()` functions throughout Subject CS1 and Subject CS2. The aim is to predict median house prices, which are in the column `medv` in the `Boston` data set. The code `medv~.` indicates that we are predicting `medv` using all available input variables. The full stop is shorthand for including all the other columns of the data set. The data argument is set to the Boston data set filtered to only include those data points in the training data set.

This code constructs a single decision tree only.

Then evaluate the predictions of the model on unseen data

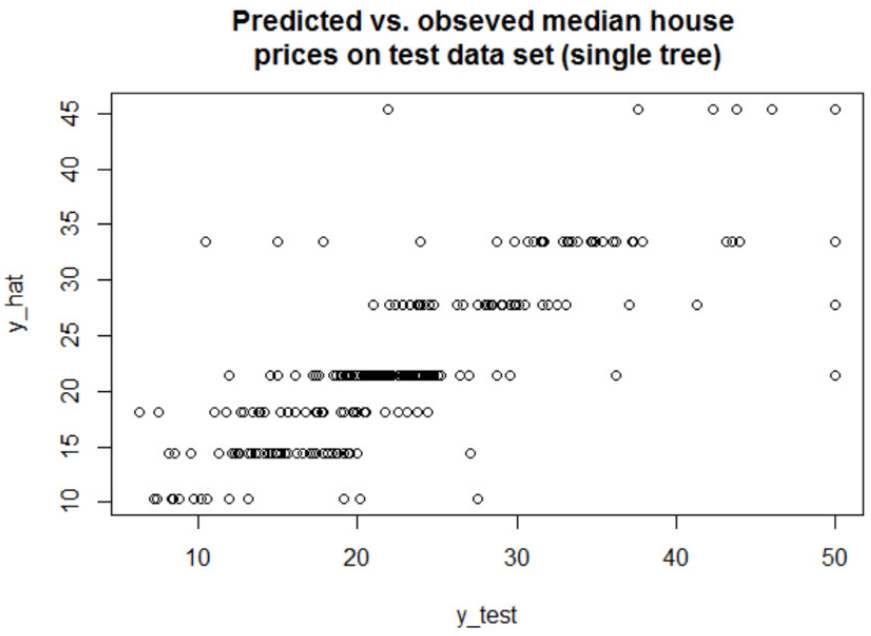
```
y_test <- Boston[-train, "medv"]
```

Here we select all the median house prices for the data points in the test data set, *ie* we select those not in the training rows by using negative indexing.

```
y_hat <- predict(boston_tree, newdata = Boston[-train,])
```

The predict function is used to predict the median house prices for all the data points in the test data set, based on the values of the other columns. We can then plot a graph of predicted vs. observed values in the test data set:

```
plot(y_test, y_hat, main = "Predicted vs. obseved median house
prices on test data set (single tree)")
```



We can then calculate the mean squared error of the test set, which is representative of the average squared distance between observed and predicted values for unseen observations.

```
mean((y_test-y_hat)^2)
```

```
[1] 35.28688
```

When plotting, note the common predicted value for all observations in the same region.

For a single decision tree, values that fall within the same terminal node (sub-rectangle of the input space) have the same predicted value. So, this is why there are many points with the same value of `y_hat`.

Now we make predictions using (by default) 500 bootstrapped trees. The number of trees used can be adjusted using the `ntree` argument. **To see the effect of bagging in isolation, we use the `randomForest` command and specify `mtry=13` to include all 13 input variables.**

Recall that bagging is when we average predictions over a number of different decision trees. Setting `mtry = 13` means that all 13 input variables are considered at each split point. When we introduced random forests in Section 0, we discussed considering subsets of the input variables at each split point. Here we first use all the input variables as candidates at each split to investigate the impact of bagging along.

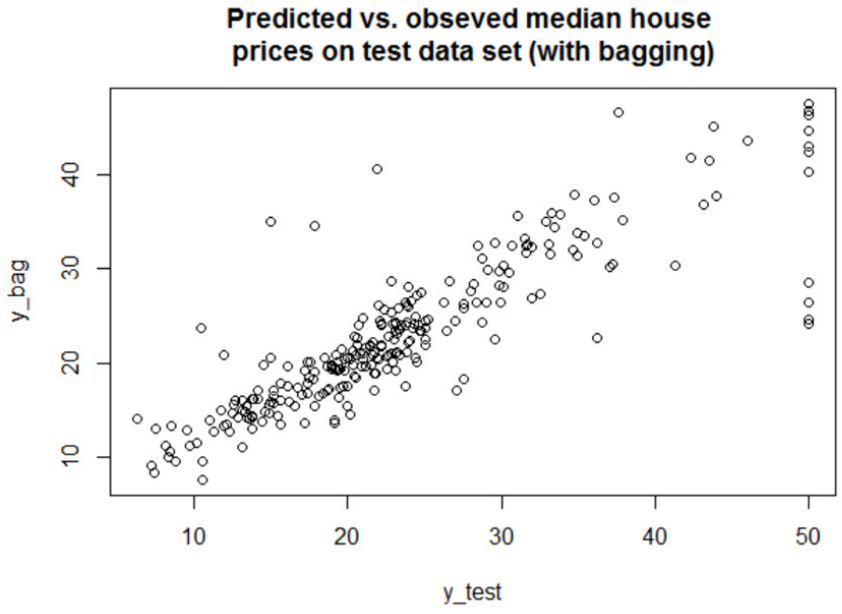
```
boston_bag <- randomForest(medv~., data = Boston[train,],mtry = 13)
```

We can then calculate predictions for each data point in the testing data:

```
y_bag <- predict(boston_bag, newdata = Boston[-train,])
```

We can then plot the predictions against the observations of the testing data set:

```
plot(y_test,y_bag, main = "Predicted vs. obseved median house
prices on test data set (with bagging)")
```



Comparing this graph to the predictions from a single tree, we can see that there is much better correspondence between predicted values and observations when averaging over multiple predictions.

We can also once again consider the mean squared error:

```
mean((y_test - y_bag)^2)
```

```
[1] 23.33324
```

The mean square error on unseen observations is much lower than when using a single decision tree.

Now by omitting `mtry` the algorithm's default choice (for a regression problem) is $\frac{d}{3}$, which leads to a further improvement in MSE.

Now only 4 variables (chosen at random) are considered at each split point when constructing each tree.

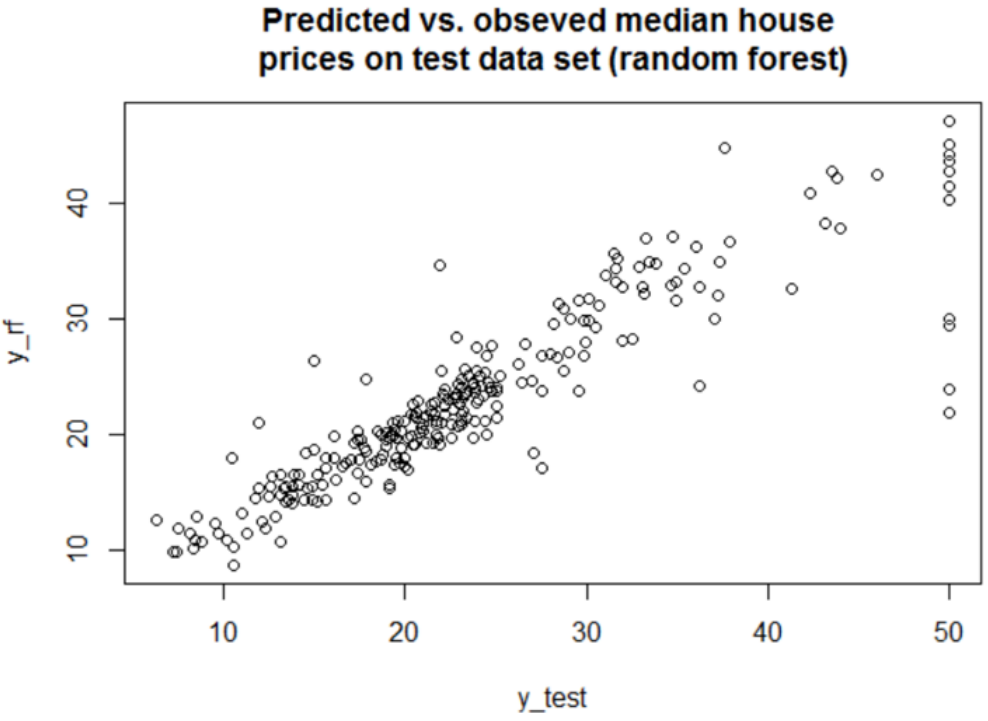
```
boston_rf <- randomForest(medv~., data = Boston[train,])
```

Once again we can calculate predictions for each data point in the testing data:

```
y_rf <- predict(boston_rf, newdata = Boston[-train,])
```

Again, we can then plot the predictions against the observations of the testing data set:

```
plot(y_test,y_rf, main = "Predicted vs. obseved median house
prices on test data set (random forest)")
```



This graph looks similar to the previous plot of bagged predictions. However, it looks even less spread out around the line $y = x$, suggesting that there has been an even further improvement when taking this approach.

Once again calculating the mean squared error:

```
      mean((y_test-y_rf)^2)
[1] 18.32787
```

This has further reduced the MSE.

4 Unsupervised learning

We saw previously that supervised learning methods seek the patterns in a collection of variables (the inputs) that best predict another variable (the output), *eg* we seek the combinations of salary and debt that are most associated with higher risk of mortgage default. In contrast, unsupervised learning methods determine the most dominant patterns within a collection of variables, without reference to any external target.

With *unsupervised learning*, there is no output variable to aim at, we are instead trying to identify patterns solely in the input variables, $x_1, \ldots, x_d$.

Unsupervised learning methods are used to identify major components of variability, *eg*:

- **Are observations clustered into distinct sub-groups, where observations within a group are more similar to each other than to other observations in the sample?**

  Trying to identify distinct sub-groups within a data set is known as clustering. An example clustering method is the K -means algorithm discussed in the next section.

- **Are there strong relationships among the variables, so that the entire dataset can be approximately described by a much smaller number of variables?**

  An example method is principal components analysis, which is covered in Subject CS1 Chapter 11.

Practical examples of unsupervised learning include market basket analysis, which identifies items commonly bought together, and market segmentation analysis, which identifies clusters of individuals with similar behaviour or interests.

Market basket analysis can be used by online retailers as the basis for the 'Other customers also bought …' recommendations or for promoting bundles of items that are frequently bought together.

As an example of market segmentation analysis, a business may be interested in segmenting its client base into different customer types based on demographics and past purchasing behaviour. There are no customer types specified in advance for the algorithm; instead the algorithm groups together similar customers based on the input data. The business then has to interpret the output groups and, for example, may update its marketing strategy to target the groups more effectively based on their characteristics.

Another example of a clustering problem is a system that groups together similar text documents based on, for example, the frequency of words within the documents.

5 Applications: unsupervised learning

5.1 *K*-means clustering

Suppose we have a set of data consisting of several variables (or features) measured for a group of individuals. These might relate to demographic characteristics, such as age, occupation, gender. Alternatively, they might relate to life insurance policies for which we have information such as sales channel, policy size, postcode, level of underwriting, *etc*.

We might ask whether we can identify groups (clusters) of policies which have similar characteristics. We may not know in advance what these clusters are likely to be, or even how many there are in our data.

There are a range of clustering algorithms available, but many are based on the *K-means algorithm*. This is an iterative algorithm, which starts with an initial division of the data into K clusters and adjusts that division in a series of steps designed to increase the homogeneity within each cluster and to increase the heterogeneity between clusters.

The K-means algorithm proceeds as follows. Let us suppose we have data on J variables.

1. **Choose a number of clusters, K, into which the data are to be divided. This could be done on the basis of prior knowledge of the problem. Alternatively, the algorithm could be run several times with different numbers of clusters to see which produces the most satisfactory and interpretable result. There are various measures of within- and between-group heterogeneity, often based on within-groups sums of squares. Comparing within-groups sums of squares for different numbers of clusters might identify a value of K beyond which no great increase in within-group homogeneity is obtained.**

2. **Initialise K cluster centres in the J-dimensional space occupied by the data. Specific problems may suggest an initial choice of cluster centres. More commonly, each case is initially assigned to a cluster at random, and the initial centres determined according to step 4 below.**

3. **Assign cases to the cluster centre that is nearest to them, using some measure of distance. One common measure is Euclidean distance:**

$$\text{dist}(x,k)=\sqrt{\sum_{j=1}^{J}(x_j-k_j)^2}$$

Here x_j is the standardised value of variable j for case x, and k_j is the value of variable j at the centre of cluster k ($k=1,\ldots,K$). Note that it is often necessary to standardise the data before calculating any distance measure, as in Section 2.8.

This is because the measure of distance is based purely on the numerical values of the variables. If some of the variables take large values because of the units of measurement that have been adopted, these variables will dominate the calculations and the other variables will effectively be ignored.

4. **Calculate the centroid of each cluster, using the mean values of the data points assigned to that cluster. This centroid becomes the new centre of each cluster.**

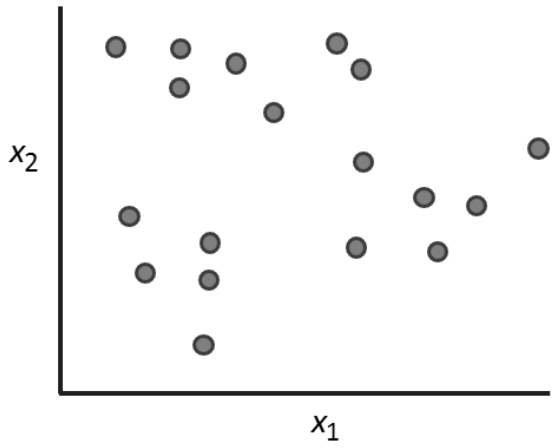
   The centroid is the centre of gravity of the data points when each has the same weight. To calculate it, we find the average of each 'coordinate' in the data set.

5. **Re-assign cases to the nearest cluster centre using the new cluster centres.**

6. **Iterate steps 4 and 5 until no re-assignment of cases takes place.**

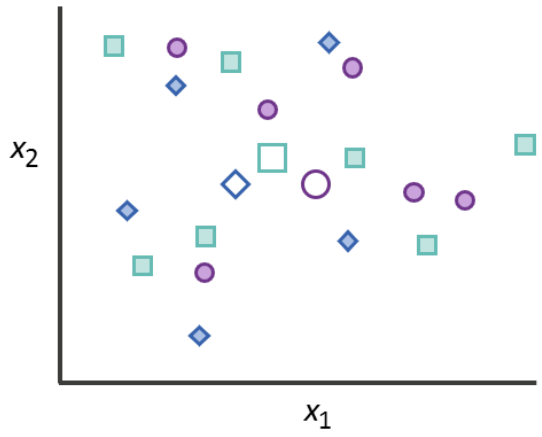
Example

The graph below shows the values of two variables, x_1 and x_2, for 18 data points (after appropriate scaling).



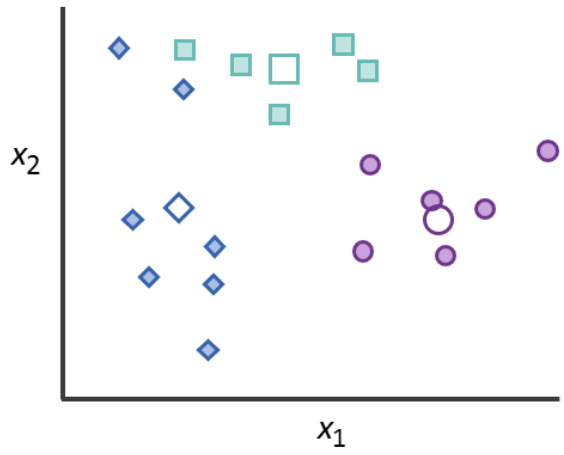
We can use the K-means algorithm with $k=3$ to try and identify any natural clusters in the data. We start by allocating each of the data points to one of the 3 clusters at random.

We've used squares, circles and diamonds to distinguish the three groups. We then need to calculate the position of the centroid of each cluster by averaging the x and y coordinates of the data points within each cluster. The centroids are indicated by the larger hollow shapes.

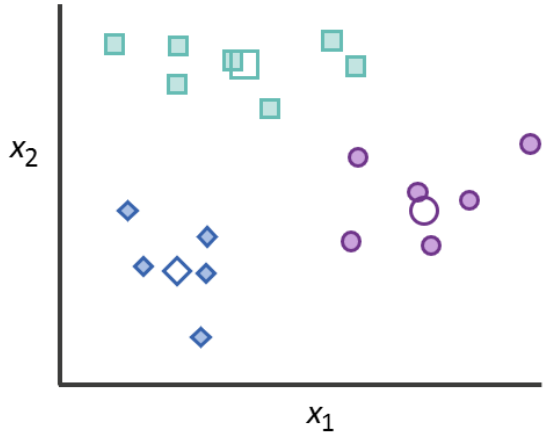


The three centroids happen to be all quite close together at this stage.

We now reallocate each data point to the cluster whose centroid it is nearest to, and then recalculate the positions of the centroids of the new groups.



Again, we reallocate each data point to the cluster whose centre it is nearest to, and then recalculate the centres:



At this point, no more re-assignments take place. So, the centres do not move, and the algorithm has converged. The final allocation of each point is indicated by the shapes and the final cluster centres are indicated by the larger hollow shapes.

We could try repeating the algorithm with a different number of clusters, though it is not necessarily straightforward to decide on the number to use. We can sometimes be informed by prior knowledge of the data.

The table below shows the strengths and weaknesses of the K-means algorithm.

Strengths	Weaknesses
Uses simple principles for identifying clusters which can be explained in non-statistical terms	**Less sophisticated than more recent clustering algorithms**
Highly flexible and can be adapted to address nearly all its shortcomings with simple adjustments	**Not guaranteed to find the optimal set of clusters because it incorporates a random element**
Fairly efficient and performs well	**Requires a reasonable guess as to how many clusters naturally exist in the data**

Source: B. Lantz, *Machine Learning with R* (Birmingham, Packt Publishing, 2013), p. 271

The interpretation and evaluation of the results of K-means clustering can be somewhat subjective. If the K-means exercise has been useful, the characteristics of the clusters will be interpretable within the context of the problem being studied and will either confirm that the pre-existing opinion about the existence of homogeneous groups has an evidential base in the data, or provide new insights into the existence of groups that were not seen before. One objective criterion that can be examined is the size of each of the clusters. If one cluster contains the vast majority of the cases, or there are clusters with only a few cases, this may indicate that the algorithm has failed to find K meaningful groups.

Another way to judge the effectiveness of the algorithm is to repeat the process several times with different random allocations at the start. If similar groupings are obtained each time, it is likely that there is a valid basis underlying the clusters.

One function that can perform K-means clustering in R is `kmeans()`. There are also several machine learning packages that can carry out K-means clustering.

5.2 Principal component analysis

This is covered in Subject CS1 Chapter 11. Where K-means clustering is to be applied to a high-dimensional data set, it is often advisable to perform principal components analysis as a pre-processing step and apply K-means using just the first few principal components.

Principal components analysis (PCA) is a technique that can be used to reduce the dimensionality of a data set. Given a data set containing d variables, PCA represents the data using a different set of d variables (components) that are uncorrelated linear combinations of the original set. If strong relationships exist between the original variables, then the new components are constructed in such a way that the full data set can be reasonably accurately represented using only a subset of these components.

In order to choose how many components are retained, we might decide to keep as many as necessary to 'explain' a desired proportion of the total variance within the data set. For example, we may wish to retain the smallest number of components, $c < d$, that explain at least 90% of the variation in the original data set.

PCA can be a useful tool in machine learning, where data sets can be extremely large and therefore computationally expensive.

References

Chalk and C. McMurtrie 'A practical introduction to Machine Learning concepts for actuaries', *Casualty Actuarial Society E-forum*, Spring 2016.

Wickham, Tidy Data: https://vita.had.co.nz/papers/tidy-data.pdf

James et al, An Introduction to Statistical Learning with R: https://www.statlearning.com/

6 Appendix: derivation of the bias-variance decomposition

The formula given in Section 2.5 is:

$$E\left[\left(y_0-\hat{f}(x_0)\right)^2\right]=\text{var}\left(\varepsilon_0\right)+\text{var}\left[\hat{f}(x_0)\right]+\left\{f(x_0)-E\left[\hat{f}(x_0)\right]\right\}^2$$

To derive this, we first write $y_0=f(x_0)+\varepsilon_0$ in the LHS above and use $\hat{f}$ and f to represent $\hat{f}(x_0)$ and $f(x_0)$ respectively:

$$E\left[(y_0-\hat{f})^2\right]=E\left[(f+\varepsilon_0-\hat{f})^2\right]$$

We now recognise the RHS as being of the form $E(X^2)$ to rewrite it as follows:

$$E\left[(f+\varepsilon_0-\hat{f})^2\right]=\text{var}(f+\varepsilon_0-\hat{f})+\left[E(f+\varepsilon_0-\hat{f})\right]^2$$

using $E(X^2)=\text{var}(X)+[E(X)]^2$.

Considering each of these terms:

$$\begin{aligned}
\text{var}(f+\varepsilon_0-\hat{f}) &= \text{var}(\varepsilon_0-\hat{f}) && \text{as } f \text{ is a constant}\\
&= \text{var}(\varepsilon_0)+\text{var}(\hat{f}) && \text{as } \hat{f} \text{ and } \varepsilon_0 \text{ are independent}\\
E(f+\varepsilon_0-\hat{f}) &= E(f)+E(\varepsilon_0)-E(\hat{f}) && \text{using linearity of expectation}\\
&= f+0-E(\hat{f}) && \text{as } f \text{ is a constant and } E(\varepsilon_0)=0\\
&= f-E(\hat{f})
\end{aligned}$$

So:

$$E\left[(y_0-\hat{f})^2\right]=\text{var}(\varepsilon_0)+\text{var}(\hat{f})+\left[f-E(\hat{f})\right]^2$$

as required.

Recall that $\hat{f}$ here (in random variable form) is an estimator of the true function form, f. So:

$$\left[f-E(\hat{f})\right]^2=\left[E(\hat{f})-f\right]^2=bias(\hat{f})^2$$

This gives:

$$E\left[(y_0-\hat{f})^2\right]=\text{var}(\varepsilon_0)+\text{var}(\hat{f})+bias(\hat{f})^2$$

The sum of the last two terms, $\text{var}(\hat{f})+bias(\hat{f})^2$, is an alternative representative of the mean squared error (MSE) of the estimator $\hat{f}$. MSE is covered in Chapter 8 of Subject CS1.

The chapter summary starts on the next page so that you can keep all the chapter summaries together for revision purposes.

Chapter 21 Summary

What is machine learning?

Machine learning is a collection of methods for the automatic detection and exploitation of patterns in data. Examples can be found in many areas of everyday life, *eg* in targeting online advertising.

Machine learning is increasingly being used in the areas of finance and insurance, *eg* for predicting which borrowers are most likely to default on a loan.

Broadly, machine learning problems can be divided into *supervised learning* problems and *unsupervised learning* problems.

In a typical supervised learning problem, the aim is to determine the relationship between a collection of d input variables, $x_1,...,x_d$, and an output variable, y, based on a sample of training data $\left(\underline{x}_i, y_i\right)\ i=1,2,...,n$.

With *unsupervised learning*, the algorithm aims to identify patterns in a data set, without being given a specific target. So, there is no output variable, and we are instead trying to identify patterns in the input variables, $x_1,...,x_d$.

Supervised learning concepts

The functional relationship

We write the relationship between the inputs and output as:

$$y_i = f(\underline{x}_i) + \varepsilon_i$$

where f is the functional relationship and ε_i is an error term. The aim of supervised learning is to estimate $f(\underline{x}_i)$ based on the available data.

Prediction vs inference

If prediction is the main aim, then we may not be that interested in the underlying structure of the relationship between input and output, only how accurate a model is.

However, we are often also interested in inference: determining which inputs, or combinations of inputs, are most closely associated with the output; discriminating between different hypotheses about the data, and making statements about the uncertainty in predictions.

Typical supervised learning problems

Problems where the output variable to be predicted is a numerical variable, such as an amount of revenue, an individual's survival time, or the number of employees in a firm, are said to be regression problems.

Problems where the output variable is qualitative (categorical), such as when predicting credit default (yes/no), are said to be classification problems. The most common classification tasks are binary classification tasks, where the outcome variable takes one of two distinct values.

Machine learning methods that make assumptions about the function relationship (*eg* assuming it is linear, as in linear regression) are called parametric methods. Those that make no (or few) such assumptions are non-parametric methods.

Evaluating performance for regression problems

The mean square error (MSE) compares a model's predictions with the true output values. It is defined as:

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

The lower the MSE, the closer the predictions are to the true values. More precisely, this might be called the training MSE because it is evaluated on the data used to train the model. Given that we want to use a model to predict the results for unseen data, this measure does not quite match up with our requirements. Given enough parameters, we could easily make a model that matches the training data with zero training error, but such a model would likely perform badly when used to predict new observations.

Any model should have a sufficient number of parameters to be flexible enough to capture underlying trends, but not so many that it reflects specific features of the training set that we would not expect to be present in new data.

The bias-variance decomposition

For a given (unseen) value x_0, the expected mean square error for the corresponding output value y_0 can be decomposed into three interpretable terms:

$$E\left[\left(y_0 - \hat{f}(x_0)\right)^2\right] = \text{var}\left(\varepsilon_0\right) + \text{var}\left[\hat{f}(x_0)\right] + \left\{f(x_0) - E\left[\hat{f}(x_0)\right]\right\}^2$$

$\text{var}\left(\varepsilon_0\right)$ is the irreducible error and reflects the uncertainty in y_0 even if the true functional relationship is known.

$\text{var}\left[\hat{f}(x_0)\right]$ is the variance of the estimator of the functional relationship. It reflects the variation in the estimates across different training samples.

$\left\{f(x_0)-E\left[\hat{f}(x_0)\right]\right\}^2$ is the square of the bias of the estimator of the functional relationship.

It represents the underlying difference between the structure of the fitted model and the true functional relationship.

Evaluating binary classifiers

Classification models that result in Yes or No outputs can be assessed using a confusion matrix for the test set, which shows:

True Positives (TP)	False Negatives (FN)
False Positives (FP)	True Negatives (TN)

We can use this to calculate:

$$\text{Precision}=\frac{TP}{TP+FP} \qquad \text{Recall (sensitivity)}=\frac{TP}{TP+FN} \qquad F_1=\frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$$

$$\text{false positive rate}=\frac{FP}{TN+FP} \qquad \text{accuracy}=\frac{TP+TN}{TP+TN+FP+FN}$$

These all take values in the range 0% (worst) to 100% (best). One minus the false positive rate is also known as the specificity.

A receiver operating characteristic (ROC) curve can be used to compare models. This plots the true positive rate against the false positive rate for different thresholds (the rule for assigning predicted categories).

Train-validation-test

A common approach is to split the available data into three parts:

- a training data set, on which the model is trained (*eg* 60%)
- a validation data set, with which hyperparameters can be tuned (*eg* 20%)
- a test data set, to evaluate out-of-sample performance (*eg* 20%).

An alternative is to split the data into K subsets of roughly equal size. K different instances of the model are then trained, with instance i using all data except the i th subset. Subset i is then used to evaluate the out-of-sample error of model instance i. The average of the K different out-of-sample errors can then be used as a proxy for the true out-of-sample error.

Supervised learning workflow

Machine learning tasks can be broken down into the following stages:

- collecting data
- exploring and preparing the data
- feature scaling (to ensure that the input variables have similar ranges of values)
- splitting the data
- training the model
- evaluation.

Reproducibility of research

It is important for data analysis to be reproducible and well-documented. For stochastic models, which include random elements, it is normally sufficient that the distribution of results is reproducible.

Supervised learning techniques

Penalised generalised linear models

Penalised regression is an adaptation of the method of maximum likelihood where a penalty is applied to constrain the estimated values of the parameters to improve their reliability for making predictions. The method involves maximising the penalised likelihood:

$$l(\beta_0, \beta_1, \ldots, \beta_d \mid \underline{x}_1, \ldots, \underline{x}_n) - \lambda g(\beta_0, \beta_1, \ldots, \beta_d)$$

We could use $\sum_{i=1}^{d} \beta_i^2$ (*ridge*) or $\sum_{i=1}^{d} |\beta_i|$ (*Lasso*) for the penalty function $g(\beta_0, \beta_1, \ldots, \beta_d)$.

To try to select an appropriate number of parameters, d, (with sample size n), we can minimise:

$$\text{AIC} = \text{deviance} + 2d \quad \text{(Akaike information criterion)}$$

or $$\text{BIC} = \text{deviance} + d\ln(n) \quad \text{(Bayesian information criterion)}$$

Naïve Bayes algorithm

The *naïve Bayes* algorithm uses Bayes' formula to classify items by determining the relative likelihood of each of the possible options for the given values of the covariates $x_{1i},...,x_{di}$. The method assumes that the conditional probabilities of the covariates given an outcome are independent. For example, for the outcome $y_i = 1$, we have:

$$P(y_i = 1 \mid x_{1i},...,x_{di}) \propto P(y_i = 1)\prod_{j=1}^{d} P(x_{ji} \mid y_i = 1)$$

Decision trees (CART analysis)

Decision trees, also known as classification and regression trees (CART), classify items by asking a series of questions about the inputs that attempt to differentiate between the categories. The simplest method of construction is to use greedy splitting where, at each stage, the best partition of the data is chosen to maximise the reduction in a loss function. For classification problems, a common loss function is the Gini index. For regression problems, squared error loss is commonly used.

Gini index for a decision tree for a classification problem

The Gini index is defined as: $\sum_{j=1}^{J} n_j \sum_{k=1}^{K} p_{jk}(1-p_{jk})$

where j indexes over the external nodes, p_{jk} is the proportion of sample items of class k present at the jth external node and n_j is the number of data points present at the jth external node. When using greedy splitting, the aim at each stage is to find the partition that maximises the reduction in the loss function. For any particular split point, this is equivalent to minimising the following weighted sum of the impurity scores of the child notes of the proposed partition:

$$\sum_{c=1}^{2} n_c \sum_{k=1}^{K} p_{ck}(1-p_{ck})$$

where c indexes over the two child nodes, p_{ck} is the proportion of sample items of class k present at the cth child node and n_c is the number of data points present at the cth child node.

The individual impurity score for a given external or child node is

$\sum_{k=1}^{K} p_{ck}(1-p_{ck}) = 1 - \sum_{k=1}^{K} p_{ck}^2$. This gives a value between 0 ('pure') and $1-\frac{1}{K}$ ('mixed') where K is the number of distinct classes. As $K \to \infty$, the upper limit tends to 1.

Squared error loss for a decision tree for a regression problem

The squared error is: $\sum_{j=1}^{J}\sum_{i=1}^{n_j}\left(y_i-\hat{y}_j\right)^2$

where y_i are the observed values and $\hat{y}_j$ is the predicted value for the jth external node.

When using greedy splitting, we want to maximise the reduction in the loss function. This is equivalent to obtaining the partition that minimises the sum of squared errors for the child nodes resulting from the proposed split, *ie* the partition that minimises:

$$\sum_{c=1}^{2}\sum_{i=1}^{n_c}(y_i-\hat{y}_c)^2$$

Bagged decision trees and random forests

Averaging predictions from decision trees built on bootstrapped samples (bagging) typically leads to much better predictions than would be obtained from a single decision tree in isolation. In particular, bagging reduces the variance of the predictions.

Random forests are also constructed by building trees using bootstrapped samples; however, now only a random sample of the input variables are considered at each split point. This compensates for the greedy nature of the tree-building algorithm, and so allows for better exploration of the predictor space. This further reduces the variance of predictions.

Unsupervised learning techniques

K-means clustering

The K-means clustering algorithm involves modelling the data values as points in space. Starting from an initial cluster allocation (commonly random), the method repeatedly finds the centroid of the data points that have been allocated to each cluster and then reallocates the points to the cluster whose centroid they are nearest to. When this process reaches a stage where no further changes are made, the algorithm has converged.

This method has the advantages that:

- it uses a simple principle that can easily be explained
- it is highly flexible and can easily be adapted to address any shortcomings
- it is efficient and performs well.

However, it also has the disadvantages that:

- it is less sophisticated than more recent clustering algorithms
- it is not guaranteed to find the optimal set of clusters (because of the random element)
- it requires a reasonable guess as to how many clusters naturally exist in the data.

Principal components analysis

Principal components analysis is a technique that can be used to reduce the dimensionality of a data set. Given a data set containing d variables, PCA represents the data using a different set of d variables (components) that are uncorrelated linear combinations of the original set. If strong relationships exist between the original variables, then the new components are constructed in such a way that the full data set can be reasonably accurately represented using only a subset of these components.

The practice questions start on the next page so that you can keep the chapter summaries together for revision purposes.

Chapter 21 Practice Questions

21.1 (i) Explain the distinction between supervised and unsupervised learning in the context of machine learning.

(ii) State whether each of the following applications involves supervised or unsupervised learning, and suggest a suitable machine learning algorithm that could be used in each case:

(a) predicting a university graduate's salary at age 40 based on the subject they studied, the grade they obtained in their degree and their sex

(b) grouping car insurance policyholders based on geographical area, premium paid and claims experience

(iii) Explain the following terms relating to machine learning:

(a) hyperparameters

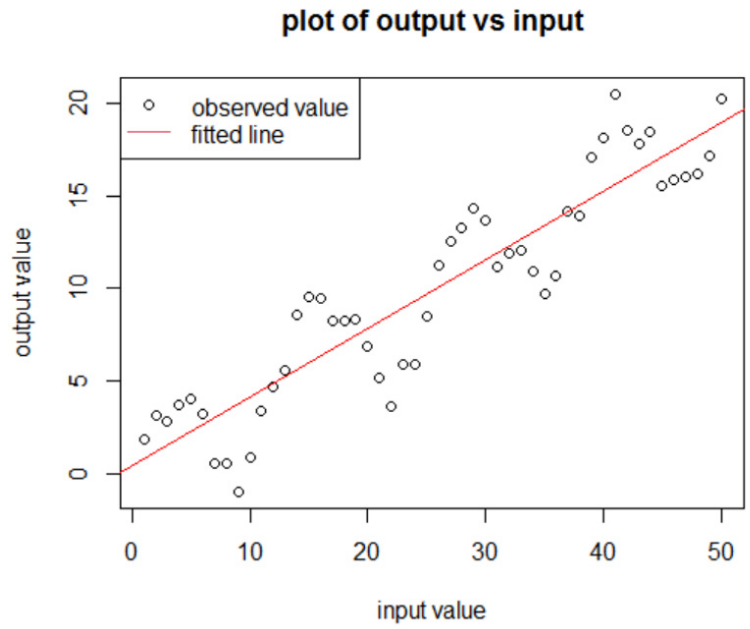
(b) CART

(c) greedy splitting

(d) clustering.

21.2 Describe the three components of the bias-variance decomposition.

21.3 (a) Describe why the following model does not fit the observed data well:



(b) Suggest how this could be addressed.

21.4 In a sample of size 100, 10% of individuals have a particular feature.

(a) Draw up a confusion matrix for a test that can identify this feature perfectly.

(b) Calculate the precision, recall, F_1 score, false positive rate, and accuracy, based on the numbers in your matrix.

21.5 A test is available to detect a certain virus. The test has sensitivity (*ie* recall) 95% and specificity 99.5% (*ie* false positive rate 0.5%). The prevalence of the virus is 0.2%.

(a) Construct the confusion matrix for this test based on a population of 100,000 people.

(b) Calculate the F_1 score for the test.

21.6 A random sample $x_1,\ldots,x_n$ of values has been taken from a $N(\mu,\sigma_0^2)$ distribution, where the value of σ_0 is known but the value of μ is unknown. However, it is believed that the value of μ is close to 100. It has been suggested that μ could be estimated using a penalised log-likelihood function.

(i) Explain the rationale behind this method.

(ii) Suggest why the penalty function $\lambda(\mu-100)^2$ would be suitable to use in this case.

(iii) Show that the estimate of μ derived using this method with the penalty function in part (ii) is:

$$\hat{\mu}=\left(\frac{n}{\sigma_0^2}\bar{x}+200\lambda\right)\Big/\left(\frac{n}{\sigma_0^2}+2\lambda\right)$$

(iv) Comment on how the value of $\hat{\mu}$ calculated using the formula in part (iii) is influenced by the value chosen for the regularisation parameter.

(v) Explain why this method might be preferable in some circumstances to the basic method of maximum likelihood.

21.7 A warehouse stores four types of single malt whisky in identical bottles in an underground storeroom, before they are labelled and distributed. Recently the warehouse experienced a major flood in which the stock records were destroyed and the handwritten descriptions on some of the cases were washed off, so that they could no longer be identified.

The warehouse manager has asked you to supervise the process of identifying the type of whisky each of these cases belongs to, so that they can be correctly labelled. A professional whisky taster has provided a report based on a single bottle from each case, which he has classified based on three criteria: Smoky, Fruity, Colour (each on a scale of 1 to 3).

The standard descriptions of the four types are shown in the table below. These can be considered to have a probability of 80% of being correct, while any other description has a probability of 10%. The warehouse manager has also indicated the proportions of each type she believes were in stock at the time of the flood.

Case	Smoky	Fruity	Colour	Proportion in stock
Mactavish	1	3	1	40%
Western Isle	2	1	1	30%
Glenragh	2	2	3	10%
Dogavulin	3	2	2	20%

(i) Show that, under the assumptions of the naïve Bayes model:

$$P(y = A \mid x_1, x_2, x_3) \propto P(y = A) \times P(x_1 \mid y = A) P(x_2 \mid y = A) P(x_3 \mid y = A)$$

The taster has described the sample bottle from one of the cases as:

Smoky = 2, Fruity = 2, Colour = 2

(ii) Use the formula from part (i) to estimate how likely this case is to be of each of the four types, and hence recommend how it should be labelled assuming equal misclassification costs.

(iii) State two advantages and one disadvantage of the naïve Bayes classification method as a machine learning technique.

21.8 A doctor is using K -means clustering with $K=5$ to classify her patients by height and weight. The raw data shows the patients' statistics in m and kg, but she has converted the heights to cm. She used a method based on Euclidean distance, which converged after 3 iterations, giving the following final centroids of the clusters:

Group	1	2	3	4	5
Height (cm)	165	160	175	150	185
Weight (kg)	55	65	80	90	100

(i) Explain what the value of K represents.

(ii) Explain the reason for the doctor's unit conversion.

(iii) Explain what convergence means in this context.

(iv) Three of the patients in the data set have the following data values:

Mr Blobby: (1.64m, 91kg)

Miss Twiggy: (1.87m, 54kg)

Mrs Average: (1.66m, 64kg)

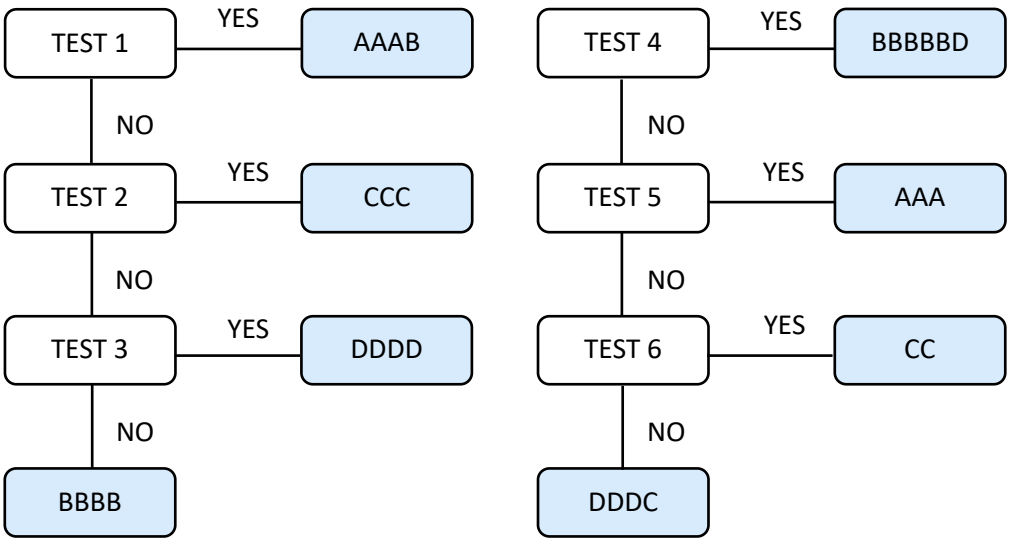
By using a graph, or otherwise, identify the clusters to which these patients belong based on their heights and weights.

(v) State whether the results in part (iv) would have differed if the clusters had been obtained using the absolute distance metric $D_{abs}(x,k)=\sum_{j=1}^{J}\left|x_j-k_j\right|$.

21.9 (i) (a) If $p_1 + p_2 + \cdots + p_K = 1$, prove the identity $\sum_{k=1}^{K} p_k \sum_{i=1, i\neq k}^{K} p_i = 1 - \sum_{k=1}^{K} p_k^2$.

(b) Explain how this identity can be used to calculate a measure of the effectiveness of a proposed split point when constructing decision tree.

A researcher is considering two possible decision trees to classify items of four different types labelled A, B, C and D. A sample of 15 items classified using each of the trees gave the results shown below.



(ii) (a) Calculate the Gini index for the initial split point of each tree.

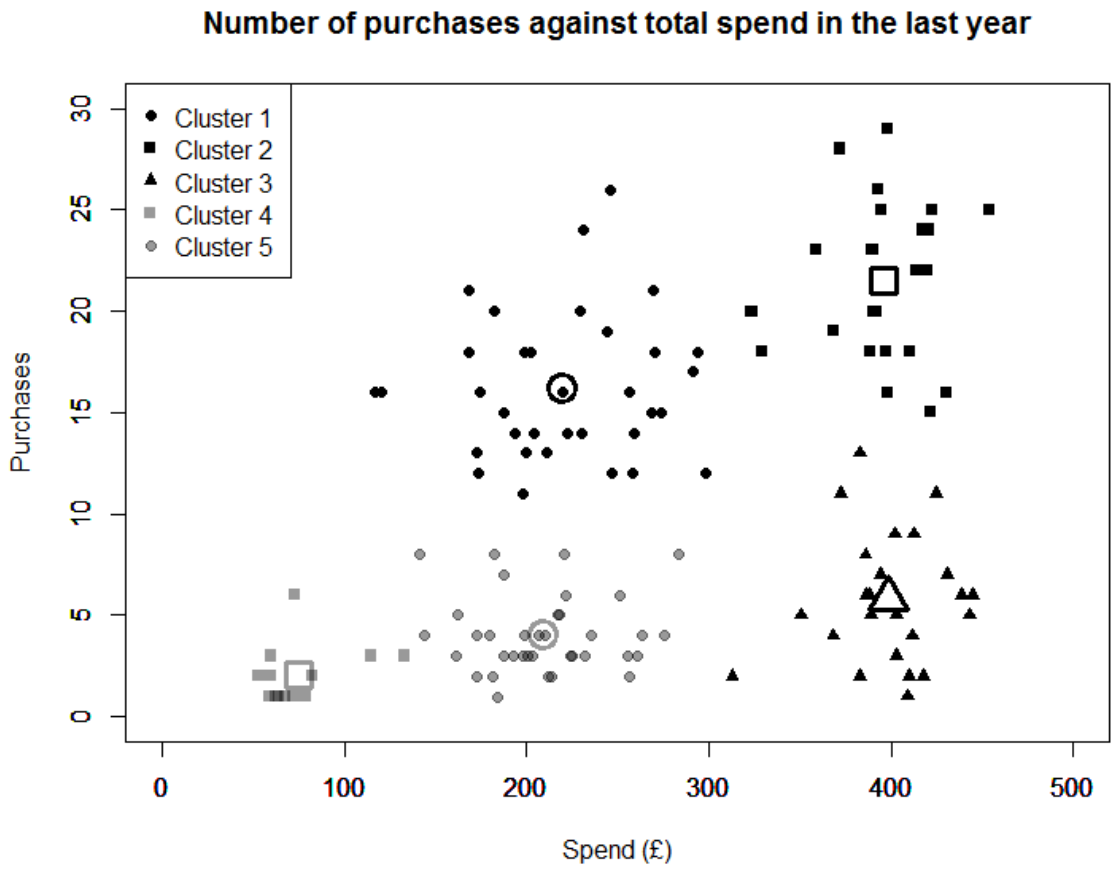
(b) Calculate the accuracy of each tree, assuming that the predicted category for each final leaf node is the type making up the majority in that node.

(c) Comment on your answers to part (ii)(a) and part (ii)(b).

21.10 An online retailer ran the K-means algorithm with $K=5$ on its customer base, using the following variables:

- total spend in the last year
- number of items purchased in the last year.

The output is shown below:



The centres of each cluster are plotted as the larger hollow shapes.

(i) Describe the 5 clusters.

The retailer's current marking strategy is a monthly email to all customers outlining its 'top picks' across its entire product range.

(ii) Suggest how the retailer could market more specifically to customers in Cluster 3.

Chapter 21 Solutions

21.1 (i) ***Supervised and unsupervised learning***

With supervised learning, the desired outcome for each data point is specified in advance and the algorithm aims to reproduce these as closely as possible.

With unsupervised learning, the desired outcome for each data point is not specified in advance, as the data are unlabelled. The algorithm aims to identify patterns within the data.

(ii) ***Applications***

(a) Here we would aim to reproduce the salaries for a sample of graduates as closely as possible based on the three variables. So, this would be supervised learning. We could use a multiple linear regression model here.

(b) Here we would hope that the algorithm can find suitable homogeneous groups that we don't know in advance. So, this would be unsupervised learning. We could use the K-means clustering algorithm here.

(iii) ***Terminology***

(a) As well as the 'internal' parameters that a model estimates from the data and uses to calculate predictions, machine learning methods often also require hyperparameters, which are external to the model and whose values are set in advance based on the user's knowledge and experience, in order to produce a model that works well. An example is the number of clusters to aim for with the K-means algorithm.

(b) CART is an abbreviation for classification and regression trees, which is another name for decision trees. When used for classification problem, these categorise data points by asking a series of questions that attempt to home in on a classification.

(c) Greedy splitting is a method of constructing a decision tree. At each stage, we choose the split that leads to the largest reduction in the loss function. In other words, we choose the split that appears to be the most effective at separating the remaining elements, without thinking ahead to the consequences this might have for the later splits.

(d) Clustering refers to grouping data into a set of homogeneous groups or clusters, which can be done using methods such as the K-means algorithm.

21.2 *The expected mean square error for the output value for a given unseen input value can be represented as:*

$$E\left[\left(y_0-\hat{f}(x_0)\right)^2\right]=\text{var}\left(\varepsilon_0\right)+\text{var}\left[\hat{f}(x_0)\right]+\left\{f(x_0)-E\left[\hat{f}(x_0)\right]\right\}^2$$

The three components are:

$\text{var}\left(\varepsilon_0\right)$, which is the irreducible error and reflects the uncertainty in y_0 even if the true functional relationship is known.

$\text{var}\left[\hat{f}(x_0)\right]$, which is the variance of the estimator of the functional relationship. It reflects the variation in the estimates across different training samples.

$\left\{f(x_0)-E\left[\hat{f}(x_0)\right]\right\}^2$, which is the square of the bias of the estimator of the functional relationship. It represents the underlying difference between the structure of the fitted model and the true functional relationship.

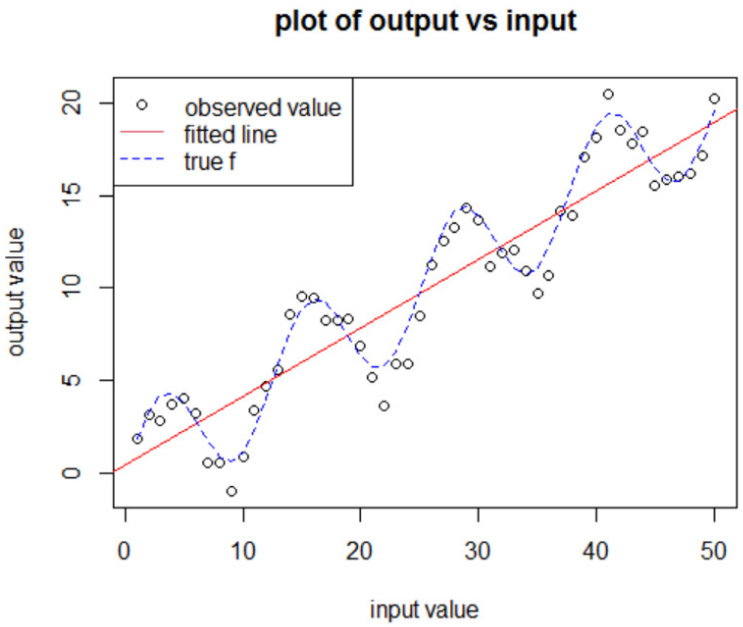
21.3 (a) ***Mode of failure***

The fitted model is a straight line and is not flexible enough to capture what appears to be a somewhat cyclical pattern observed in the data. This model has bias – there is a fundamental difference between the structure of the fitted model and the true functional relationship.

(b) ***Improvements***

We could try a model with more parameters to try to better capture the patterns in the data.

A plot showing the true functional relationship is provided below:



21.4 (a) The confusion matrix looks like this:

PREDICTED / ACTUAL	YES	NO	TOTAL
YES	TP = 10	FN = 0	10
NO	FP = 0	TN = 90	90
TOTAL	10	90	100

(b) The four measures are:

$$\text{precision} = \frac{\text{TP}}{\text{TP}+\text{FP}} = \frac{10}{10+0} = 100\%, \quad \text{recall} = \frac{\text{TP}}{\text{TP}+\text{FN}} = \frac{10}{10+0} = 100\%$$

$$F_1 \text{ score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{2 \times 100\% \times 100\%}{100\% + 100\%} = 100\%$$

$$\text{false positive rate} = \frac{\text{FP}}{\text{TN}+\text{FP}} = \frac{0}{90+0} = 0\%$$

$$\text{accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}} = \frac{10+90}{10+90+0+0} = 100\%$$

21.5 (a) ***Confusion matrix***

The number of true positives is: $0.95 \times 200 = 190$

The number of false positives is: $0.005 \times 99,800 = 499$

The number of false negatives is: $200 - 190 = 10$

The number of true negatives is: $99,800 - 499 = 99,301$

The confusion matrix looks like this:

TEST RESULT / ACTUAL	YES	NO	TOTAL
YES	190	10	200
NO	499	99,301	99,800
TOTAL	689	99,311	100,000

(b) ***F1 score***

We need the precision and recall in order to calculate the F_1 score. The recall is given in the question as 95%. The precision is:

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}} = \frac{190}{190+499} = \frac{190}{689} = 27.6\%$$

The F_1 score is then:

$$F_1 \text{ score} = \frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}} = \frac{2\times 27.6\%\times 95\%}{27.6\%+95\%} = 42.7\%$$

21.6 (i) ***Rationale***

This method is based on the method of maximum likelihood where we choose the parameter values to maximise the log-likelihood of the data available. This gives the parameter values that best explain the values in the data.

However, we can also apply a penalty function, which is chosen to make the method more likely to produce parameter estimates close to a set of target values that we are expecting. The penalty is subtracted from the log-likelihood and we maximise this adjusted function instead.

(ii) ***Penalty function***

We expect the value of μ to be close to 100. The function $\lambda(\mu-100)^2$ would be suitable to use here because it takes a large positive value if μ is a long way from 100 (in either direction).

(iii) ***Formula***

The likelihood function for the sample is:

$$L = \prod_{i=1}^{n} \frac{1}{\sigma_0\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x_i-\mu}{\sigma_0}\right)^2\right] = \sigma_0^{-n} \exp\left[-\frac{1}{2\sigma_0^2}\sum_{i=1}^{n}(x_i-\mu)^2\right] \times \text{constant}$$

So the log-likelihood is:

$$\log L = -n\log\sigma_0 - \frac{1}{2\sigma_0^2}\sum_{i=1}^{n}(x_i-\mu)^2 + \text{constant}$$

and the penalised log-likelihood is:

$$(\log L)^* = -n\log\sigma_0 - \frac{1}{2\sigma_0^2}\sum_{i=1}^{n}(x_i-\mu)^2 + \text{constant} - \lambda(\mu-100)^2$$

To maximise this, we equate the derivative with respect to the parameter μ to zero:

$$\frac{\partial(\log L)^*}{\partial\mu} = +\frac{2}{2\sigma_0^2}\sum_{i=1}^{n}(x_i-\mu) - 2\lambda(\mu-100) = \frac{n}{\sigma_0^2}(\bar{x}-\mu) - 2\lambda(\mu-100) = 0$$

Rearranging this gives:

$$\left(\frac{n}{\sigma_0^2}+2\lambda\right)\mu = \frac{n}{\sigma_0^2}\bar{x} + 200\lambda$$

So the estimate of μ is:

$$\hat{\mu} = \left(\frac{n}{\sigma_0^2}\bar{x}+200\lambda\right)\Big/\left(\frac{n}{\sigma_0^2}+2\lambda\right)$$

(iv) ***Influence of the regularisation parameter***

The regularity parameter λ will be assigned a non-negative value. (Otherwise, it would correspond to a *reward* rather than a penalty.)

If λ were set equal to zero, there would be no penalty and the method would reduce to maximum likelihood estimation. As expected, the formula in part (iii) would then give the usual formula for the MLE of the mean of a normal distribution:

$$\hat{\mu} = \left(\frac{n}{\sigma_0^2}\bar{x}+0\right)\Big/\left(\frac{n}{\sigma_0^2}+0\right) = \frac{n}{\sigma_0^2}\bar{x}\Big/\frac{n}{\sigma_0^2} = \bar{x}$$

If λ were given a very high value, the penalty would dominate the calculations and, in the limit, we would have:

$$\hat{\mu} = \lim_{\lambda\to\infty}\left(\frac{n}{\sigma_0^2}\bar{x}+200\lambda\right)\Big/\left(\frac{n}{\sigma_0^2}+2\lambda\right) = \lim_{\lambda\to\infty}\left(\frac{n}{\lambda\sigma_0^2}\bar{x}+200\right)\Big/\left(\frac{n}{\lambda\sigma_0^2}+2\right) = \frac{200}{2} = 100$$

In this case, the estimate is equal to the target value of 100.

(v) ***Why this method might be preferable***

The basic (unpenalised) method of maximum likelihood can sometimes lead to unreliable results. The estimated values of the parameters can be very sensitive to the sample data and can vary wildly.

This is most likely to happen when the sample size is small or the likelihood function is very flat so that changes in the parameter values make very little difference to the log-likelihood.

Applying a penalty function encourages the method to produce parameter estimates that are close to the values that would be expected from prior expectations.

21.7 (i) ***Formula for naïve Bayes method***

$$P(y=A\mid x_1,x_2,x_3)=\frac{P(y=A,x_1,x_2,x_3)}{P(x_1,x_2,x_3)} \quad \text{(definition of conditional probability)}$$

$$=\frac{P(x_1,x_2,x_3\mid y=A)P(y=A)}{P(x_1,x_2,x_3)} \quad \text{(definition of conditional probability in reverse)}$$

$$=\frac{P(x_1\mid y=A)P(x_2\mid y=A)P(x_3\mid y=A)P(y=A)}{P(x_1,x_2,x_3)} \quad \text{(independence assumption)}$$

$$\propto P(y=A)\times P(x_1\mid y=A)P(x_2\mid y=A)P(x_3\mid y=A) \quad \text{(ignore constant factor)}$$

(ii) ***Probabilities for each type***

Let y denote the type and let x_1,x_2,x_3 denote the three descriptions (Smoky, Fruity, Colour).

We can then apply the result from part (i) to calculate the probability that this case is a Mactavish whisky $(y=M)$, given that it has been described as Smoky $(S=2)$, Fruity $(F=2)$ and is medium Colour $(C=2)$:

$$P(y=M\mid S=2,F=2,C=2)\propto P(y=M)\times P(S=2\mid y=M)P(F=2\mid y=M)P(C=2\mid y=M)$$

$$=0.4\times0.1\times0.1\times0.1=0.0004$$

Similarly, for the other three types:

$$P(y=W\mid S=2,F=2,C=2)\propto P(y=W)\times P(S=2\mid y=W)P(F=2\mid y=W)P(C=2\mid y=W)$$

$$=0.3\times0.8\times0.1\times0.1=0.0024$$

$$P(y=G\mid S=2,F=2,C=2)\propto P(y=G)\times P(S=2\mid y=G)P(F=2\mid y=G)P(C=2\mid y=G)$$

$$=0.1\times0.8\times0.8\times0.1=0.0064$$

$$P(y=D\mid S=2,F=2,C=2)\propto P(y=D)\times P(S=2\mid y=D)P(F=2\mid y=D)P(C=2\mid y=D)$$

$$=0.2\times0.1\times0.8\times0.8=0.0128$$

So the probabilities for each type are in the ratio:

$$\text{M}:\text{W}:\text{G}:D \;=\; 4:24:64:128=\tfrac{4}{220}:\tfrac{24}{220}:\tfrac{64}{220}:\tfrac{128}{220}=0.0182:0.1091:0.2909:0.5818$$

So the recommendation would be that this case is a Dogavulin whisky, as this has a far higher probability (58%) than the other three types.

(iii) ***Advantages and disadvantages***

Advantages of the naïve Bayes method include:

- it is easy to apply
- it requires very little data.

The main disadvantage is that it assumes that the conditional probabilities are independent (which can be a poor approximation when the variables are correlated).

21.8 (i) ***Meaning of K***

K is a hyperparameter specifying the number of clusters the algorithm should aim to produce – in this case, 5.

(ii) ***Choice of units***

The weights in the original data provided were given in units of kilograms. These cover a range of values of about 50kg.

The heights in the original data provided were given in units of metres. These cover a range of values of about 0.50m.

So with units of (kg, m) the range for the weights is about 100 times greater, which would mean that the weights would totally dominate the calculations and the heights would effectively be ignored.

However, when the doctor converts the heights to centimetres, the range of values is then about 50cm, which is numerically very similar to the range for the weights. This gives the two variables a similar weighting in the calculations.

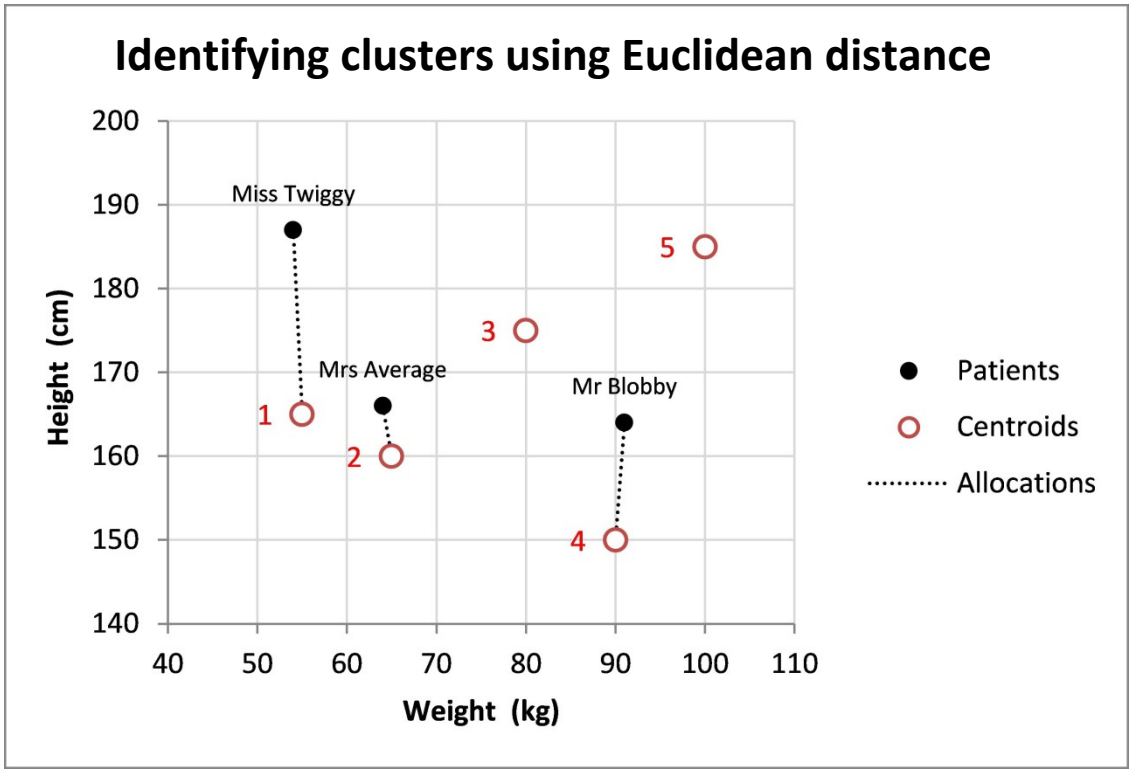
In practice both variables would be standardised by, for example, subtracting the mean and dividing by the standard deviation.

(iii) ***Convergence***

The algorithm involves repeatedly finding the centroid of the data points that have been allocated to each cluster and then reallocating the points to the cluster whose centroid they are nearest to. When this process reaches a stage where no further changes are made, the algorithm has converged.

(iv) ***Clusters of the patients***

If we plot the patients and the centroids for the clusters on a graph, we can easily see which centroid each patient is nearest to, and hence identify the cluster for these patients.



If we do the calculations, we find that the shortest Euclidean distances D_i (for the centroids $i = 1,2,\ldots,5$) are:

Mr Blobby: (1.64m, 91kg) $\rightarrow D_4 = \sqrt{(164-150)^2 + (91-90)^2} = 14.04$

Miss Twiggy: (1.87m, 54kg) $\rightarrow D_1 = \sqrt{(187-165)^2 + (54-55)^2} = 22.02$

Mrs Average: (1.66m, 64kg) $\rightarrow D_2 = \sqrt{(166-160)^2 + (64-65)^2} = 6.08$

(v) ***Absolute distance***

The absolute distance measures the distance between points assuming that we can only move horizontally or vertically.

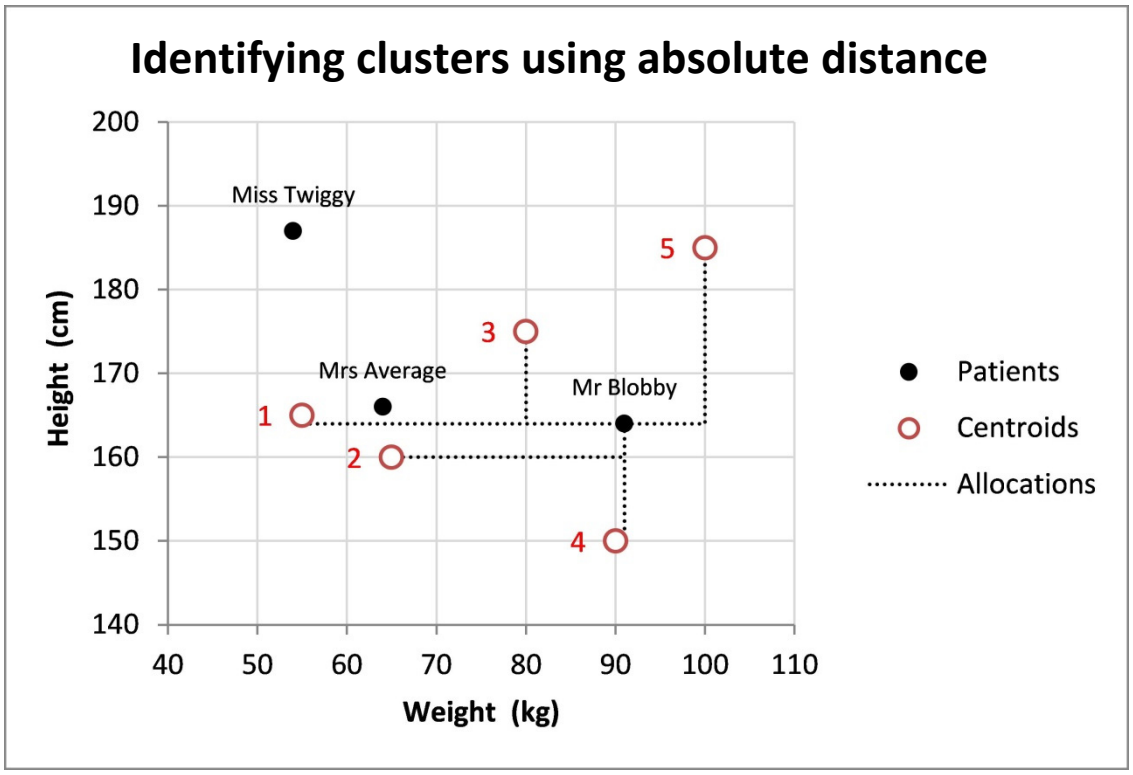
Here we have two dimensions (the two variables Weight and Height), so $J = 2$. With this metric the distance between the two points (x_1, x_2) and (k_1, k_2) is:

$$|x_1 - k_1| + |x_2 - k_2|$$

With this metric, Mr Blobby's distance from the centroid for cluster 4 is:

Mr Blobby: (1.64m, 91kg) $\rightarrow D_4 = |164 - 150| + |91 - 90| = 14 + 1 = 15$

The diagram below shows the distance to each centroid for Mr Blobby.



We can see that if the clusters had been constructed using absolute distances, we would get the same answers as if they had been constructed using Euclidean distance.

21.9 (i)(a) ***Prove the identity***

$$\sum_{k=1}^{K} p_k \sum_{i=1, i\neq k}^{K} p_i = \sum_{k=1}^{K} p_k \left(1-p_k\right) = \sum_{k=1}^{K} \left(p_k - p_k^2\right) = \sum_{k=1}^{K} p_k - \sum_{k=1}^{K} p_k^2 = 1 - \sum_{k=1}^{K} p_k^2$$

(i)(b) ***Measuring the effectiveness of a proposed split point***

We can measure the effectiveness of a proposed split point by examining the 'purity' of the data in each of the child nodes and then calculating an overall measure of the purity of the split.

This can be done by multiplying the proportion of items of type k at each child node by the proportions for each other type $i \neq k$ and summing. These values are then weighted by the number of items at that node to calculate an overall measure that we want to minimise. Using the identity in part (i)(a) leads to the following equivalent expression that we want to minimise:

$$\sum_{c=1}^{2} n_c \left[1 - \sum_{k=1}^{K} p_{ck}^2 \right]$$

(ii)(a) ***Calculate the Gini index for the initial split***

First tree

In Tree 1, the top final node contains AAAB, *ie* 3 A's and 1 B. This is the first child node of the initial split. For this child node, the proportions are $p_A = \frac{3}{4}$ and $p_B = \frac{1}{4}$. So the Gini impurity score for this node is:

$$G_1 = 1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2 = \frac{3}{8}$$

The second child node of the initial split contains all the remaining data points. The proportions are $p_B = \frac{4}{11}$, $p_C = \frac{3}{11}$ and $p_D = \frac{4}{11}$. So the Gini impurity score for this node is:

$$G_2 = 1 - \left(\frac{4}{11}\right)^2 - \left(\frac{3}{11}\right)^2 - \left(\frac{4}{11}\right)^2 = \frac{80}{121}$$

The overall Gini index after this first split then the weighted sum of these scores:

$$G = 4 \times \frac{3}{8} + 11 \times \frac{80}{121} = \frac{193}{22} = 8.772$$

Second tree

In Tree 2, the top final node contains BBBBBD, *ie* 5 B's and 1 D. This is the first child node of the initial split. For this child node, the proportions are $p_B = \frac{5}{6}$ and $p_D = \frac{1}{6}$.

So the Gini impurity score for this node is:

$$G_1 = 1 - \left(\frac{5}{6}\right)^2 - \left(\frac{1}{6}\right)^2 = \frac{5}{18}$$

The second child node of the first split contains all the remaining data points. The proportions are $p_A = \frac{3}{9} = \frac{1}{3}$, $p_C = \frac{1}{3}$ and $p_D = \frac{1}{3}$. So the Gini impurity score for this node is:

$$G_2 = 1 - \left(\frac{1}{3}\right)^2 - \left(\frac{1}{3}\right)^2 - \left(\frac{1}{3}\right)^2 = \frac{2}{3}$$

The overall Gini index after this initial split is then the weighted sum of these scores:

$$G = 6 \times \frac{5}{18} + 9 \times \frac{2}{3} = \frac{23}{3} = 7.667$$

(ii)(b) ***Calculate the accuracy for each tree***

Assuming that the assigned labels for each node are given by the majority type within them, the predicted values for the first tree, going in order down the tree, are A, C, D and B. There are 14 out of 15 correctly predicted types. So the accuracy is:

$$\frac{14}{15} = 0.933$$

The predicted values for the second tree, going in order down the tree, are B, A, C and D. There are 13 out of 15 correctly predicted types. So the accuracy is:

$$\frac{13}{15} = 0.867$$

(ii)(c) ***Comment***

The Gini index for the initial split is lower for the second tree than it is for the first. This means that, when taking a greedy approach to tree construction, the initial split from the second tree would be selected over the initial split from the first tree.

However, the first tree has a higher overall prediction accuracy than the second. So, this tree is likely to be preferred overall.

This illustrates how the greedy approach doesn't necessarily produce the best overall tree. Here, the initial split in the second tree has a lower Gini index for the initial split because it separates out 5 B's and 1 D compared to the 3 A's and 1 B in the first tree. However, after the initial split, the subsequent subtree of the first tree outperforms that of the second, perfectly classifying the remaining items.

21.10 (i) ***Describing the clusters***

Customers in Cluster 1 made more purchases than the average customer, but overall spent less than average. Assuming the purchases were spread across the year, these customers appear to be using the online retailer regularly to make mainly small to medium-sized purchases.

Customers in Cluster 2 made lots of purchases and spent a lot of money. They appear to also be regularly purchasing the cheaper to mid-range items, but more frequently than customers in Cluster 1.

Customers in Cluster 3 spent a lot of money on a few items. So, they don't make purchases often but when they do, they are buying the very expensive items.

Customers in Cluster 4 appear to have made very few mid-range to expensive purchases. They may not be regular visitors to the retailer or could be newer customers.

Customers in Cluster 5 spent a similar amount to customers in Cluster 1 but over fewer purchases. They appear to have made a few more expensive purchases over the year.

(ii) ***Marketing to Cluster 3***

From part (i), customers in Cluster 3 are those that spend a lot of money on a few purchases, suggesting that they are more interested in the expensive items. The retailer could email these customers about their top-end products only, rather than the entire product range.

End of Part 5

What next?

1. Briefly **review** the key areas of Part 5 and/or re-read the **summaries** at the end of Chapters 17 to 21.
2. Ensure you have attempted some of the **Practice Questions** at the end of each chapter in Part 5. If you don't have time to do them all, you could save the remainder for use as part of your revision.
3. Attempt **Assignment X5**.
4. **Read** the Chapter 17 to 21 material (copulas, reinsurance, risk models and machine learning) of the **Paper B Online Resources (PBOR).**
5. Attempt **Assignment Y2.**

Time to consider …

… 'rehearsal' products

Mock Exam and marking – You can attempt the Mock Exam and get it marked using *Mock Exam Marking* or more flexible *Marking Vouchers*.

Additional Mock Pack (AMP) and Marking Vouchers – The AMP includes two additional mock exam papers that you can attempt and get marked using *Marking Vouchers*.

Results of surveys suggest that attempting the mock exams and having them marked improves your chances of passing the exam. Students have said:

> *'I find the mock a useful tool in completing my pre-exam study. It helps me realise the areas I am weaker in and where I need to focus my study.'*

> *'Overall the marking was extremely useful and gave detailed comments on where I was losing marks and how to improve on my answers and exam technique. This is exactly what I was looking for – thank you!'*

You can find lots more information on our website at www.ActEd.co.uk.

Buy online at www.ActEd.co.uk/estore

And finally ...

Good luck!