A Little Book of R For Bayesian Statistics

Release 0.1

Avril Coghlan

April 22, 2013

CONTENTS

By Avril Coghlan, Wellcome Trust Sanger Institute, Cambridge, U.K. Email: alc@sanger.ac.uk

This is a simple introduction to Bayesian statistics using the R statistics software.

There is a pdf version of this booklet available at: https://github.com/avrilcoghlan/LittleBookofRBayesianStatistics/raw/master/_buil

If you like this booklet, you may also like to check out my booklets on using R for biomedical statistics, http://a-little-book-of-r-for-biomedical-statistics.readthedocs.org/, using R for time series analysis, http://a-little-book-of-r-for-time-series.readthedocs.org/, and using R for multivariate analysis, http://little-book-of-r-for-multivariate-analysis.readthedocs.org/.

Contents:

CHAPTER

ONE

HOW TO INSTALL R

1.1 Introduction to R

This little booklet has some information on how to use R for time series analysis.

R (www.r-project.org) is a commonly used free Statistics software. R allows you to carry out statistical analyses in an interactive mode, as well as allowing simple programming.

1.2 Installing R

To use R, you first need to install the R program on your computer.

1.2.1 How to check if R is installed on a Windows PC

Before you install R on your computer, the first thing to do is to check whether R is already installed on your computer (for example, by a previous user).

These instructions will focus on installing R on a Windows PC. However, I will also briefly mention how to install R on a Macintosh or Linux computer (see below).

If you are using a Windows PC, there are two ways you can check whether R is already isntalled on your computer:

1. Check if there is an "R" icon on the desktop of the computer that you are using. If so, double-click on the "R" icon to start R. If you cannot find an "R" icon, try step 2 instead.
2. Click on the "Start" menu at the bottom left of your Windows desktop, and then move your mouse over "All Programs" in the menu that pops up. See if "R" appears in the list of programs that pops up. If it does, it means that R is already installed on your computer, and you can start R by selecting "R" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.10.0) from the list.

If either (1) or (2) above does succeed in starting R, it means that R is already installed on the computer that you are using. (If neither succeeds, R is not installed yet). If there is an old version of R installed on the Windows PC that you are using, it is worth installing the latest version of R, to make sure that you have all the latest R functions available to you to use.

1.2.2 Finding out what is the latest version of R

To find out what is the latest version of R, you can look at the CRAN (Comprehensive R Network) website, http://cran.r-project.org/.

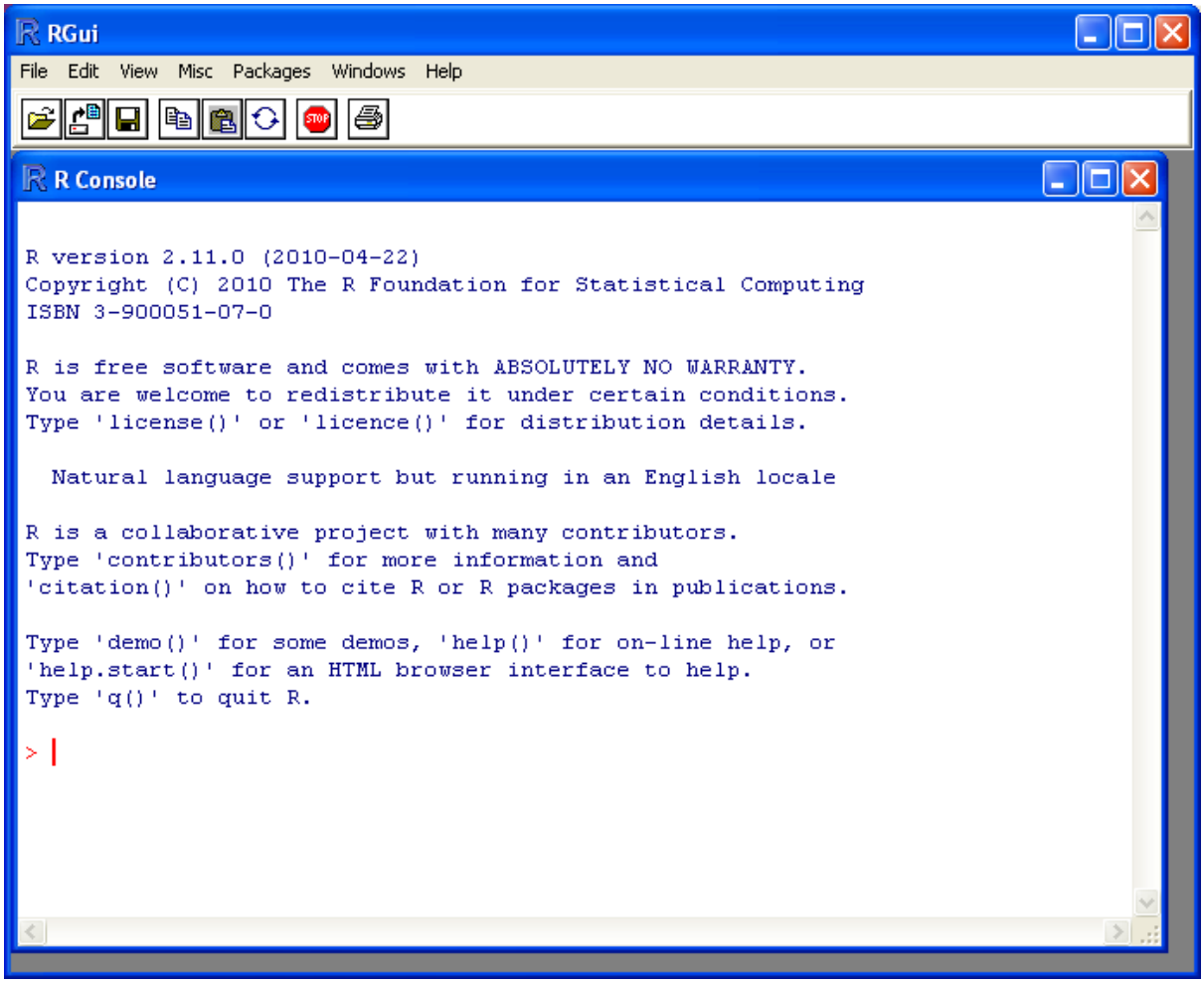
Beside "The latest release" (about half way down the page), it will say something like "R-X.X.X.tar.gz" (eg. "R-2.12.1.tar.gz"). This means that the latest release of R is X.X.X (for example, 2.12.1).

New releases of R are made very regularly (approximately once a month), as R is actively being improved all the time. It is worthwhile installing new versions of R regularly, to make sure that you have a recent version of R (to ensure compatibility with all the latest versions of the R packages that you have downloaded).

1.2.3 Installing R on a Windows PC

To install R on your Windows computer, follow these steps:

1. Go to http://ftp.heanet.ie/mirrors/cran.r-project.org.
2. Under "Download and Install R", click on the "Windows" link.
3. Under "Subdirectories", click on the "base" link.
4. On the next page, you should see a link saying something like "Download R 2.10.1 for Windows" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.11.1). Click on this link.
5. You may be asked if you want to save or run a file "R-2.10.1-win32.exe". Choose "Save" and save the file on the Desktop. Then double-click on the icon for the file to run it.
6. You will be asked what language to install it in - choose English.
7. The R Setup Wizard will appear in a window. Click "Next" at the bottom of the R Setup wizard window.
8. The next page says "Information" at the top. Click "Next" again.
9. The next page says "Information" at the top. Click "Next" again.
10. The next page says "Select Destination Location" at the top. By default, it will suggest to install R in "C:\Program Files" on your computer.
11. Click "Next" at the bottom of the R Setup wizard window.
12. The next page says "Select components" at the top. Click "Next" again.
13. The next page says "Startup options" at the top. Click "Next" again.
14. The next page says "Select start menu folder" at the top. Click "Next" again.
15. The next page says "Select additional tasks" at the top. Click "Next" again.
16. R should now be installed. This will take about a minute. When R has finished, you will see "Completing the R for Windows Setup Wizard" appear. Click "Finish".
17. To start R, you can either follow step 18, or 19:
18. Check if there is an "R" icon on the desktop of the computer that you are using. If so, double-click on the "R" icon to start R. If you cannot find an "R" icon, try step 19 instead.
19. Click on the "Start" button at the bottom left of your computer screen, and then choose "All programs", and start R by selecting "R" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.10.0) from the menu of programs.
20. The R console (a rectangle) should pop up:



1.2.4 How to install R on non-Windows computers (eg. Macintosh or Linux computers)

The instructions above are for installing R on a Windows PC. If you want to install R on a computer that has a non-Windows operating system (for example, a Macintosh or computer running Linux, you should download the appropriate R installer for that operating system at http://ftp.heanet.ie/mirrors/cran.r-project.org and follow the R installation instructions for the appropriate operating system at http://ftp.heanet.ie/mirrors/cran.r-project.org/doc/FAQ/R-FAQ.html#How-can-R-be-installed_003f).

1.3 Installing R packages

R comes with some standard packages that are installed when you install R. However, in this booklet I will also tell you how to use some additional R packages that are useful, for example, the "rmeta" package. These additional packages do not come with the standard installation of R, so you need to install them yourself.

1.3.1 How to install an R package

Once you have installed R on a Windows computer (following the steps above), you can install an additional package by following the steps below:

1. To start R, follow either step 2 or 3:
2. Check if there is an "R" icon on the desktop of the computer that you are using. If so, double-click on the "R" icon to start R. If you cannot find an "R" icon, try step 3 instead.

3. Click on the "Start" button at the bottom left of your computer screen, and then choose "All programs", and start R by selecting "R" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.10.0) from the menu of programs.
4. The R console (a rectangle) should pop up.
5. Once you have started R, you can now install an R package (eg. the "rmeta" package) by choosing "Install package(s)" from the "Packages" menu at the top of the R console. This will ask you what website you want to download the package from, you should choose "Ireland" (or another country, if you prefer). It will also bring up a list of available packages that you can install, and you should choose the package that you want to install from that list (eg. "rmeta").
6. This will install the "rmeta" package.
7. The "rmeta" package is now installed. Whenever you want to use the "rmeta" package after this, after starting R, you first have to load the package by typing into the R console:

```
> library("rmeta")
```

Note that there are some additional R packages for bioinformatics that are part of a special set of R packages called Bioconductor (www.bioconductor.org) such as the "yeastExpData" R package, the "Biostrings" R package, etc.). These Bioconductor packages need to be installed using a different, Bioconductor-specific procedure (see How to install a Bioconductor R package below).

1.3.2 How to install a Bioconductor R package

The procedure above can be used to install the majority of R packages. However, the Bioconductor set of bioinformatics R packages need to be installed by a special procedure. Bioconductor (www.bioconductor.org) is a group of R packages that have been developed for bioinformatics. This includes R packages such as "yeastExpData", "Biostrings", etc.

To install the Bioconductor packages, follow these steps:

1. To start R, follow either step 2 or 3:
2. Check if there is an "R" icon on the desktop of the computer that you are using. If so, double-click on the "R" icon to start R. If you cannot find an "R" icon, try step 3 instead.
3. Click on the "Start" button at the bottom left of your computer screen, and then choose "All programs", and start R by selecting "R" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.10.0) from the menu of programs.
4. The R console (a rectangle) should pop up.
5. Once you have started R, now type in the R console:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite()
```

6. This will install a core set of Bioconductor packages ("affy", "affydata", "affyPLM", "annaffy", "annotate", "Biobase", "Biostrings", "DynDoc", "gcrma", "genefilter", "geneplotter", "hgu95av2.db", "limma", "marray", "matchprobes", "multtest", "ROC", "vsn", "xtable", "affyQCReport"). This takes a few minutes (eg. 10 minutes).
7. At a later date, you may wish to install some extra Bioconductor packages that do not belong to the core set of Bioconductor packages. For example, to install the Bioconductor package called "yeastExpData", start R and type in the R console:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("yeastExpData")
```

8. Whenever you want to use a package after installing it, you need to load it into R by typing:

```
> library("yeastExpData")
```

1.4 Running R

To use R, you first need to start the R program on your computer. You should have already installed R on your computer (see above).

To start R, you can either follow step 1 or 2: 1. Check if there is an "R" icon on the desktop of the computer that you are using.

If so, double-click on the "R" icon to start R. If you cannot find an "R" icon, try step 2 instead.

2. Click on the "Start" button at the bottom left of your computer screen, and then choose "All programs", and start R by selecting "R" (or R X.X.X, where X.X.X gives the version of R, eg. R 2.10.0) from the menu of programs.

This should bring up a new window, which is the *R console*.

1.5 A brief introduction to R

You will type R commands into the R console in order to carry out analyses in R. In the R console you will see:

```
>
```

This is the R prompt. We type the commands needed for a particular task after this prompt. The command is carried out after you hit the Return key.

Once you have started R, you can start typing in commands, and the results will be calculated immediately, for example:

```
> 2*3
[1] 6
> 10-3
[1] 7
```

All variables (scalars, vectors, matrices, etc.) created by R are called *objects*. In R, we assign values to variables using an arrow. For example, we can assign the value 2*3 to the variable *x* using the command:

```
> x <- 2*3
```

To view the contents of any R object, just type its name, and the contents of that R object will be displayed:

```
> x
[1] 6
```

There are several possible different types of objects in R, including scalars, vectors, matrices, arrays, data frames, tables, and lists. The scalar variable *x* above is one example of an R object. While a scalar variable such as *x* has just one element, a vector consists of several elements. The elements in a vector are all of the same type (eg. numeric or characters), while lists may include elements such as characters as well as numeric quantities.

To create a vector, we can use the c() (combine) function. For example, to create a vector called *myvector* that has elements with values 8, 6, 9, 10, and 5, we type:

```
> myvector <- c(8, 6, 9, 10, 5)
```

To see the contents of the variable *myvector*, we can just type its name:

```
> myvector
[1]  8  6  9 10  5
```

The [1] is the index of the first element in the vector. We can extract any element of the vector by typing the vector name with the index of that element given in square brackets. For example, to get the value of the 4th element in the vector *myvector*, we type:

```
> myvector[4]
[1] 10
```

In contrast to a vector, a list can contain elements of different types, for example, both numeric and character elements. A list can also include other variables such as a vector. The list() function is used to create a list. For example, we could create a list *mylist* by typing:

```
> mylist <- list(name="Fred", wife="Mary", myvector)
```

We can then print out the contents of the list *mylist* by typing its name:

```
> mylist
$name
[1] "Fred"

$wife
[1] "Mary"

[[3]]
[1]  8  6  9 10  5
```

The elements in a list are numbered, and can be referred to using indices. We can extract an element of a list by typing the list name with the index of the element given in double square brackets (in contrast to a vector, where we only use single square brackets). Thus, we can extract the second and third elements from *mylist* by typing:

```
> mylist[[2]]
[1] "Mary"
> mylist[[3]]
[1]  8  6  9 10  5
```

Elements of lists may also be named, and in this case the elements may be referred to by giving the list name, followed by "$", followed by the element name. For example, *mylist$name* is the same as *mylist[[1]]* and *mylist$wife* is the same as *mylist[[2]]*:

```
> mylist$wife
[1] "Mary"
```

We can find out the names of the named elements in a list by using the attributes() function, for example:

```
> attributes(mylist)
$names
[1] "name" "wife" ""
```

When you use the attributes() function to find the named elements of a list variable, the named elements are always listed under a heading "$names". Therefore, we see that the named elements of the list variable *mylist* are called "name" and "wife", and we can retrieve their values by typing *mylist$name* and *mylist$wife*, respectively.

Another type of object that you will encounter in R is a *table* variable. For example, if we made a vector variable *mynames* containing the names of children in a class, we can use the table() function to produce a table variable that contains the number of children with each possible name:

```
> mynames <- c("Mary", "John", "Ann", "Sinead", "Joe", "Mary", "Jim", "John", "Simon")
> table(mynames)
mynames
   Ann    Jim    Joe   John   Mary  Simon Sinead
     1      1      1      2      2      1      1
```

We can store the table variable produced by the function table(), and call the stored table "mytable", by typing:

```
> mytable <- table(mynames)
```

To access elements in a table variable, you need to use double square brackets, just like accessing elements in a list. For example, to access the fourth element in the table *mytable* (the number of children called "John"), we type:

```
> mytable[[4]]
[1] 2
```

Alternatively, you can use the name of the fourth element in the table ("John") to find the value of that table element:

```
> mytable[["John"]]
[1] 2
```

Functions in R usually require *arguments*, which are input variables (ie. objects) that are passed to them, which they then carry out some operation on. For example, the log10() function is passed a number, and it then calculates the log to the base 10 of that number:

```
> log10(100)
2
```

In R, you can get help about a particular function by using the help() function. For example, if you want help about the log10() function, you can type:

```
> help("log10")
```

When you use the help() function, a box or webpage will pop up with information about the function that you asked for help with.

If you are not sure of the name of a function, but think you know part of its name, you can search for the function name using the help.search() and RSiteSearch() functions. The help.search() function searches to see if you already have a function installed (from one of the R packages that you have installed) that may be related to some topic you're interested in. The RSiteSearch() function searches all R functions (including those in packages that you haven't yet installed) for functions related to the topic you are interested in.

For example, if you want to know if there is a function to calculate the standard deviation of a set of numbers, you can search for the names of all installed functions containing the word "deviation" in their description by typing:

```
> help.search("deviation")
Help files with alias or concept or title matching
'deviation' using fuzzy matching:

genefilter::rowSds
                    Row variance and standard deviation of
                    a numeric array
nlme::pooledSD      Extract Pooled Standard Deviation
stats::mad          Median Absolute Deviation
stats::sd           Standard Deviation
vsn::meanSdPlot     Plot row standard deviations versus row
```

Among the functions that were found, is the function sd() in the "stats" package (an R package that comes with the standard R installation), which is used for calculating the standard deviation.

In the example above, the help.search() function found a relevant function (sd() here). However, if you did not find what you were looking for with help.search(), you could then use the RSiteSearch() function to see if a search of all functions described on the R website may find something relevant to the topic that you're interested in:

```
> RSiteSearch("deviation")
```

The results of the RSiteSearch() function will be hits to descriptions of R functions, as well as to R mailing list discussions of those functions.

We can perform computations with R using objects such as scalars and vectors. For example, to calculate the average of the values in the vector *myvector* (ie. the average of 8, 6, 9, 10 and 5), we can use the mean() function:

```
> mean(myvector)
[1] 7.6
```

We have been using built-in R functions such as mean(), length(), print(), plot(), etc. We can also create our own functions in R to do calculations that you want to carry out very often on different input data sets. For example, we can create a function to calculate the value of 20 plus square of some input number:

```
> myfunction <- function(x) { return(20 + (x*x)) }
```

This function will calculate the square of a number (x), and then add 20 to that value. The return() statement returns the calculated value. Once you have typed in this function, the function is then available for use. For example, we can use the function for different input numbers (eg. 10, 25):

```
> myfunction(10)
[1] 120
> myfunction(25)
[1] 645
```

To quit R, type:

```
> q()
```

1.6 Links and Further Reading

Some links are included here for further reading.

For a more in-depth introduction to R, a good online tutorial is available on the "Kickstarting R" website, cran.r-project.org/doc/contrib/Lemon-kickstart.

There is another nice (slightly more in-depth) tutorial to R available on the "Introduction to R" website, cran.r-project.org/doc/manuals/R-intro.html.

1.7 Acknowledgements

For very helpful comments and suggestions for improvements on the installation instructions, thank you very much to Friedrich Leisch and Phil Spector.

1.8 Contact

I will be very grateful if you will send me (Avril Coghlan) corrections or suggestions for improvements to my email address alc@sanger.ac.uk

1.9 License

CHAPTER

TWO

USING R FOR BAYESIAN STATISTICS

2.1 Bayesian Statistics

This booklet tells you how to use the R statistical software to carry out some simple analyses using Bayesian statistics.

This booklet assumes that the reader has some basic knowledge of Bayesian statistics, and the principal focus of the booklet is not to explain Bayesian statistics, but rather to explain how to carry out these analyses using R.

If you are new to Bayesian statistics, and want to learn more about any of the concepts presented here, I would highly recommend the Open University book "Bayesian Statistics" (product code M249/04), available from from the Open University Shop.

There is a pdf version of this booklet available at https://github.com/avrilcoghlan/LittleBookofRBayesianStatistics/raw/master/_build/latex/BayesianStats.pdf.

If you like this booklet, you may also like to check out my booklets on using R for biomedical statistics, http://a-little-book-of-r-for-biomedical-statistics.readthedocs.org/, using R for time series analysis, http://a-little-book-of-r-for-time-series.readthedocs.org/, and using R for multivariate analysis, http://little-book-of-r-for-multivariate-analysis.readthedocs.org/.

2.2 Using Bayesian Analysis to Estimate a Proportion

Bayesian analysis can be useful for estimating a proportion, when you have some rough idea of what the value of the proportion is, but have relatively little data.

2.2.1 Specifying a Prior for a Proportion

An appropriate prior to use for a proportion is a Beta prior.

For example, if you want to estimate the proportion of people like chocolate, you might have a rough idea that the most likely value is around 0.85, but that the proportion is unlikely to be smaller than 0.60 or bigger than 0.95.

You can find the best Beta prior to use in this case by specifying that the median (50% percentile) of the prior is 0.85, that the 99.999% percentile is 0.95, and that the 0.001% percentile is 0.60:

```
> quantile1 <- list(p=0.5, x=0.85)     # we believe the median of the prior is 0.85
> quantile2 <- list(p=0.99999,x=0.95) # we believe the 99.999th percentile of the prior is 0.95
> quantile3 <- list(p=0.00001,x=0.60) # we believe the 0.001st percentile of the prior is 0.60
```

We can then use the findBeta() function below to find the most appropriate Beta prior to use.

```
> findBeta <- function(quantile1,quantile2,quantile3)
  {
     # find the quantiles specified by quantile1 and quantile2 and quantile3
     quantile1_p <- quantile1[[1]]; quantile1_q <- quantile1[[2]]
```

```
    quantile2_p <- quantile2[[1]]; quantile2_q <- quantile2[[2]]
    quantile3_p <- quantile3[[1]]; quantile3_q <- quantile3[[2]]

    # find the beta prior using quantile1 and quantile2
    priorA <- beta.select(quantile1,quantile2)
    priorA_a <- priorA[1]; priorA_b <- priorA[2]

    # find the beta prior using quantile1 and quantile3
    priorB <- beta.select(quantile1,quantile3)
    priorB_a <- priorB[1]; priorB_b <- priorB[2]

    # find the best possible beta prior
    diff_a <- abs(priorA_a - priorB_a); diff_b <- abs(priorB_b - priorB_b)
    step_a <- diff_a / 100; step_b <- diff_b / 100
    if (priorA_a < priorB_a) { start_a <- priorA_a; end_a <- priorB_a }
    else                     { start_a <- priorB_a; end_a <- priorA_a }
    if (priorA_b < priorB_b) { start_b <- priorA_b; end_b <- priorB_b }
    else                     { start_b <- priorB_b; end_b <- priorA_b }
    steps_a <- seq(from=start_a, to=end_a, length.out=1000)
    steps_b <- seq(from=start_b, to=end_b, length.out=1000)
    max_error <- 10000000000000000000
    best_a <- 0; best_b <- 0
    for (a in steps_a)
    {
       for (b in steps_b)
       {
          # priorC is beta(a,b)
          # find the quantile1_q, quantile2_q, quantile3_q quantiles of priorC:
          priorC_q1 <- qbeta(c(quantile1_p), a, b)
          priorC_q2 <- qbeta(c(quantile2_p), a, b)
          priorC_q3 <- qbeta(c(quantile3_p), a, b)
          priorC_error <- abs(priorC_q1-quantile1_q) +
                          abs(priorC_q2-quantile2_q) +
                          abs(priorC_q3-quantile3_q)
          if (priorC_error < max_error)
          {
            max_error <- priorC_error; best_a <- a; best_b <- b
          }
       }
    }
    print(paste("The best beta prior has a=",best_a,"b=",best_b))
}
```

To use the findBeta() function, you first need to copy and paste it into R. The findBeta() function makes use of the beta.select() function from the LearnBayes R package, so you first need to install the LearnBayes package (for instructions on how to install an R package, see How to install an R package).

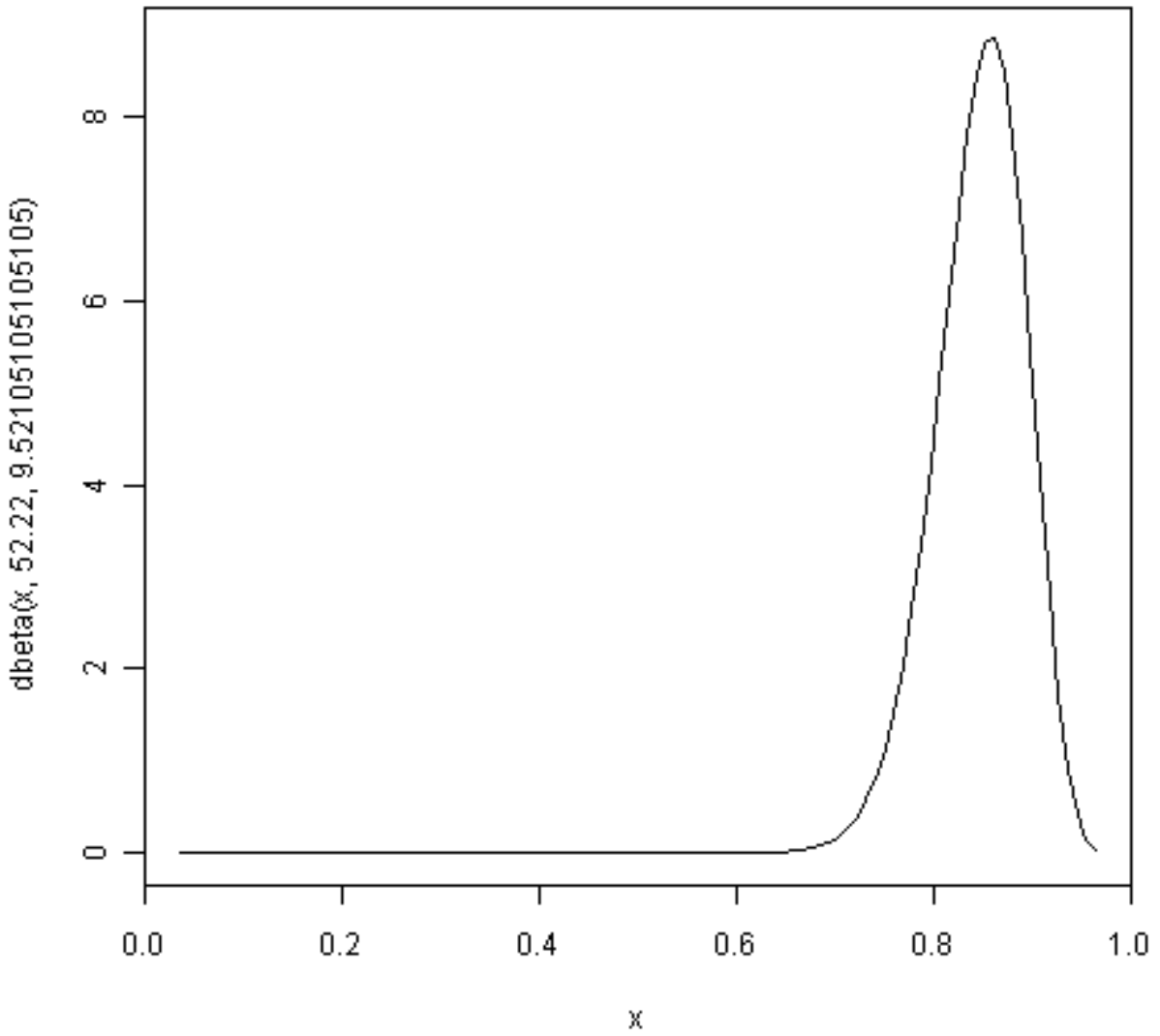
You can then load the LearnBayes package, and use findBeta() to find the best Beta prior for a proportion. For example, to find the best Beta prior for the proportion of individuals who like chocolate, where you believe the most likely value of the proportion is 0.85, and the value is almost definitely between 0.60 and 0.95, you can type:

```
> library("LearnBayes")
> findBeta(quantile1,quantile2,quantile3)
  [1] "The best beta prior has a= 52.22 b= 9.52105105105105"
```

This tells us that the most appropriate prior to use for the proportion of individuals who like chocolate is a Beta prior with a=52.22 and b=9.52, that is, a Beta(52.22, 9.52) prior.

We can plot the prior density by using the "curve" function:

```
> curve(dbeta(x,52.22,9.52105105105105)) # plot the prior
```



Note that in the command above we use the "dbeta()" function to specify that the density of a Beta(52.22,9.52105105105105) distribution.

We can see from the picture of the density for a Beta(52.22,9.52105105105105) distribution that it represents our prior beliefs about the proportion of people who like chocolate fairly well, as the peak of the distribution is at about 0.85, and the density lies almost entirely between about 0.68 and 0.97.

2.2.2 Calculating the Likelihood Function for a Proportion

Say you want to estimate a proportion, and you have a small data set that you can use for this purpose. For example, if you want to estimate the proportion of people who like chocolate, you may have carried out a survey of 50 people, and found that 45 say that they like chocolate.

This small data set can be used to calculate the conditional p.m.f. (probability mass function) of the proportion given the observed data. This is called the likelihood function. It represents how likely the possible values of the proportion are, given the observed data.

If you want to estimate a proportion, and have a small data set, you can calculate the likelihood function for the proportion using the function calcLikelihoodForProportion() below:

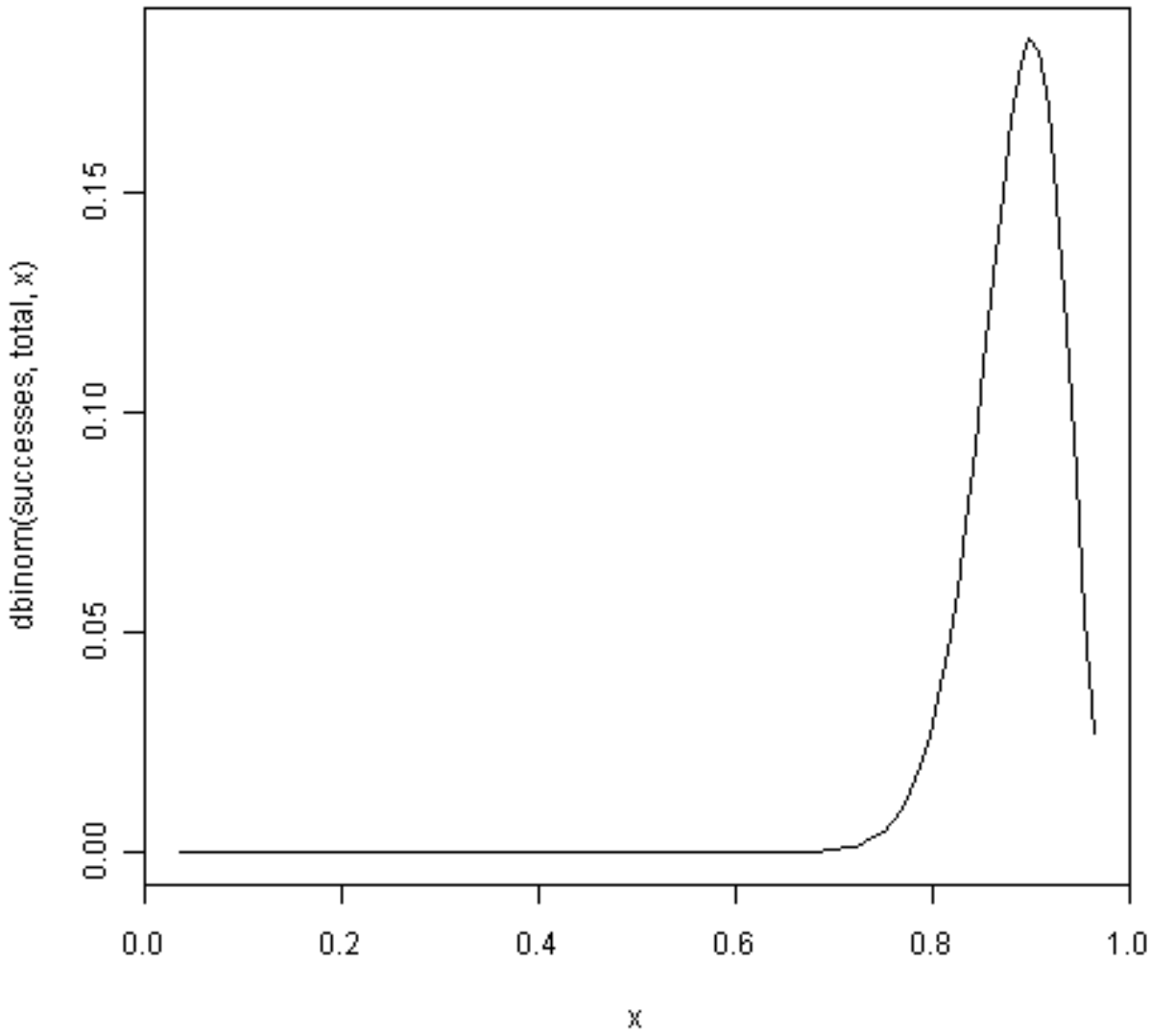
```
> calcLikelihoodForProportion <- function(successes, total)
  {
     curve(dbinom(successes,total,x)) # plot the likelihood
  }
```

The function calcLikelihoodForProportion() takes two input arguments: the number of successes observed in the sample (eg. the number of people who like chocolate in the sample), and the total sample size.

You can see that the likelihood function is being calculated using the Binomial distribution (using the R "dbinom()" function). That is, the likelihood function is the probability mass function of a B(total,successes) distribution, that is, of a Binomial distribution where the we observe "successes" successes out of a sample of "total" observations in total.

For example, if we did a survey of 50 people, and found that 45 say they like chocolate, then our total sample size is 50 and we have 45 "successes". We can calculate the likelihood function for the proportion of people who like chocolate by typing:

```
> calcLikelihoodForProportion(45, 50)
```



You can see that the peak of the likelihood distribution is at 0.9, which is equal to the sample mean (45/50 = 0.9). In other words, the most likely value of the proportion, given the observed data, is 0.9.

2.2.3 Calculating the Posterior Distribution for a Proportion

Say you are trying to estimate a proportion, and have a prior distribution representing your beliefs about the value of that proportion. If you have collected some data, you can also calculate the likelihood function for the proportion given the data.

However, after observing the data, you may wish to update the prior distribution for the proportion, taking the data into consideration. That is, you may wish to calculate the conditional distribution of the proportion given the data and the prior. This is is called the posterior distribution for the proportion.

The posterior distribution ssummarises what is known about the proportion after the data has been observed, and combines the information from the prior and the data.

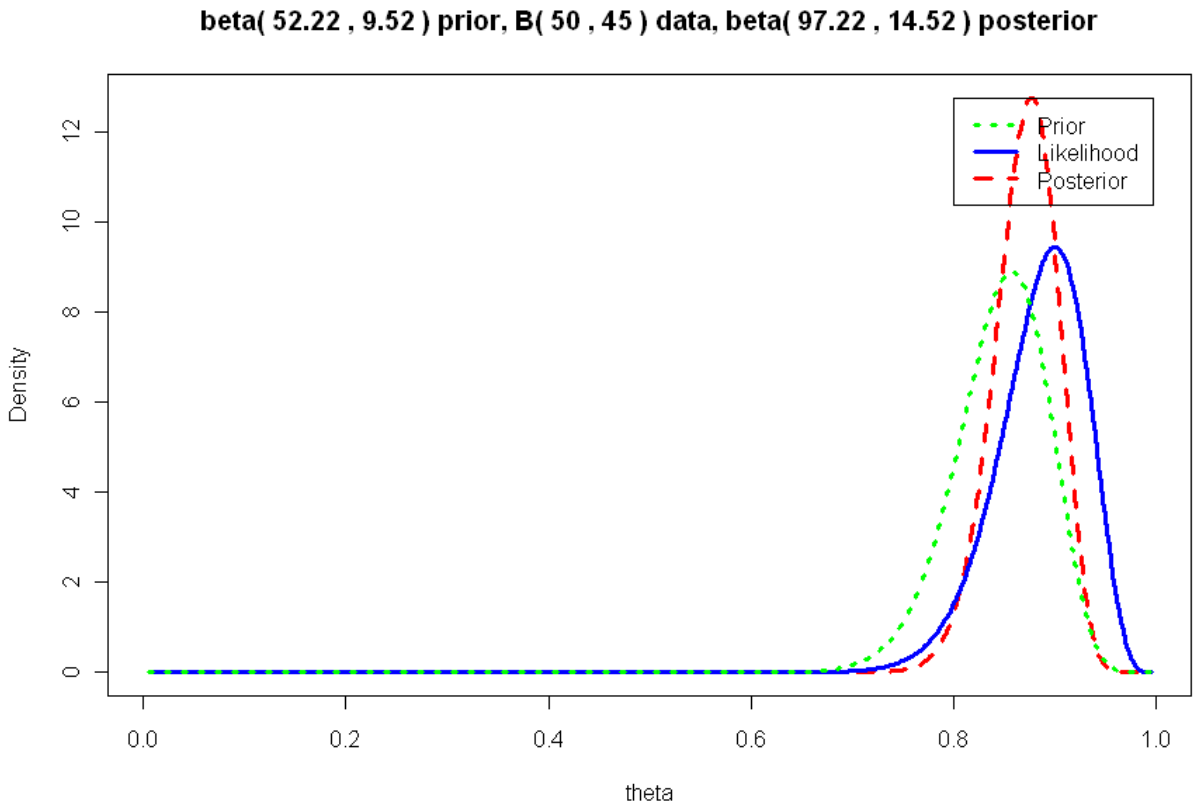
In our example of estimating the proportion of people who like chocolate, we have a Beta(52.22,9.52) prior distribution (see above), and have some data from a survey in which we found that 45 out of 50 people like chocolate. We can calculate the posterior distribution for the proportion given the prior and data using the calcPosteriorForProportion() function below (which I adapted from "triplot" in the LearnBayes package):

```
> calcPosteriorForProportion <- function(successes, total, a, b)
  {
     # Adapted from triplot() in the LearnBayes package
     # Plot the prior, likelihood and posterior:
     likelihood_a = successes + 1; likelihood_b = total - successes + 1
     posterior_a = a + successes;  posterior_b = b + total - successes
     theta = seq(0.005, 0.995, length = 500)
     prior = dbeta(theta, a, b)
     likelihood = dbeta(theta, likelihood_a, likelihood_b)
     posterior  = dbeta(theta, posterior_a, posterior_b)
     m = max(c(prior, likelihood, posterior))
     plot(theta, posterior, type = "l", ylab = "Density", lty = 2, lwd = 3,
          main = paste("beta(", a, ",", b, ") prior, B(", total, ",", successes, ") data,",
          "beta(", posterior_a, ",", posterior_b, ") posterior"), ylim = c(0, m), col = "red")
     lines(theta, likelihood, lty = 1, lwd = 3, col = "blue")
     lines(theta, prior, lty = 3, lwd = 3, col = "green")
     legend(x=0.8,y=m, c("Prior", "Likelihood", "Posterior"), lty = c(3, 1, 2),
          lwd = c(3, 3, 3), col = c("green", "blue", "red"))
     # Print out summary statistics for the prior, likelihood and posterior:
     calcBetaMode <- function(aa, bb) { BetaMode <- (aa - 1)/(aa + bb - 2); return(BetaMode); }
     calcBetaMean <- function(aa, bb) { BetaMean <- (aa)/(aa + bb); return(BetaMean); }
     calcBetaSd   <- function(aa, bb) { BetaSd <- sqrt((aa * bb)/(((aa + bb)^2) * (aa + bb + 1)))
     prior_mode      <- calcBetaMode(a, b)
     likelihood_mode <- calcBetaMode(likelihood_a, likelihood_b)
     posterior_mode  <- calcBetaMode(posterior_a, posterior_b)
     prior_mean      <- calcBetaMean(a, b)
     likelihood_mean <- calcBetaMean(likelihood_a, likelihood_b)
     posterior_mean  <- calcBetaMean(posterior_a, posterior_b)
     prior_sd        <- calcBetaSd(a, b)
     likelihood_sd   <- calcBetaSd(likelihood_a, likelihood_b)
     posterior_sd    <- calcBetaSd(posterior_a, posterior_b)
     print(paste("mode for prior=",prior_mode,", for likelihood=",likelihood_mode,", for posterio
     print(paste("mean for prior=",prior_mean,", for likelihood=",likelihood_mean,", for posterio
     print(paste("sd for prior=",prior_sd,", for likelihood=",likelihood_sd,", for posterior=",pos
  }
```

To use the "calcPosteriorForProportion()" function, you will first need to copy and paste it into R. It takes four arguments: the number of successes and total sample size in your data set, and the a and b values for your Beta prior.

For example, to estimate the proportion of people who like chocolate, you had a Beta(52.22,9.52) prior and had observed in a survey that 45 out of 50 people like chocolate. Therefore, the number of successes is 45, the sample size is 50, and a and b for the prior are 52.22 and 9.52 respectively. Therefore, we can calculate the posterior for the proportion of people who like chocolate, given the data and prior, by typing:

```
> calcPosteriorForProportion(45, 50, 52.22, 9.52)
  [1] "mode for prior= 0.857381988617342 , for likelihood= 0.9 , for posterior= 0.876799708401677
  [1] "mean for prior= 0.845804988662132 , for likelihood= 0.884615384615385 , for posterior= 0.87
  [1] "sd for prior= 0.0455929848904483 , for likelihood= 0.0438847130123102 , for posterior= 0.0
```



Since the prior and posterior are distributions, the area under their densities is 1. The likelihood has been scaled so that the area underneath it is also 1, so that it is easy to compare the likelihood with the prior and posterior.

Therefore, the prior and likelihood curves should look the same shape as those plotted before (see above), but the y-axis scale is different for the likelihood scale compared to the plot made using calcLikelihoodForProportion() above.

Note that the peak of the posterior always lies somewhere between the peaks of the prior and the likelihood, because it combines information from the prior and the likelihood (which is based on the data).

In our example of estimating the proportion of people who like chocolate, the peak of the posterior is roughly half-way between the peaks of the likelihood and prior, indicating that the prior and the data contribute roughly equally to the posterior.

2.3 Links and Further Reading

Here are some links for further reading.

For a more in-depth introduction to R, a good online tutorial is available on the "Kickstarting R" website, cran.r-project.org/doc/contrib/Lemon-kickstart.

There is another nice (slightly more in-depth) tutorial to R available on the "Introduction to R" website, cran.r-project.org/doc/manuals/R-intro.html.

To learn about Bayesian Statistics, I would highly recommend the book "Bayesian Statistics" (product code M249/04) by the Open University, available from the Open University Shop.

There is a book available in the "Use R!" series on using R for multivariate analyses, Bayesian Computation with R by Jim Albert.

2.4 Acknowledgements

Many of the examples in this booklet are inspired by examples in the excellent Open University book, "Bayesian Statistics" (product code M249/04), available from the Open University Shop.

2.5 Contact

I will be grateful if you will send me (Avril Coghlan) corrections or suggestions for improvements to my email address alc@sanger.ac.uk

2.6 License

CHAPTER

THREE

ACKNOWLEDGEMENTS

Thank you to Noel O'Boyle for helping in using Sphinx, http://sphinx.pocoo.org, to create this document, and github, https://github.com/, to store different versions of the document as I was writing it, and readthedocs, http://readthedocs.org/, to build and distribute this document.

CHAPTER

FOUR

CONTACT

I will be very grateful if you will send me (Avril Coghlan) corrections or suggestions for improvements to my email address alc@sanger.ac.uk

CHAPTER

FIVE

LICENSE