Statistical Computing Chapter_1

Henry Bourne

2022-10-17

Reproducibility

In science it is important to conduct **reproducible research**: where we are able to reproduce the analysis and conclusions of a particular investigation given the same raw data. In most science reproducible research is seen as very important and as a result most scientific research is done in such a way that it is reproducible. However, scientific research involving computation for the most part is not reproducible. The reason for this is probably that historically carrying out reproducible research with computers was very hard as it used to be very difficult to create code that works on many different computer systems. Another reason could be that computer research can often be very interactive making it hard to track the final set of steps that were used for the finished product. However, increasingly we see computational scientific research becoming more reproducible, for example now we often see code being supplied alongside the research.

To carry out reproducible computing the following is important:
1. Readable code.
2. Accessible data.
3. Ability for the code to run on any machine (Or a guide on what is needed in order to run the code).
4. Version control.

Some of the above may or may not be possible depending on the research. For example, if you are working with sensitive data you may not be able to make the data accessible and if you are working with unique hardware for which your code is optimized then it will be difficult to create a guide for how to implement the code and may not even be worthwhile. However, wherever possible we should strive to implement the above. Having readable code is very important for reproducibility (one must be able to decipher what the code is doing if they are to use it) and there are a number of ways one can go about this, we will explore how to do this shortly. Making your data accessible usually is done by uploading it to an online repository and often fields will have specific online repositories that they use. Making your code able to run on any machine often involves writing it in high-level languages and specifying system pre-requisites such as which packages are needed in order to run the code. Version control is usually implemented using the git version control software and an online repository such as Github.com, we will visit this in more depth in the next chapter. Version control is important as it allows someone to follow how the code was built and therefore deduce the steps that where taken to create the code. By fulfilling the above we get closer to truly reproducible computing.

Readable Code and Literate Programming

There are a number of ways to help facilitate the creation of easily understandable code. One such way of doing this is by **including comments** withing the code itself. By including comments within code we can in-situ give explanation to what we are trying to do with specific pieces of code. Including comments in your code allows you to give context to the human reader as to what the code you have written is instructing the computer to do, and clearly is very important when trying to write code that is easily parseable by the reader. Another step that can be taken to make code more easily useable is through creating **documentation**. Documenting our code is where we create a separate document (to our code) where we describe what it is that our code does. There also exist tools out there which help you to create good documentation such as "Doxygen" which lets you create documentation straight from your source code, the advantage of this being

that it is easier to keep your documentation up to date with your source code.

The last way we will discuss of creating readable reproducible code is **Literate Programming**: involves writing source code that can be processed to produce:
1. A document explaining what the program does.
2. A program that can be executed.
RMarkdown for example is a use of literate programming, so is Jupyter Notebook, also note that automated forms of documentation creation can also be considered literate programming (such as using Doxygen). The advantages of literate programming are:
- Code and documentation are consistent with each other (as both are in the same file).
- One can focus on readability of the output document.
- Can emphasize the thought process behind the code more clearly.
Specifically in R there are three main ways of doing this. One way is through writing an R script with specially delimited Markdown chunks that are ignored by R as comments as they are parsed, but can optionally be converted into a literate programming document using "knitr::spin". Another way would be to incorporate R code into LaTeX. Lastly we can also write in RMarkdown, in fact, this document and the rest of the documents for the statistical computing unit shall be written in RMarkdown. We will now give an example of Literate programming in action.

Literate Programming example

In this section we will investigate the "Prostate cancer dataset". We will produce a linear Least Squares (LS) solver to produce a model for the dataset that predicts the outcome given the predictor variables. We will then calculate the cross validation error of our model and then analyse the importance of each predictor variable in our model. From this analysis we find that the lcavol, lweight and svi predictor variables do not impact performance when removed from the model and therefore suggest that we can reduce the dimensionality of our problem by removing said predictor variables from our model.

Throughout this document we will make sure to keep the code reproducible and will be conducting literate programming in order to make the code easily readable and understandable. We also will not be using any special packages outside of the default packages needed for R such that this code can be ran on any computer. Except, do make sure that you have the formatR package installed so the comments in the chunks are wrapped when this file is knitted.

Finding the Least squares estimator

We begin by loading in the dataset directly from the URL so this can be run on any computer without first downloading the file. We also print out the first 6 rows to check the structure of the data.

```
D <- read.table(url("https://hastie.su.domains/ElemStatLearn/datasets/prostate.data"))
head(D)
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
##   train
## 1  TRUE
## 2  TRUE
## 3  TRUE
## 4  TRUE
## 5  TRUE
## 6  TRUE
```

We extract the predictor variable y from the dataset and print it to check it has the correct structure.

```
y <- D$lpsa
y
```

```
##  [1] -0.4307829 -0.1625189 -0.1625189 -0.1625189  0.3715636  0.7654678
##  [7]  0.7654678  0.8544153  1.0473190  1.0473190  1.2669476  1.2669476
## [13]  1.2669476  1.3480731  1.3987169  1.4469190  1.4701758  1.4929041
## [19]  1.5581446  1.5993876  1.6389967  1.6582281  1.6956156  1.7137979
## [25]  1.7316555  1.7664417  1.8000583  1.8164521  1.8484548  1.8946169
## [31]  1.9242487  2.0082140  2.0082140  2.0215476  2.0476928  2.0856721
## [37]  2.1575593  2.1916535  2.2137539  2.2772673  2.2975726  2.3075726
## [43]  2.3272777  2.3749058  2.5217206  2.5533438  2.5687881  2.5687881
## [49]  2.5915164  2.5915164  2.6567569  2.6775910  2.6844403  2.6912431
## [55]  2.7047113  2.7180005  2.7880929  2.7942279  2.8063861  2.8124102
## [61]  2.8419982  2.8535925  2.8535925  2.8820035  2.8820035  2.8875901
## [67]  2.9204698  2.9626924  2.9626924  2.9729753  3.0130809  3.0373539
## [73]  3.0563569  3.0750055  3.2752562  3.3375474  3.3928291  3.4355988
## [79]  3.4578927  3.5130369  3.5160131  3.5307626  3.5652984  3.5709402
## [85]  3.5876769  3.6309855  3.6800909  3.7123518  3.9843437  3.9936030
## [91]  4.0298060  4.1295508  4.3851468  4.6844434  5.1431245  5.4775090
## [97]  5.5829322
```

We now create a function which given the dataset, D, will return the model matrix X.

```
mm <- function(D) {
    X <- as.matrix(D[1:(ncol(D) - 2)])
    ones <- rep(1, nrow(D))
    X <- cbind(ones, X)
    X
}
```

And we now use that function to obtain the model matrix, X, from the dataset, D.

```
X <- mm(D)
head(X)
```

```
##   ones     lcavol  lweight age      lbph svi       lcp gleason pgg45
## 1    1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0
## 2    1 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0
## 3    1 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20
## 4    1 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0
## 5    1  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0
## 6    1 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0
```

Now we have the model matrix, X, and predictor variable, y, we have everything we need to find the LS estimate. We now write a function that takes in the model matrix and target variable and returns the LS solution.

```
LLS <- function(X, y) {
    w <- solve(t(X) %*% X) %*% t(X) %*% y
    w
}
```

And we now call that function using our X and y to give us the LS estimator. We also print our solution.

```
w_LS <- LLS(X, y)
w_LS
```

```
##                 [,1]
## ones     0.181560845
## lcavol   0.564341280
## lweight  0.622019788
## age     -0.021248185
## lbph     0.096712522
## svi      0.761673402
## lcp     -0.106050939
## gleason  0.049227934
## pgg45    0.004457512
```

Cross Validation

We now are going to assess the performance of our estimator by calculating the cross-validation error.

We begin by randomly shuffling our data and denoting this shuffled data by D_dash

```
D_dash <- D[sample(nrow(D)), ]
head(D_dash)
```

```
##        lcavol  lweight age       lbph svi        lcp gleason pgg45     lpsa
## 22  2.0592388 3.501043  60  1.4747630   0  1.3480732       7    20 1.658228
## 41  0.6205765 3.141995  60 -1.3862944   0 -1.3862944       9    80 2.297573
## 92  2.5329028 3.677566  61  1.3480732   1 -1.3862944       7    15 4.129551
## 44  1.7715568 3.896909  61 -1.3862944   0  0.8109302       7     6 2.374906
## 12 -1.3470736 3.598681  63  1.2669476   0 -1.3862944       6     0 1.266948
## 26  1.4469190 3.124565  68  0.3001046   0 -1.3862944       6     0 1.766442
##    train
## 22 FALSE
## 41  TRUE
## 92  TRUE
## 44 FALSE
## 12  TRUE
## 26 FALSE
```

We now create a list which indexes the rows in our dataset, we will use this to select groups of certain rows from the dataset, hence what we have effectively done here is partition the dataset (randomly - as we shuffled the dataset previously) into groups. We split the dataset into 10 groups as we will be performing 10-fold cross validation.

```
subsets <- cut(seq(1, nrow(D)), breaks = 10, labels = FALSE)
subsets
```

```
##  [1]  1  1  1  1  1  1  1  1  1  1  2  2  2  2  2  2  2  2  2  2  3  3  3  3  3
## [26]  3  3  3  3  4  4  4  4  4  4  4  4  4  4  5  5  5  5  5  5  5  5  5  5  6
## [51]  6  6  6  6  6  6  6  6  7  7  7  7  7  7  7  7  7  7  8  8  8  8  8  8  8
## [76]  8  8  9  9  9  9  9  9  9  9  9  9 10 10 10 10 10 10 10 10 10 10
```

We now write a function that will carry out 10-fold cross validation given the dataset, D, and the predictor variable, y. Note that within this function we include comments to assist the reader in understanding what exactly is happening in this larger function.

```
cross_val <- function(D, y) {

    # We randomly shuffle the indices of the rows and using
    # these randomized indices shuffle the rows of the
    # dataset and the target variable (in the same order)
    ind <- sample(nrow(D))
```

```
    D_dash <- D[paste(ind), ]
    y_dash <- y[ind]

    # We now create a list which indexes the rows in our
    # dataset, we will use this to select groups of certain
    # rows from the dataset, hence what we have effectively
    # done here is partition the dataset (randomly - as we
    # shuffled the dataset previously) into groups. We
    # split the dataset into 10 groups as we will be
    # performing 10-fold cross validation.
    subsets <- cut(seq(1, nrow(D)), breaks = 10, labels = FALSE)

    # We create a vector to store our cross-val errors
    errors <- c()

    # Loop for carrying out 10-fold cross-val
    for (i in 1:10) {

        # We segment our data into testing, D.test, and
        # training, D.train, datasets
        testIndexes <- which(subsets == i, arr.ind = TRUE)
        D.test <- D_dash[testIndexes, ]
        D.train <- D_dash[-testIndexes, ]

        # We get the model matrix and predictor variables
        # for the training data and then we find the LS
        # estimator
        X.train <- mm(D.train)
        y.train <- y_dash[-testIndexes]
        w <- LLS(X.train, y.train)

        # We get the model matrix and predictor variables
        # for the testing data and then we calculate the
        # least squares testing error
        X.test <- mm(D.test)
        y.test <- y_dash[testIndexes]
        test.error <- norm(y.test - X.test %*% w, type = "2")^2

        # We add the testing error to our error vector
        errors[i] <- test.error

    }

    # We find and return the cross-val error
    error.CV <- sum(errors)/10
    error.CV
}
```

Finally, we carry out cross validation by passing our data to the function we just wrote and find the cross-validation error.

```
cross_val(D, y)
```

```
## [1] 5.188135
```

Dimensionality reduction

Now we would like to see if all the predictor variables are important, to check this for each predictor variable we will remove it from the dataset and calculate the cross-validation error. Examining the cross-validation errors we can then check to see if removing certain features corresponds to obtaining larger errors and therefore if certain features are more crucial to keep in the model than others.

Here we loop through the dataset 8 times, each time removing a different column (predictor variable) and then calculating and storing the error in a vector. We do this 100 times and return the average cross-validation error.

```
total_error <- rep(0, 8)
for (j in 1:100) {
    errors_j <- c()
    for (i in 1:8) {
        D_drop <- D[-i]
        errors_j[i] <- cross_val(D_drop, y)
    }
    total_error <- total_error + errors_j
}
total_error/100
```

```
## [1] 7.582563 5.709965 5.414487 5.345992 5.764937 5.296518 5.204049 5.252070
```

From our results we see that removing column 1 leads to a large increase in the cross-validation error. However, removing any of the other columns leads to more minor changes in the error. This may suggest that the other features may not be needed in the model. To test this we will calculate the cross validation error obtained by the LS estimator by just using column 1 (lcavol).

```
D_drop <- D[-c(2, 3, 4, 5, 6, 7, 8)]
cross_val(D_drop, y)
```

```
## [1] 6.215014
```

We notice that the cross-validation error produced is higher than when we had all the features present in our model. Looking back at the errors found when dropping features we notice that column 2 (lweight) and column 5 (svi) have larger errors. So lets do the same as before but now also dropping these features.

```
D_drop <- D[-c(3, 4, 6, 7, 8)]
cross_val(D_drop, y)
```

```
## [1] 5.118062
```

We see that the error we obtain is around the value we got when using all the features. This suggests that the lcavol, lweight and svi predictor variables are potentially not needed in our model for us to have good predictions for the outcome. Hence we can reduce the dimensionality of our problem (and need less observations/ produce a better model on the number of observations) whilst being able to produce just as good a model (than with all the features still present).

The source file for this should run on any computer with R and its main packages installed, the data is easily accessible and does not even need to be downloaded prior to running the file, is written using literate programming for ease of readability and has been written using version control and is available on Github.