Universidad Carlos III de Madrid

Regression Methods

An introduction to R Language

Phd. in Business Administration
&
Phd. in Mathematical Engineering

Last Revision Date: January 2, 2006 Version 1.0

Table of Contents

1. Install R

An up-to-date version of R may be downloaded from http://cran.r-project.org/ or from the nearest mirror site.

Information about the program at:

- http://www.r-project.org/
- Official R Manuals

1.1. R Packages

- http://cran.r-project.org/src/contrib/PACKAGES.html

2. Basics

R works with data structures. The simplest data structure that R works with is a vector. (For more details see Dalgaard (2002) and Maindonald and Braun (2003))

2.1. Vectorized Arithmetic

- **Data Vectors**

The construct `c()` is used to define vectors. A data vector is simply and array of numbers and a vector variable can be constructed like this:

```
> weight<-c(60,72,57,90,95,72)
> weight
[1] 60 72 57 90 95 72
```

You can do calculations with vectors just like ordinary numbers, as long they are of the same length.

```
> height<-c(1.75,1.80,1.65,1.90,1.74,1.91)
> height
[1] 1.75 1.80 1.65 1.90 1.74 1.91

> bmi<- weight/height^2
> bmi # body mass index
[1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

These are some typical statistical calculations in R language:

```
> length(weight)  # length of vector weight
[1] 6
> sum(weight)  # summatory of the elements of the vector
[1] 446
> mean(weight)  # calculates the mean of weight
[1] 74.33333
> sd(weight)  # calculates the standard deviation of weight
[1] 15.42293
```

There are most easily created from vectors using the `matrix()` function:

```
> X=matrix(c(1,2,3,4,5,6),nrow=3)
> X
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

```
> Y=matrix(c(1,2,3,4,5,6), nrow=3, byrow=T)
> Y
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

In each case a vector of length 6 is turned into a 3×2 matrix. You can specify `nrow=3` or `ncol=2` or both; all three choices are equivalent. The vector fills up the matrix in successive columns unless `byrow=T` is specified, in which case the matrix is filled up by successive rows.

- **Operations on vectors**

To select an element of a vector, you can use `[]`

```
> x <-c(2,5)
[1] 2 5
> x[2]
[1] 5
```

To select multiple elements, use `[]` with a vector enclosed:

```
> x <-c(11:20)
> x
 [1] 11 12 13 14 15 16 17 18 19 20

> y <- x[c(3,5,8:10)]
> y
 [1] 13 15 18 19 20
```

Alternatively, you can concatenate commands to do this example in a single step:

```
> y <-c(11:20)[c(3,5,8:10)]
> y
[1] 13 15 18 19 20
```

- **Character vectors**

A *character vector* is a vector of text strings, whose elements are specified and printed in quotes:

```
> c("Huey","Dewey","Louie")
[1] "Huey"  "Dewey" "Louie"
```

- **Logical vectors**

Logical vectors can take the value `TRUE` or `FALSE` (or `NA`). In input, you may use the convenient abbreviations `T` and `F` Logical vectors are constructed using the `c` function just like the other vector types:

```
> c(T,T,F,T)
 [1]  TRUE  TRUE FALSE  TRUE
```

Actually, you will not often have to specify logical vectors in the above manner. It is much more common to use single logical values to turn an option on or off in a function call. Vectors of more than one value most often result from *relational expressions*:

```
> bmi
 [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630

> bmi > 25

 [1] FALSE FALSE FALSE FALSE  TRUE FALSE

> bmi > 20 & bmi < 25

 [1] FALSE  TRUE  TRUE  TRUE FALSE FALSE
```

In practical data analysis a data point is frecuently unavailable. Statistical software needs ways to deal with this. R allows vectors to contain a special `NA` value. This value is carried through in computations so that operations on `NA` yield `NA` as the result.

- **Concatenation - joining vector objects**

We can concatenate two vectors:

```
> x <- c(2, 3, 5, 2, 7, 1)
> x
 [1] 2 3 5 2 7 1

> y <- c(10, 15, 12)
> y
 [1] 10 15 12

> z <- c(x,y)
> z
 [1]  2  3  5  2  7  1 10 15 12
```

- **Subsets of vectors**

Note two common ways to extract subsets of vectors.

1. Specify the indices of the elements that are to be extracted, e.g.,

```
> x <-c(3, 11, 8, 15, 12)
> x[c(2,4)]  # Elements in position 2 and 4 only
[1] 11 15
```

2. Use negative subscripts to omit the elements in nominated subscript positions (take care not to mix positive and negative subscripts):

```
> x[-c(2,3)]} # Remove the elements in positions 2 and 3
  [1] 3 15 12
```

2.2. Functions that create vectors

Here we introduce three functions, `c`, `seq` and `rep`, which are used to create vectors in various situations.

- `c( )`

The function `c( )` is a short for *'concatenate'*, that is, joining items end to end, which is exactly what the function does:

```
> c(42,57,12,39,1,3,4)
[1] 42 57 12 39  1  3  4
```

- `seq( )`

Function `seq( )` ('sequence') is used for equidistant series of

```
> seq(4,9)
 [1] 4 5 6 7 8 9
```

yields, as seen, the integers from 4 to 9. If you want a sequence in jumps of 2, write

```
> seq(4,10,2)
 [1] 4 6 8 10
```

Also

```
> seq(from=4, to=10, by=2)
 [1]  4  6  8 10
```

This kind of vector is frequently needed, particularly for graphics.

The case with step size equal to 1 can also be written using a special syntax:

```
> 4:9
[1] 4 5 6 7 8 9
```

- `rep( )`

Function `rep( )` ('replicate') is used to generate repeated values. It is used in two variants, depending on whether the second argument is a vector or a single number:

```
> oops <- c(7,9,13)
> rep(oops,3)   # repeats the entire vector 'oops' three times
  [1]  7  9 13  7  9 13  7  9 13

> rep(oops,1:3) # this function has the number 3 replaced by a vector with
the three values (1,2,3) indicating that 7 should be repeated
  once, 9 twice and 13 three times.

[1]  7  9  9 13 13 13
```

The `rep` function is often used for things like group codes. If it is known that the first 10 observations are men and the last 15 ones are women, you can use

```
> rep(1:2,c(10,15))
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

to form a vector that for each observation indicates whether it is from a man or a woman.

To repeat the sequence (2,3,5) four times over, enter

```
> rep(c(2,3,5), 4)
 [1] 2 3 5 2 3 5 2 3 5 2 3 5
```

Patterned character vectors are also possible

```
> c(rep("female",3), rep("male",2))
 [1] "female" "female" "female" "male"   "male"
```

2.3. Matrices and arrays

In R matrices and arrays are represented as vectors with dimensions:

- `dim( )`

The `dim` assignment function sets or changes the *dimension attribute* of `x`, causing R to treat the vector of 12 numbers as a 3×4 matrix.

```
> x<- 1:12
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12

> dim(x)<-c(3,4)
> x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Notice that the storage is column-major, that is, the elements of the first column are followed by those of the second, etc.

A convenient way to create matrices is to use the matrix function:

```
> X <- matrix(1:12,nrow=3,byrow=T)
> X
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

Notice how the `byrow=T` switch causes the matrix to be filled in a rowwise fashion rather than columnwise.

- **`cbind( ) and rbind( )`**

You can attach vectors together, columnwise or rowwise, using the `cbind` and `rbind` functions. If X and Y contain the matrices $\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$ and $\begin{bmatrix}5 & 6\\7 & 8\end{bmatrix}$ then

`cbind(x,y)` and `rbind(x,y)` contain the matrices

$$\begin{bmatrix}1 & 2 & 5 & 6\\3 & 4 & 7 & 8\end{bmatrix} \text{ and } \begin{bmatrix}1 & 5\\3 & 6\\5 & 7\\7 & 8\end{bmatrix}$$

respectively. If X and/or Y is a vector, it is interpreted as a column in `cbind` and as a row in `rbind`.

Useful functions that operate on matrices include `rownames`, `colnames` and the transposition function `t`, which turns rows into columns and vicecersa:

```
> rownames(x) <- LETTERS[1:3]
> x
  [,1] [,2] [,3] [,4]
A    1    2    3    4
B    5    6    7    8
C    9   10   11   12

> t(x)
     A B  C
[1,] 1 5  9
[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
```

The character vector `LETTERS` is a built-in variable that contains the capital letters A-Z. Similar useful vectors are `letters`, `month.name`, and `month.abb` with lowercase letters, month names and abbreviated month names.

```
> rownames(x) <- month.name[1:3]
> x
         [,1] [,2] [,3] [,4]
January     1    2    3    4
February    5    6    7    8
March       9   10   11   12
```

2.4. Factors

It is common in statistical data to have categorical variables, indicating some subdivision of data, such as social class, primary diagnosis, tumor stage, etc. Typically, these are input using a numeric code. Such variables should be specified as *factors* in R. This is a data structure that (among other things) makes it possible to assign meaningful names to the categories.

A factor is stored internally as a numeric vector with values 1,2,3,...,k. The value k is the numbers of levels. The levels are character strings.

Consider a survey that has 691 females and 692 males. If the first 691 are females and the next 692 are males, we can create a vector of strings that holds the values thus:

```
> gender<-c(rep("female",691),rep("male",692))
```

We can change this vector to a factor, by entering

```
> gender<- factor(gender)
```

Internally, the factor `gender` is stores as 691 1's, followed by 692 2's. The values `female` and `male` are the levels of the factor.

```
> levels(gender) # Extracts the names of the levels
[1] "female"  "male"
```

Another example:

```
> pain <- c(0,3,2,2,1)    # This command creates a numerical vector 'pain',
# encoding the pain level of five patients.

> fpain <- factor(pain, levels=0:3)
> levels(fpain) <- c("none","mild","medium","severe")
> fpain
[1] none   severe medium medium mild
Levels: none mild medium severe
```

You can create a factor `fpain` to treat the vector as a categorical variable. The argument `levels=0:3` indicates that the input coding uses the values 0,1,2 and 3.

```
> as.numeric(fpain)
 [1] 1 4 3 3 2

> levels(fpain) <- c("none", "mild", "medium", "severe")
 [1] "none"   "mild"   "medium" "severe"
```

The function `as.numeric` extracts the numerical encoding as numbers 1-4. The internal representation of a factor always starts at 1.

If, in `factor()`, you do not specify a `levels` argument, the levels will by default be the sorted unique values represented in the vector. This is not always desirable when dealing with text variables, since the sorting is alphabetical. Consider, for instance,

```
> text.pain<-c("none","severe","medium","medium","mild")
> factor(text.pain)
[1] none   severe medium medium mild
Levels: medium mild none severe
```

One advantage of factors is that the memory required for storage is less than for the corresponding character vector when there are multiple values for each factor level, and the levels are long character strings.

2.5. Lists

It is sometimes useful to combine a collection of objects into a larger composite object. Consider a set of data concerning pre- and postmenstrual energy intake in a group of women.

```
> intake.pre <- c(5260, 5470, 5640, 6180, 6390,
+ 6515, 6805, 7515, 7515, 8230, 8770)

> intake.post <-c(3910, 4220, 3885, 5160, 5645,
+ 4680, 5265, 5975, 6790, 6900, 7335)
```

To combine these individual vectors into a list, you can say

```
> mylist <- list(before=intake.pre, after=intake.post)
> mylist
$before
 [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770

$after
 [1] 3910 4220 3885 5160 5645 4680 5265 5975 6790 6900 7335
```

The components of the list are named according to the argument names used in `list`. Named components may be extracted like this:

```
> mylist$before
 [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770
```

2.6. Data frames

A data frame corresponds to what other statistical packages call a "data matrix" or a "data set". It is a list of vectors and/or factors of the same length, which are related "across", such that data in the same position come from the same experimental unit (subject, animal, etc...). In addition, it has unique set of row names.

Data frames are fundamental to the use of the `R` and graphics functions. All elements of any column must have the same mode, i.e. all numeric, or all factor, or all character, or all logical.

You can create data frames from preexisting variables:

```
> d<-data.frame(intake.pre, intake.post)
> d
   intake.pre intake.post
1        5260        3910
2        5470        4220
3        5640        3885
4        6180        5160
5        6390        5645
6        6515        4680
7        6805        5265
8        7515        5975
9        7515        6790
10       8230        6900
11       8770        7335
```

As with lists, variables are accesible using the `$` notation:

```
> d$intake.pre
 [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770
```

Example:

```
> library(DAAG)
> data(Cars93.summary)
> Cars93.summary
        Min.passengers Max.passengers No.of.cars abbrev
Compact              4              6         16      C
Large                6              6         11      L
Midsize              4              6         22      M
Small                4              5         21     Sm
Sporty               2              4         14     Sp
Van                  7              8          9      V

> row.names(Cars93.summary)   # row labels
[1] "Compact" "Large"   "Midsize" "Small"   "Sporty"  "Van"

> names(Cars93.summary)   # column names
[1] "Min.passengers" "Max.passengers" "No.of.cars"     "abbrev"
```

There are several ways to access the columns of a data frame:

```
> type <- Cars93.summary$abbrev
> type1 <- Cars93.summary[,4]}
> type2 <- Cars93.summary[,"abbrev"]
> type3 <- Cars93.summary[[4]]    # Take the object that
# is stored in the 4th list element
```

In each case, one can view the contents of the object `type` by entering `type` at the command line, thus:

```
> type
 [1] C  L  M  Sm Sp V
Levels: C L M Sm Sp V
```

It is often convenient to use the `attach()` function:

```
> attach(Cars93.summary)   # Now you can access the columns directly
> abbrev
 [1] C  L  M  Sm Sp V
Levels: C L M Sm Sp V

> detach(Cars93.summary)
```

In Windows versions, use of `edit()` allows to access to a spreadsheet-like display of a data frame or of a vector. Users can then manipulate individual entries or perform data entry operations as with a spreadsheet.

```
> Cars93.summary <-edit(Cars93.summary)
```

To close the spreadsheet, click on the `file` menu then on `close`

2.7. Graphics

One of the most important aspects of the representations and analysis of data is the generation of proper graphics.

The functions `plot()`, `points()`, `lines()`, `text()`, `mtext()`, `axis()`, `identify()`, etc,... form a suite that plot graphs and add features to the graph. To see some of the possibilities that `R` offers, enter

```
> demo(graphics)
```

Press Enter key to move to each new graph.

For more details: http://cran.r-project.org/doc/manuals/R-intro.html#Graphics

3. Matrix algebra review

Let `A` be a 3×4 matrix

```
> A <-matrix(c(1:12),nrow=3)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

and B another 4×3 matrix

```
> B <-matrix(c(5:16),nrow=4)
> B
     [,1] [,2] [,3]
[1,]    5    9   13
[2,]    6   10   14
[3,]    7   11   15
[4,]    8   12   16
```

3.1. Basic algebra

- **Matrix multiplication**
  - `%*%`: Returns the product of two matrices,

    ```
    > A%*%B
    ```

  - `%x%`: Returns the Kronecker product ($A \otimes B$),

    ```
    > A%x%B
    ```

- **Determinant of a matrix**
  - `det()`: Returns the determinant of a matrix.
- **Inverse of a matrix**
  - `solve()`: Returns the inverse of a matrix.

    Remember the inverse of a matrix A is the matrix A^{-1} such that $AA^{-1} = A^{-1}A = I$. However when we calculate A^{-1} with R we don't get exactly zero in all of the off-diagonal elements. This is because the numerical approximation used are not exactly[1].
- **Transpose of a matrix**
  - `t()`: Returns the transpose of a matrix.
- **Diagonal function**

The R function `diag()` diag performs differently depending on what argument is passed to the function. If the function is applied to a **matrix** then the diagonal elements of that matrix are returned.

```
> C <- matrix(c(1,1,3,2),2,2)
> C
     [,1] [,2]
[1,]    1    3
[2,]    1    2

> diag(C)
[1] 1 2
```

[1] When a matrix is symmetric there is a more efficient way to calculate the inverse, which is based on the Cholesky decomposition (3.2)

If the function is applied to a **vector**, it creates a diagonal matrix with the elements on the diagonal.

```
> x <-c(1,2,1,3)
> x
[1] 1 2 1 3

> diag(x)
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    2    0    0
[3,]    0    0    1    0
[4,]    0    0    0    3
```

If you give `diag` a single number (scalar), then it creates an **identity matrix**.

```
> I=diag(4)
> I
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
```

- **Trace of a matrix**

We can find the trace of a matrix en R as follows

```
> sum(diag(A))
[1] 34
```

3.2. Cholesky Decomposition

If the matrix A is positive definite it can be factored as $A = T^tT$, where T is an upper triangular matrix[2]. One way to find such a factorization is the Cholesky decomposition, which is implemented in `R` in the `chol()` function. Suppose we want to factor the matrix

[2]Cholesky decomposition may also be decomposed as $A = LL^t$, where in this case L is an lower triangular matrix with positive diagonal elements. Recall that, $L = T^t$ and $L^t = T$, so $A = T^tT = LL^t$

```
> A=1+diag(5)
> A
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    1    1    1    1
[2,]    1    2    1    1    1
[3,]    1    1    2    1    1
[4,]    1    1    1    2    1
[5,]    1    1    1    1    2

> A.chol=chol(A)
         [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 1.414214 0.7071068 0.7071068 0.7071068 0.7071068
[2,] 0.000000 1.2247449 0.4082483 0.4082483 0.4082483
[3,] 0.000000 0.0000000 1.1547005 0.2886751 0.2886751
[4,] 0.000000 0.0000000 0.0000000 1.1180340 0.2236068
[5,] 0.000000 0.0000000 0.0000000 0.0000000 1.0954451
```

To get back to the original matrix form the Cholesky decomposition we simply must calculate

```
> t(A.chol)%*%A.chol

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

You can also easily find the determinant of A using the Cholesky decomposition by nothing that $|A| = |T|^2$ and that determinant of un upper triangular matrix is simply the product of the diagonal elements. Therefore $|A| = (\Pi_{i=1}^{n} T_{ii})^2$

```
> prod(diag(A.chol))^2
[1] 6
```

Applying the function `chol2inv()` to the Cholesky decomposition of the matrix A will return A^{-1}.

```
> chol2inv(chol(A))
           [,1]       [,2]       [,3]       [,4]       [,5]
[1,]  0.8333333 -0.1666667 -0.1666667 -0.1666667 -0.1666667
```

```
[2,] -0.1666667  0.8333333 -0.1666667 -0.1666667 -0.1666667
[3,] -0.1666667 -0.1666667  0.8333333 -0.1666667 -0.1666667
[4,] -0.1666667 -0.1666667 -0.1666667  0.8333333 -0.1666667
[5,] -0.1666667 -0.1666667 -0.1666667 -0.1666667  0.8333333
```

3.3. Eigenvalues and Eigenvectors

R approximates the eigenvectors and eigenvalues of any square matrix with the function `eigen()`. It returns a *list* (2.5) rather than just a matrix. A *list* can contain any number of objects, in this case `eigen` returns two objects: a vector named `values` which contains all eigenvalues for the matrix; and a matrix whose columns contain the eigenvectors.

```
> A.eigen <- eigen(A)
> A.eigen
$values
[1] 6 1 1 1 1

$vectors
          [,1]       [,2]       [,3]          [,4]       [,5]
[1,] 0.4472136  0.8944272  0.0000000  0.000000e+00  0.0000000
[2,] 0.4472136 -0.2236068  0.8660254 -2.347975e-17  0.0000000
[3,] 0.4472136 -0.2236068 -0.2886751 -7.191508e-17  0.8164966
[4,] 0.4472136 -0.2236068 -0.2886751 -7.071068e-01 -0.4082483
[5,] 0.4472136 -0.2236068 -0.2886751  7.071068e-01 -0.4082483
```

Recall that to access only the values object of the list you type `A.eigen$values`. Similarly use `A.eigen$vectors` to access the eigenvectors.

3.4. Spectral Decomposition

The spectral decomposition decomposes the matrix $A = CD_\lambda C'$ where C is the matrix whose columns are made up of the eigenvectors and D_λ is the matrix with the eigenvalues on the diagonal. So

```
> C=A.eigen$vectors
> C
          [,1]       [,2]       [,3]          [,4]       [,5]
[1,] 0.4472136  0.8944272  0.0000000  0.000000e+00  0.0000000
[2,] 0.4472136 -0.2236068  0.8660254 -2.347975e-17  0.0000000
[3,] 0.4472136 -0.2236068 -0.2886751 -7.191508e-17  0.8164966
[4,] 0.4472136 -0.2236068 -0.2886751 -7.071068e-01 -0.4082483
[5,] 0.4472136 -0.2236068 -0.2886751  7.071068e-01 -0.4082483

> D=diag(A.eigen$values)
```

```
> D
     [,1] [,2] [,3] [,4] [,5]
[1,]    6    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1

> C%*%D%*%t(C)
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    1    1    1    1
[2,]    1    2    1    1    1
[3,]    1    1    2    1    1
[4,]    1    1    1    2    1
[5,]    1    1    1    1    2
```

which is exactly A. Similarly the square root matrix can be found as follows

```
> A.sqrt=C%*%sqrt(D)%*%t(C)
> A.sqrt
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 1.2898979 0.2898979 0.2898979 0.2898979 0.2898979
[2,] 0.2898979 1.2898979 0.2898979 0.2898979 0.2898979
[3,] 0.2898979 0.2898979 1.2898979 0.2898979 0.2898979
[4,] 0.2898979 0.2898979 0.2898979 1.2898979 0.2898979
[5,] 0.2898979 0.2898979 0.2898979 0.2898979 1.2898979

> A.sqrt%*%A.sqrt
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    1    1    1    1
[2,]    1    2    1    1    1
[3,]    1    1    2    1    1
[4,]    1    1    1    2    1
[5,]    1    1    1    1    2
```

3.5. Singular Value Decomposition

The singular value decomposition of a $m \times n$ matrix is given by:

$$A = U\Sigma V^t$$

where U and V are orthogonal matrices and $\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_n)$ where $\sigma_1 \geq \sigma_1 \geq ...\sigma_n$ are the singular values of A.

The singular value decomposition can be found by setting U equal to matrix of the eigenvector of AA', Σ equal to the square-root of the non-zero eigenvalues of AA' and defining V by

$$V = A'U\Sigma^{-1}$$

then

$$A = U\Sigma V^{-1}$$

```
> B<-matrix(c(3,2,1,1,1,2,),3,2)
> B
     [,1] [,2]
[1,]    3    1
[2,]    2    1
[3,]    1    2

> BBt.eigen<-eigen(B%*%t(B))

> BBt.eigen$values
[1] 1.806226e+01 1.937742e+00 2.099612e-15

> BBt.eigen$vectors
          [,1]        [,2]       [,3]
[1,] 0.7286301 -0.46038608  0.5070926
[2,] 0.5251205 -0.09981375 -0.8451543
[3,] 0.4397121  0.88208949  0.1690309

> U<-BBt.eigen$vectors[,1:2]
> U
          [,1]        [,2]
[1,] 0.7286301 -0.46038608
[2,] 0.5251205 -0.09981375
[3,] 0.4397121  0.88208949

> D<-diag(sqrt(BBt.eigen$values)[1:2])
> D
         [,1]     [,2]
[1,] 4.249971 0.000000
[2,] 0.000000 1.392028

> V=t(B)%*%U%*%solve(D)
> V
          [,1]       [,2]
[1,] 0.8649101 -0.5019268
[2,] 0.5019268  0.8649101
```

```
> U%*%D%*%t(V)
     [,1] [,2]
[1,]    3    1
[2,]    2    1
[3,]    1    2
```

The singular value decomposition and the spectral decomposition can also be found using the R function svd[3].

```
> svd(B)
$d
[1] 4.249971 1.392028

$u
           [,1]        [,2]
[1,] -0.7286301  0.46038608
[2,] -0.5251205  0.09981375
[3,] -0.4397121 -0.88208949

$v
           [,1]       [,2]
[1,] -0.8649101  0.5019268
[2,] -0.5019268 -0.8649101
```

[3]Recall that if v is a normalized eigenvector, then so is $-v$

4. Multiple Regression

The basic model for multiple regression[4] analysis is

$$y_i = \beta_0 + \beta_{i1}x_1 + \ldots + \beta_{ik}x_{ik} + \epsilon_i \tag{1}$$

4.1. Dwaine studios example

Dwaine studios, Inc. operates portrait studios in 21 cities of medium size. These studios specialize in portraits of children. The company is considering an expansion into other cities of medium sizes and wishes to investigate whether sales (Y) in a community can be predicted from the number of persons aged 16 or younger in the community (X_1) and the per capita disposable personal income in the community (X_2).

- Y_i is *sales* in city i, in thousand of dollars.
- X_1: population aged 16 and under (in 1000's of people)
- X_2: per capita disposable income, expressed in thousands of dollars.
- The starting regression model is:

$$Y_i = \beta_0 + \beta_{i1}x_1 + \beta_{i2}x_{i2} + \epsilon_i \tag{2}$$

```
> data <- read.table("dwaine")

> sales <- data[,1]   # sales in a community
> targtpop <- data[,2]   # number of persons aged 16 or younger
> dispoinc <- data[,3]   # per capita disposable personal income

> Dwaine <- cbind(sales, targtpop, dispoinc)
# use "cbind" to add extra columns

> dimnames(Dwaine) <- list(NULL, c("sales", "targtpop", "dispoinc"))
# Assign column heading > Dwaine

> Dwaine
      sales targtpop dispoinc
 [1,] 174.4     68.5     16.7
 [2,] 164.4     45.2     16.8
 ... ...  ... ...
[21,] 166.5     52.3     16.0
```

[4] more details about Regression using R in http://www.stat.lsa.umich.edu/ faraway/book/pra.pdf

4.2. Plotting multivariate data

You can obtain pairwise scatterplots between all the variables in the data set. This is done using the function `pairs` (Figure 1).

```
> pairs(Dwaine)
```

```
> pairs(Dwaine, panel=function(x,y){points(x,y);abline(lm(y~x),col=2)})
```

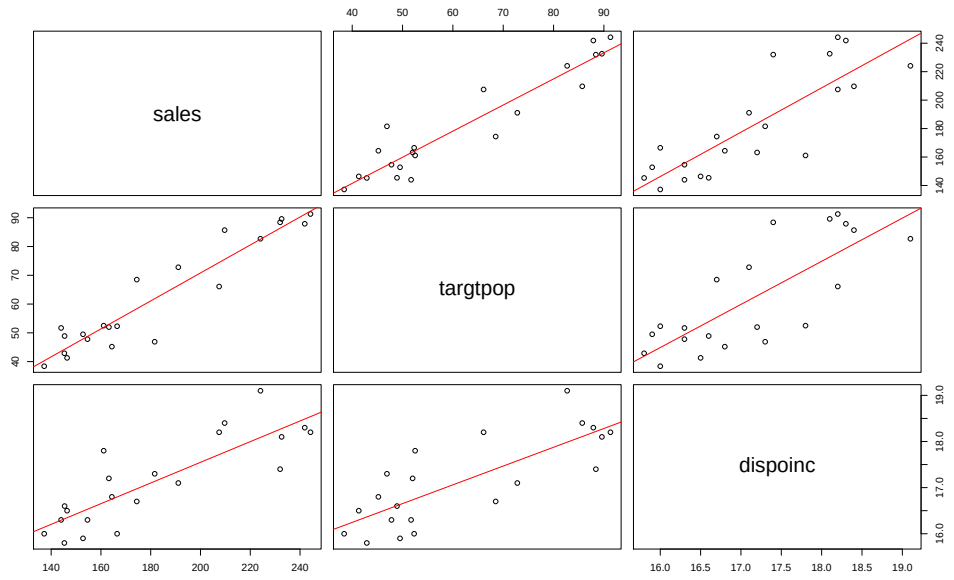

Figure 1: Pairwise scatterplots - Dwaine studios example

The scatter plot indicates that the relationship `between` sales and `targtpop (dispoinc)` is reasonably linear, note in Figure 1 the relationship between the two predictor variables. This is also confirmed by the correlation matrix:

```
> cor(Dwaine)
             sales  targtpop  dispoinc
sales    1.0000000 0.9445543 0.8358025
targtpop 0.9445543 1.0000000 0.7812993
dispoinc 0.8358025 0.7812993 1.0000000
```

Now you can create a 3D scatter plot of the data (Figure 2) (download package `scatterplot3d` from `R` homepage at http://cran.r-project.org/src/contrib/Descriptions/scatterplot3d.html)

```
> scatterplot3d(targtpop, dispoinc, sales, pch=20)
```

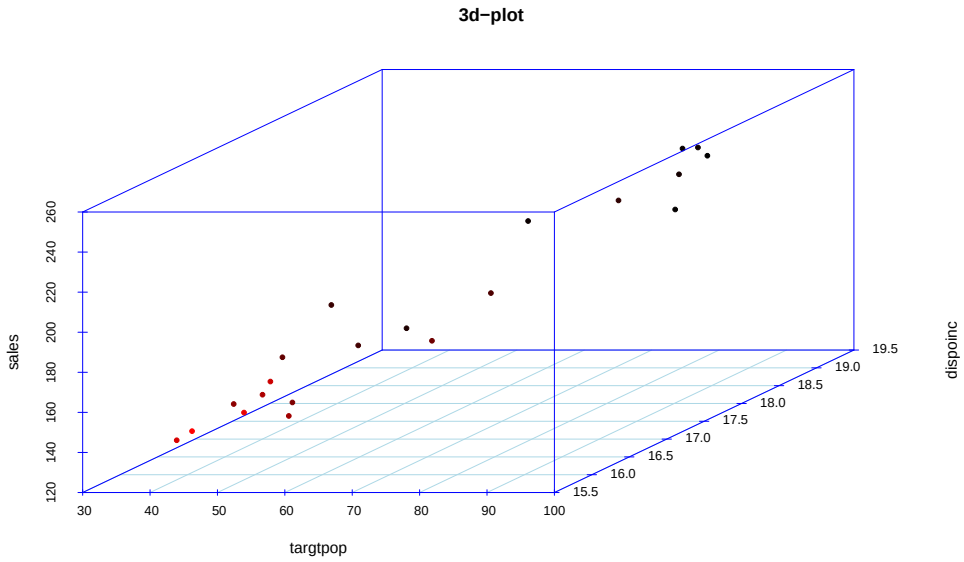

Figure 2: 3D scatter plot - Dwaine studios example

4.3. Model specification

- **Matrix operation and multiple regression**

To set up matrix forms of Y and X

```
> Y <- sales
> Xmatrix <-cbind(rep(1,length(Y)),targtpop,dispoinc)
> Xmatrix
        targtpop dispoinc
 [1,] 1     68.5     16.7
 [2,] 1     45.2     16.8
 ... ...     ...      ...
[21,] 1     52.3     16.0
```

The **regression coefficients**, $\widehat{\beta}$ are obtained by using the normal equations formula[5]

$$\widehat{\beta} = \begin{pmatrix} \widehat{\beta}_0 \\ \widehat{\beta}_1 \\ \vdots \\ \widehat{\beta}_{p-1} \end{pmatrix} = (X'X)^{-1} X'Y \tag{3}$$

```
> beta <- solve(t(Xmatrix)%*%Xmatrix)%*% t(Xmatrix)%*%Y
```

[5]Pay attention to the special case where the determinant of $X'X$ tends to 0, which leads to *multicollinearity*, see Section 4.8.

```
> beta
                [,1]
         -68.857073
targtpop   1.454560
dispoinc   9.365500
```

Vector of **fitted values**, $\widehat{Y}$

$$\widehat{Y} = \begin{pmatrix} \widehat{Y}_1 \\ \vdots \\ \widehat{Y}_n \end{pmatrix} = X\widehat{\beta}$$

```
> Y_fitted <- Xmatrix%*%beta
> Y_fitted
          [,1]
 [1,] 187.1841
 [2,] 154.2294
 ...     ...
[21,] 157.0644
```

Vector of **residuals**, e

$$\begin{aligned} e &= \begin{pmatrix} Y_1 - \widehat{Y}_1 \\ \vdots \\ Y_n - \widehat{Y}_n \end{pmatrix} \\ &= Y - \widehat{Y} \\ &= Y - X\widehat{\beta} \\ &= \left[I - X(X'X)^{-1}X'\right] Y \end{aligned}$$

```
> resid <- Y - Xmatrix%*% beta
> resid
             [,1]
 [1,] -12.7841146
 [2,]  10.1705737
 ...        ...
[21,]   9.4356009
```

- `lm()`

In R language the specification of a multiple regression analysis is done by setting up a model formula with + between the explanatory variables [6]. The name `lm` stands for linear model. The syntax is:

```
> myfit <- lm(sales ~ targtpop + dispoinc, qr=T)
```

The result of `lm()` is a list of objects that summarize the least-squares regression fit. To get a nice printout of the results of the regression, use the `summary()` function:

```
> summary(myfit)

Call:
lm(formula = sales ~ targtpop + dispoinc, qr = T)

Residuals:
     Min       1Q   Median       3Q      Max
-18.4239  -6.2161   0.7449   9.4356  20.2151

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -68.8571    60.0170  -1.147   0.2663
targtpop      1.4546     0.2118   6.868    2e-06 ***
dispoinc      9.3655     4.0640   2.305   0.0333 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.01 on 18 degrees of freedom
Multiple R-Squared: 0.9167,     Adjusted R-squared: 0.9075
F-statistic:  99.1 on 2 and 18 DF,  p-value: 1.921e-10
```

It follows from the above summary that

$$\begin{aligned} \widehat{\beta}_0 &= -68.8571 \\ \widehat{\beta}_1 &= 1.4546 \\ \widehat{\beta}_2 &= 9.3655 \end{aligned}$$

and the estimated regression model is

$$\widehat{Y}_i = -68.8571 + 1.4546 X_{1i} + 9.3655 X_{2i},$$

which is a plane in the three-dimensional space.

To create a three-dimensional plot, use `persp()`, the perspective plot. Input is two sets of coordinate points $(x1, x2)$ and an array of corresponding heights (y) of the three-dimensional surface to be plotted (Figure 3).

[6] for more details type in R `help("lm")` or `?lm`

```
> x1 <- seq(min(dispoinc), max(dispoinc), length=100)
> x2 <- seq(min(targtpop), max(targtpop), length=100)
> y <- outer(x1,x2, function(x1,x2){-68.857+1.455*x1+9.366*x2})
> y[is.na(y)] <- 1
> persp(x1,x2,y,theta=30, phi=30, expand=0.5, col=7)
```

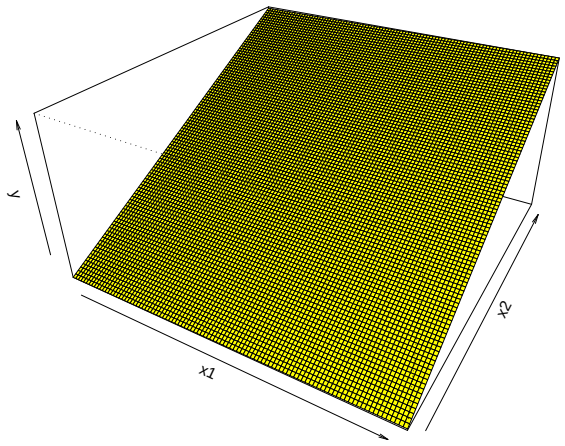

Figure 3: Perspective plot - Dwaine studios example

To obtain residuals and fitted values, use

```
> residuals <- myfit$residuals # or > residuals <- resid(myfit)

> fittedvalues <- myfit$fitted  # or > fittedvalues <- fitted(myfit)
```

- `lsfit()`

The function `lsfit()` find the Least Squares Fit[7]

```
> Dwaine.lsfit<-lsfit(cbind(Dwaine[,2],Dwaine[,3]),Dwaine[,1])
```

where `cbind(Dwaine[,2],Dwaine[,3])` is a matrix of predictors (X) and `Dwaine[,1]` is the vector of responses (Y). The length of the vector of responses should be the same as the number of rows of the matrix of predictors. Unless you say otherwise, `R` will automatically append a column of $1's$ onto X so that the regression model will include a intercept. The result of this function is a list that contains various quantities summarizing the least-squares fit. For example,

[7] for more details in `R` write `help("lsfit")` or `?lsfit`.

```
> Dwaine.lsfit$coef
 Intercept          X1          X2
-68.857073    1.454560    9.365500
```

will print the fitted coefficients ($\widehat{\beta}$), a vector of length $ncol(X) + 1$, and

```
> Dwaine.lsfit$residuals

 [1] -12.7841146  10.1705737   9.8036764   1.2714690  20.2150722   9.7585779
 [7]   0.7449178  -4.6666316 -12.3381967   0.3539791  11.5126289  -6.0849209
[13]   9.3142995   3.7815611 -13.1132116 -18.4238900   0.6530062 -15.0013134
[19]   1.6129777  -6.2160615   9.4356009
```

will print the residuals, a vector of the same length as Y_i

The function `ls.print()` will automatically generate this kind of printout:

```
> ls.print(Dwaine.lsfit)
Residual Standard Error=11.0074
R-Square=0.9167
F-statistic (df=2, 18)=99.1035
p-value=0
           Estimate Std.Err t-value Pr(>|t|)
Intercept -68.8571 60.0170 -1.1473   0.2663
X1          1.4546  0.2118  6.8682   0.0000
X2          9.3655  4.0640  2.3045   0.0333
```

One of the disadvantages of using `lsfit(...)` for linear regression is that if some of your predictor variables are categorical, you need to create dummy variables for them and put them into the X matrix yourself. If there are a large number of categorical predictors, this can be very tedious.

- `ls.diag()`

The function `ls.diag` is used to obtain many standard regression diagnostics[8]. It was originally used with an argument that was the output of the function `ls.fit`, but if you use `qr=T` in the `lm` command, you can use `ls.diag()` with lm's output too.

```
> Dwaine.ls.diag <- ls.diag(Dwaine.lsfit)
```

The output is a list with the following numeric components:

```
> names(Dwaine.ls.diag)
 [1] "std.dev"   "hat"           "std.res"     "stud.res"      "cooks"
 [6] "dfits"     "correlation"   "std.err"     "cov.scaled"    "cov.unscaled"
```

[8] for more details in R write `help("ls.diag")` or `?ls.diag`

- **Residuals**

Once you have specified a multiple regression, you can estimate the residuals.

- Residuals from least squares fit:

```
> ls.resid <- Dwaine.lsfit$residuals
```

- Standardized residuals:

```
> std.resid <- Dwaine.ls.diag$std.res
```

- Studentized residuals:

```
> stud.resid <- Dwaine.ls.diag$stud.resid
```

- **Fitting other models**

In multiple regression applications, there is frecuently the possibility of interaction effects being present, so that the response function of the type

$$E[Y] = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2$$

might be more appropriate.

In R language the tilde character (~) means, *is modeled as a function of*. So the formula Y ~ X1 + X2 + ... means that Y is modeled as an additive linear function of $X1, X2, ...$ To include the interaction between $X1$ and $X2$, you would say:

```
> summary(lm(sales ~ targtpop + dispoinc + targtpop:dispoinc, qr=T))

Call:
lm(formula = sales ~ targtpop + dispoinc + targtpop:dispoinc,
    qr = T)

Residuals:
     Min       1Q   Median       3Q      Max
-18.9159  -7.3276   0.4411   9.4115  20.8075

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       -12.47627  227.96284  -0.055    0.957
targtpop            0.53195    3.59800   0.148    0.884
dispoinc            6.09330   13.40395   0.455    0.655
targtpop:dispoinc   0.05288    0.20585   0.257    0.800

Residual standard error: 11.3 on 17 degrees of freedom
Multiple R-Squared: 0.9171,     Adjusted R-squared: 0.9024
F-statistic: 62.66 on 3 and 17 DF,  p-value: 2.128e-09
```

A shorthand version of the same thing is

```
> summary(lm(sales ~ targtpop*dispoinc, qr=T))
```

where $X1 * X2$ tells R to include both main effects X1 and X2 as well as the interaction X1:X2.

You can also omit the intercept

```
> summary(lm(sales ~ targtpop + dispoinc -1, qr=T))

Call:
lm(formula = sales ~ targtpop + dispoinc - 1, qr = T)

Residuals:
     Min       1Q   Median       3Q      Max
-17.2762  -8.1955  -0.3514   5.9801  23.3577

Coefficients:
         Estimate Std. Error t value Pr(>|t|)
targtpop   1.6217     0.1549  10.466 2.51e-09 ***
dispoinc   4.7504     0.5832   8.145 1.28e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.1 on 19 degrees of freedom
Multiple R-Squared: 0.9968,     Adjusted R-squared: 0.9964
F-statistic:  2917 on 2 and 19 DF,  p-value: < 2.2e-16
```

- **Diagnostic Plots**

To examine the appropriateness of regression model (2) , you can consider several diagnostic plots (Figure 4):

```
> e.star <- ls.diag(Dwaine.lsfit)$stud.res
> par(mfrow=c(2,3))

> boxplot(e.star, ylab="stud.res")
> title("Fig 1: Boxplot of the \n studentized residuals")

> plot(fitted(myfit),e.star)
> abline(h=0)
> title("Fig 2: studentized resids VS y.hat")

> plot(targtpop, e.star)
> abline(h=0)
```

```
> title("Fig 3: studentized resids VS X1")

> plot(dispoinc,e.star)
> abline(h=0)
> title("Fig 4: studentized resids VS X2")

> plot(targtpop*dispoinc,e.star)
> abline(h=0)
> title("Fig 5: studentized resids VS X1*X2")

> qqnorm(e.star)
> abline(0,1)
> title("Fig 6: qqplot of studentized resids")
```

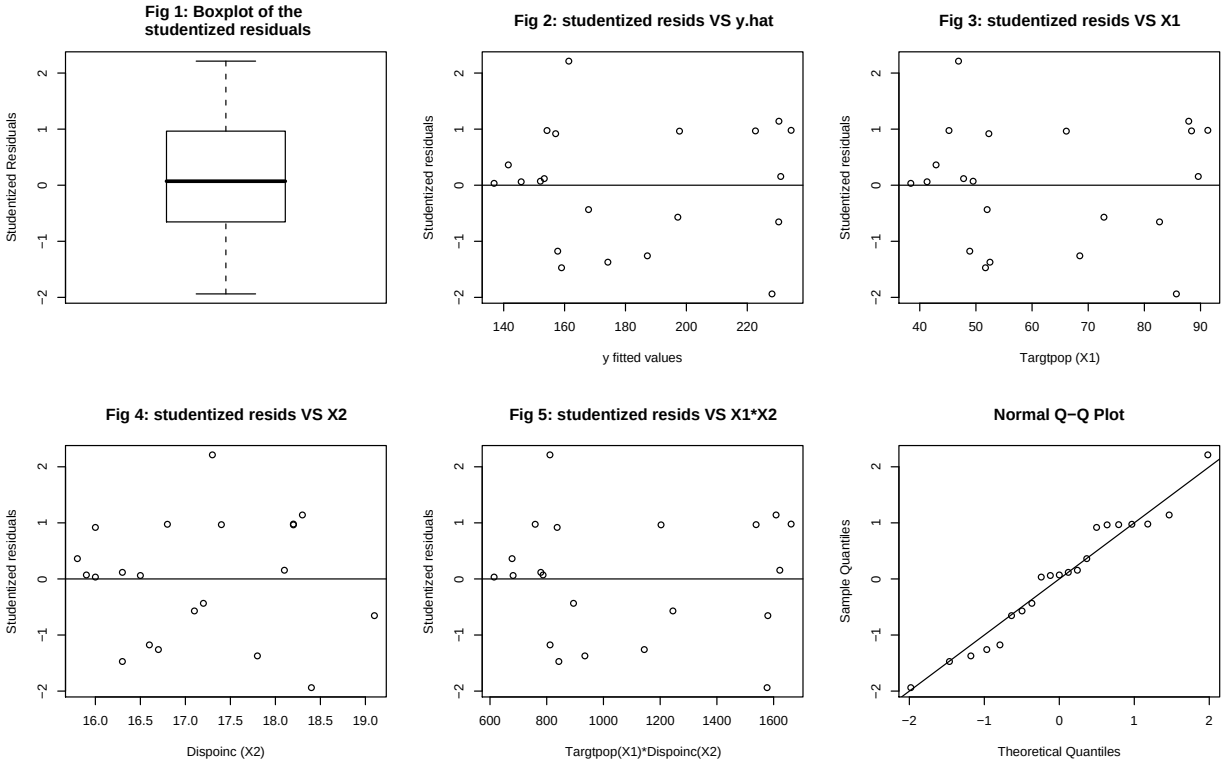

Figure 4: Diagnostic Plots for sales - Dwaine studios example

4.4. Detecting outliers

It is common practice to distinguish between two types of outliers. Outliers in the response variable represent model failure. Such observations are called *outliers*. Outliers with respect to the predictors are called *leverage points*. They can affect the regression model, too. Their response variables need not be outliers. However, they may almost uniquely determine regression coefficients. They may also cause the standard errors of regression coefficients to be much smaller than they would be if the observation were excluded.

The *ordinary or simple residuals* (*observed - predicted values*) are the most commonly used measures for detecting outliers. The ordinary residuals sum to zero but do not have the same standard deviation. Many other measures have been offered to improve on or complement simple residuals. *Standardized Residuals* are the residuals divided by the estimates of their standard errors. They have mean 0 and standard deviation 1. There are two common ways to caculate the standardized residual for the ith observation. One uses the *residual mean square error* from the model fitted to the *full dataset* (internally studentized residuals). The other uses the residual mean square error from the model fitted to the *all of the data except the ith observation* (externally studentized residuals). The externally standardized residuals follow a t distribution with $n-p-2$ degrees of freedom. They can be thought of as testing the hypothesis that the corresponding observation does not follow the regression model that describes the other observations.

In practice, the ordinary residuals are the most useful. While the standard deviations of the residuals are different, they are usually not different enough to matter when looking for outliers. They have the advantage of being in the same scale as the response.

4.5. Detecting Influential Observations

We have seen so far how to detect potential problems with model building. We will focus now on detecting potential observations that have significant impact on our model. There are several reasons that we need to detect influential observations. First, there are possibly data entry errors. Secondly influential observations may be of interest by themselves for us to study. After all, influential data points may badly skew our regression estimation.

We take up three measures of influence that are widely used in practice, each based on the omission of a single case to measure its influence. Kutner et al. (2005)

- **Influence on single fitted value: *DFFITS***

A useful measure of the influence that case i has on the fitted value $\widehat{Y}_i$ is given by:

$$DFFITS_i = \frac{\widehat{Y}_i - \widehat{Y}_{i(i)}}{\sqrt{MSE_{(i)}h_{ii}}} \tag{4}$$

The numerator of (4) is the difference between the fitted value $\widehat{Y}_i$ for the case ith case when all n cases are used in fitting the regression function and the predicted value $\widehat{Y}_{i(i)}$ for the ith case obtained when the ith case is omitted in fitting the regression function. The denominator of (4) is the estimated standard deviation of $\widehat{Y}_i$, but it uses the error mean square when the ith case is ommited in fitting the regression function for estimating the error variance σ^2. The denominator provides a standardization so that the value $DFFITS_i$ for the ith case represents the number of the estimated standard deviation of $\widehat{Y}_i$ that the fitted value $\widehat{Y}_i$ increases or decreases with the inclusion of the ith case in fitting the regression model.

For identifying influential cases, many authors suggest considering a case influential if the absolute value of $DFFITS$ exceeds 1 for small to medium data sets and $2\sqrt{p/n}$ for large data sets. Some analysts suggest Some analysts suggest investigating observations for which $|DFFITS_i|$ is greater than $2\sqrt{[(p+1)/(n-p-1)]}$

In R[9]:

```
> dffits.Dwaine<- dffits(myfit)
```

- **Influence on all fitted values: Cook's Distance**

In contrast to the $DFFITS$ measure in (4), which considers the influence of the ith case on the fitted value $\widehat{Y}_i$ for this case, Cook's distance measure considers the influence of the ith case on all n fitted values, Cook's distance measure, denoted by D_i, is an aggregate influence measure, showing the effect of the ith case on all n fitted values:

$$D_i = \frac{\sum_{j=1}^{n} \left(\widehat{Y}_j - \widehat{Y}_{j(i)}\right)}{p \cdot MSE} \tag{5}$$

Note that the numerator involves similar differences as in the $DFFITS$ measure, but here each of the n fitted values $\widehat{Y}_j$ is compared with the corresponding fitted value $\widehat{Y}_{j(i)}$ when the ith case is deleted in fitting the regression model. These differences are then squared and summed, so that the aggregate influence of the ith case is measured without regard to the signs of the effects. Finally, the denominator serves as a standardizing measure.

In R[10]

```
> cooks.Dwaine <- cooks.distance(myfit)
```

- **Influence on the regression coeficients: *DFBETAS***

A measure of the influence of the ith case on each regression coefficient β_k $(k = 0, 1, ..., p-1)$ is the diference between the estimated regression coefficient β_k based on all n cases and the regression coefficient obtained when the ith case is omitted, to be denoted by $\beta_{k(i)}$. When this difference is divided by an estimated of the standard deviation of β_k, we obtain the measure $DFBETAS$:

$$DFBETAS_{\beta_{k(i)}} = \frac{\beta_k - \beta_{k(i)}}{\sqrt{MSE_{(i)} c_{kk}}} \tag{6}$$

where c_{kk} is the kth diagonal element of $(X'X)^{-1}$.

The $DFBETAS$ value by its sign indicates wheter inclusion of a case leads to a increase or decrease in the estimated regression coeficient, and its absolute magnitude shows the size of the difference relative to the estimated standard deviation of the regression coefficient.

Large absolute value of $(DFBETAS)_{k(i)}$ is indicative of a large impact of the ith case on the kth regression coefficient. It is recommended for identifyng influential cases to consider a case influential if the absolute value of $DFBETAS$ exceeds 1 for small to medium data sets and $2/\sqrt{n}$ for large data sets.

In R,

[9] It is also possible to obtain the *DFFITS* statistic using `ls.diag()`, see •, by `Dwaine.lsfit$dfits`

[10] It is also obtained by `Dwaine.lsfit$cooks`, see `ls.diag()` •

```
> dfbetas.Dwaine <- dfbetas(myfit)
```

Leverage points do not necessarily correspond to outliers. There are a few reasons why this is so. First, an observation with sufficiently high leverage might exert enough influence to drag the regression equation close to its response and mask the fact that it might otherwise be an outlier.

In R, the function `influence.measures()` computes regression deletion diagnostics

```
> infl.Dwaine <- influence.measures(myfit)
Influence measures of
         lm(formula = sales ~ targtpop + dispoinc, qr = T) :

     dfb.1_ dfb.trgt dfb.dspn   dffit cov.r   cook.d    hat inf
1  -0.34451 -0.33922  0.35063 -0.4689 1.034 7.10e-02 0.1218
2  -0.09131 -0.23149  0.13006  0.3328 1.126 3.70e-02 0.1044
3   0.01082  0.27824 -0.05368  0.4488 1.219 6.73e-02 0.1737
4   0.01579 -0.00374 -0.01198  0.0361 1.296 4.60e-04 0.0863
5  -0.60147 -0.81264  0.68954  0.9727 0.663 2.59e-01 0.1620
...   ... ...   ...    ...      ...     ...
17 -0.00200 -0.01389  0.00483  0.0221 1.340 1.72e-04 0.1150
18 -0.27757 -0.06871  0.24048 -0.4554 0.907 6.49e-02 0.0873
19  0.00409  0.04324 -0.01040  0.0682 1.409 1.64e-03 0.1611
20  0.33339  0.12840 -0.31721 -0.4068 1.528 5.70e-02 0.2788   *
21  0.29455  0.14115 -0.27291  0.3666 1.190 4.52e-02 0.1373

> summary(infl.Dwaine)

Potentially influential observations of
         lm(formula = sales ~ targtpop + dispoinc, qr = T) :

   dfb.1_ dfb.trgt dfb.dspn dffit cov.r   cook.d hat
20  0.33   0.13    -0.32    -0.41  1.53_*  0.06   0.28
```

Figure 5 presents four different plots for regression diagnostics. The first panel shows residuals versus fitted values. The second is a $Q-Q$ normal distribution plot of standardized residuals. The third plot is of the square root of the absolute value of the standardized residuals; this reduces the skewness of the distribution and makes it much easier to detect if there might be a trend in the dispersion. The fourth plot is of Cook's distance

```
> windows(width=12, height=12)
> par(mfrow=c(2,2), mgp=c(1.5, 0.5,0), mar=c(3,3,2,1))
> plot(myfit, pch=16, cex=0.5, cex.lab=1.5, cex.id=1)
```

The function `rstandard(myfit)` gives the standardized residuals discused above. There is also `rstudent(myfit)`, which gives *leave-out-residuals*, in which the fitted value is calculated omitting the current point; if the model is correct, then these will follow a t distribution.

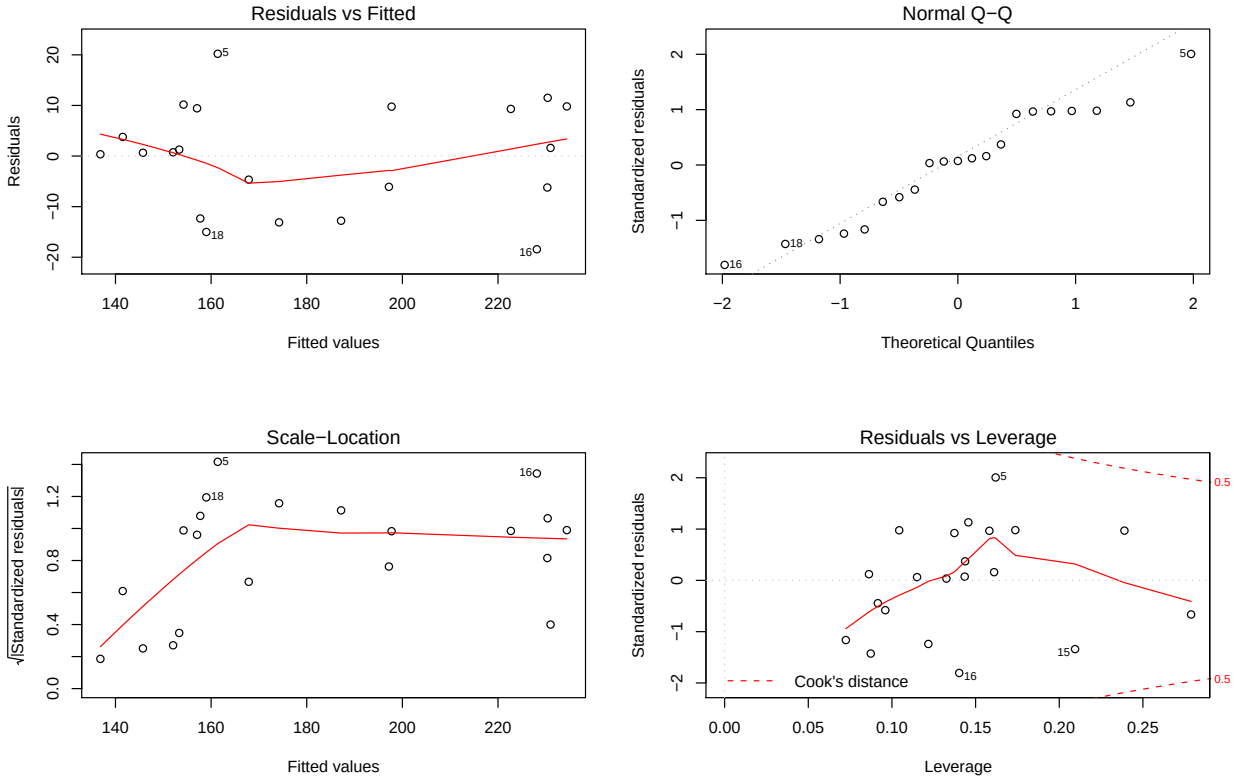

Figure 5: Standard diagnostic plots - Dwaine studios example

4.6. Leverage and the hat matrix

Recall that the leverage of the ith observation is the ith diagonal element of the so-called *hat matrix* that can be derived from the model matrix. Large values represent high leverage[11].

h_{ii} is a measure of how much Y_i is contributing to the prediction of $\widehat{Y}_i$. This depends on the distance between the X values for the ith case and the means of the X values. Observations with extreme values for the predictors will have more influence. I always holds:

$$0 \leq h_{ii} \leq 1$$

and

$$\sum h_{ii} = p$$

where p is the number of regression parameters in the regression funtion including the intercept term. In addition, it can be shown that h_{ii} is a measure of the distance between the X values for the i^{th} case and the means of the X values for all n cases.

A large value of h_{ii} suggests that the ith case is distant from the center of all $X's$. The average value is $\frac{p}{n}$. Values far from this average (say, twice as large) point to cases that

[11]For more detailes see Venables and Ripley (1997)

should be examined carefully because they may have a substantial influence on the regression parameters.

The diagonal h_{ii} in this context is called the *leverage* (in terms of the X values) of the ith case.

If the ith case is outlying in terms of its X observations and therefore has a large leverage value h_{ii}, it exercises substantial leverage in determining the fitted value $\widehat{Y}_i$. This is so for the following reasons:

1. The fitted value $\widehat{Y}_{ii}$

```
> plot(Dwaine.ls.diag$hat, myfit$residuals,
+ xlab="hat matrix diagonal", ylab="residuals", pch=20, lwd=3)
```

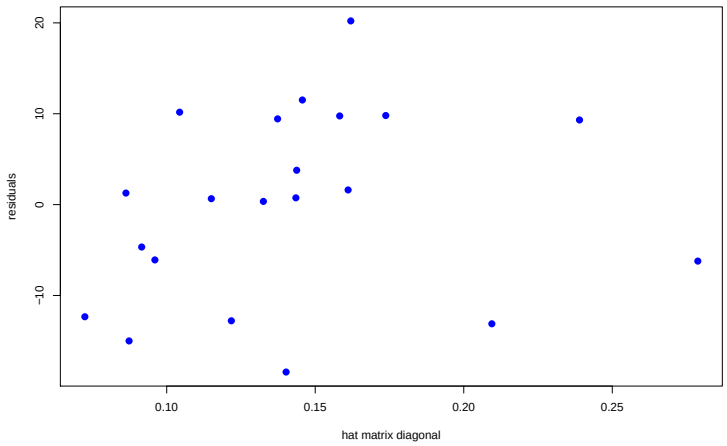

Figure 6: Leverage points VS Residuals - Dwaine example

4.7. Variable Selection: Selection of Predictor Variables

- Response variable: Y
- Set of potential variables: $X_1, X_2, ..., X_{p-1}$

$\Rightarrow$ **GOAL:** Find a 'good' new subsets of predictor variables needed for estimating and interpreting the response variable. (see Faraway, 2002, chap. 10)

Questions:

1. Why we are interested in some subsets but not the poll of potential predictor variables?
   - We want to explain the data in a simpler way - redundant or unnecessary predictor variables should be removed.
2. What is a *'good'* subset?

- It is based on a criterion we employ.

3. From where we start our search for optimum subsets?
   - There are various procedures, including:
     - *All-possible-regressions* procedure *Step-type* procedures (Forward selection, backward elimination, stepwise selection, et al.)
   - Does variable selection guarantee that the chosen subsets of predictor variables are correct for the population?
     - Maybe not. For instance, the existence of multicollinearity would lead us to choose a wrong subset.

4.8. Multicollinearity

In multiple regression analysis, the nature and significance if the relations between the predictor or explanatory variables and the response variable are often of particular interest.

Some later variables may be exact, or very nearly, linear combinations of earlier variables. Technically, this is known as multicollinearity. For each multicollinear relationship, there is one redundant variable.

- What we want:
  - the response variable is highly correlated with the predictor variables.
- What we do not want:
  - the predictor (or explanatory) variables themselves are highly correlated.

- **How to identify the existence of multicollinearity?**

Diagnostic tools:

1. Correlation matrix of the predictor variables (a pairwise-checking approach)
2. Scatterplot matrix of the predictor variables (a pairwise-checking approach)
3. Eigensystem analysis of $X'X$
4. Variance Inflation Factor (VIF)

- **Body fat example**

```
> data <- read.table("bodyfat")

> y  <- data[,4] # body fat
> x1 <- data[,1] # triceps skinfold thickness
> x2 <- data[,2] # thigh circumference
> x3 <- data[,3] # midarm circumference
```

The correlation matrix of X_1, X_2, and X_3 is given by

- ***Correlation matrix of the predictor variables:***

```
> cor(cbind(x1,x2,x3))
          x1        x2        x3
x1 1.0000000 0.9238425 0.4577772
x2 0.9238425 1.0000000 0.0846675
x3 0.4577772 0.0846675 1.0000000
```

Here, X_1 and X_2 are highly correlated. It can also be seen from the scatterplot matrix of X_1, X_2, X_3, (See Fig. 7)

- ***Scatterplot matrix of the predictor variables:***

```
> pairs(cbind(x1,x2,x3))
# or
> pairs(cbind(x1,x2,x3),panel=function(x,y){points(x,y);
+ abline(lm(y~x),col=2)})
```

It is interesting to see that X_3 is highly correlated with X_1 and X_2 together from

```
> summary(lm(x3~x1+x2))

Call:
lm(formula = x3 ~ x1 + x2)

Residuals:
     Min       1Q   Median       3Q      Max
-0.58200 -0.30625  0.02592  0.29526  0.56102

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 62.33083    1.23934   50.29   <2e-16 ***
x1           1.88089    0.04498   41.82   <2e-16 ***
x2          -1.60850    0.04316  -37.26   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

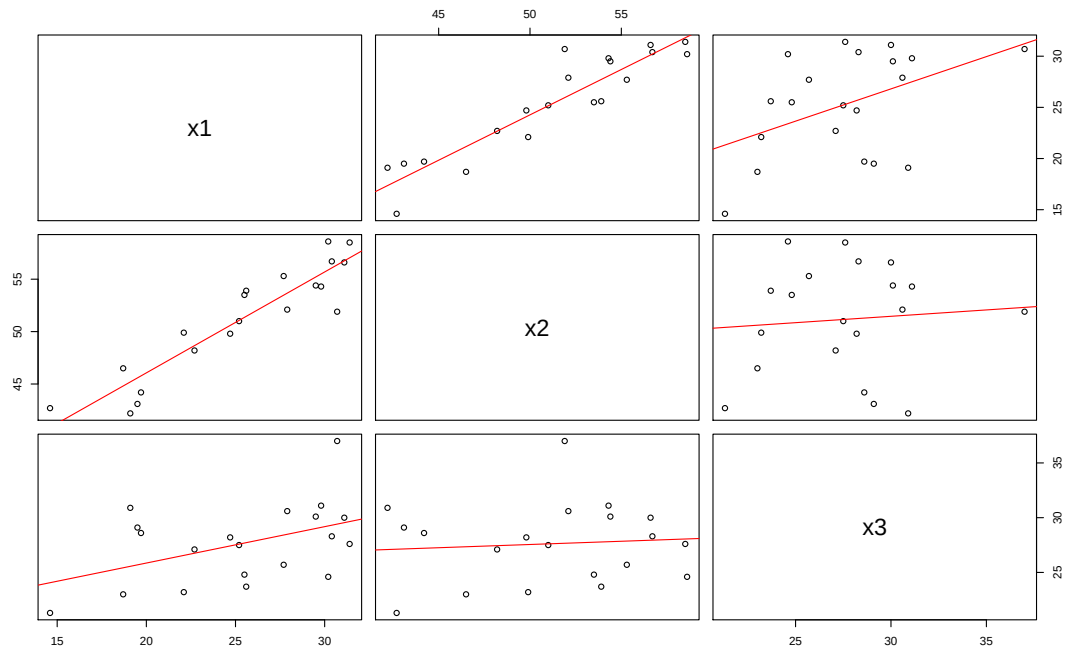

Figure 7: Scatterplot matrix of X variables - Body Fat example

```
Residual standard error: 0.377 on 17 degrees of freedom
Multiple R-Squared: 0.9904,     Adjusted R-squared: 0.9893
F-statistic: 880.7 on 2 and 17 DF,  p-value: < 2.2e-16
```

- ***Eigensystem analysis of*** $X'X$***:***

  The eigenvalues of $X'X$, say $\lambda_1, ..., \lambda_p$ can be used to measure the extent of multicollinearity in the data. Obviously, $\lambda_1, ..., \lambda_p$ are non-negative and the extreme case is one or more $\lambda_k = 0$.

  If there are one or more near linear dependencies in the data, then one or more eigenvalues will be close to zero.

  On the other hand, one or more small eigenvalues imply that there are near linear dependencies among the columns of X.

```
> Xmatrix <- cbind(rep(1,length(y)), x1, x2, x3)
> XprimeX <- t(Xmatrix)%*%Xmatrix
> eigen(XprimeX)

$values
[1] 8.129024e+04 2.942499e+02 1.198158e+02 6.165867e-04
```

```
$vectors
            [,1]        [,2]        [,3]        [,4]
[1,] -0.01560414  0.00965285 -0.03861731  0.99908560
[2,] -0.40266387 -0.18165656  0.89663553  0.03012348
[3,] -0.80607618 -0.39389683 -0.44093021 -0.02582705
[4,] -0.43342763  0.90097337 -0.01157473 -0.01592177
```

In this case, one of four eigenvalues of $X'X$ is close to 0. One of this side effects is to produce the least-square estimators of the regression coefficients that are too large in absolute value.

The **condition index** summarizes this idea, and a common rule of thumb is that a condition index over 15 indicates a possible multicollinearity problem and a condition index over 30 suggests a serious multicollinearity problem.

$$\text{Cond. Index} = \sqrt{\frac{\text{largest eigenvalue of } X'X}{\text{smallest eigenvalue of } X'X}} = \sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}} \tag{7}$$

```
> max.eig <- max(eigen(XprimeX)$values)
> min.eig <- min(eigen(XprimeX)$values)
> sqrt(max.eig/min.eig)
[1] 11482.12
```

- ***Variance Inflation Factor (VIF)*** The variance inflation factor (VIF) quantifies the effect of the correlation with other variables in inflating the standard error of a regression coefficient.

$$VIF_k = \frac{1}{1-R_k^2}, \quad k = 1, 2, ..., p-1 \tag{8}$$

where R_k^2 is the coefficient of multiple determination when X_k is regressed on the $p-2$ other X variables in the model.

  - When $R_k^2 = 0$, i.e., X_k is not linearly related to the other X variables, $VIF_k = 1$;
  - when $R_k^2 > 0, VIF_k > 1$;
  - when $R_k^2 \to 1, VIF_k \to +\infty$.

A maximum VIF value in excess of 10 is frequently taken as an indication of the existence of multicollinearity.

In the *Body fat example*, to find VIF_1, in `R` type:

```
> summary(lm(x1 ~ x2+x3))

Call:
lm(formula = x1 ~ x2 + x3)

Residuals:
     Min       1Q   Median       3Q      Max
-0.31341 -0.16002 -0.00444  0.14890  0.31944

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -33.013062   0.545938  -60.47   <2e-16 ***
x2            0.855480   0.008773   97.51   <2e-16 ***
x3            0.526544   0.012592   41.82   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1995 on 17 degrees of freedom
Multiple R-Squared: 0.9986,     Adjusted R-squared: 0.9984
F-statistic:  6017 on 2 and 17 DF,  p-value: < 2.2e-16


> vif.1 <- (1-summary(lm(x1~x2+x3))$r.squared)^(-1)
> vif.1
[1] 708.8429
```

Similarly,

```
> vif.2 <- (1-summary(lm(x2~x1+x3))$r.squared)^(-1)
> vif.2
[1] 564.3434
> vif.3 <- (1-summary(lm(x3~x1+x2))$r.squared)^(-1)
> vif.3
[1] 104.606
```

In DAAG package[12], there is the `vif( )` function for the calculations

```
> library(DAAG)
> body.fat <- lm(y ~x1+x2+x3)
> vif(body.fat)
    x1     x2     x3
708.84 564.34 104.61
> mean(vif(body.fat))
[1] 459.2633
```

[12] *Data Analysis and Graphics Package - http://cran.r-project.org/src/contrib/Descriptions/DAAG.html*

Table 1 contains the estimated standardized regression coefficients and the VIF values for the body fat example with three predictor variables. The maximum of the VIF values is 708.84 and their mean value is $\overline{VIF} = 459.26$. Thus, the expected sum of the squared errors in the least squares standardized regression coefficients is nearly 460 times as large as it would be if th X variables were uncorrelated. In addition, all three VIF values greatly exceed 10, which indicates that a serious multicollinearityu problems exist.

Variable	$\widehat{\beta}_k$	VIF_k
X_1	4.334	708.84
X_2	-2.857	564.34
X_3	-2.186	104.61

Table 1: Variance Inflation Factor - Body fat example with tree predictor variables

A limitation of variance inflations factors for detecting multicollinearities is that they cannot distinguish between several simultaneous multicollinearities.

- **Multicollinearity remedial measures**

When the sample data for regression exhibit multicollinearity, the least squares estimates of β coefficients may be unstable. Measures:

- Drop one or more predictor variables to decrease the multicollinearity.
- Add some cases that break the pattern of multicollinearity.
- Principal components regression. Form one or more composite indexes based on the highly correlated predictor variables, an index being a linear combination of the correlated predictor variables.
- Ridge regression.

Ridge regression

Ridge regression is a technique developed for stabilizing the regression coefficients in the presence of multicollinearity. The ridge estimator is found by solving a slightly modified version of the normal equations, and defined by:

$$\widehat{\beta}_R = (X'X + \lambda I)^{-1} X'Y, \tag{9}$$

where I is an identity matrix, and $\lambda \geq 0$ is a constant called a *biasing constant*. Note that the constant λ reflects the amount of bias in the estimators. When $\lambda = 0$, equation (9) reduces to the ordinary least squares regression coefficients, as given in (3).

Ridge regression modifies the method of least squares to allow biased estimators of the regression coefficients. When an estimator has only a small bias and is substantially more

precise than an unbiased estimator, it may well be the preferred estimator since it will have a large probability of being close to the true parameter value.

How to choose an appropiate λ?

A commonly used method is based on examining

- The *ridge trace*: a simultaneous plot for the values of the $p-1$ estimated ridge standardized regression coefficients for different values of $\lambda \in [0,1]$. Extensive experience has indicated that the estimated regression coefficients $\widehat{\beta}_R$ may fluctuate widely as λ is changed slightly from 0 and some may even change signs. Gradually, however, these wide fluctuations cease and the magnitudes of the regression coefficients tend to move slowly toward zero as λ is increased further. See Figure 8.

```
> body <- cbind(y,x1,x2,x3)
> body <- as.data.frame(body) # convert the matrix into a data.frame
> library(MASS) # load MASS library for lmridge function
> body.ridge <- lm.ridge(y~x1+x2+x3, body,lambda = seq(0,1,0.001))
> body.ridge <- lm.ridge(y~., body,lambda = seq(0,1,0.001))

> matplot(body.ridge$lambda,t(body.ridge$coef),
+ type="l",xlab=expression(lambda),ylab=expression(hat(beta)))
> abline(h=0,lwd=2)
> select(body.ridge)

modified HKB estimator is 0.008505093
modified L-W estimator is 0.3098511
smallest value of GCV  at 0.019

> body.ridge$coef[, body.ridge$lambda == 0.02]

body.ridge$coef[,1]
```

- the VIF values for different values of $\lambda \in [0,1]$: choose the smallest value of λ where it is deemed that the regression coefficients first become stable in the ridge trace and the VIF values have become sufficiently small.

4.9. A strategy for fitting a multiple regression model

An essential first step is to make a careful scrutiny of the explanatory variables. This may lead to any or all of:

- Transformation of some or all variables.
- Replacement of existing variables by newly constructed variables that are a better summary of the information.

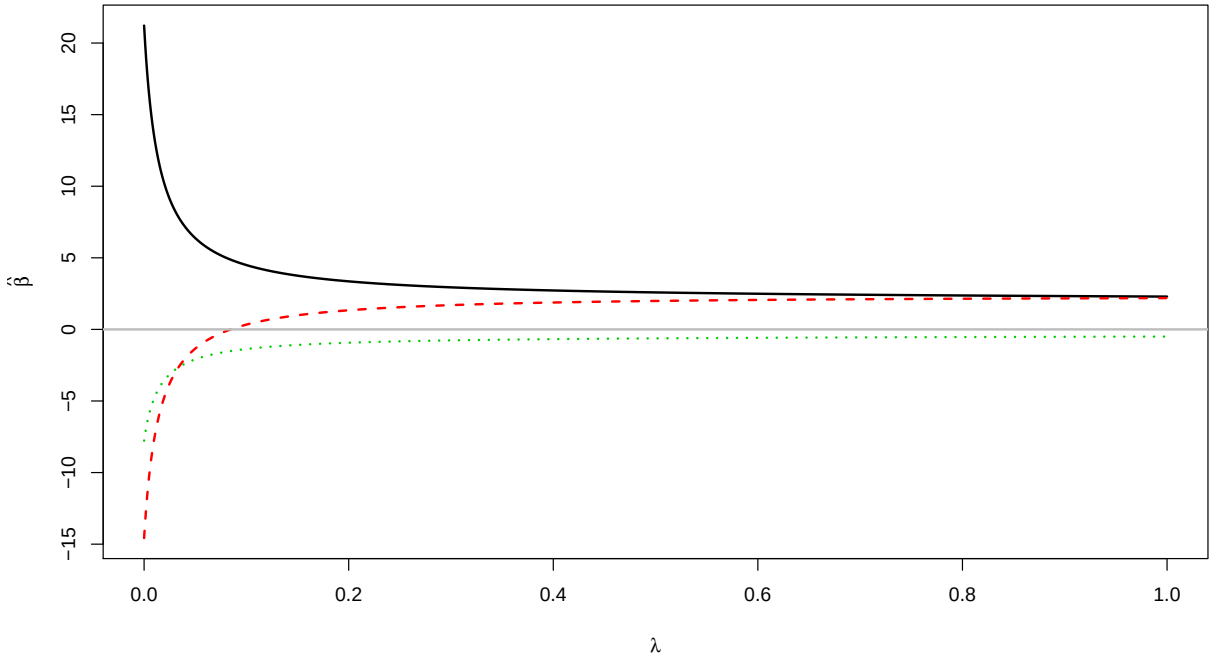

Figure 8: Plot of regression coefficients VS $\lambda \in [0, 1]$

- Omission of some variables.

Here are steps that are reasonable to follow. They involve examination of the distribution of values of explanatory variables, and of the pairwise scatterplots.

- Examine the distribution of each of the explanatory variables, and of the dependent variable. Look for any instances where distributions are highly skew, or where there are outlying values. Check whether any outlying values may be mistakes.
- Examine the scatterplot matrix involving all the explanatory variables. Look first for evidence of non-linearity in the plots of explanatory variables against each other. Look for values that appear as outliers in any of the pairwise scatterplots.
- Note the ranges of each of the explanatory variables. Do they vary sufficiently to affect values of the dependent variable? If values of a variable are essentially constant, there can be no apparent effect from varying levels of that variable.
- Where the distribution is skew, consider transformations that may lead to a more symmetric distribution. If some pairwise plots show evidence of non-linearity, examine the possible use of transformation(s) to give more nearly linear relationships.
- Look for pairs of explanatory variables that seem highly correlated.

If relationships between explanatory variables are approximately linear, perhaps after transformation, it is then possible to interpret plots of predictor variables against the response

variable with confidence. Contrary to what might be expected, this is more helpful than looking at the plots of the response variable against explanatory variables. If there is non-linearity in these plots, examine whether a transformation of the dependent variable would help.

It is not always possible to obtain pairwise linear scatterplots. This can create problems both for the interpretation of the diagnostic plots for the fitted regression relationships and for the interpretation of the coefficients in the fitted regression equation.

4.10. Model Fitting

Here is a suggested procedure for fitting and checking the regression relationship (Figure 5):

- Fit the multiple regression equation.
- Examine the Cook's distance statistics. If it is helpful, examine standarized versions of the $drop - 1$ coefficients directly, using `dfbetas()`. It may be necessary to delete influential points that, although residuals are large, have little influence on the fitted model.
- Plot residuals against fitted values. Check for patterns in the residuals.
- For each explanatory variable, do a component plus residual plot, in order to check whether any of the explanatory variables require transformation.

5. Multiple Logistic Regression

5.1. Model

$Y_{i1} = 1, ..., n$ are independent Bernoulli random variables with

$$E(Y_i) = \pi_i = \frac{exp(\beta_0 + \beta_1 X_{i1} + ... + \beta_{p-1} X_{i,p-1})}{1 + exp(\beta_0 + \beta_1 X_{i1} + ... + \beta_{p-1} X_{i,p-1})}$$

where $\beta_0, \beta_1, ..., \beta_{p-1}$ are parameters, and X observations are assumed to be known constants. Or

$$E[Y] = \frac{exp(X'\beta)}{1 + exp(X'\beta)}.$$

The logit transformation:

$$\pi' = Ln\left(\frac{\pi}{1-\pi}\right)$$

now leads to the logit response function, or linear predictor:

$$\pi' = X'\beta$$

5.2. Disease outbreak example

In a health study to investigate an epidemic outbreak of a disease that is spread by mosquitoes, individuals were randomly sampled within two sectors in a city to determine if the person had recently contracted the disease under study. This was ascertained by the interviewer, who asked pertinent questions to asses whether certain specific symptoms associated with the disease were present during the specified period. The response variable Y was coded 1 if this disease was determined to have been present, and 0 if not.

Three predictor variables were included in the study, representing known or potential risk factors. They are age, socioeconomic status of household, and sector within city. Age (X_1) is quantitative variable.

City sector is also a categorical variable. Since there were only two sectors in the study, one indicator variable (X_4) was used, defined so that $X_4 = 0$ for sector 1 and $X_4 = 1$ for sector 2.[13]

```
> disease <- read.table("diseaseoutbreak")
> y   <- disease[,5][1:98] # disease status
> x1  <- disease[,2][1:98] # age
> x4  <- disease[,4][1:98] # city sector
> soc <- disease[,3][1:98] # socioeconomic status
```

Here socioeconomic status is a categorical variable with three levels. It is represented by two indicator variables (X_2 and X_3), as follows:

[13]The reason why the upper socioeconomic class was chosen as the reference class is that it was expected that this class would have the lowest disease rate among the socioeconomic classes.

		Status		City Sector
Case (i)	**Age** (X_{i1})	$\mathbf{X}_{i2}$	$\mathbf{X}_{i3}$	$\mathbf{X}_{i4}$
1	33	0	0	0
2	35	0	0	0
3	6	0	0	0
4	60	0	0	0
⋮	⋮	⋮	⋮	⋮
98	35	0	1	0

Table 2: Portion of Model - Building Data Set - Disease Outbreak Example.

Class	$\mathbf{X}_2$	$\mathbf{X}_3$
Upper	0	0
Middle	1	0
Lower	0	1

The first 98 cases were selected for fitting the model. Table 2 in columns 1-5 contains the data for a portion of the 98 cases used for fitting the model. Note the use of the indicator variables as just explained for the two categorical variables. The primary purpose of the study was to assess the strength of the association between each of the predictor variables and the probability of a person having contracted disease.

```
> x2 <- 1:length(y)
> x2[soc==1] <- 0
> x2[soc==2] <- 1
> x2[soc==3] <- 0

> x3 <- 1:length(y)
> x3[soc==1] <- 0
> x3[soc==2] <- 0
> x3[soc==3] <- 1

> x4[x4==1] <- 0
> x4[x4==2] <- 1
```

A first-order multiple logistic regression model with the three predictor variables was considered *a priori* to be reasonable:

$$E[Y] = [1 + exp\,(-X'\beta)]^{-1}$$

where:

$$X'\beta = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4$$

This model was fitted by the method of maximum likelihood to the data for the 98 cases.

```
> fit1 <- glm(y~x1+x2+x3+x4, family=binomial(link="logit"))
Call:
glm(formula = y ~ x1 + x2 + x3 + x4, family = binomial(link = "logit"))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6552  -0.7529  -0.4788   0.8558   2.0977

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.31293    0.64259  -3.599 0.000319 ***
x1           0.02975    0.01350   2.203 0.027577 *
x2           0.40879    0.59900   0.682 0.494954
x3          -0.30525    0.60413  -0.505 0.613362
x4           1.57475    0.50162   3.139 0.001693 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)
```

The note about the dispersion parameter is related to the fact that the binomial variance depends entirely on the mean. There is no scale parameter like the variance in the normal distribution.

```
    Null deviance: 122.32  on 97  degrees of freedom
Residual deviance: 101.05  on 93  degrees of freedom
AIC: 111.05
```

`Residual deviance` corresponds to the residual sum of squares in ordinary regression analyses which is used used to estimate the standard deviation of the observations in known, and you can therefore use the deviance in a test for model especification.

The `AIC` (Akaike information Criterion) is a measure of goodness of fit which takes the number of fitted parameters into account.

`R` is reluctant to associate $p - value$ with the deviance. Just as well, because no exact $p - value$ can be found, only an approximation that is valid for large expected counts. The asymptotic distribution of the residual deviance is a χ^2 distribution with the stated degrees of freedom.

```
Number of Fisher Scoring iterations: 4
```

This refers to the fitting procedure. Normally, `glm` halts the fitting procedure if the number of iterations exceeds 10, but it is possible to configure the limit.

A table of correlations between parameter estimates can be obtained via the optional argument `corr=T` to `summary`, (i.e. `summary(fit1,corr=T)`).

```
Correlation of Coefficients:
   (Intercept) x1    x2    x3
x1 -0.66
x2 -0.48        0.14
x3 -0.52        0.09  0.41
x4 -0.51        0.05  0.04  0.21

> beta.fit1 <- coefficients(fit1)
> beta.fit1
(Intercept)          x1          x2          x3          x4
-2.31293482  0.02975009  0.40879024 -0.30525456  1.57474923
```

The estimated logistic response function is:

$$\widehat{\pi} = [1 + exp(2.31293 - .02975X_1 - .40879X_2 + 0.30525X_3 - 1.57475X_4)]^{-1}$$

The interpretation of the estimated regression coefficients in the fitted first-order multiple logistic response function parallels that for the simple logistic response function: $exp(b_k)$ is the estimated odds ratio for predictor variable X_k.

```
> fit1$fitted
```

contains the fitted values $\widehat{\pi}_i$, calculated as for instance, the estimated mean response for case $i = 1$, where $X_{11} = 33, X_{12} = 0, X_{13} = 0, X_{14} = 0$, is:

$$\widehat{\pi}_1 = [1 + exp(2.31293 - .02975 \times 33 - .40879 \times 0 + 0.30525 \times 0 - 1.57475 \times 0)]^{-1} = .209$$

```
> fit1$fitted[1]
        1
0.2089640
> fit1$residuals[1:5]
        1         2         3         4         5
-1.264162 -1.280358 -1.118302 -1.589826  9.025881

> fit1$deviance
[1] 101.0542
```

Deviance tables corresponds to ANOVA tables for multiple regression analyses and are generated like this with the `anova` function. To undertand the output, you need to know about the *'deviances'*. A deviance has a role very similar to a sum of squares in regression. Thus we have:

Regression	**Logistic Regression**
degrees of freedom	degrees of freedom
Sum of Squares	Deviance
Mean Sum of Squares (divide by d.f.)	Mean Deviance (divide by d.f)
We prefer models with a small mean Residual Sum of Squares	We prefer models with a small Mean Deviance

Estimated Coefficients, Std. Deviation and Odds Ratios			
Regression Coeff.	Estimated Reg. Coeff.	Estimated Std. Dev.	Estimated Odds Ratio
β_0	-2.3129	.64259	-
β_1	.02975	.01350	1.030
β_2	-.30525	.60413	.7369
β_3	.40879	.59900	.737
β_4	1.57475	.50162	4.829

Table 3: MLE estimates of logistic regression function - Disease Outbreak Function Example

```
> anova(fit1, test="Chisq")
Analysis of Deviance Table

Model: binomial, link: logit

Response: y

Terms added sequentially (first to last)

     Df Deviance Resid. Df Resid. Dev P(>|Chi|)
NULL                    97    122.318
x1    1    7.405        96    114.913     0.007
x2    1    1.804        95    113.109     0.179
x3    1    1.606        94    111.502     0.205
x4    1   10.448        93    101.054     0.001

> exp(beta.fit1)
(Intercept)          x1          x2          x3          x4
 0.09897037  1.03019705  1.50499600  0.73693576  4.82953038
```

Another way of doing this is treat the variables X_4 and *soc* as factors:

```
> x4 <-factor(x4)
> soc <- factor(soc)

> fit2 <- glm(y~x1+soc+x4, family=binomial(link="logit"))

> summary(fit2)

> beta.fit2 <- coefficients(fit2)
> beta.fit2

(Intercept)          x1        soc2        soc3          x4
-2.31293482  0.02975009  0.40879024 -0.30525456  1.57474923

> exp(beta.fit2)
```

```
(Intercept)          x1        soc2        soc3         x42
 0.09897037  1.03019705  1.50499600  0.73693576  4.82953038
```

The result is the same as in `fit1`.

- **Prediction**

The `predict` function works for generalized linear models too[14]:

```
> predict.glm(fit1,data.frame(x1=35,x2=0,x3=0,x4=1,level=0.95,
+ interval="prediction",type="response"))
```

To obtain the $(n \times 1)$ *vector of linear predictors* $\widehat{\pi}$.

```
> predict.glm(fit1,level=0.95,interval="prediction",type="response") # or
> pred.fit <- as.matrix(predict(fit1, type="response"))
> pred.fit

          [,1]
1   0.20896395
2   0.21896953
3   0.10579477
    .......
97  0.09187623
98  0.17122984
```

5.3. Logistic Regression Diagnostics

In this section we take up the analysis of residuals and the identification of influencial cases for logistic regression. (See Weisberg (2005))

- **Logistic Regression Residuals**

Residual analysis for logistic regression is more difficult than for linear regression models because the responses Y_i can take on only the values 0 and 1. Consequently, the ith ordinary residual, e_i will assume one of two values:

$$e_i = \begin{cases} 1 - \widehat{\pi}_i & \text{if } Y_i = 1 \\ -\widehat{\pi}_i & \text{if } Y_i = 0 \end{cases}$$

In `R`, the ordinary residuals e_i are:

```
> e <- residuals(fit1, "response");
> e[1:5]
         1          2          3          4          5
-0.2089640 -0.2189695 -0.1057948 -0.3710000  0.8892091
```

[14] for more details, in `R`, type `?predict.glm`

The ordinary residuals will not be normally distributed and, indeed, their distribution under the assumption that the fitted model is correct is unknown. Plots of ordinary residuals against fitted values or predictor variables will generally be uninformative.

1. **Pearson Residuals:** The ordinary residuals can be made more comparable by dividing them by the estimated standard error of Y_i, namely, $\sqrt{\widehat{\pi}_i(1-\widehat{\pi}_i)}$. The resulting *Pearson residuals* are given by:

$$r_{p_i} = \frac{Y_i - \widehat{\pi}_i}{\sqrt{\widehat{\pi}_i(1-\widehat{\pi}_i)}} = \frac{e_i}{\sqrt{\widehat{\pi}_i(1-\widehat{\pi}_i)}} \tag{10}$$

   Pearson residuals are defined to be the standardized difference between the observed frequency and the predicted frequency. It measures the relative deviations between the observed and fitted values. That's why they are directly related to Pearson chi-square goodness of fit statistic.

   In the disease outbreak example (Section 5.2) , the resulting Pearson residuals, r_{p_i} for $i = 1, 2, 3, 4, 5$ are:

```
> rp <- residuals(fit1, "pearson");
> rp[1:5]
        1          2          3          4          5
-0.5139697 -0.5294901 -0.3439644 -0.7680007  2.8330216
```

2. **Studentized Pearson Residuals:** The Pearson residuals do not have unit variance since no allowance has been made for the inherent variation in the fitted value $\widehat{\pi}_i$. A better procedure is to divide the ordinary residuals by their estimated standard deviation. This value is approximated by $\sqrt{\widehat{\pi}_i(1-\widehat{\pi})(1-h_{ii})}$, where h_{ii} is the ith diagonal element of the $n \times n$ estimated hat matrix for logistic regression.

$$H = \widehat{W}^{\frac{1}{2}} X \left(X' \widehat{W} X \right)^{-1} X' \widehat{W}^{\frac{1}{2}}$$

   Here, $\widehat{W}$ is the $n \times n$ diagonal matrix with elements $\widehat{\pi}(1-\widehat{\pi})$, X is the usual $n \times p$ design matrix, and $\widehat{W}^{\frac{1}{2}}$ is a diagonal matrix with diagonal elements equal to the square roots of those in $\widehat{W}$. The resulting *studentized Pearson residuals* are defined as:

$$r_{SP_i} = \frac{r_{p_i}}{\sqrt{1-h_{ii}}} \tag{11}$$

   Recall that for multiple linear regression (Section 4), the hat matrix satisfies the matrix expression $\widehat{Y} = HY$. The hat matrix for logistic regression is developed in analogous fashion; it satisfies approximately the expression $\widehat{\pi}' = HY$, where $\widehat{\pi}'$ is the $(n \times 1)$ vector of linear predictors.

```
> h <- hatvalues(fit2)  # hat matrix diagonal elements
> h[1:5]
         1          2          3          4          5
0.03873504 0.04040802 0.03319736 0.08951342 0.02518569
```

Calculating (11) in R:

```
> rsp <- rp/sqrt(1-h)  #  Studentized Pearson Residuals
> rsp[1:5]
         1          2          3          4          5
-0.5242229 -0.5405235 -0.3498200 -0.8048683  2.8693858
```

3. **Deviance Residuals:** measure the disagreement between the maxima of the observed and the fitted loglikelihood functions. In multiple linear regression, the residual sum of squares provides the basis for tests for comparing mean functions. Since the logistic regression uses the maximal likelihood principle, the goal in logistic regression is to minimize the sum of deviance residuals, which is often called G^2. For binary data the model deviance is:

$$DEV(X_0, ..., X_{p-1}) = -2 \sum_{i=1}^{n} [Y_i \log_e(\widehat{\pi}_i) + (1 - Y_i) \log_e(1 - \widehat{\pi}_i)] \tag{12}$$

The *deviance residual* for case i, denoted by dev_i is defined as the signed square root of the contribution of the ith case to the model deviance DEV in (12):

$$dev_i = sign(Y_i - \widehat{\pi}_i) \sqrt{-2 [Y_i \log_e(\widehat{\pi}_i) + (1 - Y_i) \log_e(1 - \widehat{\pi}_i)]} \tag{13}$$

where the sign is positive when $Y_i \geq \widehat{\pi}_i$. Thus the sum of the squared deviance residuals equals the model deviance in (12) in:

$$\sum_{i=1}^{n} (dev_i)^2 = DEV(X_0, ..., X_{p-1})$$

Therefore the square of each deviance residual measures the contribution of each binary response to the deviance goodness of fit statistic (12).

In R, (12) is calculated by:

```
> residuals(fit2, "deviance")[1:5]
         1          2          3          4          5
-0.6847069 -0.7030521 -0.4729058 -0.9629372  2.0976704
```

Table 4, list in columns:

- The response Y_i,
- the predicted mean response $\widehat{\pi}_i$,
  `> pi <- fit2$fitted`
- the ordinary residual e_i,
  `> e <- residuals(fit2, "response")`
- the Pearson residual r_{P_i},
  `> rp <- residuals(fit2, "pearson")`
- the deviance residual dev_i and
  `> dev <- residuals(fit2, "deviance");`
- the hat matrix diagonal elements h_{ii}.
  `> h <- hatvalues(fit2)`

i	Y_i	$\widehat{\pi}_i$	e_i	r_{p_i}	r_{sp_i}	dev_i	h_{ii}
1	0	0.209	-0.209	-0.514	-0.524	-0.685	0.039
2	0	0.219	-0.219	-0.529	-0.541	-0.703	0.040
3	0	0.106	-0.106	-0.344	-0.350	-0.473	0.033
4	0	0.371	-0.371	-0.768	-0.805	-0.963	0.090
5	1	0.111	0.889	2.833	2.869	2.098	0.025
...	...	...	...	...	...	...	...
96	0	0.114	-0.114	-0.358	-0.363	-0.491	0.025
97	0	0.092	-0.092	-0.318	-0.322	-0.439	0.024
98	0	0.171	-0.171	-0.455	-0.463	-0.613	0.036

Table 4: Disease Outbreak Example's Residuals and Hat matrix Diagonal elements.

Figure 9 can be plotted in R, like this:

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

> # (a) Est. Probability VERSUS Ord. Residuals
> plot(pi,e, xlab="Estimated Probability",
+ ylab="Ordinary Residuals",main="(a)")

> # (b) Est. Probability VERSUS Pearson Residuals
> plot(pi,rp, xlab="Estimated Probability",
+ ylab="Pearson Residuals",main="(b)")

> # (c) Est. Probability VERSUS Stud. Pearson Residuals
> plot(pi,rp, xlab="Estimated Probability",
+ ylab="Studentized Pearson Residuals",main="(c)")
```

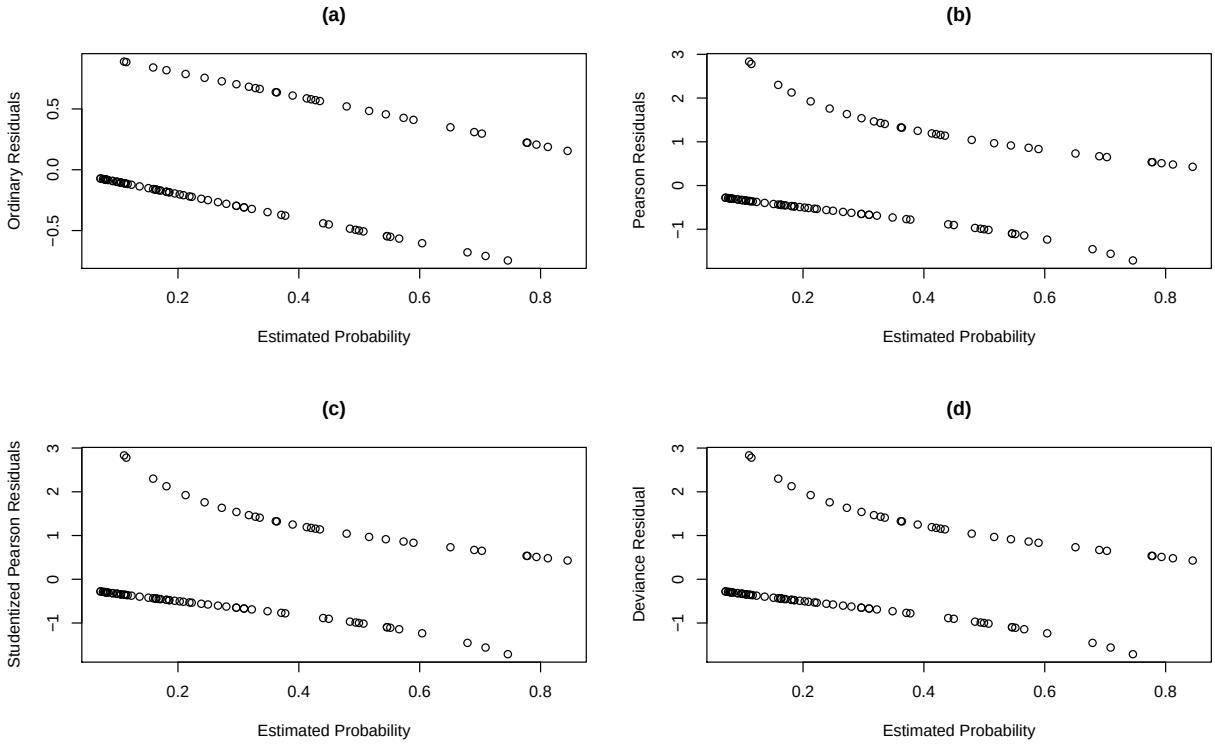

Figure 9: Selected residuals plotted against predicted mean response - Disease Outbreak Example

```
> # (d) Est. Probability VERSUS Dev. Residuals
> plot(pi,rp, xlab="Estimated Probability",
+ ylab="Deviance Residual",main="(d)")
```

5.4. Detection of Influential Observations

In ordinary least regression, we have several types of residuals and influence measures that help us understand how each observation behaves in the model, such as if the observation is too far away from the rest of the observations, or if the observation has to much leverage on the regression line (See Section 4.5). Similar techniques have been developed in logistic regression.

In logistic regression, there are three measures that can be used to identify influential observations. Consider the influence of individual binary cases on three aspects of the analysis:

1. The Pearson Chi-square statistic.
2. The deviance statistic.
3. The fitted linear predictor.

- **Influence on Pearson Chi-Squared and the Deviance Statistics**

Let X^2 and DEV denote the Pearson and deviance statistics (12), based on the full data set, and let $X^2_{(i)}$ and $DEV_{(i)}$ denote the values of these statistics when case i is deleted. The ith *delta chi-square statistic* is defined as the change in the Pearson statistic when the ith case is deleted:

$$\Delta X_i^2 = X^2 - X^2_{(i)}$$

Similarly the ith *delta deviance statistic* is defined as the change in the deviance statistic when the ith case is deleted:

$$\Delta dev_i = DEV - DEV_{(i)}$$

It is possible to approximate these two statistics by:

$$\Delta X_i^2 = r^2_{SP_i} \tag{14}$$

$$\Delta dev_i = h_{ii} \times r^2_{SP_i} + dev_i^2 \tag{15}$$

(14) and (15) give the change in the Pearson Chi-square and deviance statistics, respectively, when the ith case is deleted. They therefore provide measures of the influence of the ith case on these summary statistics. Usually the delta chi-square and delta deviance statistics are plotted against case number i, against $\widehat{\pi}_i$, or against $\widehat{\pi}'_i$. Extreme values appear as spikes when plotted against case number, or as outliers in the upper corners of the plot when plotted against $\widehat{\pi}_i$ or $\widehat{\pi}'_i$.

Figure 10a and 10b provide index of the delta chi-square (14) and delta deviance (15) statistics for the disease outbreak example (See section 5.2).

In R,

```
> par(mfrow=c(1,2))
> plot(case, deltaX2, xlab="Case", ylab="Delta Chi-Square", main="(a)")
> lines(case,deltaX2)
> identify(case, deltaX2)
[1]  5 14

> plot(case, deltadev, xlab="Case", ylab="Delta Deviance", main="(b)")
> lines(case,deltadev)
> identify(case, deltadev)
[1]  5 14
```

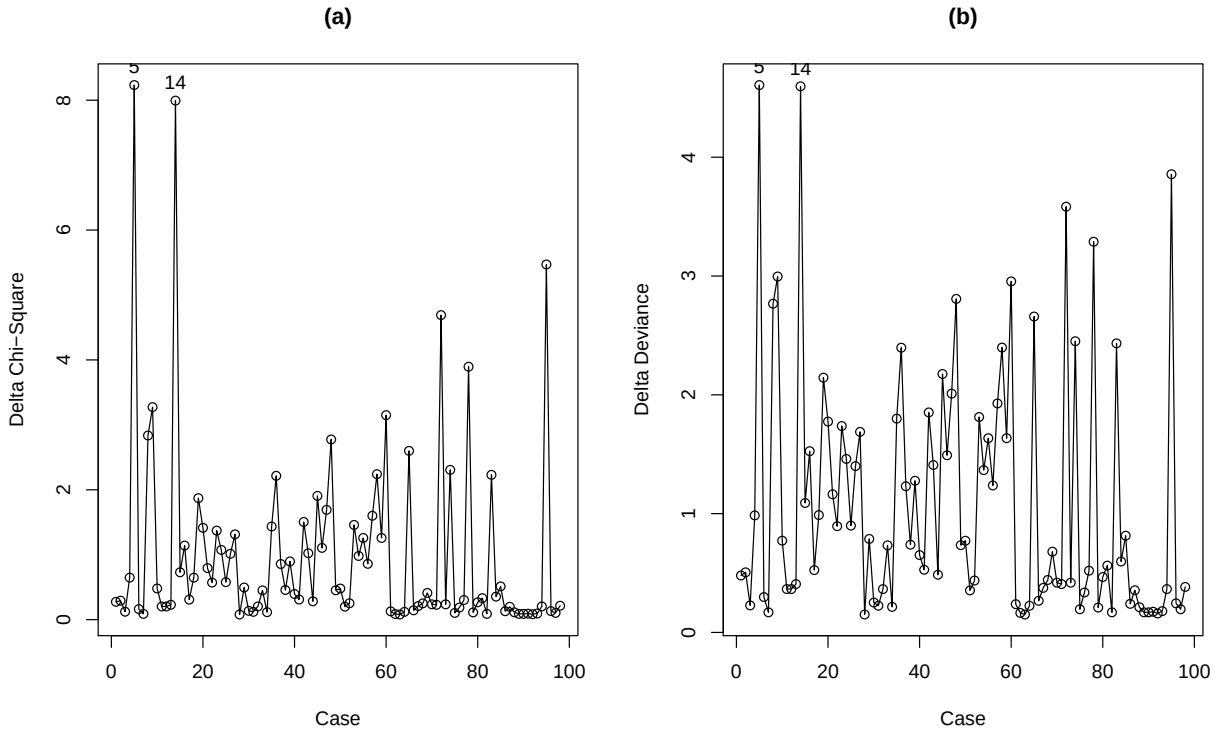

Figure 10: Delta Chi-Square and Delta Deviance Plots - Disease Outbreak Example

In Figure 10, the two peaks corresponding to cases 5 and 14 indicate clearly that these cases have the largest values of the delta deviance and the delta chi-square statistics. These results suggest that cases 5 and 14 may substantively affect the conclusions.

- **Influence on the Fitted Linear Predictor: Cook's Distance**

Recall that in Section 4.5, we introduced Cook's distance statistic, D_i for the identification of influential observations. Similarly, for logistic regression Cook's distance measures the

standardized change in the linear predictor $\widehat{\pi}_i$ when the ith case is deleted. The following approximation is used:

$$D_i = \frac{r_{P_i}^2 h_{ii}}{p(1-h_{ii})^2} \tag{16}$$

Index plots of leverage values h_{ii} are useful for identifying outliers in the X space, and index plots of D_{ii} can be used to identify cases that have a large effect on the fitted linear predictor.[15]

In R,

```
> D <- (rp^2*h)/(p*(1-h)^2) # Cook's Distance
```

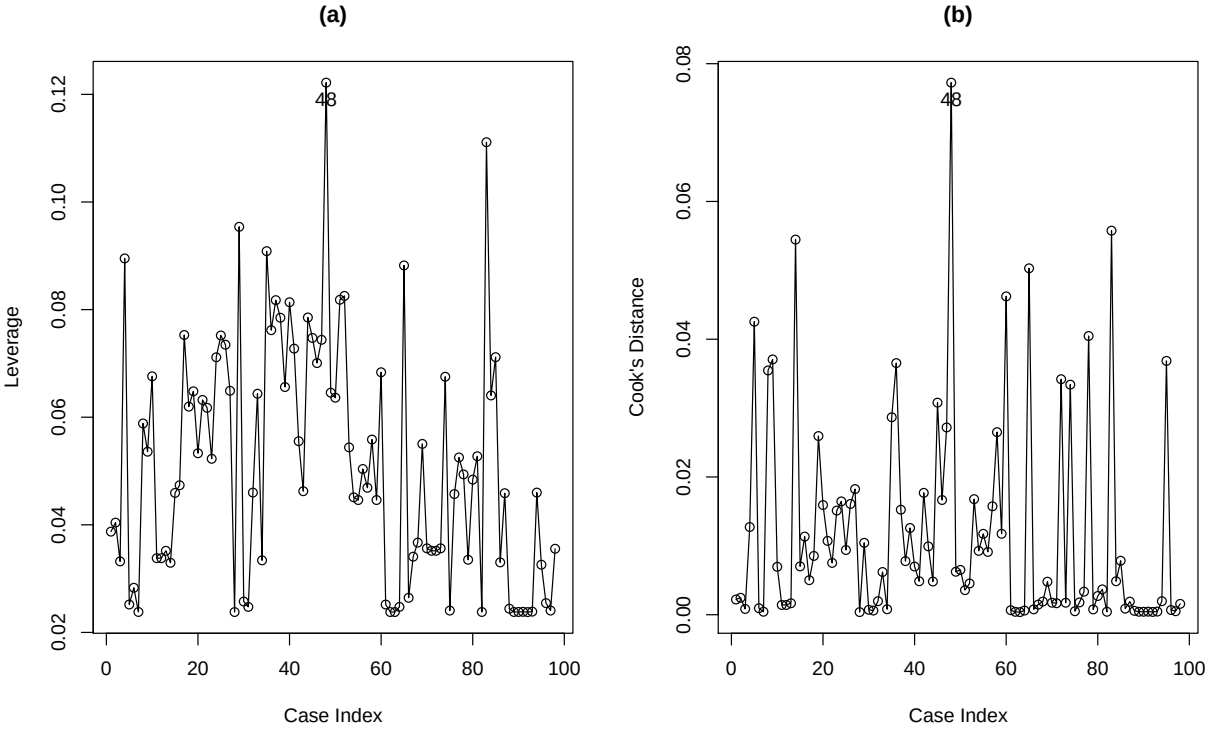

Figure 11: Index Plots of Leverage Values and Cook's distances - Disease Outbreak Example

The leverage plot of Figure 11a identifies case 48 as being somewhat outlying and therefore influential, and the plot 11b of te Cook's distances indicates that case 48 is indeed the most influential in terms of effect on the linear predictor. Note that cases 5 and 14 - previously identified as most influential in terms of their effect on the Pearson Chi-square and deviance statistics - have relatively less influence on the linear predictor. This is shown also by the proportional-influence plot in Figure 12.

[15] p is the number of regression coefficients, in the disease outbreak example, $p = 5$.

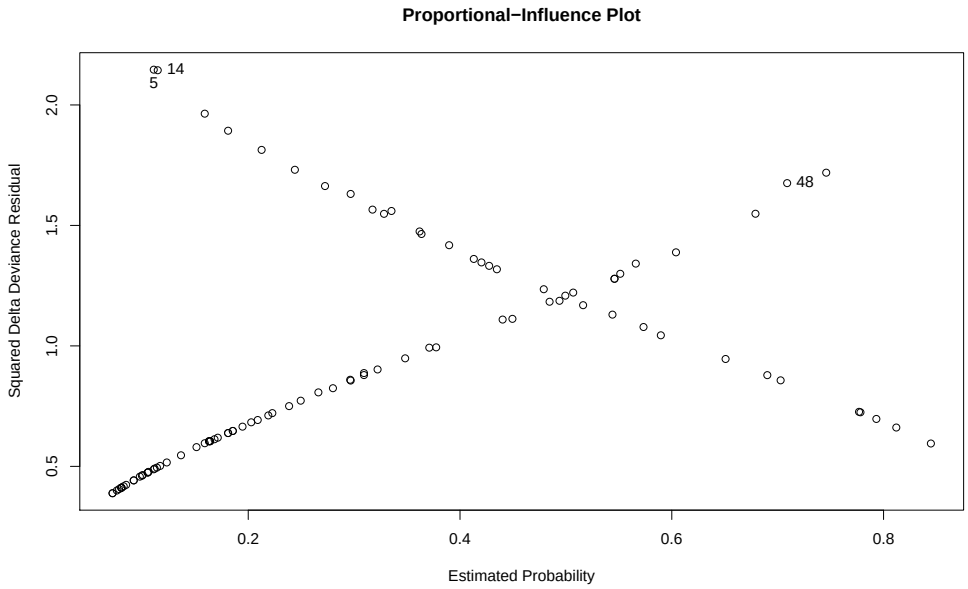

Figure 12: Index Plot of Proportional-Influence Plot of delta deviance statistic - Disease Outbreak Example

References

Dalgaard, P. (2002). *Introductory Statistics with R.* Statistics and Computing. Springer, New York. 4

Faraway, J. (2002). *Practical Regression and Anova using R (electronic book).* http://www.stat.lsa.umich.edu/faraway/book/. 36

Kutner, M., Nachtsheim, C., Neter, J., and Li, W. (2005). *Applied Linear Statistical Models.* 5th edition. 32

Maindonald, J. and Braun, J. (2003). *Data Analysis and Graphics Using R - An Example-based Approach.* Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge. 4

Venables, W. and Ripley, B. (1997). *Modern Applied Statistics with S-plus.* Statistics and Computing. Springer-Verlag, New York, 2nd edition. 35

Weisberg, S. (2005). *Applied Linear Regression.* John Wiley & Sons, Inc. Publication, 3rd edition. 51