Non-life — Assignment NL2

Niels Keizer* and Robert Jan Sopers†

September 23, 2016

1 Simulating an insurance portfolio-App. A3

Q1

How many bytes does it take to store $1, \ldots, 10, 1000, 100000$ logical values `TRUE/FALSE`?

We assume that $1, \ldots, 10$ means all the integers from 1 to 10. To how many bytes are needed in `R`, we use the function `object.size()`.

```
> for (n_values in c(1,2,3,4,5,6,7,8,9,10,1000,100000)){
+   hh <- rep(TRUE,n_values)
+   rr <- sample(c(TRUE,FALSE),n_values,repl=TRUE,prob=c(1,1))
+   af <- as.factor(rr)
+   print(c(n_values, object.size(hh), object.size(rr), object.size(af)))
+ }
[1]    1  48   48 464
[1]    2  48   48 464
[1]    3  56   56 528
[1]    4  56   56 528
[1]    5  72   72 544
[1]    6  72   72 488
[1]    7  72   72 544
[1]    8  72   72 544
[1]    9  88   88 560
[1]   10  88   88 560
[1] 1000 4040 4040 4512
[1] 100000 400040 400040 400512
```

The first column of the output is the length of the vector. The second column indicates the size in bytes of a vector filled with only `TRUE` values. The third with a random selection of `TRUE` and `FALSE`. The final column represents the size of the randomized vector, after it has been turned into a factor object.

*Student number: 10910492
†Student number: 0629049

Q2

To obtain the y vector, we first need to run the following code:

```
> n.obs <- 10000; set.seed(4)
> # n.obs <- 10000; set.seed(4) # Gebruik deze regel voor een grotere sample size.
> sx <- as.factor(sample(1:2, n.obs, repl=TRUE, prob=c(6,4)))
> jb <- as.factor(sample(1:3, n.obs, repl=TRUE, prob=c(3,2,1)))
> re.tp <- sample(1:9, n.obs, repl=TRUE, prob=c(.1,.05,.15,.15,.1,.05,.1,.1,.2))
> tp <- as.factor(c(1,2,3,1,2,3,1,2,3)[re.tp])
> re <- as.factor(c(1,1,1,2,2,2,3,3,3)[re.tp])
> mo <- 3 * sample(1:4, n.obs, repl=TRUE, prob=c(1,1,0,8))
> mu <- 0.05 * c(1,1.2)[sx] *
+              c(1,1,1)[jb] *
+              c(1,1.2,1.44)[re] *
+              1.2^(0:2)[tp] * mo/12
> y <- rpois(n.obs, mu)
> table(y)
y
   0    1    2    3
9276  702   20    2
```

Which is then inspected by calculating `mean(y)`, `var(y)` and the overdispersion factor `var(y)/mean(y)`.

```
> cbind(mean=mean(y),variance=var(y),phi=var(y)/mean(y))
       mean  variance       phi
[1,] 0.0748 0.0744124 0.9948182
```

The overdispersion factor is smaller than 1. This is possible because we are looking at a relatively small sample, with low probabilities. If we would take a much larger sample, the value would be larger than 1. We check this by running the same code, but with a sample 100 times larger. This gives a result with an overdispersion factor larger than 1.

```
> table(y)
y
     0      1      2      3      4
931128  66053   2734     82      3
> cbind(mean=mean(y),variance=var(y),phi=var(y)/mean(y))
         mean   variance      phi
[1,] 0.071779 0.07262285 1.011756
```

Q3

We create a dataframe by using the function `aggregate()`.

```
> aggr <- aggregate(list(Expo=mo/12,nCl=y,nPol=1), list(Jb=jb,Tp=tp,Re=re,Sx=sx), sum)
```

Then we compare the sizes.

```
> object.size(aggr)
5336 bytes
> object.size(mo)
80040 bytes
> object.size(y)
40040 bytes
> object.size(jb) + object.size(tp) + object.size(re) + object.size(sx)
162240 bytes
```

The amount of memory gained is equal to $80040 + 40040 + 162240 - 5336 = 276984$ bytes.

Q4

According to MART Sec. 3.9.3, the maximum likelihood estimate $\hat{\lambda}_{3,3,3,2}$ is equal to the number of claims divided by the exposure.

```
> aggr[54,]
   Jb Tp Re Sx   Expo nCl nPol
54  3  3  3  2 115.75  13  130
> lambda3332 <- aggr$nCl[54]/aggr$Expo[54]
> lambda3332
[1] 0.112311
```

In the first command, we show that observation 54 contains the desired aggregated values to calculate the estimate, which is then determined at 0.112.

2 Exploring the automobile portfolio of Sec. 9.5

First we execute the following code in R to generate the portfolio.

```
> rm(list=ls(all=TRUE))
> n <- scan(n=54) ## read 54 numbers into vector n
1:   1  8 10  8  5 11 14 12 11 10  5 12 13 12 15 13 12 24
19: 12 11  6  8 16 19 28 11 14  4 12  8 18  3 17  6 11 18
37: 12  3 10 18 10 13 12 31 16 16 13 14  8 19 20  9 23 27
Read 54 items
> expo <- scan(n=54) ## the number of policies
1:  10 22 30 11 15 20 25 25 23 28 19 22 19 21 19 16 18 29
19: 25 18 20 13 26 21 27 14 16 11 23 26 29 13 26 13 17 27
37: 20 18 20 29 27 24 23 26 18 25 17 29 11 24 16 11 22 29
Read 54 items
> expo <- 7 * expo ## each policy is in force during a 7-year period
> sex <- gl(2,27); region <- gl(3, 9, 54); type <- gl(3, 3, 54); job <- gl(3, 1, 54)
```

Q5

We are asked to comment on the difference between to lines of R code.

```
> str(type)
Factor w/ 3 levels "1","2","3": 1 1 1 2 2 2 3 3 3 1 ...
> str(rep(1:3, each=3, len=54))
int [1:54] 1 1 1 2 2 2 3 3 3 1 ...
```

The `str()` function compactly displays the structure of an arbitrary R object. `type` contains a `Factor` object, with 3 ordered levels (or categories), and a list of integers which indicate which element is at that position. `rep(1:3, each=3, len=54)` creates a vector of integers of three ones, three twos and three threes, repeated to a length of 54. Both objects

Q6

First we take a sample from a dataframe which contains the portfolio.

```
> set.seed(1); subset <- sort(sample(1:54,15))
> data.frame(sex, region, type, job, n, expo)[subset,]
sex region type job  n expo
3    1      1    1   3 10  210
8    1      1    3   2 12  175
10   1      2    1   1 10  196
11   1      2    1   2  5  133
15   1      2    2   3 15  133
16   1      2    3   1 13  112
20   1      3    1   2 11  126
29   2      1    1   2 12  161
30   2      1    1   3  8  182
31   2      1    2   1 18  203
32   2      1    2   2  3   91
45   2      2    3   3 16  126
46   2      3    1   1 16  175
47   2      3    1   2 13  119
48   2      3    1   3 14  203
```

We are asked to check if the covariates of the first two cells have the right value. We print the right values of cells 3 and 8 using this code.

```
> cbind(sex=sex[3],region=region[3],type=type[3],job=job[3],n=n[3],expo=expo[3])
     sex region type job  n expo
[1,]   1      1    1   3 10  210
> cbind(sex=sex[8],region=region[8],type=type[8],job=job[8],n=n[8],expo=expo[8])
     sex region type job  n expo
[1,]   1      1    3   2 12  175
```

We conclude that these are equal to those in the dataframe.

Q7

We construct two analysis of deviance tables. One where `type` is added before `region` and the other way around.

```
> anova(glm(n/expo ~ type*region, quasipoisson, wei=expo))
Analysis of Deviance Table

Model: quasipoisson, link: log

Response: n/expo

Terms added sequentially (first to last)


            Df Deviance Resid. Df Resid. Dev
NULL                           53    104.732
type         2   36.367        51     68.365
region       2   23.424        49     44.940
type:region  4    2.529        45     42.412
> anova(glm(n/expo ~ region*type, quasipoisson, wei=expo))
Analysis of Deviance Table

Model: quasipoisson, link: log

Response: n/expo

Terms added sequentially (first to last)


            Df Deviance Resid. Df Resid. Dev
NULL                           53    104.732
region       2   21.597        51     83.135
type         2   38.195        49     44.940
region:type  4    2.529        45     42.412
```

What we see is that the order in which these terms are added does not matter for the result. After both `type` and `region` are added, the resulting degrees of freedom and residual deviance is the same. We do of course see a difference between the analysis of only adding `region` or `type`.

Q8

We are asked to explain the similarities and the differences between the following `R` code.

```
> (g.wei <- glm(n/expo ~ region*type, poisson, wei=expo))

Call:  glm(formula = n/expo ~ region * type, family = poisson, weights = expo)

Coefficients:
  (Intercept)        region2        region3          type2          type3
     -2.98873        0.14988        0.42165        0.43376        0.45195
region2:type2  region3:type2  region2:type3  region3:type3
     -0.08084       -0.02230        0.25559        0.10860
```

```
Degrees of Freedom: 53 Total (i.e. Null);  45 Residual
Null Deviance:     104.7
Residual Deviance: 42.41  AIC: Inf
There were 50 or more warnings (use warnings() to see the first 50)
> (g.off <- glm(n ~ 1+region+type+region:type+offset(log(expo)),
+                family=poisson(link=log)))

Call:  glm(formula = n ~ 1 + region + type + region:type + offset(log(expo)),
    family = poisson(link = log))

Coefficients:
  (Intercept)        region2        region3          type2          type3
     -2.98873        0.14988        0.42165        0.43376        0.45195
region2:type2  region3:type2  region2:type3  region3:type3
     -0.08084       -0.02230        0.25559        0.10860

Degrees of Freedom: 53 Total (i.e. Null);  45 Residual
Null Deviance:     104.7
Residual Deviance: 42.41  AIC: 290.7
```

The output of `g.off` and `g.wei` contain the same coefficients, degrees of freedom, null deviance and residual deviance. The AIC for g.off is 290.7, however, for g.wei this is Inf. Also, g.wei throws warnings, on further inspection these arise from having non-integer x values in calls to dpois. This is what prevents the `glm` function from computing the AIC.

Q9

We define the dummy functions `region2` and `type3` as follows:

$$\texttt{region2} = \begin{cases} 1 & \texttt{region} = 2 \\ 0 & \texttt{region} \neq 2 \end{cases} \tag{1}$$

$$\texttt{type3} = \begin{cases} 1 & \texttt{type} = 3 \\ 0 & \texttt{type} \neq 3 \end{cases} \tag{2}$$

Multiplying these functions gives a new function

$$\texttt{region2} \cdot \texttt{type3} = \begin{cases} 1 & \texttt{region} = 2 \wedge \texttt{type} = 3 \\ 0 & \texttt{region} \neq 2 \vee \texttt{type} \neq 3 \end{cases} \tag{3}$$

Here $\wedge$ is the logical AND operator and $\vee$ is the logical OR operator. We see that this function equals 1 when `region` equals 2 and `type` equals 3, zero otherwise. It is therefore the same function as the dummy function `region2:type3`.

Q10

We run the following `R` code to generate `g.main`.

```
> g.main <- glm(n/expo ~ region+type, quasipoisson, wei=expo)
> coef(g.main)
(Intercept)     region2     region3       type2       type3
-3.0313238   0.2314097   0.4604585   0.3941889   0.5833108
```

a)

If `region` = 1 and `type` = 1, then the indicators for `region2`, `region3`, `type2` and `type3` are 0. Thus we only have to calculate:

```
> exp(g.main$coefficients["(Intercept)"])
(Intercept)
0.04825172
```

The first row of the dataset has region=1 and type=1, so we check against the fitted values from the glm.

```
> g.main$fitted.values[1]
1
0.04825172
```

Which is the same.

b)

We run the following code to determine the worst `type/region` combination.

Assuming all `type/region` combinations already exist in the model data (which is true):

```
> max(g.main$fitted.value)
[1] 0.1370301
```

By going through all possible combinations using a max function:

```
> exp(g.main$coefficients[1]+max(0,g.main$coefficients[2:3])+max(0,g.main$coefficients[4:5]))
(Intercept)
0.1370301
```

The maximum with 0 is taken in case both coefficients for `region` and/or `type` are negative. In that case, the baseline `region` = 1 and/or `type` = 1 would be the worst case.

Showing all possible combinations:

```
> exp(g.main$coefficients[1]+matrix(c(0,g.main$coefficients[2:3]),3,3)
+                            +t(matrix(c(0,g.main$coefficients[4:5]),3,3)))
           [,1]       [,2]       [,3]
[1,] 0.04825172 0.07156602 0.08646522
[2,] 0.06081528 0.09020005 0.10897864
[3,] 0.07646934 0.11341785 0.13703011
```

All three methods show that the estimated annual number of claims for the worst `type/region` combination equals 0.1370301. The third method shows that this is the case when `region` = 3 and `type` = 3.

Q11

Here we reconstruct the vector of fitted values using `R`. We also compare the results to the results from the model itself to show that the calculation is correct.

```
> cbind(g.off$family$linkinv(model.matrix(g.off) %*% coef(g.off) + g.off$offset),
+       fitted.values(g.off))
        [,1]      [,2]
1   3.524590  3.524590
2   7.754098  7.754098
3  10.573770 10.573770
4   5.982456  5.982456
5   8.157895  8.157895
6  10.877193 10.877193
7  13.846154 13.846154
8  13.846154 13.846154
9  12.738462 12.738462
10 11.464567 11.464567
11  7.779528  7.779528
12  9.007874  9.007874
13 11.071942 11.071942
14 12.237410 12.237410
15 11.071942 11.071942
16 13.292308 13.292308
17 14.953846 14.953846
18 24.092308 24.092308
19 13.432836 13.432836
20  9.671642  9.671642
21 10.746269 10.746269
22 10.540541 10.540541
23 21.081081 21.081081
24 17.027027 17.027027
25 25.411765 25.411765
26 13.176471 13.176471
27 15.058824 15.058824
28  3.877049  3.877049
29  8.106557  8.106557
30  9.163934  9.163934
31 15.771930 15.771930
32  7.070175  7.070175
33 14.140351 14.140351
34  7.200000  7.200000
35  9.415385  9.415385
36 14.953846 14.953846
```

37	8.188976	8.188976
38	7.370079	7.370079
39	8.188976	8.188976
40	16.899281	16.899281
41	15.733813	15.733813
42	13.985612	13.985612
43	19.107692	19.107692
44	21.600000	21.600000
45	14.953846	14.953846
46	13.432836	13.432836
47	9.134328	9.134328
48	15.582090	15.582090
49	8.918919	8.918919
50	19.459459	19.459459
51	12.972973	12.972973
52	10.352941	10.352941
53	20.705882	20.705882
54	27.294118	27.294118