Assignment 2. Statistical Theory and Modelling

Caroline Birkehammar, Pablo Paras Ochoa, Steven Hiram Rubio Vasquez

Statistical Theory and Modelling - Assignment Part 2

Import libraries

```
[1] "es_ES.UTF-8/es_ES.UTF-8/es_ES.UTF-8/C/es_ES.UTF-8/en_US.UTF-8"
```

Part 4

Problem 4a

Letting $a = (1, 0, 2)^T$ and $b = (1, 0, -1)^T$, we perform the dot product.

```
a = c(1,0,2)
b = c(1,0,-1)
(a%*%b)[1]
```

```
[1] -1
```

For two orthogonal vectors, the dot product will be equal to 0. Since our result is -1, we conclude that a and b are not orthogonal vectors.

Problem 4b

First we simulate a 10x3 matrix (X) with standard normal $N(0,1)$ random variables and a matrix $b = (1,1,2)^T$.

```
set.seed(22)
X = matrix(rnorm(30, mean = 0, sd = 1), nrow = 10, ncol = 3, byrow = TRUE)
X
```

```
              [,1]        [,2]       [,3]
 [1,] -0.512139088  2.48518368  1.0078262
 [2,]  0.292814572 -0.20895936  1.8580924
 [3,] -0.066026405 -0.16276495 -0.1998607
 [4,]  0.300561734 -0.76390728  0.0819619
 [5,]  0.743028275 -0.08402219 -0.7928945
 [6,] -0.922153631  0.86156238  2.0029422
 [7,]  0.936551013 -1.61573487 -0.5750566
 [8,] -0.003973089 -0.67611260 -1.0496283
 [9,] -0.543280568  0.55614453  0.2528377
[10,] -0.901814675  0.82439136 -1.5602798
```

```
b = c(1, 1, 2)
b
```

```
[1] 1 1 2
```

Then we perform matrix-vector multiplication, using the operator

```
mu = (X%*%b)
mu
```

```
            [,1]
 [1,]  3.9886969
 [2,]  3.8000400
 [3,] -0.6285127
 [4,] -0.2994217
 [5,] -0.9267830
 [6,]  3.9452931
 [7,] -1.8292970
 [8,] -2.7793422
 [9,]  0.5185394
[10,] -3.1979828
```

The first element of the matrix $\mu_{[1,1]} = 3.988697$ is obtained by calculating the dot product between the first row of X $(-0.512139088, 2.48518368, 1.0078262)$ and the vector b. We can calculate it by hand to corroborate the result.

```
mu_1 = (1*-0.512139088) + (1*2.48518368) + (2*1.0078262)
mu_1
```

```
[1] 3.988697
```

Problem 4c

We simulate the vector of errors ϵ from a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 0.1$.

```
epsilon = matrix(rnorm(10, mean = 0, sd = 0.1), nrow = 10, ncol = 1, byrow = TRUE)
epsilon
```

```
              [,1]
 [1,]  0.053799400
 [2,] -0.126835389
 [3,]  0.064051983
 [4,] -0.053576182
 [5,] -0.101964282
 [6,] -0.080788151
 [7,]  0.005682522
 [8,]  0.095021140
 [9,] -0.112676350
[10,] -0.020116829
```

With the error vector, we can create a vector of response observations on the variable Y:

```
y = mu + epsilon
y
```

```
            [,1]
 [1,]  4.0424963
 [2,]  3.6732046
 [3,] -0.5644607
 [4,] -0.3529979
 [5,] -1.0287472
 [6,]  3.8645050
 [7,] -1.8236145
 [8,] -2.6843211
 [9,]  0.4058630
[10,] -3.2180997
```

Finally, we compute the least squares estimate for our particular vectors

```
b_hat = solve(t(X) %*% X) %*% t(X) %*% y
b_hat
```

```
          [,1]
[1,] 0.9974657
[2,] 1.0091050
[3,] 1.9676249
```

This result of our least squares estimation of the coefficients, $\hat{\beta}$, is similar to the input vector that we used, which was $(1, 1, 2)^T$.

Problem 4d

For this task, we first calculate the vector or residuals.

```
error = y - X%*%b_hat
error
```

```
              [,1]
 [1,]  0.062502339
 [2,] -0.064034888
 [3,]  0.058896125
 [4,] -0.043205578
 [5,] -0.124986213
 [6,] -0.026124311
 [7,]  0.004149783
 [8,]  0.067185296
 [9,] -0.110931240
[10,] -0.080422518
```

Then we estimate the variance of the errors as follows.

```
n = nrow(X)
p = ncol(X)
var_e = (t(error) %*% error)/(n - p)
var_e[1]
```

```
[1] 0.007564446
```

At last we obtain the covariance matrix and extract the square root of the values in the diagonal.

```
CoVar = var_e[1] * solve(t(X) %*% X)
CoVar_ind = matrix(c(sqrt(CoVar[1]),
                     sqrt(CoVar[5]),
                     sqrt(CoVar[9])),
                  nrow = 1, ncol = 3)
CoVar_ind
```

```
          [,1]       [,2]       [,3]
[1,] 0.0665725 0.03974297 0.02574523
```

Part 5

Import data

```
data = read.csv("https://github.com/StatisticsSU/STM/raw/main/assignment/bugs.csv",
header = TRUE)
y = data$nBugs
X = data[,-1]
X = as.matrix(X)
```

Problem 5a

If we model our data using a Negative Binomial distribution, Y (nBugs) with a mean estimate of $\hat{\mu} = \bar{y} \approx 5.2528$ is the number of Bernoulli trials until r "successes" or "failures". Whether r respresents successes or failures depends on the data and the research question. We do not have access to information about the dataset and what its parameters represent, so we leave out the interpretation of the results in any other aspect than purely statistical.

In this task, we are interested in finding the maximum likelihood estimate of r given our data on Y. To do this, we create a function to find the negative log-likelihood function of our data, and then find the minimum value of that function (the optim function in R performs minimization by default, so we negate the log-likelihood function).

```
# Negative binomial log likelihood function
loglik_negbin <- function(r, y){
  return(-sum(dnbinom(y, r, mu = 5.2528, log = TRUE)))
}

opt <- optim(par = 1, gr = NULL, fn = loglik_negbin, y, method = "L-BFGS-B",
             lower = 0.0001, hessian = TRUE)

rhat <- opt$par
rhat
```

```
[1] 1.473745
```

The maximum likelihood estimate of r given our data is $\hat{r} \approx 1.4737$.

Problem 5b

The task is to find the standard error of the maximum likelihood estimate of r, which is the same as the standard deviation in the sampling distribution of the estimator. In large samples (n $>$ 30) we can approximate the sampling distribution of the estimator with a normal distribution, so that $\hat{r} \sim N\left(r_0, \frac{1}{-l''(\hat{r})}\right)$ approximately, and this is what we use to calculate the standard error below.

```
mle_se <- 1 / sqrt(opt$hessian[1])
mle_se
```

```
[1] 0.2874746
```

The standard error of our maximum likelihood estimate is approximately $SE_r \approx 0.2875$.

Problem 5c

To estimate both μ and r using the maximum likelihood method we use the same method as before, but insert a vector of the two parameters to be estimated into the log-likelihood function instead of only one parameter. Then we find the best estimators by finding the minimum values of the negative log-likelihood functions.

```
# Joint negative log-likelihood function
loglik_negbin_2 <- function(param) {
  r  <- param[1]
  mu <- param[2]
  return(-sum(dnbinom(y, r, mu = mu, log = TRUE)))
}

opt_2 <- optim(par = c(1, 1), fn = loglik_negbin_2, method = "L-BFGS-B",
               lower = c(0.0001, 0.0001), hessian = TRUE)

rhat  <- opt_2$par[1]
muhat <- opt_2$par[2]

rhat
```

```
[1] 1.473745
```

```
muhat
```

```
[1] 5.252747
```

The estimated value for μ is approximately 5.2528, and the estimated value for r is approximately 1.4737, the same value that we received before when we estimated only one parameter.

Problem 5d

Similarly to before, we use the second derivatives to calculate the standard errors of the parameter estimates. This time they are stored in a Hessian matrix however, so the code has to be adjusted slightly.

```
se_rhat  <- sqrt(solve(opt_2$hessian)[1, 1])
se_muhat <- sqrt(solve(opt_2$hessian)[2, 2])

se_rhat
```

```
[1] 0.2874746
```

```
se_muhat
```

```
[1] 0.5132815
```

The standard error for the mean estimate SE_μ is 0.5133, and the standard error for the r estimate SE_r is 0.2875. Both standard errors are rounded to four decimals.

Part 6

Problem 6a

The model fit is in the code below. As can be seen, the estimates obtained through numerical optimization using log likelihood very closely match those obtained using the pre-built glm function.

```
# Modelito is for verification purposes
modelito <- glm(nBugs ~ nCommits + propC + propJava + complexity,
                 family = poisson, data = data)

# This is the actual regression using numerical optimization
y <- data$nBugs

X <- cbind(data$intercept, data$nCommits, data$propC, data$propJava, data$complexity)

log_like_pois_reg <- function(betas, y, X) {
  lambda <- exp(X %*% betas)
  logLik <- sum(dpois(y, lambda = lambda, log = T))
  return(logLik)
}

init_vals <- c(0, 0, 0, 0, 0)

reg_opt <- optim(init_vals, log_like_pois_reg, gr = NULL, method = c("BFGS"), y, X,
                 control = list(fnscale = -1), hessian = T)

b_con_sombrerito <- reg_opt$par


# Comparing both results to ensure the results from our function are accurate
b_con_sombrerito
```

```
[1] -1.117837428 -0.009543965  1.456838249  3.200732352  2.145565785
```

```r
summary(modelito)
```

```

Call:
glm(formula = nBugs ~ nCommits + propC + propJava + complexity, 
    family = poisson, data = data)

Coefficients:
             Estimate Std. Error z value             Pr(>|z|)    
(Intercept) -1.117947   0.422681  -2.645              0.00817 ** 
nCommits    -0.009529   0.013663  -0.697              0.48555    
propC        1.456763   0.573667   2.539              0.01110 *  
propJava     3.200850   0.665862   4.807           0.00000153 ***
complexity   2.145435   0.255053   8.412 < 0.0000000000000002 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 405.18  on 90  degrees of freedom
Residual deviance: 304.64  on 86  degrees of freedom
AIC: 587.23

Number of Fisher Scoring iterations: 5
```

Problem 6b

Only one of the covariates does not seem to be statistically different from 0 (the number of commits). All other confidence intervals do not cross over 0 but the confidence interval for nCommits does.

```r
cov_b_sombrero <- -solve(reg_opt$hessian)

var_b_sombrero <- diag(cov_b_sombrero)

sd_b_sombrero <- sqrt(var_b_sombrero)

menor <- b_con_sombrerito - 1.96*sd_b_sombrero
```

```
mayor <- b_con_sombrerito + 1.96*sd_b_sombrero

conf_int <- tibble(
  beta = b_con_sombrerito,
  std_err = sd_b_sombrero,
  lower = menor,
  upper = mayor
)

print(conf_int)
```

```
# A tibble: 5 x 4
      beta std_err   lower   upper
     <dbl>   <dbl>   <dbl>   <dbl>
1 -1.12      0.423  -1.95   -0.289
2 -0.00954  0.0137 -0.0363  0.0172
3  1.46      0.574   0.332   2.58
4  3.20      0.666   1.90    4.51
5  2.15      0.255   1.65    2.65
```

Problem 6c

The predicted number of bugs for release 92 is around bugs 19.15 given both the covariate vector for release 92 and our model.

```
x92 <- c(1, 10, 0.45, 0.5, 0.89)

pred <- exp(x92 %*% b_con_sombrerito)

print(pred)
```

```
         [,1]
[1,] 19.14841
```

```
# Testing with modelito to verify the answer
x92_plus <- data.frame(
  nCommits = 10,
  propC = 0.45,
  propJava = 0.5,
  complexity = 0.89
```

```r
)

pred <- predict(modelito, newdata = x92_plus, type = "response")

print(pred)
```

```
      1
19.1475
```

References

Source code: https://github.com/hiramRV/PreBayesA1