The Cox proportional hazard model is probably the most commonly applied model for survival analysis in epidemiological research. Cox models offer a efficient way to adjust for the underlying time scale \(t\) which otherwise requires more complex parametric models. In those models e.g. in the Poisson or Weibull model there are two ways of adjusting for the underlying time scale: either one uses a time splitting approach where the time \(t\) gets split into multiple time intervals which are added as covariates to the model or one uses a spline function to model the effect of the underlying time scale. Both approaches need assumptions about the shape of the effect of the underlying time scale e.g. the time points used for the splitting or the degrees of freedom of the spline function. Additionally, both approaches add multiple parameters to the model. Fitting these more complex parametric models might be a challenge when facing limited data. The Cox model instead only needs one assumption to be able to adjust for the underlying time scale i.e. the effect of the exposure is assumed to be proportional over time. This assumption is known as the proportional hazard assumption and is subject to many debates.

Furthermore, the Cox model does not need any additional parameters for adjusting for the time scale \(t\). Hence, it can also be used to analyse more limited data.

As in my previous blog posts, we will use the lung cancer data set included in the {survival}

package as an example. For more information on this data set please take a look at the help file

?survival::lung Specifically, we will model the effect of sex and age on the survival of lung cancer patients in this data set. To estimate this effect, we will use a Cox model.

The Cox model follows the general form of

\[ \lambda(t|X) = \lambda\_0(t) \exp(\beta X). \]

In our case we will fit the following Cox model including the independent variables female

\((x\_1)\) and age \((x\_2)\).

\[ \lambda(t|X) = \lambda\_0(t) \exp(\beta\_1 x\_{1} + \beta\_2 x\_{2}). \]

So now let’s get started with loading the data set and setting up the variables.

# 1. Prefix

# Remove all files from ls rm(list = ls())

# Loading packages require(survival) require(optimx) require(numDeriv) require(purrr) require(dplyr) require(tibble) require(broom)

# 2. Loading data set

-

#Reading the example data set lung from the survival package lung <- as.data.frame(survival::lung)

#Recode dichotomous variables

lung$female <- ifelse(lung$sex == 2, 1, 0)

lung$status\_n <- ifelse(lung$status == 2, 1, 0)

The Cox model in its basic form requires unique event time i.e. no ties, as we will see later. To deal with this assumption we will randomly add or subtract small amounts of time from each event time in the data set, as seen in the code below. In an upcoming blog post I will address different approaches to deal with ties in Cox models.

#Removes time ties in data set set.seed(2687153)

lung$time <- map\_dbl(lung$time, function(x){x + runif(1, -0.1, +0.1)}) #Check if no ties are left

lung %>% count(time) %>%

arrange(desc(n)) %>% head(5)

## time n

## 1 5.042231 1

|  |  |  |  |
| --- | --- | --- | --- |
| ## | 2 | 10.917351 | 1 |
| ## | 3 | 11.016322 | 1 |
| ## | 4 | 11.020302 | 1 |
| ## | 5 | 11.989413 | 1 |

Great! We don’t have any ties in our data set left. Now we can proceed with the definition of the log-likelihood function of our Cox model. The log-likelihood function of the Cox model generally follows the form

\[ \ln L(\beta) = \sum d\_i \bigg( X\_i \beta - \ln \sum\_{j:t\_j\geq t\_i} \theta\_j \bigg) \]

where \(\theta = \exp(\beta X)\) and \(d\_i\) is the event indicator for the \(i^{th}\) subject. If we plug in our independent variables from above we yield \(\theta = \exp(\beta\_1 x\_1 + \beta\_2 x\_2)\) for our specific case.

The \(\sum\_{j:t\_j\geq t\_i} \theta\_j\) part of the formula above makes the computation a little bit more complicated, since it requires a certain order of the observations. We need to sum

\(\theta\_j\) for all observations \(j\) that have an event later or at the exact same time as our

\(i^th\) observation. Basically this the cumulative sum of \(\theta\_j\) across the event times \(t\) in descending order. To calculate this quantity in R we can sort the data set by descending event times and afterwards use the base::cumsum() function to calculate the cumulative sum.

Lets put this all together and define our log-likelihood function in R.

# 3. Define log-likelihood function for Cox regression model

negll <- function(par){

#Extract guesses for beta1 and beta2 beta1 <- par[1]

beta2 <- par[2]

#Define dependent and independent variables m <- data.frame(t = lung$time

d = lung$status\_n, x1 = lung$female, x2 = lung$age) #Calculate theta

m$theta <- exp(beta1 \* m$x1 + beta2 \* m$x2 #Calculate cumulative sum of theta with descending t

m <- m %> arrange(desc(t)) %>%

mutate(thetaj = cumsum(theta) #Estimate negative log likelihood value

val <- -sum(m$d \* ((m$x1 \* beta1 + m$x2 \* beta2) - log(m$thetaj))) return(val)

}

To improve our optimisation we should also pass the gradient functions for our model to the

optimx() later. The gradient function for the Cox model in general follows

\[ \ln L'(\beta) = \sum d\_i \bigg(X\_i - \frac{\sum\_{j:t\_j\geq t\_i} \theta\_j X\_j}{\sum\_{j:t\_j\geq t\_i}

\theta\_j} \bigg)\]

In our case we yield the following two gradient functions for \(\beta\_1\) and \(\beta\_2\).

\[ \ln L'(\beta\_1) = \sum d\_i \bigg(x\_{1i} - \frac{\sum\_{j:t\_j\geq t\_i} \theta\_j x\_{1j}}{\sum\_{j:t\_j\geq t\_i} \theta\_j} \bigg)\]

\[ \ln L'(\beta\_2) = \sum d\_i \bigg(x\_{2i} - \frac{\sum\_{j:t\_j\geq t\_i} \theta\_j x\_{2j}}{\sum\_{j:t\_j\geq t\_i} \theta\_j} \bigg)\]

We can use this function to get the following gradient function for our Cox model. Note that we have to use the base::cumsum() twice in the code below to calculate the cumulative sum of

\(\theta\_j x\_{1j}\) and \(\theta\_j x\_{2j}\).

# 4. Define gradient function for Cox regression model

negll\_grad <- function(par){ #Extract guesses for beta1 and beta2 beta1 <- par[1]

beta2 <- par[2] #Create output vector n <- length(par[1])

gg <- as.vector(rep(0, n))

#Define dependent and independent variables m <- data.frame(t = lung$time

d = lung$status\_n, x1 = lung$female, x2 = lung$age)

#Calculate theta, thetaj, thetajx1 and thetajx2 m$theta <- exp(beta1 \* m$x1 + beta2 \* m$x2

m <- m %> arrange(desc(t)) %>%

mutate(thetaj = cumsum(theta) thetajx1 = cumsum(theta \* x1), thetajx2 = cumsum(theta \* x2)) #Calculate partial gradient functions

gg[1] <- -sum(m$d \* (m$x1 - (m$thetajx1 / m$thetaj))) gg[2] <- -sum(m$d \* (m$x2 - (m$thetajx2 / m$thetaj))) return(gg)

}

Lets just check if our gradient function is correct by comparing it with the approximation of the

gradient function calculated with the numDerive::grad() function.

# 4.1 Compare gradient function with numeric approximation of gradient

========

# compare gradient at 1, 0, 0, 0 mygrad <- negll\_grad(c(0, 0)

numgrad <- grad(x = c(0, 0), func = negll) all.equal(mygrad, numgrad)

## [1] TRUE

Looks like we get the same numbers and our gradient functions works fine.

Now we pass both our log-likelihood and gradient function on to our optimx() call.

# 5. Find minimum of log-likelihood function

# Passing names to the values in the par vector improves readability of results

opt <- optimx(par = c(beta\_female = 0, beta\_age = 0), fn = negll,

gr = negll\_grad, hessian = TRUE,

control = list(trace = 0, all.methods = TRUE))

# Show results for optimisation algorithms, that converged (convcode != 9999)

summary(opt, order = "value") %>% rownames\_to\_column("algorithm") %>% filter(convcode != 9999) %>% arrange(value) %>%

select(algorithm, beta\_female, beta\_age, value) %>% head(7)

## algorithm beta\_female beta\_age value

|  |  |  |  |
| --- | --- | --- | --- |
| ## | 1 | Rcgmin -0.5133011 0.01703863 | 742.7912 |
| ## | 2 | nlminb -0.5133013 0.01703864 | 742.7912 |
| ## | 3 | nlm -0.5133009 0.01703866 742.7912 | |
| ## | 4 | L-BFGS-B -0.5133009 0.01703860 742.7912 | |
| ## | 5 | BFGS -0.5133055 0.01703866 742.7912 | |
| ## | 6 | Nelder-Mead -0.5134599 0.01702275 742.7912 | |
| ## | 7 | CG -0.4557679 0.01735647 742.8464 | |

Six of the optimisation algorithms implemented in the {optimx} package yielded equal maximum likelihood values up to four decimals. This suggests neglectable differences between these models. If we would print more decimals of the maximum log-likelihood values we would probably see some slight differences between them. However, all models suggest that females have about half the risk of dying from lung cancer compared to males and the risk of dying increases with increasing age.

Let us check which estimates we would get for sex and age if we would fit a Cox model using the survival::coxph() function and compare those with our estimates.

# 6. Estimate regression coefficients using coxph

cox\_model <- coxph(Surv(time, status\_n == 1) ~ female + age,

data = lung)

# 7. Comparing results from optimx and coxph

coef\_coxph <- unname(coef(cox\_model)) coef\_opt <- coef(opt) lapply(1:nrow(coef\_opt), function(i){

opt\_name <- attributes(coef\_opt)$dimnames[[1]][i] diff\_beta\_1 <- (coef\_opt[i, 1] - coef\_coxph[1]) diff\_beta\_2 <- (coef\_opt[i, 2] - coef\_coxph[2]) mean\_dif <- mean(diff\_beta\_1, diff\_beta\_2

na.rm = TRUE) data.frame(opt\_name, mean\_dif)

}) %>%

bind\_rows() %>% filter(!is.na(mean\_dif)) %>% mutate(mean\_dif = abs(mean\_dif)) %>% arrange(mean\_dif)

## opt\_name mean\_dif

## 1 Rcgmin 3.497854e-08

## 2 nlminb 1.140406e-07

|  |  |  |
| --- | --- | --- |
| ## | 3 | L-BFGS-B 2.315549e-07 |
| ## | 4 | nlm 2.650035e-07 |
| ## | 5 | BFGS 4.366428e-06 |
| ## | 6 | Nelder-Mead 1.587632e-04 |
| ## | 7 | CG 5.753333e-02 |
| ## | 8 | Rvmmin 5.133012e-01 |

We can see that the mean difference between our estimates and the estimates yielded with the survival::coxph() model is neglectable for most of our models. It seems as everything worked well. However, as a good researcher we would of course also like to obtain some estimates of uncertainty. So let us take the model we fitted with the Rcgmin algorithm from our optimx() output and calculate the standard error for our estimates using the hessian matrix.

# 8. Estimate the standard error ------------------------------

#Extract hessian matrix for the Rcgmin optimisation

hessian\_m <- attributes(opt)$details["Rcgmin", ][["nhatend"]] # Estimate se based on hessian matrix

fisher\_info <- solve(hessian\_m) prop\_se <- sqrt(diag(fisher\_info))

# Compare the estimated se from our model with the one from the coxph mode

ses <- data.frame(se\_rcgmin = prop\_se, se\_coxph = tidy(cox\_model)[["std.error"]]) %>% print()

## se\_rcgmin se\_coxph

## 1 0.167457336 0.167457337

## 2 0.009224444 0.009224444

all.equal(ses[,"se\_rcgmin"], ses[, "se\_coxph"]) ## [1] TRUE

Based on the standard error, we can calculate the confidence intervals for our estimates now.

# 9. Estimate 95%CIs using estimation of SE

-

# Extracting estimates from the Rcgmin optimisaiton coef\_test <- coef(opt)["Rcgmin",]

# Compute 95%CIs

upper <- coef\_test + 1.96 \* prop\_se lower <- coef\_test - 1.96 \* prop\_se # Print estimate with 95%CIs data.frame(Estimate = coef\_test, CI\_lower = lower,

CI\_upper = upper, se = prop\_se) %>% round(4)

## Estimate CI\_lower CI\_upper se

## beta\_female -0.5133 -0.8415 -0.1851 0.1675

## beta\_age 0.0170 -0.0010 0.0351 0.0092

Great! We obtained our own Cox model with confidence intervals.

To summarise, we specified our log-likelihood function and its gradient function, and optimised it using the optimx::optimx() function. Based on the output of the optimx() call we were able to obtain the standard error and confidence intervals for our estimates.