Contents

20.3 Influence Diagnostics

- Cook's Distance
- Likelihood Distance

Preparatory Steps

The `formula()` can recall the definition of the model defining the mean structure.

Auxillary Function `logLik1()` which is designed to calculate a contribution of a given subject to the overall likelihood for a given model.

The number of degrees of freedom reported by loglik is equal to 8. This corresponds to the total number of parameters in the model.

Part A

```
 <- update(fm16.5)



logLik()
```

Part B

```
beta0 <- fixef(fm16.5ml)

beta0

colnames(vcovb) <- names(beta0)
vcovb
```

We extract the β estimates and their estimated variance covariance matrix.

Towards this end, we use the functions `fixef()` and `vcov()` respectively, we can save these estimates and the matrix as objects `beta0` and `vcovb` respectively.

With the help of the `abbreviate` function the names of the beta estimates are shortened to simplify the display of content.

An auxillary function `logLik1()`

we used the `logLik()` function to obtain the value of the log-likelihood for the fitted model. It should be noted that the function returns the log-likelihood evaluated at the set of the

The auxillary functon `logLik1()` has been included in the package nlmeU.

estimated fixed effects and variance-covariance parameters and for the data set, to which the model is fitted. In the context of influence diagnostics, we need a more general function that allows

`modfit` an object of class *lme* representing the lme model fited to a given dataset using ML estimation.

`dt1` a dataframe with data for one subject, for whom the likelihood function to be evaluated

`dtInit` an optional auxillary data frame

The dataframe provided in the argument `dt1` is typically created by choosing a subset with one subject from the data used to obtain the model fit object specified in the `modfit` argument. However, in general, any pluasible fata for one subject , not necessarily from the dataset used to fit the model, can be used.

The auxillary data provided in the argument dtInit is temporarily appended to the dt1 data. during the `logLik()` function execution.

The `logLik1()` function returns the numeric contributions of the single subject, with the data specified in the dt1 argument, to the log likelihood for the model specified in the modfit argument.

Contributions of Individual Subjects to the log-likelihood for fitted model

```
lLik.i <- as.vector(lLik.i)

lLik.i[1:5]
sum(lLik.i)
```

Plot of individual contributions to the log-likelihood (traditional graphics)

```
subject.c <-
subject.x <-
plot()
points()
text()
```

next we use the function `logLik1()` to compute the loglikelihood contributions for all subjects.

We present the syntax to plot the per-observation individual log-likelihood contributions. First, with the help of the by() function, we create the array `nx` , which contains the number of observations

1 Influence Diagnostics

We use the results of the preparatory steps to perform influence-diagnostic calculations for the model. More specifically we evaluate the influence of every subject included in the data set.

We create a list containing the results of fitted model the "leave-one-subject-out" (LOO) datasets and explore its contents.

We define the function `lmeU()`, which fits the model to the data from the armd

when the function `lmeU()` is executed, and LOO data frame, named dfU, is created with thesubject, indicated by the cx argument.

Subsequently model is fitted by dfI by applying the

next, with the help of the function lapply(), we apply the lmeU() to the consecutive elements of the character vector subject.c.

As a result, we obtain the list lmeUall, with lme-class model fit objects as elements. The model-fit objects contains the result of fitting model M16.5.

Finally, we name the components of the `lmeUall()` list using the subjects idenifier stored in the vector subject.c.

This technique is computationally expensive, as it required the model to be fitted m number of times, omitting one of the m cases each time.

Execution time can be improved if we decided to perform a reduced number of likelihood iterations, instead of performing iterations until there is convergence.

The values, based on the first few iterations, are expected to give a fairly good approximation of the LOO estimates.

The names of the first six components are printed out using the function `names()`.

To extract the LOO data frame for, e.g., the subject "6", we refer to the "6" compoment of the `lmeUall` list.

The extracted data frame is stored in the object `dataU6`. By using the function dim() we can check the dimensions of the data frame.

Model is fitted to a sequence of "leave-one-subject-out" out data sets.

```
 # back - what is cx?
 # cx is Case identity
lmeU <- function(cx){
dfU <- subset(myData, subject !=cx)
update(mymodel,data=dfU)
}

 # what is lmeU?

lmeUall <- lapply(subject.c, lmeU)
```

Exploring the contents of the lmeUall object.

```
names(lmeUall)
dataU6 <- lmeUall[["6"]]$data
dim(dataU6)
unique(dataU6$subject)[1:6]
```

Galecki presents the code used to calculate and plot individial likelihood displacements.

For an LMM, it is required the computation of the full log-likelihood for $\hat{\theta}$, the ML estimate for θ obtained by fitting the model to all data, and for $\hat{\theta}_(- i)$, the ML estimate obtained by fitting the model to the data with the $i-$th subject excluded.

Note that both values of the log-likelihood, used in the definition of the likelihood displacement, should be calculated taking into account all observations.

Galeck creates an auxillary function `lLik()` which , for a given subject indicated by the main argument, extracts the lme model fit object for the corresponding LOO data.

The corresponding log-likelihood function is extracted from the lmeU with the help of the `logLik()`

The returned value `lLikU + lLik.s` is the log-likelihood evaluated for all observations, using the displaced estimates of the model parameters.

Calculation of the likelihood displacement

```
lLik <- function(cx)
   {
   lmeU <- lmeUall[[cx]]
   lLikU <- logLik(lmeU, REML = FALSE)
   df.s <- subset(armd,subject == cx)

   lLik.s <- logLik1(lmeU,df.s)

   return(lLikU + lLik.s)
   }

lLikUall <- apply(subject.c,lLik)
```

```
names(dif.2Lik) <- subject.c #subjects ids assigned
outL <- dif.2Lik > 0.5

dif.2Lik[outL]
```

```
## Plot Component

libary(lattice)

subject.f <- factor(subject.c, levels = subject.c)

myPanel <- function(x,y, ...){
  x1 <- as.numeric(x)
  panel.xyplot(x1,y, ... )
  ltext(x1[outL],y[outL], subject.c[outL]) # outlying LDis
  }

dtp <- dotplot(dif.2Lik ~subject.f, panel = myPanel, type= "h")

 # ggplot?
lxlims <- length(dtp$x.limits)

update(dtp,xlim=rep(" ", lxlims),grid= "h")
```

By applying the `summary()` function to the vector, we obtain summary statistics of the computed likelihood-displacement values.

we create the logical output vector outL which indicates the subjects with the values of the likelihood displacement exceeding say 0.5.

From the printout of the selected elements of the vectir dif.2Lik it follows that there are seven such subjects.

We then use the function `dotplot()` from the package lattice to plot the likelihood distance value for all subjects.

The x-axis of the plot is constructed using numeric representation of the `numeric.f` factor, containing values ranging from 1 to 234.

2 20.3.2.3. Cook's Distance for the β estimates

Cook's distance for the β estimates was defined in (4.26) for the classical LM. The definition can be extended to the LMMs in a straightforward manner.

Part 2 plot of cook's distance using traditional graphics

```
betaUall <- sapply(lmeUall,fixef)

vb.inv <- solve(vcovb)

cookDfun <- function(betaU){
   dbetaU <- betaU - beta0
```

```
  cookD.value <- t(dbetaU) %*% vb.inv %*% dbetaU
  }
```

Plot of Cook's Distance using traditional graphics. Outlying values annotated.

```
outD <- cookD > 0.03


#Create the Stick plot
plot(CookD ~subject.c,
  ylab="Cook's Distance", type="h")

text()
points()
```

Cook's distance for the *beta* estimates were defined bu 4.26 for the classical LM model.

Next we compute the inverse of the variance covariance matrix $\hat{\beta}$ using the `solve()` function. We store the resulting matrix in the object `vb.inv`.

Subsequently we define the function `cookDfun()` which, for a vector given in the `betaU` argument, computes the value of the numerator of Cook's Distance, as in (4.26).

The function is then applied sequentially to all columns of the matrix `betaUall`.

The resultant vector is divided by the number of the fixed effects coefficients, which, under the assumption that the design matrix is of full rank, is equivalent to the rank of the design matrix.

The outcome is stored in the vector `cookD` and contains the values of cooks distanced for all subjects.

We create the logical vector outD which indicates the subjects with cook's distance values that exceed 0.03.

Present the scatterplot matrix of the two-dimensional projections of the differences. for all pairs of the fixed-effects coefficients.

The plot was generated using the `splom` function. The main argument of the function was obtained by subtracting the beta0 vector from the rows of the transposed betaUall matrix.

The labels used in the panels located on the diagonal of the figure provide the fixed estimates of fixed effects coefficients of the model and their estimates SEs. The panels above the diagonal include points for all subjects. The points for non-outlying values are plotted using small size open circles. The five outlying values are plotted using different plotting symbols defined in the legend of the figure at the top of the graph.

The effect of removal of this subject on the estimates of the remaining fixed-effects coefficients is relatively small. In contrast, removing the subject "227" affects the estimate of all fixed-effects coefficients to a different degree and in different directions.

More specifically, the intercept is driven towards lower values : the positive slope associated with visual acuity at baseline visual0, is further increased, the negative slope associated with `time` is brough closer to zero, and the treatment effect is attenuated.

Overall we note that the effect of removing any of the subjects on the fixed effects estimates is very small, as it does not exceed 0.05 of the SE of any of the estimates.

3 Simulation of the Dependent Variable

We cosnider simualtions of the dependent variable, based on the marginal distribution implied by the fitted model. Towards this end, we have developed `simulateY()`, which can be used for objects of class lme.

We note that the function is different from simulate.lme(), available in the nlme in that the latter returns simulation-based REML and/or ML values and not the value of the dependent variable.

We demonstrate the use of the `simulateY()` function to create the empirical distribution of the β estimates. As an example, we consider the model which was fitted to the armd data.

Note however that the presented syntax is fairly general can be used for other LMEs as well.

We apply the function `simulateY()` to the object fm16.5ml.

```
library(nlmeU)
simY <- simulateY(fm16.5ml, nsim=1000)


simYsumm <- apply( simY, MARGIN = 2,
 FUN = function(y){

     }
   )


simYsumm[[1]]

# \hat{\beta} for the 1st simulation
```

There is a second argument `nsim` that indicated how many simulations

The auxillary function performs the following steps

- the dependent variable visual is the `auxDt` data frame is replaced wiht a new set of simulated values contained in the vector y.
- Model M16.5 is fitted to the modified data frame
- the vector beta with the estimates of the fixed effects coefficients is extracted from the summary of the model-fit object with the help of the `fixef()` function.
- The vector of estimates is returned as a list with one component named `beta`.

They are drawn from the marginal distribution of the dependent variable implied by the fitted model.

It should be mentioned that the creation of the simYsumm list involves refitting the model many times, and it therefore takes a long time.

Toward this end, with the help of the `sapply()` function, we extrac the vectors with the values of $\hat{\beta}$ for each simulation for the list -object `simYsumm` and bind them column-wise into the `betaE`.

Then we use the function `rowMeans()` to compute the mean values of the columns (i.e. accross the rows) of betaE.