selfmade: SELective inference For Mixed and ADditive model Estimators

David Rügamer

Objective

This vignette describes the generic use of the selfmade software to produce valid post-selection inference, or more specifically selective inference, for linear mixed and additive models after any type of variable selection mechanism, which can be repeated in a bootstrap-like manner.

Prerequisites

- The framework assumes that covariates in all models to be fixed.
- It must be possible to fit the final model with the `gam4` function of the eponymous R package or the `gamm` function from mgcv.
- It must be possible to define a deterministic function of a vector \( y \in \mathbb{R}^n \) (referred to as `selection_function` in the following) determining the selection result for which the practitioner seeks valid inference statements. In other words, the user has to define a function similar to the function `selection_function` defined below, which is deterministic in the sense that for the same input \( y \) the output should also be exactly the same.
- It must be possible to define a function, which checks the congruency of the result of the `selection_function` and the original selection given when performing model selection on the original data \( y \). This is usually trivial and just a wrapper for the `selection_function`.

```r
selection_function <- function(y) {
  # based on any input y of the same dimension as the original response
  # a model is selected and mapped to an integer value
  # ....
  best_model_index <- get_best_model(list_of_models)

  return(best_model_index)
}
```

Note that the `selection_function` should return the original result when called with the original data vector \( y \).

Approach

1. Run the experiment with the `original_response`
2. Save the model selection result (e.g., as integer indicating the selected model) as well as the final model `final_model` (after refitting the model with `gam4` if a different package has been used for model selection)
3. Define the model selection function (`selection_function`)
4. Define the wrapper (`check_congruency`) function returning a logical value whether the result of any model call of `selection_function` is equivalent to the original model selection result.
5. Run the `mocasin`-function providing selective inference as follows:

   ```r
   res <- mocasin(mod = final_model,
                   checkFun = check_congruency,
                   this_y = original_response
                   # further options
   )
   ```

Examples

- Example 1 demonstrates the package’s ability to reproduce classical inference if no model selection was done.
- Example 2 demonstrates the use of the package for model selection with only `gamm4` models.
- Example 3 demonstrates the package’s ability to calculate valid inference regardless of the type of model selection and packages involved (as long as the prerequisites are met).

Example 1

```r
library(selfmade)
library(gamm4)
set.seed(0)
dat <- gamSim(1,n=500,scale=2) ## simulate 4 term additive truth
dat$y <- 3 + dat$x0^2 + rnorm(n=500)
br <- gamm4(y~ s(x0) + s(x1), data = dat)
summary(br$gam) ## summary of gam

# do not use any selection
# - hence it's not necessary to define selection_function
# and the check_congruency always returns TRUE
checkFun <- function(yb) TRUE

# calculate selective inference, which, in this case,
# except for an approximation error, should be equivalent
# to the unconditional inference
res <- mocasin(br, this_y = dat$y,
                checkFun = checkFun,
                # further options
                nrlocs = c(0.7,1),
                nrSamples = 1000, trace = FALSE)

# we get very similar results using
do.call("rbind", res$selinf)
```

Example 2

```r
library(selfmade)
library(lme4)
```
library(lmerTest)

# use the ham data and use scaled information liking
# as response
ham$Informed.liking <- scale(ham$Informed.liking)

# We first define a function to fit a model based on response
# This function is usually not required but can be
# specified in order to define the selection_function
# as a function of the model instead of as a function
# of the response vector
modFun <- function(y)
{
  ham$y <- y
  lmer(y ~ Gender + Information * Product + (1 | Consumer) +
       (1 | Product), data=ham)
}

# define the selection_function:
# here this is done as function of a model
# which, in combination with modFun, can then
# be used as function
selFun <- function(mod) step(mod, reduce.fixed = FALSE)

# define a function which extracts the results
# of the selection procedure
extractSelFun <- function(this_mod){
  this_mod <- attr(this_mod, "model")
  if(class(this_mod)="lm")
    return(attr(this_mod$coefficients, "names")) else
    return(c(names(fixef(this_mod)),
             names(getME(this_mod, "theta"))))
}

# Now we run the initial model selection on the
# original data, which is a
# backward elimination of non-significant effects:
(step_result <- selFun(modFun(ham$Informed.liking)))
attr(step_result, "model")

# Elimination tables for random- and fixed-effect terms:
(sel <- extractSelFun(step_result))

# Now we can define the function checking the congruency
# with the original selection
checkFun <- function(yb){
  this_mod <- modFun(yb)
  setequal( extractSelFun(selFun(this_mod)), sel )
}


# and compute valid p-values conditional on the selection
# (this takes some time and will produce a lot of warnings)
res <- mocasin(attr(step_result, "model"), this_y = ham$Informed.liking,
    checkFun = checkFun, which = 1:4, nrSamples = 50, trace = FALSE)
print(res)

Example 3

# Run an AIC comparison between different additive models
# fitted with mgcv::gam
library(selfmade)
library(mgcv)
library(gamm4)

# create data and models
set.seed(2)
# use enough noise to get a diverse model selection
dat <- gamSim(1, n=400, dist="normal", scale=10)
b0123 <- gam(y-s(x0)+s(x1)+s(x2)+s(x3), data=dat)
b123 <- gam(y-s(x1)+s(x2)+s(x3), data=dat)
b013 <- gam(y-s(x0)+s(x1)+s(x3), data=dat)

# seems that the second model seems to be the most
# 'appropriate' one
which.min(AIC(b0123, b123, b013)$AIC)

# and refit the model with gamm4
b123_gamm4 <- gamm4(y~s(x0)+s(x1)+s(x3), data=dat)

# define selection_function
selection_function <- function(y)
{
  dat$y <- y
  list_of_models <- list(
    gam(y-s(x0)+s(x1)+s(x2)+s(x3), data=dat),
    gam(y-s(x1)+s(x2)+s(x3), data=dat),
    gam(y-s(x0)+s(x1)+s(x3), data=dat)
  )

  # return an integer value which model is best
  return(
    which.min(sapply(list_of_models, AIC))
  )
}

# define the congruency function
checkFun <- function(y) selection_function(y)==2

# compute inference
res <- mocasin(mod = b123_gamm4,
checkFun = checkFun,
nrlocs = 3, # test one position of the spline
nrSamples = 10
print(res)

References

