Vignette for Modeling Interactions with Treatment in High Dimensions: Survival Outcome

Abraham Apfel

Thursday, April 25, 2024

Contents

1 Purpose

The purpose of this vignette is to offer a guidance for colleagues looking to discover biomarkers whose association with an outcome may differ by treatment arm. We particularly focus on the high dimensional setting, however many of these ideas are easily applicaple to a low dimensional setting as well.

In this vignette we will cover parameterizing your model in a way to focus on identifying biomarkers which interact with treatment, on the importance of using splines when investigating interactions, and our

recommended approach to model certain biomarkers which appear to have a point of discontinuity at 0. We also focus on several challenges unique to the penalized regression setting such as including splines and interactions, scaling continuous variables appropriately for proper inference when categorical variables are also present, and appropriate visualizations.

2 Dataset

We make use of a randomized clinical trial made up of 323 patients for which we have RNA-Seq data (there were 699 patients total in the trial).

The goal of our analysis is to discover in any biomarkers from a pre-selected list of 13 differ in their association with Disease Free Survival (DFS) from the Treatment arm and the Placebo arm. Although this vignette will focus on a survival endpoint and these ideas are similarly applicable to continuous or binary endpoints with the main distinction being to apply 'grpreg' instead of 'grpsurv'.

3 Setting up Your Data

3.1 Scaling

The first step we do after loading our data is to scale all continuous variables by 2 times the standard deviation (instead of the more typically applied 1 times the sd). This is in order to allow the continuous variables to be on the same scale as the categorical ones. See (Gelman, 2008).

This particular dataset starts with 699 observations but we only make use of the 323 observations with RNA-Seq data.

```
# Load data for composite biomarker analysis


library(rms)
library(mice)
# if (!("SGL" %in% installed.packages()[, "Package"])){
#   install.packages("SGL")
# }
# library(SGL)
library(parallel)
library(glinternet)
library(knitr)
library(TunePareto)
library(survivalROC)
library(survcomp)
library(timeROC)
library(corrplot)
library(grpreg)
library(CalibrationCurves)
library(splines)
library(ellipse)
library(bmsPURR)
library(tidyverse)
```

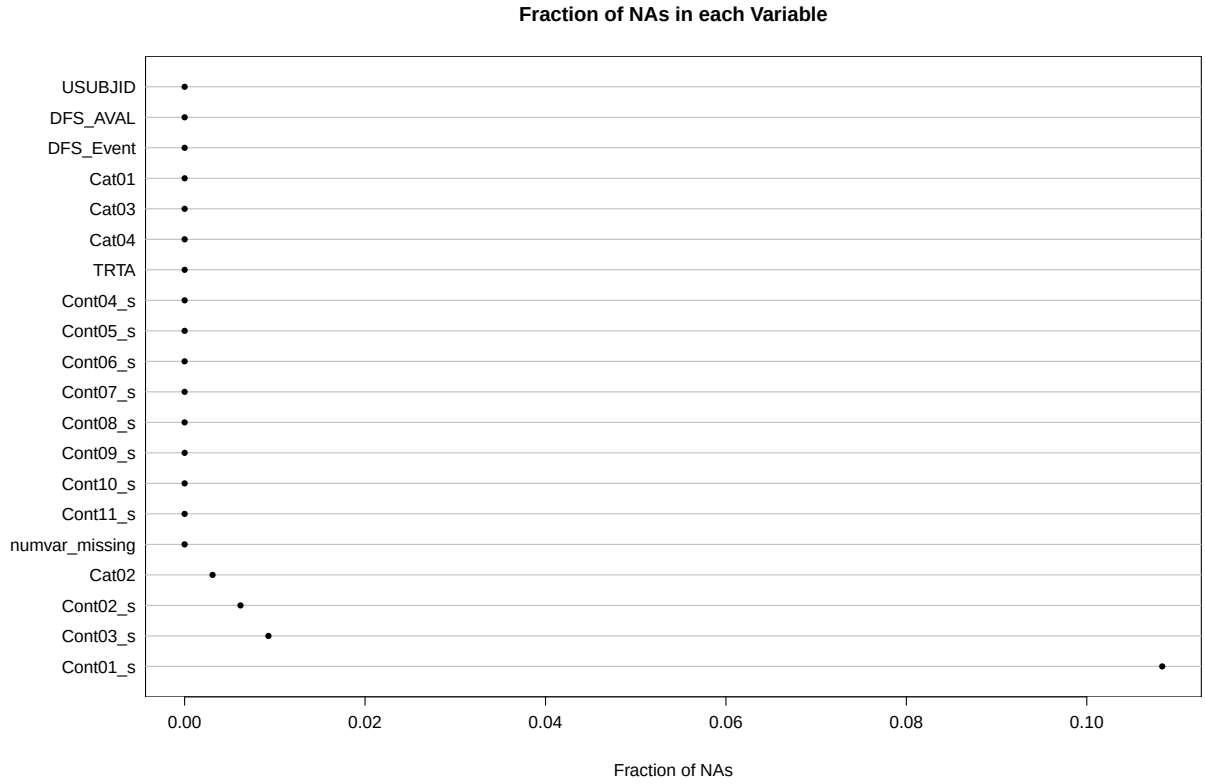
```
dat <- readRDS(file.path(results2, "Anonymized_dat.rds"))

options(contrasts=c("contr.treatment", "contr.treatment"))
```

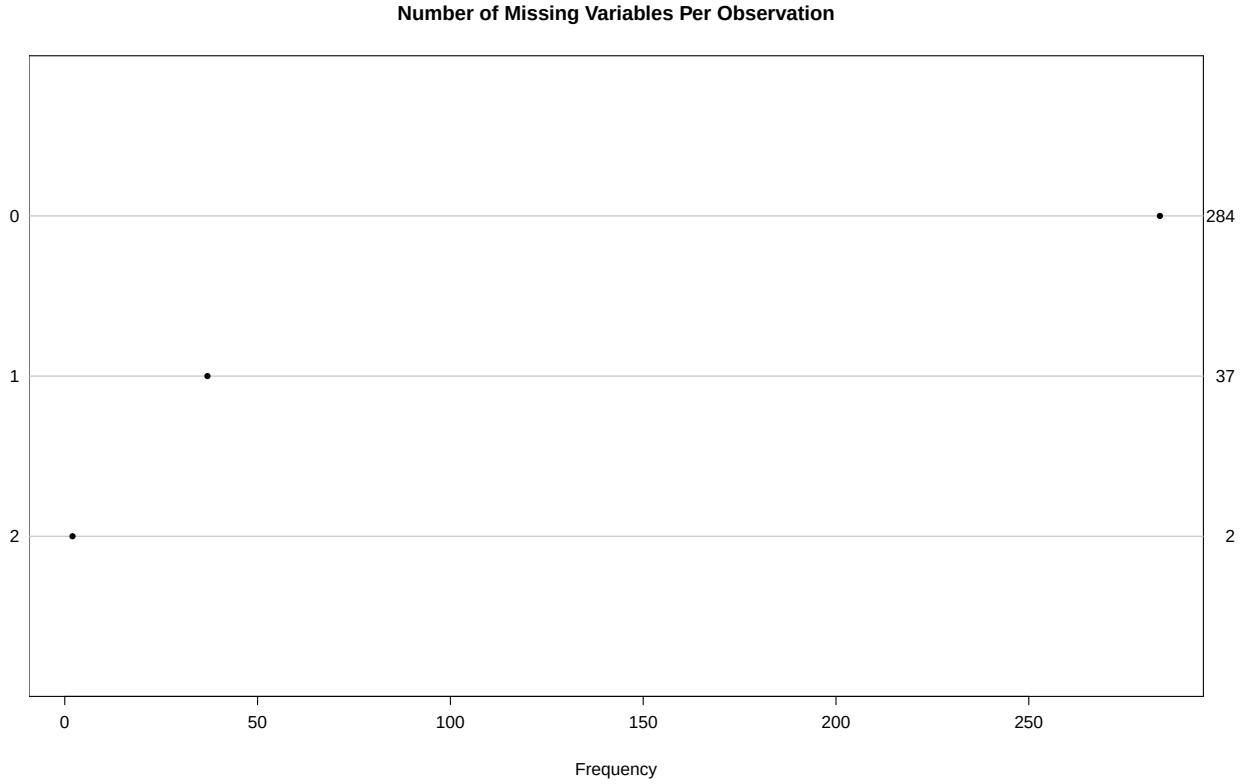
```
# Create function to scale by 2*sd (see Gelman, et al) so continuous parameters
# will be on similar scale to categorical predictors

scale2 <- function(x) {
  (x - mean(x, na.rm = T))/(2*sd(x, na.rm = T))
}


cont_bm <- c("Cont01", "Cont02", "Cont03", "Cont04", "Cont05", "Cont06",
             "Cont07", "Cont08", "Cont09",
             "Cont10", "Cont11")

dat <- mutate_at(dat, cont_bm, list(s = ~scale2(.)))


# Retain only columns of interest
```

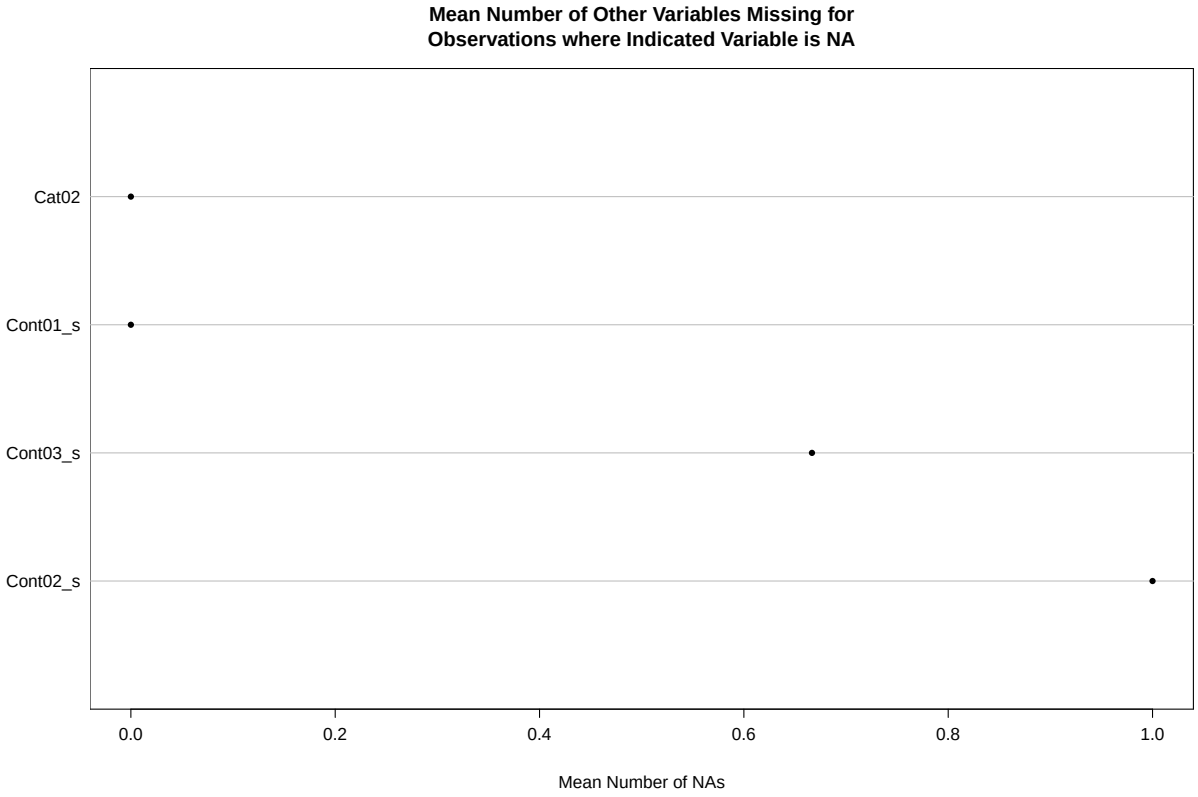
```r
comp_dat <- dat[,c("USUBJID", "DFS_AVAL", "DFS_Event",
                   "Cat01", "Cat02", "Cat03", "Cat04",
                   "TRTA",
                   "Cont01_s", "Cont02_s", "Cont03_s", "Cont04_s", "Cont05_s", "Cont06_s",
                   "Cont07_s", "Cont08_s", "Cont09_s",
                   "Cont10_s", "Cont11_s")]

# # Explore distributions of relevant variables
# describe(comp_dat)

cont_bm <- c("Cont01_s", "Cont02_s", "Cont03_s", "Cont04_s", "Cont05_s", "Cont06_s",
             "Cont07_s", "Cont08_s", "Cont09_s",
             "Cont10_s", "Cont11_s")

cat_bm <- c("Cat01", "Cat02", "Cat04", "Cat03")
```

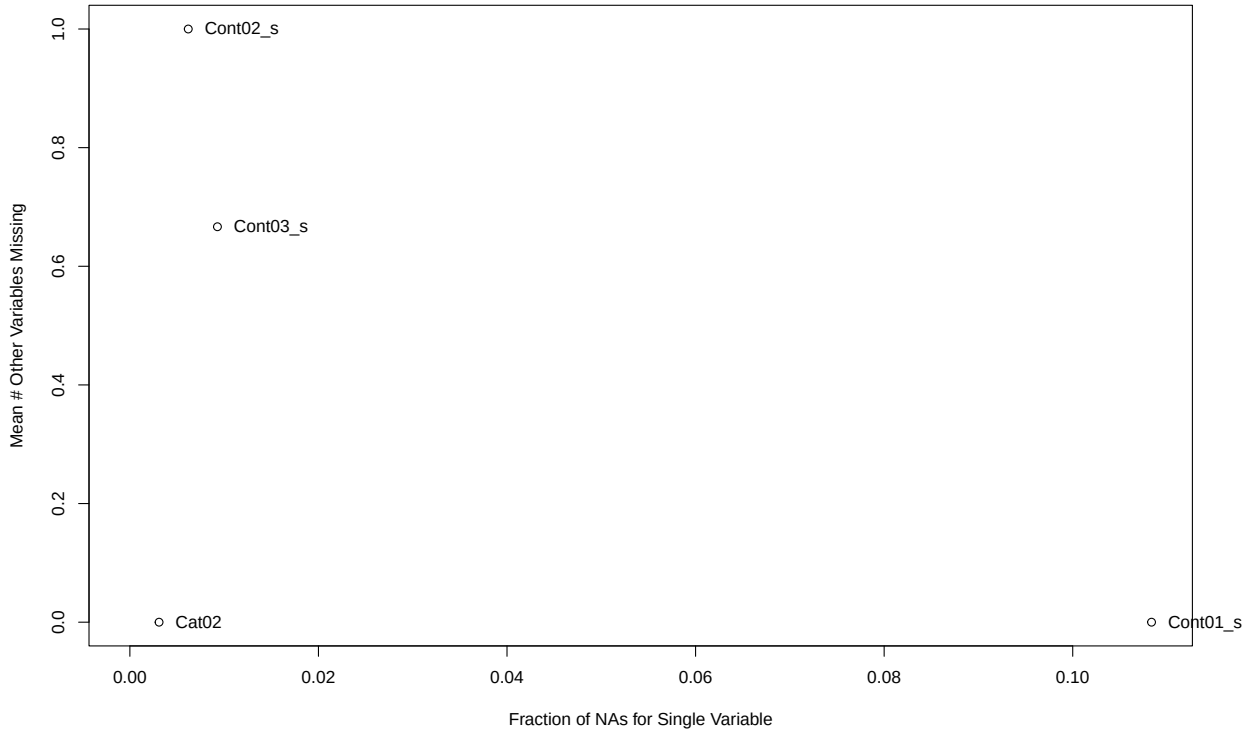
Having scaled the full cohort, we now remove all observations who do not have RNA_Seq data.

```r
# Explore missing patterns
na.patterns <- naclus(comp_dat)
comp_dat$numvar_missing <- na.patterns$na.per.obs

# Remove subjects missing on >5 variables
red_dat <- filter(comp_dat, numvar_missing <= 2)

# Explore missing patterns
na.patterns <- naclus(red_dat)
naplot(na.patterns)
```


Fraction of NAs in each Variable
USUBJID
DFS_AVAL
DFS_Event
Cat01
Cat03
Cat04
TRTA
Cont04_s
Cont05_s
Cont06_s
Cont07_s
Cont08_s
Cont09_s
Cont10_s
Cont11_s
numvar_missing
Cat02
Cont02_s
Cont03_s
Cont01_s
0.00
0.02
0.04
0.06
0.08
0.10
Fraction of NAs


Number of Missing Variables Per Observation
0
1
2
284
37
2
0
50
100
150
200
250
Frequency

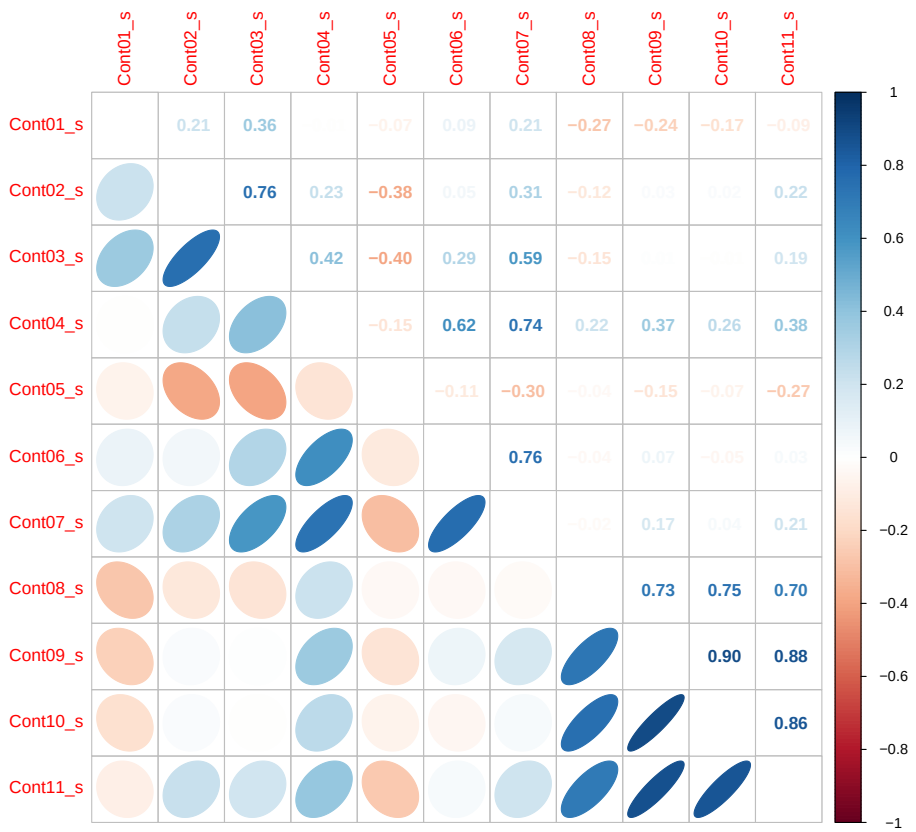

Mean Number of Other Variables Missing for
Observations where Indicated Variable is NA
Cat02
Cont01_s
Cont03_s
Cont02_s
0.0
0.2
0.4
0.6
0.8
1.0
Mean Number of NAs




```
# # Explore correlation of continuous biomarkers
# describe(red_dat)

num_dat <- as.matrix(dplyr::select(red_dat, cont_bm))
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(cont_bm)
##
##   # Now:
##   data %>% select(all_of(cont_bm))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
cor_mat <- rcorr(num_dat)

corrplot.mixed(cor_mat$r, lower = "ellipse", upper = "number", tl.pos = "lt")
```




```
# Remove Cont11, Cont10 b/c highly correlated with Cont09
# (r = 0.88, 0.90 respectively)
red_dat2 <- dplyr::select(red_dat, -Cont11_s, -Cont10_s, -numvar_missing)
```

3.2 Imputation

Since there are relatively few rows with missing data 39/323, even though in general best practice with this amount of missing (~12%) is still to employ multiple imputation, here to avoid complications stemming from combining penalized regression with multiple imputation, we will use single imputation. See Regression Modeling Strategies vignette for more details about best imputation practice and for more details about imputing survival data.

```
# Calculate Nelson-Aalen estimate for cumulative hazard for imputation purposes
red_dat2 <- mutate(red_dat2, DFS_NA_hazard = nelsonaalen(red_dat2, DFS_AVAL, DFS_Event))

set.seed(122)
MI <- aregImpute( ~ Cont08_s + Cont01_s + Cat02 + Cont06_s + Cont05_s +
                  Cat04 + TRTA + I(Cont02_s) + Cont09_s + Cont04_s +
                  Cat03 + Cat01 + Cont03_s +
                  Cont07_s,
                  data = red_dat2, n.impute = 1, tlinear = F)
```

```
## Iteration 1 Iteration 2 Iteration 3 Iteration 4
```

```
# Create dataset with imputed values
imp_dat <- as.data.frame(impute.transcan(MI, imputation=1, data=red_dat2,
                                         list.out=TRUE, pr=FALSE, check=FALSE))
```

```
# Add USUBJID, DFS_Event, and DFS_AVAL based on Cont09 (each row had unique value)
tmp <- dplyr::select(red_dat2, USUBJID, DFS_Event, DFS_AVAL, Cont09_s)
imp_dat <- inner_join(tmp, imp_dat)
```

3.3 Indicator Functions for Cont02 and Cont03

Now that we have a complete dataset (no more missing), we must make several transformations to run the most appropriate model.

While in general when modeling continuous variables we recommend keeping them in continuous form instead of dichotomizing them to avoid throwing away information, for Cont02 and Cont03 specifically, experience tells us that that there appears to be a point of discontinuity at 0. Therefore we recommend best practice to create an indicator variable for when Cont02 or Cont03 = 0 and model all values greater than 0 as continuous.

Note that this recommendation is independent of the complications of penalized regression and cubic splines we are applying in this analysis. In this specific analysis, since we already transformed Cont02 and Cont03 by taking the log and scaling we will create an indicator variable at the minimum Cont02 and Cont03 (which would map to 0 on the raw scale).

```
# Create Indicator functions for Cont02 and Cont03
imp_dat <- mutate(imp_dat, Ind_Cont02_s_Ind = ifelse(Cont02_s == min(Cont02_s), 1, 0))
imp_dat <- mutate(imp_dat, Ind_Cont03_s_Ind = ifelse(Cont03_s == min(Cont03_s), 1, 0))
```

3.4 Basis Functions for Cubic Splines

In order to perform penalized regression with splines, we must transform the continuous variables into basis functions. Here, we chose to apply splines with 3 knots, thus each continuous variable will be transformed to 2 columns of data each representing a different basis function for the spline.

In order to aid in the interpretability of the results, we will group the basis functions together so that they will undergo the same amount of shrinkage, thus we do not have to worry about one basis function for a given variable being removed from the model while the other basis function remains. We will implement this via the group lasso penalty.

```
# Set up data for group lasso modeling with splines

# Updated list of cont_bm (removed Cont10 and Stroma)
cont_bm <- c("Cont01_s", "Cont04_s", "Cont05_s", "Cont06_s",
             "Cont07_s", "Cont08_s", "Cont09_s")

# We identify the variables for which an indicator variable was created to aid
# in future transformations and visualizations
Ind_bm <- c("Cont02_s", "Cont03_s")

# Create dataframe of basis functions for splines of continuous variables
cont_dat <- list()
for(i in seq_len(length(cont_bm))){
  cont_dat[[paste0("mod_", cont_bm[i], "1")]] <- rcs(imp_dat[[cont_bm[[i]]]], 3)[,1]
  cont_dat[[paste0("mod_", cont_bm[i], "2")]] <- rcs(imp_dat[[cont_bm[[i]]]], 3)[,2]
}

for(i in seq_len(length(Ind_bm))){
  cont_dat[[paste0("Ind_", Ind_bm[i], "1")]] <- rcs(imp_dat[[Ind_bm[[i]]]], 3)[,1]
```

```
  cont_dat[[paste0("Ind_", Ind_bm[i], "2")]] <- rcs(imp_dat[[Ind_bm[[i]]]], 3)[,2]
}
```

```
## Warning in rcspline.eval(x, nk = nknots, inclx = TRUE, pc = pc, fractied = fractied): could not obta
##  Used alternate algorithm to obtain 3 knots

## Warning in rcspline.eval(x, nk = nknots, inclx = TRUE, pc = pc, fractied = fractied): could not obta
##  Used alternate algorithm to obtain 3 knots
```

```
cont_dat <- as.data.frame(cont_dat)
```

3.5 Reparameterizing Treatment

Because the goal of this analysis is to identify biomarkers which we think are likely to have a different association with DFS depending on treatment, we will parametrize the treatment variable as +1/-1 instead of the standard dummy coding. See (Tian et al., 2014).

```
# Reparameterize treatment to +1/-1 to apply Tibshirani/Simon method
imp_dat$TRT2 <- with(imp_dat, ifelse(TRTA == "TREATMENT", 1,
                             ifelse(TRTA == "PLACEBO", -1, NA)))
```

3.6 Transforming categorical variables

The software for Group Lasso (from the **grpreg** package) requires you to provide all predictors in matrix form with all categorical variables to be coded as 0/1. Thus we make the necessary transformations below.

Note that for categorical variables with >2 levels (as is the case for Cat04), we must create 2 distinct dummy variables and then we will group them together as we do with the basis functions for continuous variables and the indicator variables for Cont02 and Cont03.

```
# Set up clinical covariates for model
imp_dat <- mutate(imp_dat, cat_Cat02 = as.numeric(as.factor(Cat02)) - 1)
imp_dat <- mutate(imp_dat, cat_Cat03 = as.numeric(as.factor(Cat03)) - 1)
imp_dat <- mutate(imp_dat, cat_Cat01 = as.numeric(as.factor(Cat01)) - 1)

# Create 2 dummy variables for Cat04
imp_dat <- mutate(imp_dat,
                  status_unclear = ifelse(Cat04 == "Unclear", 1, 0))
imp_dat <- mutate(imp_dat, status_pos = ifelse(Cat04 == "Positive", 1, 0))


# To make handling of different types of variables easier
mod_Cat04 <- c("status_unclear", "status_pos")
mod_clin <- c("cat_Cat02", "cat_Cat03", "cat_Cat01")
mod_trt <- "TRT2"

# Dataframe to be used for model
mod_dat <- dplyr::select(imp_dat, mod_clin, mod_Cat04, mod_trt,
                         "DFS_Event", "DFS_AVAL", "Ind_Cont02_s_Ind", "Ind_Cont03_s_Ind")
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(mod_clin)
```

```
##
##   # Now:
##   data %>% select(all_of(mod_clin))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(mod_Cat04)
##
##   # Now:
##   data %>% select(all_of(mod_Cat04))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(mod_trt)
##
##   # Now:
##   data %>% select(all_of(mod_trt))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
mod_dat <- cbind(mod_dat, cont_dat)
mod_dat <- droplevels(mod_dat)
```

3.7 Interactions

Next we have to manually create the interactions between each of our predictors and treatment. Note that all predictors with the exception of 2-level categorical predictors have multiple columns associated with them. We need to multiple every column by the treatment variable to make the interaction. We will later group these interaction terms which each respective predictor for the group lasso penalty.

```
# Set up data to apply group lasso method
bm <- str_subset(colnames(mod_dat), "mod_")
mod_dat <- mutate_at(mod_dat, bm, list(modINT = ~. * TRT2))
INT_bm <- str_subset(colnames(mod_dat), "_modINT")

status <- str_subset(colnames(mod_dat), "status_")
mod_dat <- mutate_at(mod_dat, status, list(statusINT = ~. * TRT2))
INT_status <- str_subset(colnames(mod_dat), "_statusINT")

clin <- str_subset(colnames(mod_dat), "cat_")
```

```
mod_dat <- mutate_at(mod_dat, clin, list(catINT = ~. * TRT2))
INT_cat <- str_subset(colnames(mod_dat), "_catINT")

Ind <-  c("Ind_Cont02_s_Ind", "Ind_Cont02_s1", "Ind_Cont02_s2",
          "Ind_Cont03_s_Ind", "Ind_Cont03_s1", "Ind_Cont03_s2")
mod_dat <- mutate_at(mod_dat, Ind, list(IndINT = ~. * TRT2))
INT_Ind <- c("Ind_Cont02_s_Ind_IndINT", "Ind_Cont02_s1_IndINT", "Ind_Cont02_s2_IndINT",
             "Ind_Cont03_s_Ind_IndINT", "Ind_Cont03_s1_IndINT", "Ind_Cont03_s2_IndINT")
```

4 Group Lasso Modeling

4.1 Repeated Group Lasso

Now that the data is finally in shape, we are almost ready to run our model. When working with penalized regression, which typically requires some kind of tuning parameter selection via Cross-Validation (CV), we recommend running the CV numerous times in order to not have to rely on one arbitrary split of the data. Thus we will select the tuning parameter which has the minimum median error across the numerous runs.

To implement this recommendation, we make use of a wrapper function, `rep_grpsurv`, which can be found at https://github.com/apfela2/Utility_Functions/blob/main/code/R/grpLasso/rep_grpsurv.R

The `rep_grpsurv` function contains the following arguments:

Parameter	Details
data	dataframe
preds	Predictors to be placed in model - IMPORTANT!!! Pay attention to order of predictors so that groups are assigned correctly
time	Survival time
status	Censoring indicator: NOTE: 1 = Event, 0 = Censored
group	Vector of group number assigned to each predictor
penalty	See grpsurv documentation
lsp	Vector of proposed lambda values, recommend to keep NULL
nlambda	Number of lambda values which will be explored
n.folds	Number of folds used in inner loop of CV used for tuning parameter selection
se	See cv.grpsurv documentation
num.repeats	Number of times to repeat the CV procedure
alpha	Tuning parameter for elastic net type penalty (how much weight to put on L1 as opposed to L2)
lambda.type	Choice of how to determine "best" lambda choice of "min", "elbow", or "sparse"
numCores	Number of cores to use if parallelizing
tau	See grpsurv documentation
group.multiplier	See grpsurv documentation
warn	See grpsurv documentation
returnX	Return the standardized design matrix
returnY	See cv.grpsurv documentation
trace	see cv.grpsurv documentation

```
# Sanity check to make sure groups are properly assigned
tmp <- dplyr::select(mod_dat, c("TRT2", clin, INT_cat, status, INT_status, bm, INT_bm, Ind, INT_Ind))
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
```

```
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(clin)
##
##   # Now:
##   data %>% select(all_of(clin))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(INT_cat)
##
##   # Now:
##   data %>% select(all_of(INT_cat))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(status)
##
##   # Now:
##   data %>% select(all_of(status))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(INT_status)
##
##   # Now:
##   data %>% select(all_of(INT_status))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(bm)
##
```

```
##   # Now:
##   data %>% select(all_of(bm))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(INT_bm)
##
##   # Now:
##   data %>% select(all_of(INT_bm))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(Ind)
##
##   # Now:
##   data %>% select(all_of(Ind))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(INT_Ind)
##
##   # Now:
##   data %>% select(all_of(INT_Ind))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
grp <- c(0, rep(1:length(clin), times = 2), rep(length(clin) + 1, times = length(status)*2),
         rep(seq(length(clin) + 2, length(clin) + 1 + length(cont_bm)), each = 2, times = 2),
         rep(seq(length(clin) + 1 + length(cont_bm) + 1, length(clin) + 1 + length(cont_bm) + 2),
             each = 3, times = 2))
cbind(colnames(tmp), grp)
```

```
##                                grp
##  [1,] "TRT2"                   "0"
##  [2,] "cat_Cat02"              "1"
```

```
##  [3,] "cat_Cat03"                "2"
##  [4,] "cat_Cat01"                "3"
##  [5,] "cat_Cat02_catINT"         "1"
##  [6,] "cat_Cat03_catINT"         "2"
##  [7,] "cat_Cat01_catINT"         "3"
##  [8,] "status_unclear"           "4"
##  [9,] "status_pos"               "4"
## [10,] "status_unclear_statusINT" "4"
## [11,] "status_pos_statusINT"     "4"
## [12,] "mod_Cont01_s1"            "5"
## [13,] "mod_Cont01_s2"            "5"
## [14,] "mod_Cont04_s1"            "6"
## [15,] "mod_Cont04_s2"            "6"
## [16,] "mod_Cont05_s1"            "7"
## [17,] "mod_Cont05_s2"            "7"
## [18,] "mod_Cont06_s1"            "8"
## [19,] "mod_Cont06_s2"            "8"
## [20,] "mod_Cont07_s1"            "9"
## [21,] "mod_Cont07_s2"            "9"
## [22,] "mod_Cont08_s1"            "10"
## [23,] "mod_Cont08_s2"            "10"
## [24,] "mod_Cont09_s1"            "11"
## [25,] "mod_Cont09_s2"            "11"
## [26,] "mod_Cont01_s1_modINT"     "5"
## [27,] "mod_Cont01_s2_modINT"     "5"
## [28,] "mod_Cont04_s1_modINT"     "6"
## [29,] "mod_Cont04_s2_modINT"     "6"
## [30,] "mod_Cont05_s1_modINT"     "7"
## [31,] "mod_Cont05_s2_modINT"     "7"
## [32,] "mod_Cont06_s1_modINT"     "8"
## [33,] "mod_Cont06_s2_modINT"     "8"
## [34,] "mod_Cont07_s1_modINT"     "9"
## [35,] "mod_Cont07_s2_modINT"     "9"
## [36,] "mod_Cont08_s1_modINT"     "10"
## [37,] "mod_Cont08_s2_modINT"     "10"
## [38,] "mod_Cont09_s1_modINT"     "11"
## [39,] "mod_Cont09_s2_modINT"     "11"
## [40,] "Ind_Cont02_s_Ind"         "12"
## [41,] "Ind_Cont02_s1"            "12"
## [42,] "Ind_Cont02_s2"            "12"
## [43,] "Ind_Cont03_s_Ind"         "13"
## [44,] "Ind_Cont03_s1"            "13"
## [45,] "Ind_Cont03_s2"            "13"
## [46,] "Ind_Cont02_s_Ind_IndINT"  "12"
## [47,] "Ind_Cont02_s1_IndINT"     "12"
## [48,] "Ind_Cont02_s2_IndINT"     "12"
## [49,] "Ind_Cont03_s_Ind_IndINT"  "13"
## [50,] "Ind_Cont03_s1_IndINT"     "13"
## [51,] "Ind_Cont03_s2_IndINT"     "13"
```

```
# Store modeling preferences in list to ease reproducibility
PARAMS <- list(x = c("TRT2", clin, INT_cat, status, INT_status, bm, INT_bm, Ind, INT_Ind),
               grp = grp,
               time = "DFS_AVAL",
```

```
                status = "DFS_Event",
                penalty = "grLasso",
                lsp = NULL,
                se = "bootstrap",
                returnX = T,
                returnY = T,
                num.repeats    = 50, #inner loop of cv to fit lambda
                nfolds         = 5,  #inner loop of cv to fit lambda
                nlambda        = 100, #number of lambdas to evaluate
                lambda.type    = "min", #which lambda to select from based on inner loop cv
                                        #to apply to full model within each outer loop
                num_outer_rep  = 100, #number of repeats for outer loop for performance estimation
                outer_cv_nfolds = 5,  #number of folds for outer loop of cross-validation
                font_size   = 18,
                run.parallel    = T,
                numCores = max(1, (detectCores() - 1)),
                # response = surv.resp,
                # cont_X = str_subset(colnames(dat), "_Q"),
                alpha = 0.95,
                verbose = T)

# Run Model (and save it)

set.seed(46363)
if(!file.exists("/stash/results/dev/apfela/P04888_Vignette_grpLasso/full_Anonymous.rds")) {

  full_grpsurv <- rep_grpsurv(data = mod_dat, preds = PARAMS$x,  time = PARAMS$time,
                              status = PARAMS$status, group = PARAMS$grp, penalty = PARAMS$penalty,
                              se = PARAMS$se, nlambda = PARAMS$nlambda, lsp = PARAMS$lsp,
                              num.repeats = PARAMS$num.repeats, lambda.type = PARAMS$lambda.type,
                              n.folds = PARAMS$nfolds, alpha = PARAMS$alpha, returnY = PARAMS$returnY,
                              numCores = PARAMS$numCores, returnX = PARAMS$returnX)

  saveRDS(full_grpsurv, file = file.path(results, "full_Anonymous.rds"))
} else {
  full_grpsurv <- readRDS(file = "/stash/results/dev/apfela/P04888_Vignette_grpLasso/full_Anonymous.rds
}
```

5 Visualizing the Results

5.1 CV Error by Lambda

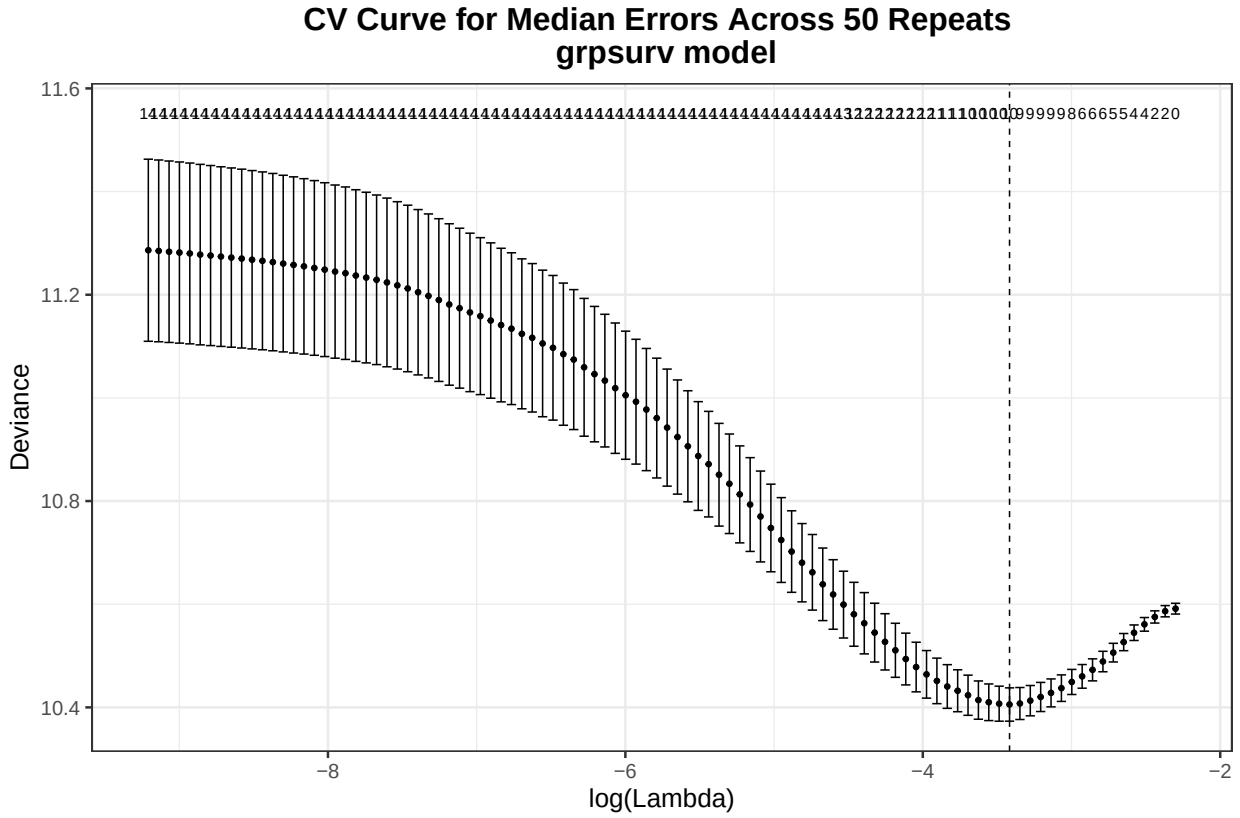
The first useful visualization is to look at the distribution of CV Error across the range of lambda values.

You would like to see a figure which shows a decrease in error as lambda increases and at a certain influx point see the error increase as lambda increases. This point of influx would be the "best lambda".

What you do NOT want to see is the CV Error curve plateau as lambda increases. This would tell us that the model with the highest value of lambda (i.e. the Null Model) is the best performing model. This implies the chosen model does not provide any meaningful information.

```
# Summarize results from full grpreg model
```

```r
# Create object for grpsurv model to making coding more efficient
full_DFS <- full_grpsurv$enet.model


# Set up data for CV Error vs lambda plot
err_DFS <- as.data.frame(full_grpsurv$err)
sum_err <- data.frame(err.median = apply(err_DFS, 2, median), err.sd = apply(err_DFS, 2, sd))

sum_err$index <- seq_len(nrow(sum_err))
sum_err$log_lambda <- log(as.numeric(rownames(sum_err)))


# Calculate number of groups with non-zero coefficients for each lambda
X  <-  mod_dat[, PARAMS$x]

ngroups <- predict(full_DFS, X = X, type = "ngroups")

# Prepare for plot
numcoef <- NULL
for(s in 1:ncol(full_DFS$beta)) {
  numcoef[s] <- ngroups[s]
}

sum_err <- cbind(sum_err, numcoef)
min_log_lambda <- sum_err[which.min(sum_err$err.median),"log_lambda"]
min_lambda <- exp(sum_err[which.min(sum_err$err.median),"log_lambda"])
index <- which.min(sum_err$err.median)

plot.title <- paste("CV Curve for Median Errors Across 50 Repeats \n grpsurv model")


ggplot(sum_err, aes(x = log_lambda, y = err.median)) + geom_point() +
  geom_errorbar(aes(ymin = err.median - err.sd, ymax = err.median + err.sd)) +
  geom_text(mapping = aes(x = log_lambda,
                          y = max(err.median + err.sd +
                                    ((max(err.median) - min(err.median)) / 10)),
                          label = numcoef)) +
  geom_vline(aes(xintercept = sum_err[which.min(sum_err$err.median),"log_lambda"]),
             linetype = "dashed") +
  labs(title = plot.title) +
  theme_bw(PARAMS$font_size) +
  ylab("Deviance") +
  xlab("log(Lambda)") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```



In our example we see a nice dip followed by a steep rise in CV Error, a sign that there is helpful information provided by our model.

5.2 Lambda Trajectory Plots

Lambda Trajectory plots help give an overview of the strength of the association between each of the variables in the model with the outcome and how that association changes across the range of lambda. In this application, we are interested in the overall association between each variable with outcome as well as the strength of the interaction between each variable and treatment with the outcome. Therefore we make 2 types of Lambda Trajectory plots. This is also useful forgetting a sense of how important each variable is relative to the others.

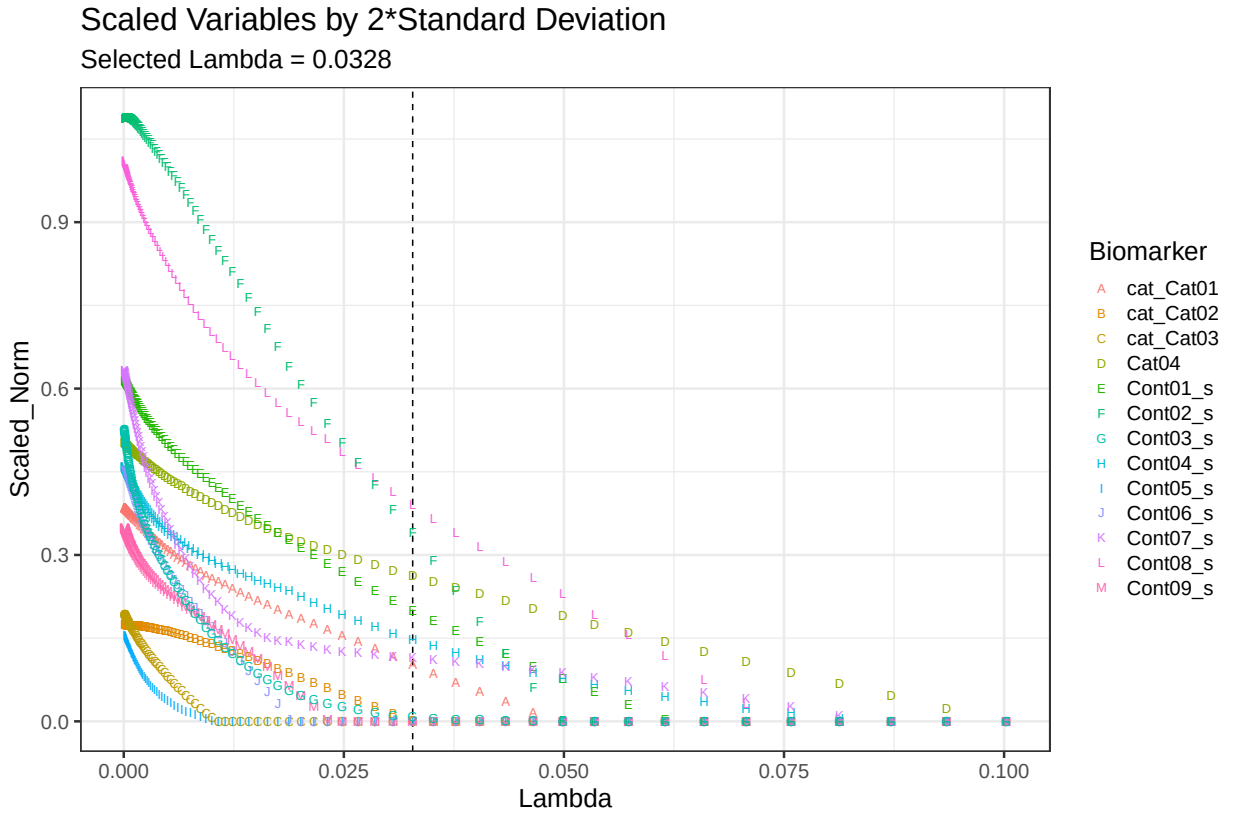
1. Overall Effect - We take the Group Norm (sqrt of the sum of the squares) of the coefficient from all terms in each "group" divided by the sqrt of the number of terms in that group.
2. Interaction Terms - We take the Group Norm of only the interaction terms from each group divided by the number of terms in that group.

We demonstrate 2 options to visualize these plots: the standard trajectory plot and a faceted version. The faceted version may be particularly useful when there is strong overlap in the tragections of many variables.

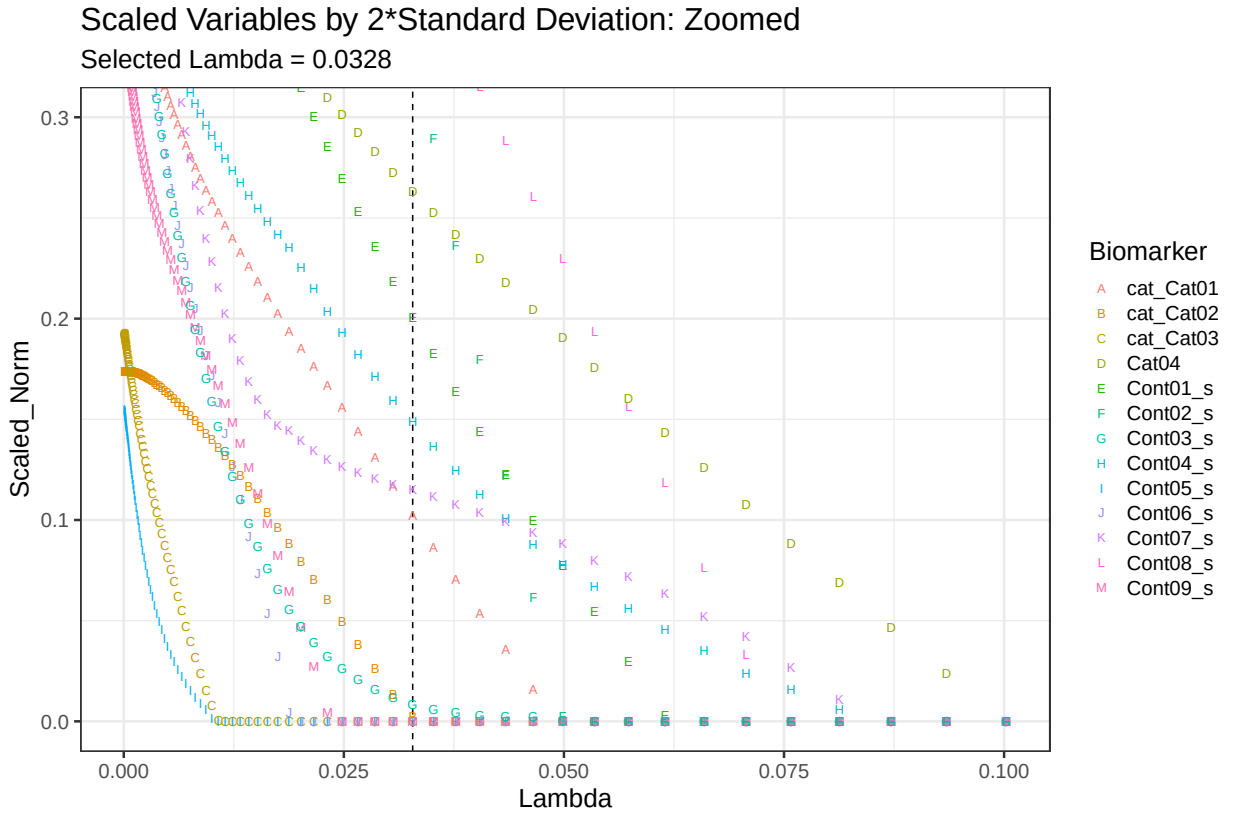
At times it might be useful to zoom in the event one variable distorts the scale of the y-axis. This wasn't the case here, but we illustrate anyway for when it turns to be useful.

5.2.1 Overall Effect

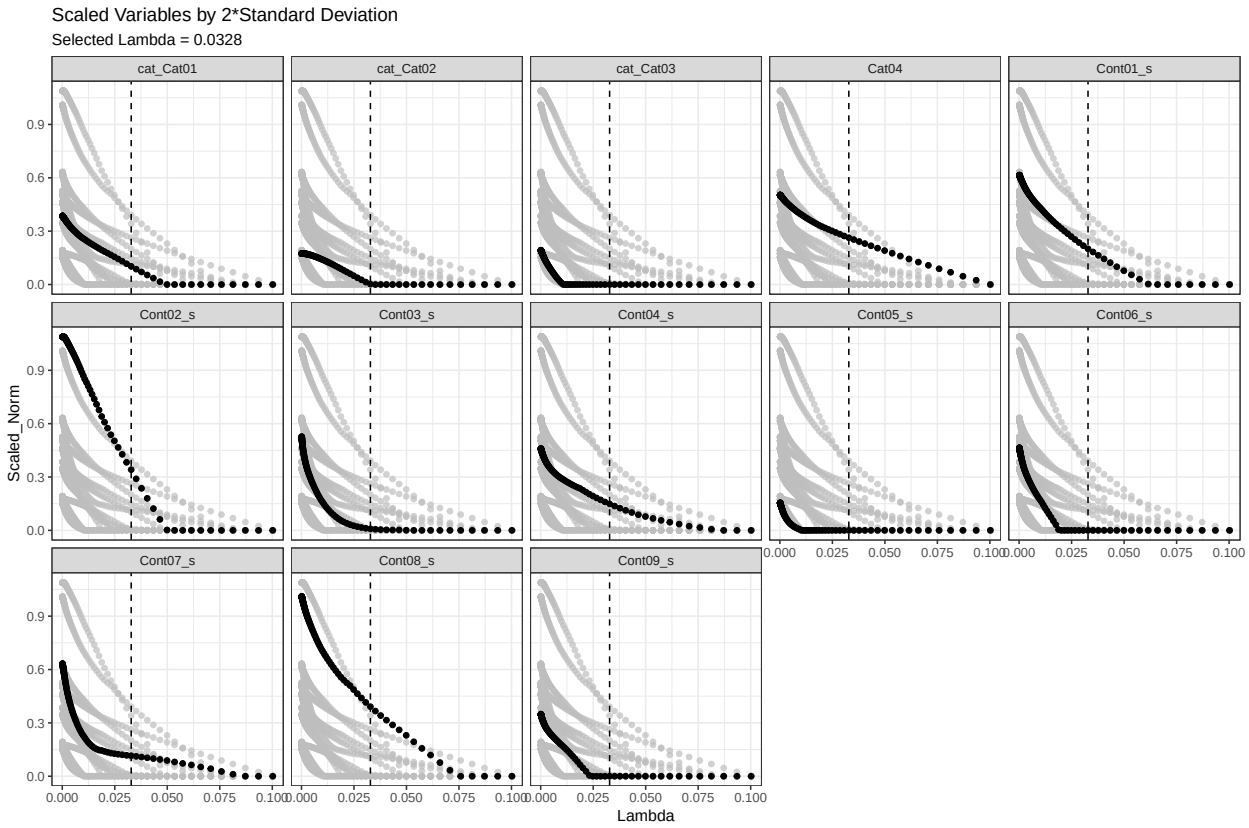
```r
# List of all variables to be included in model
mod_var <- c(mod_trt, mod_clin, "Cat04", cont_bm, Ind_bm)

# Make labels for Beta trajectory plot
labs <- paste0(0:(length(mod_var) - 1), " = ", mod_var)


# Create dataframe of lambda and group norms
lambda_group <- predict(full_DFS, X = X, type = "norm")
lambda_group <- as.data.frame(lambda_group)

# Fix column names so that lambda values aren't in scientific notation
colnames(lambda_group) <- round(full_DFS$lambda, 7)


# Set up data for plotting

# Create dataframe mapping coefficients for variables to different lambda values
lambda_group <- rownames_to_column(lambda_group, var = "Group")
lambda_group <- mutate(lambda_group, Biomarker = mod_var)
lambda_group <- filter(lambda_group, Group != 0)

# Transpose such that each row has coefficient for different variable/lambda combination
lambda_group_l <- pivot_longer(lambda_group, -c(Group, Biomarker), names_to = "Lambda",
                               values_to = "Norm")
lambda_group_l$Lambda <- as.numeric(lambda_group_l$Lambda)
lambda_group_l <- mutate(lambda_group_l, log_lambda = log(Lambda))

# Calculate number of terms in each group to appropriately calculate Scaled Norms
tmp <- data.frame(Biomarker = lambda_group$Biomarker, k = c(rep(2, length(mod_clin)),
                                                            length(status)*2,
                                                            rep(4, length(cont_bm)),
                                                            rep(6, length(Ind_bm))))
lambda_group_l <- inner_join(lambda_group_l, tmp)
lambda_group_l <- mutate(lambda_group_l, Scaled_Norm = Norm/sqrt(k))


# Make plot with Lambda on x-axis
ggplot(lambda_group_l, aes(x = Lambda, y = Scaled_Norm, color = Biomarker)) +
  scale_shape_manual(values = seq(65, length.out = (length(unique(lambda_group_l$Biomarker))))) +
  # guides(shape = F) +
  # scale_x_reverse() +
  geom_vline(xintercept = min_lambda, linetype = "dashed") +
  geom_point(aes(shape = Biomarker), size = 3) +
  ggtitle("Scaled Variables by 2*Standard Deviation", subtitle = paste0("Selected Lambda = ",
                                                                        round(min_lambda, 4))) +
  theme_bw(18)
```



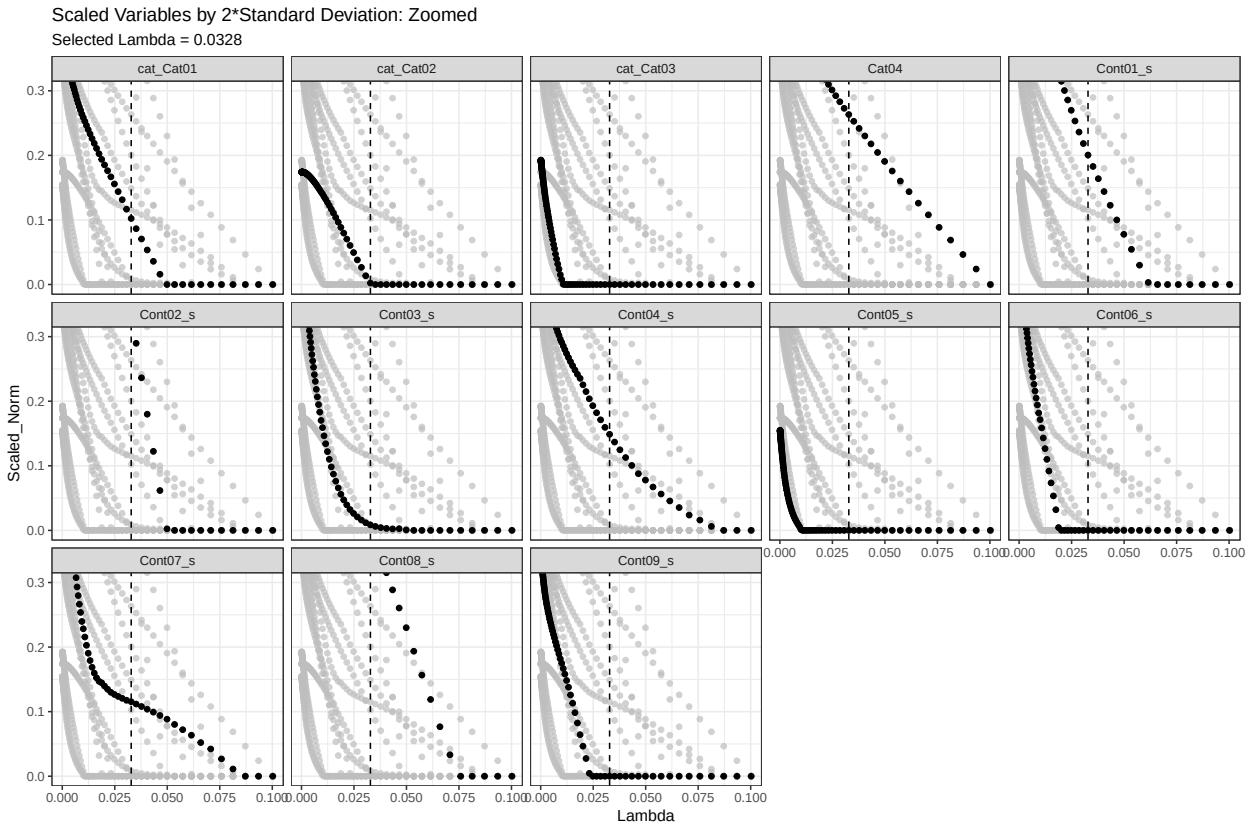

```
# Zoomed In
ggplot(lambda_group_l, aes(x = Lambda, y = Scaled_Norm, color = Biomarker)) +
  scale_shape_manual(values = seq(65, length.out = (length(unique(lambda_group_l$Biomarker))))) +
  # guides(shape = F) +
  # scale_x_reverse() +
  geom_vline(xintercept = min_lambda, linetype = "dashed") +
  geom_point(aes(shape = Biomarker), size = 3) +
  coord_cartesian(ylim = c(0, 0.3)) +
  ggtitle("Scaled Variables by 2*Standard Deviation: Zoomed",
          subtitle = paste0("Selected Lambda = ", round(min_lambda, 4))) +
  theme_bw(18)
```




```
# Faceted version
ggplot(lambda_group_l, aes(x = Lambda, y = Scaled_Norm)) +
  scale_shape_manual(values = seq(65, length.out = length(unique(lambda_group_l$Biomarker)))) +
  geom_point(data = lambda_group_l %>%
               dplyr::select(-Biomarker), colour = "grey", alpha = 0.7) +
  geom_point(aes(color = Biomarker), color = "black") +
  facet_wrap(Biomarker ~ ., nrow = 4, ncol = 5) +
  theme_bw() +
  theme(legend.position = "none") +
  ggtitle("Scaled Variables by 2*Standard Deviation",
          subtitle = paste0("Selected Lambda = ", round(min_lambda, 4))) +
  geom_vline(xintercept = min_lambda, linetype = "dashed")
```




```
# Faceted version Zoomed In
ggplot(lambda_group_l, aes(x = Lambda, y = Scaled_Norm)) +
  scale_shape_manual(values = seq(65, length.out = length(unique(lambda_group_l$Biomarker)))) +
  geom_point(data = lambda_group_l %>%
               dplyr::select(-Biomarker), colour = "grey", alpha = 0.7) +
  geom_point(aes(color = Biomarker), color = "black") +
  facet_wrap(Biomarker ~ ., nrow = 4, ncol = 5) +
  theme_bw() +
  theme(legend.position = "none") +
  ggtitle("Scaled Variables by 2*Standard Deviation: Zoomed",
          subtitle = paste0("Selected Lambda = ", round(min_lambda, 4))) +
  geom_vline(xintercept = min_lambda, linetype = "dashed") +
  coord_cartesian(ylim = c(0, 0.3))
```

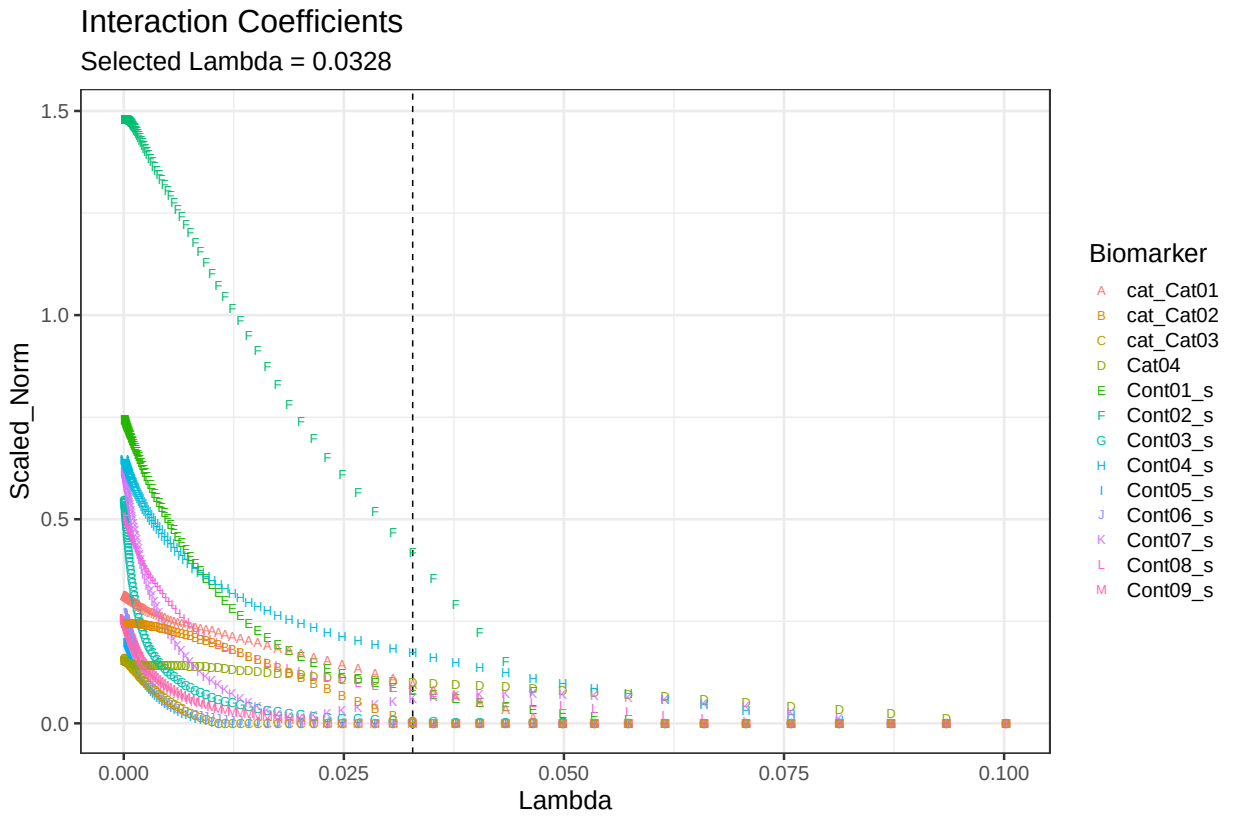


5.2.2 Interaction Plot

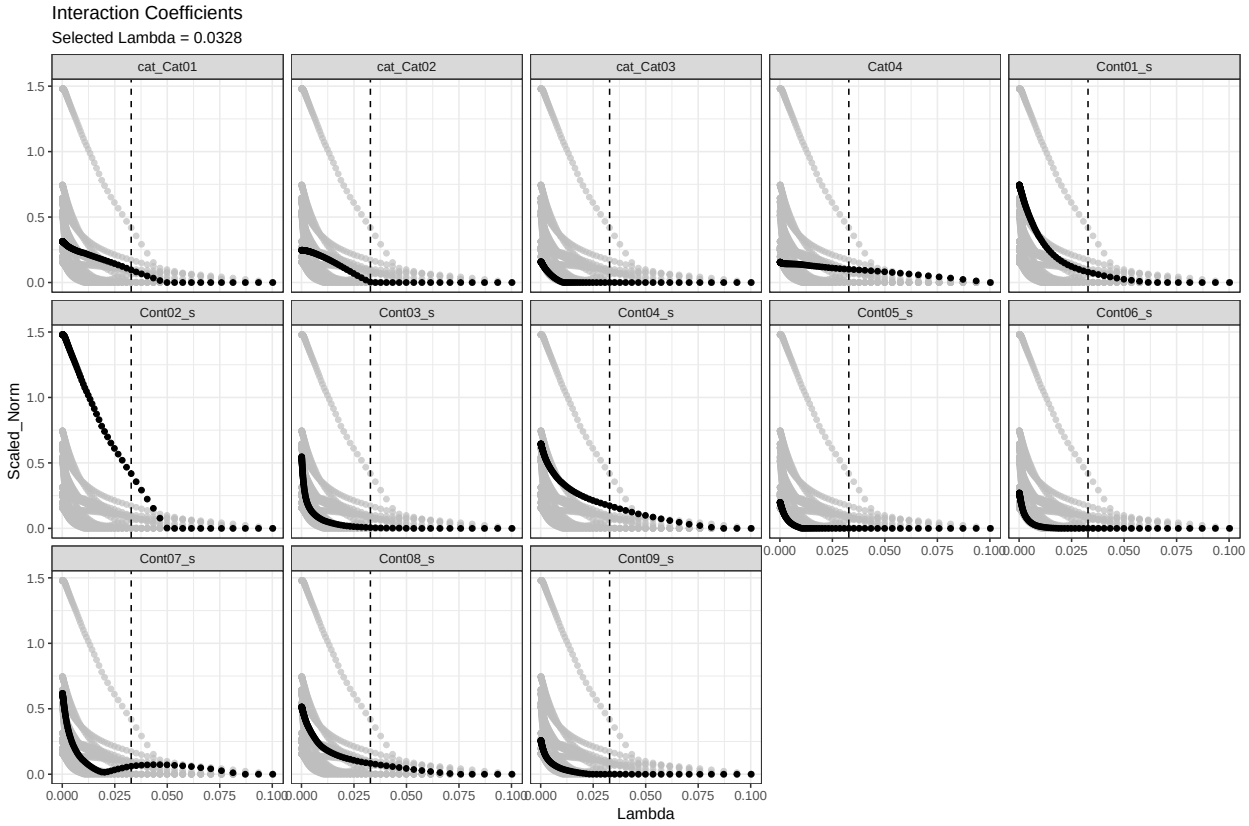
```
#################################################################
# Set up data to make lambda trajectory plot for only Interaction coefficients

# Identify categorical variables with >2 categories
mult_cat <- c("status")

# Create vector of terms included in all categorical variables
cat_var <- c("cat", mult_cat)



# Get coefficients across all lambdas
coef_lambda <- predict(full_DFS, X = X, type = "coefficients")
coef_lambda <- as.data.frame(coef_lambda)

# Fix column names so that lambda values aren't in scientific notation
colnames(coef_lambda) <- round(full_DFS$lambda, 7)

coef_lambda <- rownames_to_column(coef_lambda, var = "Var")

coef_lambda <- mutate(coef_lambda,
                      Type = ifelse(str_detect(Var, pattern = "INT"),
                                    "Interaction", "Main_Effect"))
```

```
coef_lambda <- filter(coef_lambda, Var != "TRT2")
coef_lambda <- filter(coef_lambda, Var != "(Intercept)")
coef_lambda <- mutate(coef_lambda,
                      CAT = ifelse(str_detect(Var, pattern = paste(cat_var, collapse = "|")),
                                   1, 0 ))

# Simplify Variable Names such that single name covers all columns for given variable
coef_lambda <- mutate(coef_lambda, BM =
                        ifelse(CAT == 0 & Type == "Main_Effect" &
                                         !str_detect(Var, pattern = "_Ind"),
                                       str_sub(Var, 1, -2),
                               ifelse(CAT == 0 & Type == "Main_Effect" &
                                        str_detect(Var, pattern = "_Ind"),
                                      str_sub(Var, 1, -5),
                                      ifelse(CAT == 0 & Type == "Interaction" &
                                               !str_detect(Var, pattern = "_Ind_"),
                                             str_sub(Var, 1, -9),
                                             ifelse(CAT == 0 & Type == "Interaction" &
                                                      str_detect(Var, pattern = "_Ind_"),
                                                    str_sub(Var, 1, -12),
                                                    ifelse(CAT == 1 &
                                                             Type == "Interaction" &
                                                             !str_detect(Var,
                                                                         pattern =
                                                                           paste(mult_cat,
                                                                                 collapse = "|")),
                                                           str_sub(Var, 1, -8),
                                                           ifelse(CAT == 1 &
                                                                    Type == "Interaction" &
                                                                    str_detect(Var, pattern =
                                                                                 paste(mult_cat, colla
                                                                  str_sub(Var, 1, -11), Var)))))))

# Remove prefixes of continuous variables
coef_lambda <- mutate(coef_lambda, BM2 = ifelse(CAT == 0, str_sub(BM, 5), BM))




# Only keep interaction coefficients
coef_Int <- filter(coef_lambda, Type == "Interaction")

# Remove unnecessary columns
coef_Int <- dplyr::select(coef_Int, -CAT)

# Collapse Multilevel categorical variables into one name
coef_Int <- mutate(coef_Int, group = ifelse(str_detect(Var, pattern = "status"), "Cat04", BM2))


# Calculate number of terms in each group for scaling purposes
coef_Int <- coef_Int %>% group_by(group) %>%
  mutate(k = length(BM)) %>%
  # mutate(Norm = sqrt(sum(coef^2)/k)) %>%
```

```
  ungroup()

# Remove unnecessary columns
coef_Int <- dplyr::select(coef_Int, -BM, -BM2, -Type)


# Transpose data to have column for lambda
coef_Int_l <- pivot_longer(coef_Int, -c(Var, group, k), names_to = "Lambda", values_to = "coef")
coef_Int_l$Lambda <- as.numeric(coef_Int_l$Lambda)
coef_Int_l <- mutate(coef_Int_l, log_lambda = log(Lambda))

# Calculate scaled norms
coef_Int_l <- coef_Int_l %>%
  group_by(Lambda, group) %>%
  mutate(Scaled_Norm = sqrt(sum(coef^2)/k)) %>%
  distinct(group, .keep_all = T) %>% # Remove duplicate rows of variable within a group
  arrange(Lambda, group, Var) %>%
  ungroup()

# Rename group to more intuitive name
coef_Int_l <- rename(coef_Int_l, Biomarker = group)

# Make plot with Lambda on x-axis
ggplot(coef_Int_l, aes(x = Lambda, y = Scaled_Norm, color = Biomarker)) +
  scale_shape_manual(values = seq(65, length.out = (length(unique(coef_Int_l$Biomarker))))) +
  # guides(shape = F) +
  # scale_x_reverse() +
  geom_vline(xintercept = min_lambda, linetype = "dashed") +
  geom_point(aes(shape = Biomarker), size = 3) +
  ggtitle("Interaction Coefficients",
          subtitle = paste0("Selected Lambda = ", round(min_lambda, 4))) +
  theme_bw(18)
```




```
# Faceted version
ggplot(coef_Int_l, aes(x = Lambda, y = Scaled_Norm)) +
  scale_shape_manual(values = seq(65, length.out = length(unique(coef_Int_l$Biomarker)))) +
  geom_point(data = coef_Int_l %>%
               dplyr::select(-Biomarker), colour = "grey", alpha = 0.7) +
  geom_point(aes(color = Biomarker), color = "black") +
  facet_wrap(Biomarker ~ ., nrow = 4, ncol = 5) +
  theme_bw() +
  theme(legend.position = "none") +
  ggtitle("Interaction Coefficients", subtitle = paste0("Selected Lambda = ",
                                                        round(min_lambda, 4))) +
  geom_vline(xintercept = min_lambda, linetype = "dashed")
```



5.3 Partial Effects Plots

In conjunction to seeing the strength of association between each variable and outcome, it is useful to see what is the shape of the association.

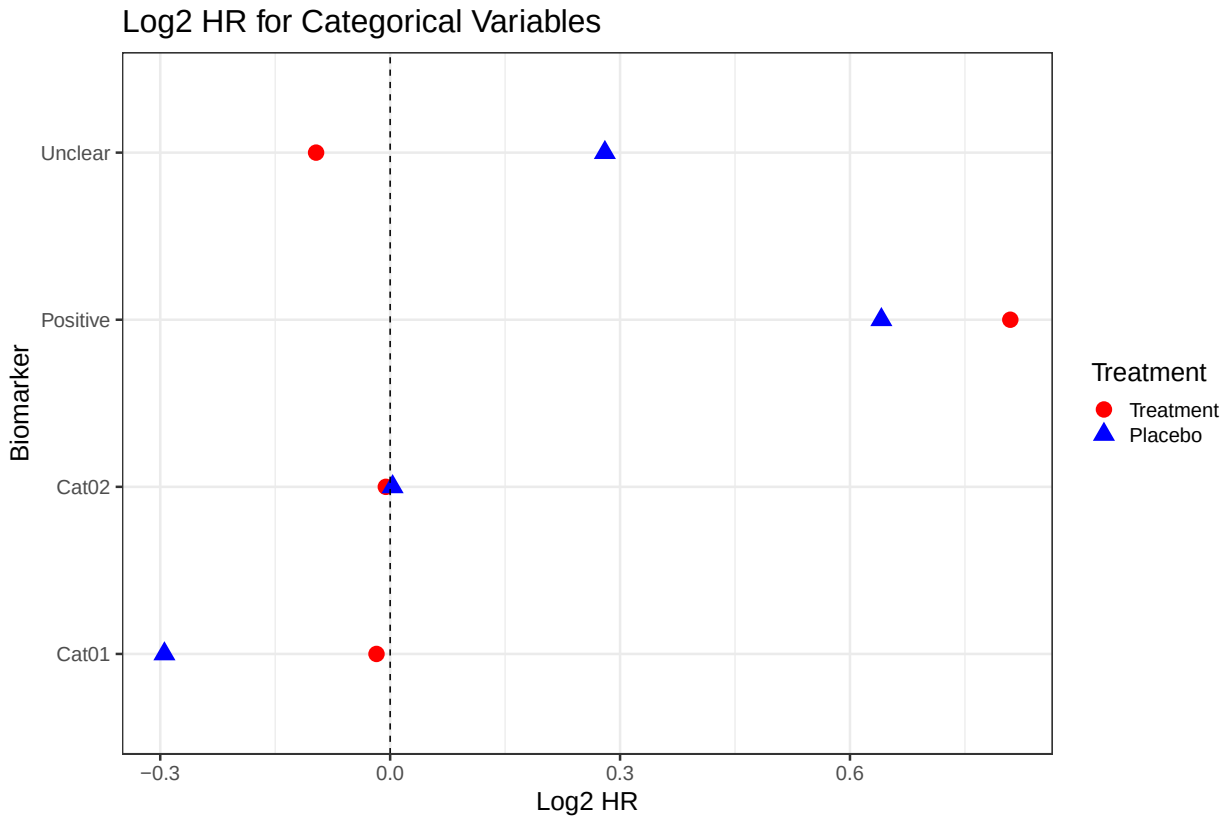
5.3.1 Categorical Variables

The associations for Categorical variables on the other hand can be summarized by the more familiar forest plots.

```
# Partial Effects Plots


# Get prediction matrix for selected lambda to use in PE plots
coef <- predict(full_DFS, X = X, type = "coefficients", which = index)
coef <- rownames_to_column(as.data.frame(coef), var = "Var")
coef <- rename(coef, coef = as.character(round(full_grpsurv$selected.mod["lambda",], 4)))
coef <- mutate(coef, Type = ifelse(str_detect(Var, pattern = "INT"),
                                   "Interaction", "Main_Effect"))
coef <- filter(coef, Var != "TRT2")
coef <- filter(coef, Var != "(Intercept)")
coef <- mutate(coef, CAT = ifelse(str_detect(Var, pattern = paste(cat_var, collapse = "|")),
                                  1, 0 ))

# Simplify Variable Names such that single name covers all columns for givevn variable
coef <- mutate(coef,
```

```
                BM = ifelse(CAT == 0 & Type == "Main_Effect" &
                              !str_detect(Var, pattern = "_Ind"), str_sub(Var, 1, -2),
                                  ifelse(CAT == 0 & Type == "Main_Effect" &
                                           str_detect(Var, pattern = "_Ind"), str_sub(Var, 1, -5),
                                  ifelse(CAT == 0 & Type == "Interaction" &
                                           !str_detect(Var, pattern = "_Ind_"),
                                         str_sub(Var, 1, -9),
                                         ifelse(CAT == 0 & Type == "Interaction" &
                                                  str_detect(Var, pattern = "_Ind_"),
                                                str_sub(Var, 1, -12),
                                         ifelse(CAT == 1 & Type == "Interaction" &
                                                  !str_detect(Var, pattern = "status"),
                                                str_sub(Var, 1, -8),
                                                ifelse(CAT == 1 & Type == "Interaction" &
                                                         str_detect(Var, pattern = "status"),
                                                       str_sub(Var, 1, -11), Var)))))))

# Remove variables which were filtered out
coef <- filter(coef, coef != 0)


############################
# Categorical


# Set up data for scatter plot of categorical coefficients
coef_CAT <- filter(coef, CAT == 1) %>% dplyr::select(-CAT)
coef_CAT <- mutate(coef_CAT, group = ifelse(str_detect(Var, pattern = "status"), "Cat04", BM))

# Calculate scaled norms
coef_CAT <- coef_CAT %>% group_by(group, Type) %>%
  mutate(k = length(coef)) %>%
  mutate(Norm = sqrt(sum(coef^2)/k)) %>%
  ungroup()

# Transpose to calculate contrasts
coef_cat_w <- pivot_wider(coef_CAT, id_cols = c(BM, group), names_from = Type,
                          values_from = coef)

# Calculate contrasts
coef_cat_w <- mutate(coef_cat_w, Placebo = Main_Effect - Interaction)
coef_cat_w <- mutate(coef_cat_w, Treatment = Main_Effect + Interaction)
coef_cat_w <- mutate(coef_cat_w, Placebo_HR = exp(Placebo))
coef_cat_w <- mutate(coef_cat_w, Treatment_HR = exp(Treatment))

coef_groups <- predict(full_DFS, X = X, type = "norm", which = index)
names(coef_groups) <- mod_var
coef_groups <- rownames_to_column(as.data.frame(coef_groups), var = "Var")

# Set up data for Table
coef_cat_table <- dplyr::select(coef_CAT, group, Type, k, Norm)
coef_cat_table <- unique(coef_cat_table)
coef_cat_table <- pivot_wider(coef_cat_table, id_cols = c(group, k),
```

```
                              names_from = Type, values_from = Norm)
coef_cat_table <- inner_join(coef_cat_table, coef_groups, by = c("group" = "Var"))
coef_cat_table <- rename(coef_cat_table, Group = coef_groups)
coef_cat_table <- mutate(coef_cat_table, Group = Group/sqrt(k))


# Make dot plot of HR for categorical variables
coef_cat_w <- pivot_longer(coef_cat_w, c(Placebo, Treatment), names_to = "Treatment",
                           values_to = "log_HR")
coef_cat_w <- mutate(coef_cat_w, log2_HR = log_HR*log2(exp(1)))

ggplot(coef_cat_w, aes(y = BM, x = log2_HR, color = Treatment, shape = Treatment)) +
  geom_point(size = 5) +
  scale_shape_discrete(labels=c("Treatment", "Placebo")) +
  scale_linetype_discrete(labels=c("Treatment", "Placebo")) +
  scale_color_manual(values = c("red", "blue"), labels = c("Treatment", "Placebo")) +
  geom_vline(xintercept = 0, linetype = "dashed") +
  scale_y_discrete(labels = c(status_unclear = "Unclear",
                              status_pos = "Positive",
                              cat_Cat02 = "Cat02", cat_Cat01 = "Cat01")) +
  ylab("Biomarker") +
  xlab("Log2 HR") +
  theme_bw(18) +
  ggtitle("Log2 HR for Categorical Variables")
```



5.3.2 Continuous Variables

Partial Effects (PE) plots are particularly important when using splines in the model for continuous variables since the relationship can no longer be summarized simply by the sign of the coefficient and its slope. Rather, since there is no linearity assumption we must visualize the whole relationship through Partial Effects plots.

In order to make PE plots, we have to make predictions across a range of values for each biomarker for each treatment while keeping everything else constant. In general we do not recommend predicting the entire range of observed values because outliers will distort the x-axis.In this case study we make predictions for the middle 90% of the observed data.

When working with splines, it is important to apply the knots at the same locations as they were in the training model.

NOTE: One very important bug to be wary of when making predictions from grpsurv models is that it assumes the variables in the newdata are in the same order as they were in the training data. If the order changes you will get wrong predictions WITHOUT WARNING!

```
# Continuous


##########################################
# Set up data for partial effects plots
##########################################

# Identify knot locations for each continuous covariate
# cont_dat_s1 <- select(cont_dat, str_subset(colnames(cont_dat), "s1"))

# Sanity checks
attributes(cont_dat$mod_Cont04_s1)$parms
```

```
## [1] -0.626292758  0.009231734  0.607766234
```

```
attributes(cont_dat$Ind_Cont02_s1)$parms
```

```
## [1] -0.3507422  0.3535277  0.9012213
```

```
# Interpolate, and apply model knot locations to create splines
cont_raw <- dplyr::select(imp_dat, cont_bm, Ind_bm)
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(Ind_bm)
##
##   # Now:
##   data %>% select(all_of(Ind_bm))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
meds <- sapply(cont_raw, median)

nintp <- 10  # Number of interpolations: Here for illustration purposes we only
             # use 10 but in practice more interpolations may be preferred.
```

```
pred_dat <- list()
pred_dat_names <- list()


# Make same names as in model data
for(i in seq_len(length(cont_bm))){
  pred_dat_names[[i]] <-  c(paste0("mod_", cont_bm[i], "1"), paste0("mod_", cont_bm[i], "2"))
}


# Start with only cont versions of Cont02 and Cont03 we'll add Ind versions later
for(i in seq_len(length(Ind_bm))){
  pred_dat_names[[i + length(cont_bm)]] <-  c(paste0("Ind_", Ind_bm[i], "1"),
                                              paste0("Ind_", Ind_bm[i], "2"))
}

pred_dat_names <- unlist(pred_dat_names)

# Create vector of ALL cont bm
all_bm <- c(cont_bm, Ind_bm)

# For 1st BM we create 2 (one for each ARM) sequences from 5% to 95% followed by
# median for remaining BM
# For 2nd BM and on we start with median for preceding BM followed by sequence
for(i in seq_len(length(cont_bm))){
  if(i == 1) {
    pred_dat[[cont_bm[i]]] <- c(rep(seq(quantile(imp_dat[[cont_bm[i]]], 0.05),
                                        quantile(imp_dat[[cont_bm[i]]], 0.95),
                                        length.out = nintp), 2), rep(meds[i],
                                                                     (length(all_bm)-i)*nintp*2))
  } else {
    pred_dat[[cont_bm[i]]] <- c(rep(meds[i], (i-1)*nintp*2),
                                rep(seq(quantile(imp_dat[[cont_bm[i]]], 0.05),
                                        quantile(imp_dat[[cont_bm[i]]], 0.95),
                                        length.out = nintp), 2),
                                rep(meds[i], (length(all_bm)-i)*nintp*2))
  }

  # Create 2 basis functions from interpolated data using same knots as in model
  # NOTE: This code assumes there were only 3 knots in model for all continuous variables
  pred_dat[[paste0("mod_", cont_bm[i], "1")]] <-
    rcs(pred_dat[[cont_bm[[i]]]],
        attributes(cont_dat[[paste0("mod_", cont_bm[i], "1")]])$parms,
        name = pred_dat_names[(2*i) - 1])[,1]
  attr(pred_dat[[paste0("mod_", cont_bm[i], "1")]], "dimnames") <- NULL
  pred_dat[[paste0("mod_", cont_bm[i], "2")]] <-
    rcs(pred_dat[[cont_bm[[i]]]],
        attributes(cont_dat[[paste0("mod_", cont_bm[i], "2")]])$parms,
        name = pred_dat_names[2*i])[,2]
  attr(pred_dat[[paste0("mod_", cont_bm[i], "2")]], "dimnames") <- NULL
}

# For Cont02/Cont03 treat as 2nd BM or later from previous chunk
```

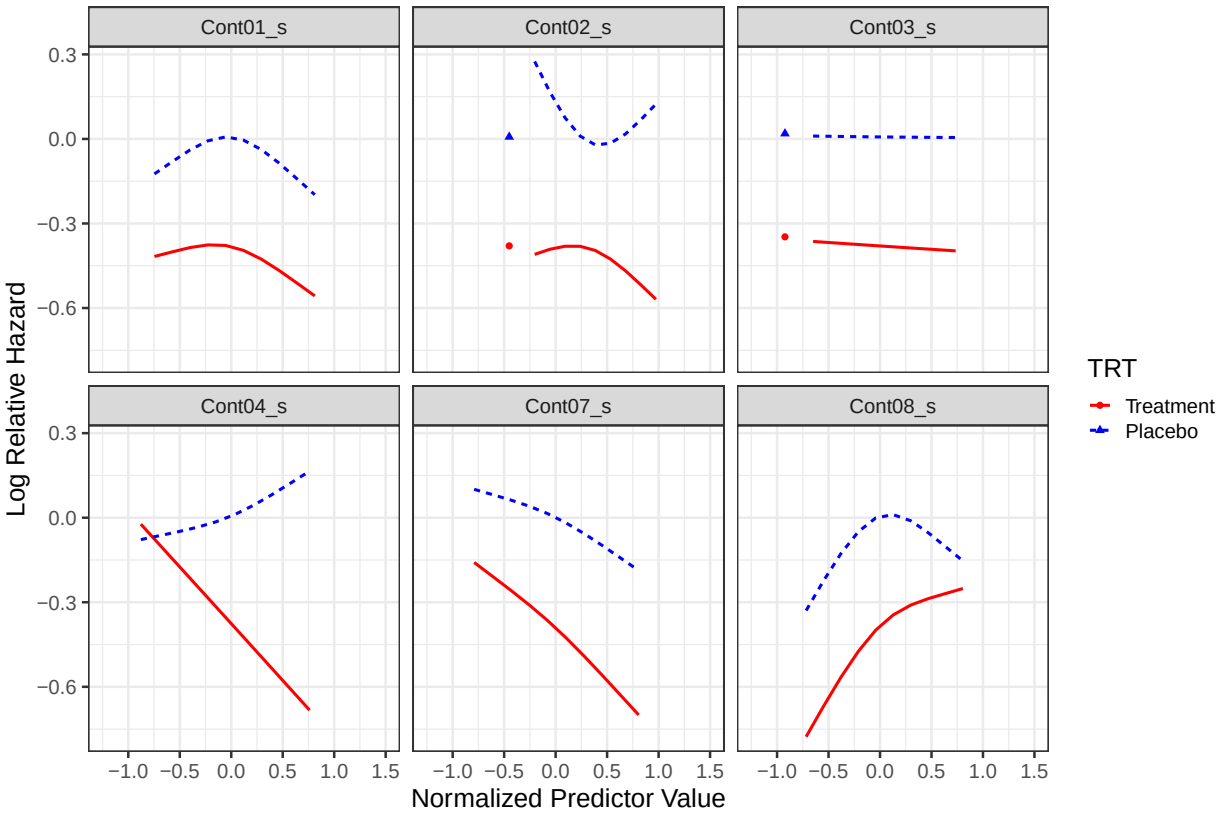
```r
for(i in seq_len(length(Ind_bm))){
  j <- length(cont_bm) + i # Convenience to recalibrate Ind_bm to cont_bm
  pred_dat[[Ind_bm[i]]] <- c(rep(meds[j], (j - 1)*nintp*2),
                             rep(seq(quantile(imp_dat[[Ind_bm[i]]], 0.05),
                                     quantile(imp_dat[[Ind_bm[i]]], 0.95),
                                     length.out = nintp), 2),
                             rep(meds[j], (length(all_bm)-j)*nintp*2))

  pred_dat[[paste0("Ind_", Ind_bm[i], "1")]] <-
    rcs(pred_dat[[Ind_bm[[i]]]],
        attributes(cont_dat[[paste0("Ind_", Ind_bm[i], "1")]])$parms,
        name = pred_dat_names[(2*j) - 1])[,1]
  attr(pred_dat[[paste0("Ind_", Ind_bm[i], "1")]], "dimnames") <- NULL
  pred_dat[[paste0("Ind_", Ind_bm[i], "2")]] <-
    rcs(pred_dat[[Ind_bm[[i]]]],
        attributes(cont_dat[[paste0("Ind_", Ind_bm[i], "2")]])$parms,
        name = pred_dat_names[2*j])[,2]
  attr(pred_dat[[paste0("Ind_", Ind_bm[i], "2")]], "dimnames") <- NULL
}


pred_dat <- as.data.frame(pred_dat)

# Create Indicator functions for Cont02/Cont03
pred_dat <- mutate(pred_dat,
                   Ind_Cont02_s_Ind = ifelse(Cont02_s == min(imp_dat$Cont02_s), 1, 0))
pred_dat <- mutate(pred_dat,
                   Ind_Cont03_s_Ind = ifelse(Cont03_s == min(imp_dat$Cont03_s), 1, 0))

# Keep only terms used in model
pred_dat <- dplyr::select(pred_dat, str_subset(colnames(pred_dat), "mod"),
                          str_subset(colnames(pred_dat), "Ind"))

# Create TRT2
pred_dat$TRT2 <- rep(c(1, -1), each = nintp, times = length(all_bm))

# Create vector of all categorical variable names used in model
cat_mod <- c(mod_clin, status)

# Create categorical variables set to reference levels
for(var in cat_mod){
  pred_dat[[var]] <- 0
}

# Create interactions for all variables
pred_dat <- mutate_at(pred_dat, bm, list(modINT = ~. * TRT2))
pred_dat <- mutate_at(pred_dat, status, list(statusINT = ~. * TRT2))
pred_dat <- mutate_at(pred_dat, clin, list(catINT = ~. * TRT2))
pred_dat <- mutate_at(pred_dat, Ind, list(IndINT = ~. * TRT2))


##################################
# Get predictions
```

```r
###################################

# !!!! BUG IN predict.grpsurv !!!!
# Must have prediction matrix IN SAME ORDER as training data!
# If not, results will be INCORRECT WITH NO WARNING!!!

# Ensure prediction matrix is same order as training data
pred_dat <- dplyr::select(pred_dat, PARAMS$x)
pred_mat <- as.matrix(pred_dat)

# Get Predictions
pred_dat$preds <- predict(full_DFS, X = pred_mat, type = "link", which = index)

# Identify which BM is getting predicted
pred_dat$BM <- rep(all_bm, each = 2*nintp)

#################################
# Make partial effects plots
#################################
# Set up data for plotting
raw_bm <- sub("_s$", "", all_bm)
plot_dat <- list()

# Create dataframe of variables on original scale, but mapping predictions
for(i in seq_len(length(all_bm))) {
  plot_dat[[all_bm[i]]] <- rep(seq(quantile(imp_dat[[all_bm[i]]], 0.05),
                                   quantile(imp_dat[[all_bm[i]]], 0.95),
                                   length.out = nintp), 2)
}


plot_dat <- as.data.frame(plot_dat)

plot_dat$TRT <-  rep(c("Treatment", "Placebo"), each = nintp)

plot_dat <- pivot_longer(plot_dat, -TRT, names_to = "Biomarker", values_to = "Value")

# Rearrange in order to match up order with pred_dat
plot_dat <- arrange(plot_dat, match(Biomarker, all_bm), TRT, Value)

# We recommend do some spot checking here to make sure everything lines up between
# plot_dat and pred_dat

# Since everything matches, we transfer preds
plot_dat$preds <- pred_dat$preds

# Identify filtered BM
filtered <-
  mod_var[!(mod_var %in% mod_var[predict(full_DFS, X = X, type = "groups", which = index)])]

# Filter out biomarkers removed
plot_dat_filtered <- filter(plot_dat, !(Biomarker %in% filtered))
```

```r
# Separate out point of discontinuity for better visualization
nuggets <- filter(plot_dat_filtered,
                  Biomarker == "Cont02_s" & Value == min(imp_dat$Cont02_s) |
                    (Biomarker == "Cont03_s" & Value == min(imp_dat$Cont03_s)))
nuggets <- mutate(nuggets, Value2 = Value - 0.1)


no_nugget <- filter(plot_dat_filtered,
                    (Biomarker == "Cont02_s" & Value != min(imp_dat$Cont02_s)) |
                      Biomarker != "Cont02_s")
no_nugget <- filter(no_nugget, (Biomarker == "Cont03_s" & Value != min(imp_dat$Cont03_s)) |
                      Biomarker != "Cont03_s")


tmp <- ggplot(no_nugget, aes(x = Value, y = preds, color = TRT, linetype = TRT)) +
  geom_line(size = 1) +
  # ylim(-3, 2) +
  xlim(-1.25, 1.5) +
  facet_wrap(~Biomarker) +
  ylab("Log Relative Hazard") +
  xlab("Normalized Predictor Value") +
  scale_shape_discrete(labels=c("Treatment", "Placebo")) +
  scale_linetype_discrete(labels=c("Treatment", "Placebo")) +
  scale_color_manual(values = c("red", "blue"), labels = c("Treatment", "Placebo")) +
  theme_bw(18)
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
tmp + geom_point(data=nuggets,
                 aes(x=Value2,y=preds, color = TRT, shape = TRT), size = 2, inherit.aes=FALSE)
```



6 References

Gelman, A., 2008. Scaling regression inputs by dividing by two standard deviations. Statistics in medicine 27, 2865–2873.

Tian, L., Alizadeh, A.A., Gentles, A.J., Tibshirani, R., 2014. A simple method for estimating interactions between a treatment and a large number of covariates. Journal of the American Statistical Association 109, 1517–1532.

7 System Information

Time required to process this report: *26.52957 secs*

R session information:

```
## R version 4.3.1 (2023-06-16)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.6 LTS
##
## Matrix products: default
## BLAS:   /opt/tbio/domino_202310/binaries/R-4.3.1/lib/R/lib/libRblas.so
## LAPACK: /opt/tbio/domino_202310/binaries/R-4.3.1/lib/R/lib/libRlapack.so;  LAPACK version 3.11.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
```

```
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## time zone: America/New_York
## tzcode source: system (glibc)
##
## attached base packages:
## [1] splines   parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] lubridate_1.9.3         forcats_1.0.0           purrr_1.0.2
##  [4] readr_2.1.4             tidyverse_2.0.0         bmsPURR_0.1.34
##  [7] ellipse_0.5.0           CalibrationCurves_2.0.0 grpreg_3.4.0
## [10] corrplot_0.92           timeROC_0.4             survcomp_1.51.0
## [13] prodlim_2023.08.28      survival_3.5-7          survivalROC_1.0.3.1
## [16] TunePareto_2.5.3        knitr_1.44              glinternet_1.0.12
## [19] mice_3.16.0             rms_6.7-1               Hmisc_5.1-1
## [22] tidyr_1.3.0             tibble_3.2.1            stringr_1.5.0
## [25] ggplot2_3.4.3           dplyr_1.1.3             conflicted_1.2.0
##
## loaded via a namespace (and not attached):
##   [1] rstudioapi_0.15.0   jsonlite_1.8.7      shape_1.4.6
##   [4] CatMisc_1.1.5       magrittr_2.0.3      TH.data_1.1-2
##   [7] SuppDists_1.1-9.7   jomo_2.7-6          farver_2.1.1
##  [10] nloptr_2.0.3        rmarkdown_2.25      fs_1.6.3
##  [13] vctrs_0.6.3         memoise_2.0.1       minqa_1.2.6
##  [16] askpass_1.2.0       base64enc_0.1-3     usethis_2.2.2
##  [19] htmltools_0.5.6     polspline_1.1.23    broom_1.0.5
##  [22] Formula_1.2-5       mitml_0.4-5         parallelly_1.36.0
##  [25] KernSmooth_2.23-22  desc_1.4.2          htmlwidgets_1.6.2
##  [28] sandwich_3.0-2      zoo_1.8-12          cachem_1.0.8
##  [31] DominoR_0.0.2       mime_0.12           lifecycle_1.0.3
##  [34] iterators_1.0.14    pkgconfig_2.0.3     Matrix_1.6-1.1
##  [37] R6_2.5.1            fastmap_1.1.1       future_1.33.0
##  [40] shiny_1.7.5         pingr_2.0.2         digest_0.6.33
##  [43] numDeriv_2016.8-1.1 colorspace_2.1-0    ps_1.7.5
##  [46] rprojroot_2.0.3     pkgload_1.3.3       labeling_0.4.3
##  [49] timechange_0.2.0    fansi_1.0.4         httr_1.4.7
##  [52] compiler_4.3.1      remotes_2.4.2.1     withr_2.5.1
##  [55] htmlTable_2.4.1     backports_1.4.1     pkgbuild_1.4.2
##  [58] pan_1.9             MASS_7.3-60         lava_1.7.2.1
##  [61] quantreg_5.97       openssl_2.1.1       sessioninfo_1.2.2
##  [64] myRepository_1.34.0 tools_4.3.1         foreign_0.8-85
##  [67] httpuv_1.6.11       future.apply_1.11.0 bootstrap_2019.6
##  [70] nnet_7.3-19         glue_1.6.2          callr_3.7.3
##  [73] nlme_3.1-163        promises_1.2.1      grid_4.3.1
##  [76] checkmate_2.2.0     getPass_0.2-2       cluster_2.1.4
##  [79] generics_0.1.3      ParamSetI_1.17.0    gtable_0.3.4
##  [82] tzdb_0.4.0          EventLogger_1.15.3  hms_1.1.3
##  [85] data.table_1.14.8   utf8_1.2.3          foreach_1.5.2
##  [88] pillar_1.9.0        later_1.3.1         dynamictable_1.5
```

```
##  [91] lattice_0.21-9      SparseM_1.81        tidyselect_1.2.0
##  [94] miniUI_0.1.1.1      gridExtra_2.3       xfun_0.40
##  [97] devtools_2.4.5      stringi_1.7.12      yaml_2.3.7
## [100] boot_1.3-28.1       pec_2023.04.12      evaluate_0.22
## [103] codetools_0.2-19    cli_3.6.1           rpart_4.1.19
## [106] xtable_1.8-4        processx_3.8.2      munsell_0.5.0
## [109] Rcpp_1.0.11         rmeta_3.0           globals_0.16.2
## [112] MatrixModels_0.5-2  ellipsis_0.3.2      prettyunits_1.2.0
## [115] profvis_0.3.8       urlchecker_1.0.1    lme4_1.1-34
## [118] listenv_0.9.0       glmnet_4.1-8        mvtnorm_1.2-3
## [121] timereg_2.0.5       scales_1.2.1        crayon_1.5.2
## [124] rlang_1.1.1         multcomp_1.4-25
```

7.1 Domino environment details

Domino User: apfela

Domino Project:

Domino Run ID:

Domino Run #: