Shape Constrained Regression for Excel

William Chiu

2022-10-27

Summary

Generalized Additive Models (gam) estimate non-linear but additive relationships between a response and a set of predictors. However, when the training data has a high noise to signal ratio, gam could estimate wiggly (non-monotonic) relationships that are difficult to explain. Shape Constrained Additive Models (scam) impose user-defined monotonic relationships that can be explained by the user.

The `mgcv` and `scam` packages are highly effective tools for developing gam and scam models. Both packages use the p-spline basis to construct non-linear features. Although p-splines have desirable properties, they are difficult to implement in other software tools like Excel. There is no closed-form equation that I could export from gam/scam and into Excel for an end-user.

Multivariate Adaptive Regression Splines (mars), from the `earth` package, allow us to approximate the scam model with linear basis functions – which are Excel-friendly.

Through a data simulation, I demonstrate how to fit gam and scam models using the `mgcv` and `scam` packages. Then I approximate the scam model and provide an Excel-friendly regression equation. Reader should be aware that MARS does *not guarantee* monotonicity. Therefore, after fitting an approximation of the scam model, reader should test the predictions by feeding extreme values of the predictors into the final model.

We also explore the `scar` package which uses monotonic binning. Then I approximate the scar model with non-linear least squares and sigmoid functions. Sigmoid functions guarantee monotonicity. However, sigmoid functions assume that the response tapers off on the tails (i.e., the relationship is S-shaped).

Approximations are achieved by turning the logistic regression problem into a least squares problem. The trick is to use a pseudo response that is continuous (rather than binary).

Data Generating Process (DGP)

The chunk generates 3 data frames: full data, training data, and test data. To demonstrate the wiggliness associated with gam, the sample size of the training data is limited to only 2000 observations.

Each data frame contains a binary response and two predictors. The first predictor has a monotonically increasing relationship with the response, while the second predictor has a monotonically decreasing relationship with the response. Both relationships are sigmoidal and hence highly non-linear.

In addition, the event rate of the response is rare.

```
library(mgcv)
library(scam)
library(tidyverse)
library(caTools)
library(MLmetrics)
```

```
library(earth)
library(plotmo)
library(furrr)
library(scar)
library(minpack.lm)
library(car)
library(cgam)

plan(multisession, workers = 6) # set to 2 for most PCs

set.seed(2001)

nobs <- 100000

x1 <- rnorm(nobs)
x2 <- rnorm(nobs)
x3 <- sample(c('A', 'B', 'C', 'D'), size=nobs, replace=TRUE,
             prob = c(0.8, 0.1, 0.05, 0.05))

# Population log odds is a function of sigmoid features

z <- -35 + 5 * SSfpl(x1, -2, 2, 0, 0.5) - 5 * SSfpl(x2, -1, 1, 0, 0.1) +
        + 5 * I(x3=='B') + 10 * I(x3=='C') + 10 * I(x3=='D') +
  rnorm(nobs,0,20)

y <- rbinom(nobs, 1, prob=boot::inv.logit(z))

full_data <- data.frame(y=y, x1=x1, x2=x2, x3=as.factor(x3))

summary(full_data)
```

```
##        y                 x1                  x2            x3
##  Min.   :0.00000   Min.   :-4.469540   Min.   :-4.349958   A:79992
##  1st Qu.:0.00000   1st Qu.:-0.672327   1st Qu.:-0.667040   B:10044
##  Median :0.00000   Median : 0.006574   Median : 0.002566   C: 5032
##  Mean   :0.06252   Mean   : 0.004819   Mean   : 0.000159   D: 4932
##  3rd Qu.:0.00000   3rd Qu.: 0.681828   3rd Qu.: 0.676167
##  Max.   :1.00000   Max.   : 4.071722   Max.   : 4.590295
```

```
train_index <- sample.split(full_data$y, SplitRatio = 2/100)

train_data <- full_data[train_index,] # small
test_data <- full_data[!train_index,] # large
```

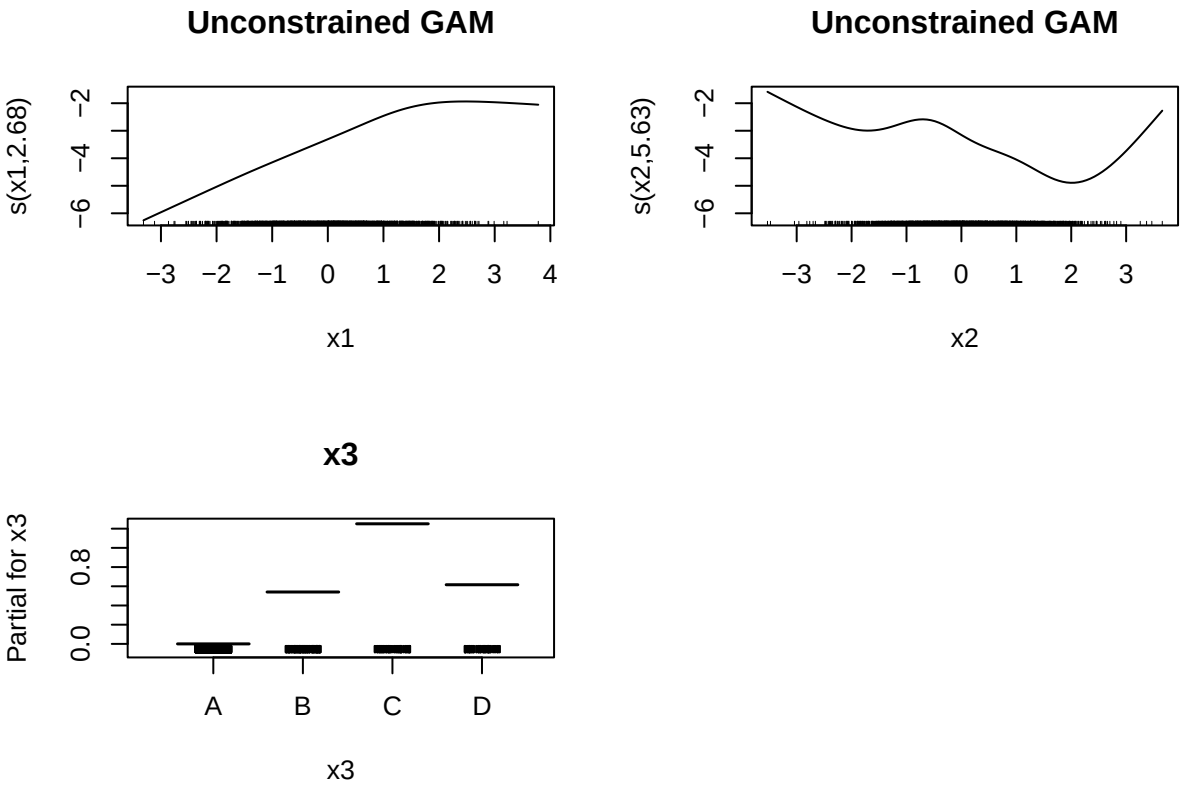
GAM

Due to the high noise to signal ratio in the training data, gam estimates very wiggly relationships between the binary response and the predictors. The y-axis is on the log-odds (or logit) scale.

```r
## fit a gam

## when population error is high, gam returns non-monotonic relationships
## even when the population relationships are monotonic

mod_gam <- gam(y ~ s(x1) + s(x2) + x3, data = train_data, family = binomial)

summary(mod_gam)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## y ~ s(x1) + s(x2) + x3
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -3.3278     0.1492 -22.299  < 2e-16 ***
## x3B           0.5406     0.2876   1.879 0.060212 .
## x3C           1.2506     0.3277   3.816 0.000136 ***
## x3D           0.6160     0.3836   1.606 0.108282
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##         edf Ref.df Chi.sq  p-value
## s(x1) 2.677  3.421  51.80  < 2e-16 ***
## s(x2) 5.634  6.807  30.41 8.49e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0685   Deviance explained = 12.6%
## UBRE = -0.57921  Scale est. = 1         n = 2000
```

```r
plot(mod_gam, pages=1, se=FALSE, main="Unconstrained GAM",
     shift=coef(mod_gam)[1], all.terms=TRUE)
```



SCAM

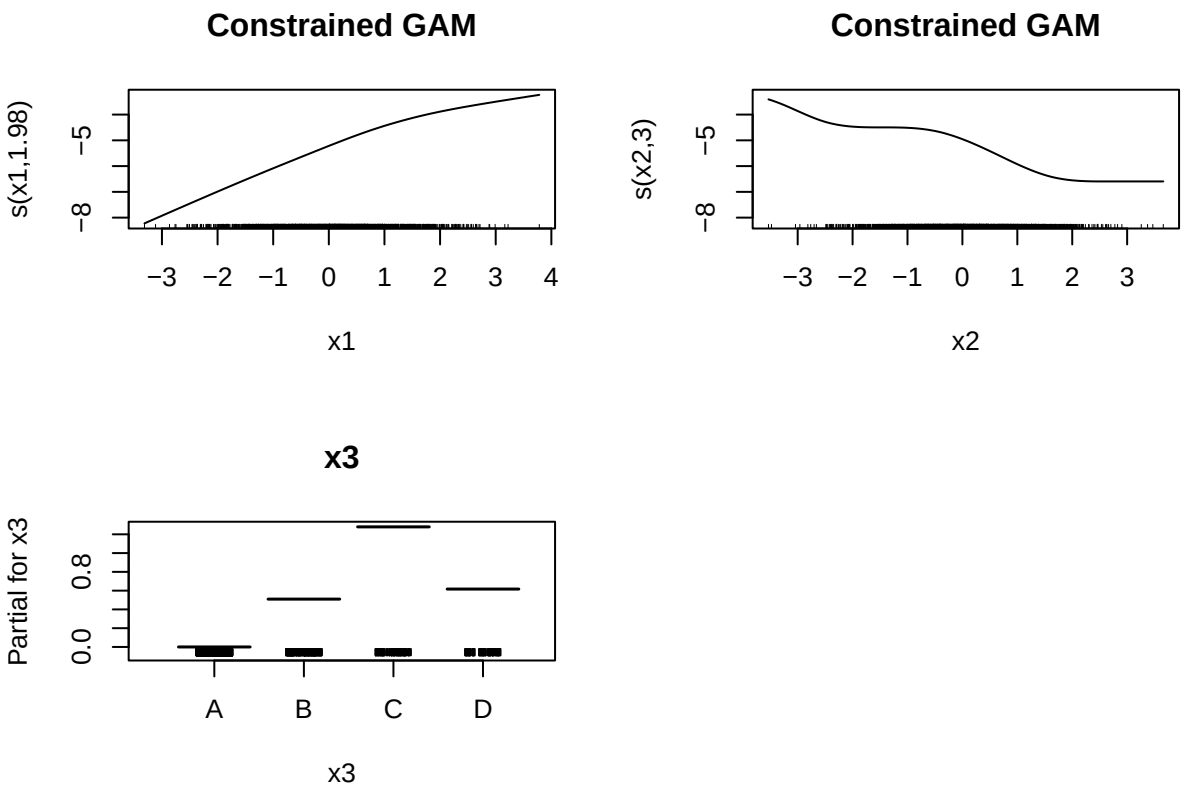
Shape constraints on each predictor reduce variance (wiggliness). Out-of-sample performance is similar between `gam` and `scam`.

```
## gam with monotonic constraints

## scam imposes monotonic relationships between each
## feature and the binary response
## mpi = monotonic p-spline increasing
## mpd = monotonic p-spline decreasing

mod_mono_gam <- scam(y ~ s(x1, bs="mpi") + s(x2, bs="mpd") + x3,
                     data = train_data, family = binomial)

summary(mod_mono_gam)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## y ~ s(x1, bs = "mpi") + s(x2, bs = "mpd") + x3
##
```

```
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -5.2813     2.9437  -1.794   0.0728 .
## x3B           0.5097     0.2875   1.773   0.0763 .
## x3C           1.2791     0.3255   3.930  8.5e-05 ***
## x3D           0.6168     0.3830   1.610   0.1073
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##        edf Ref.df Chi.sq  p-value
## s(x1) 1.98  2.466  54.97 1.08e-11 ***
## s(x2) 3.00  3.000  25.99 9.58e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0606   Deviance explained = 11.6%
## UBRE score = -0.57769  Scale est. = 1         n = 2000
```

```
plot(mod_mono_gam, pages=1, se=FALSE, main="Constrained GAM",
     shift=coef(mod_mono_gam)[1], all.terms=TRUE)
```




```
## helper function for test set performance

perf <- function(lst_preds, f_metric=caTools::colAUC, metricname="ROC-AUC"){
```

```
  map_dfr(lst_preds, function(x){
  f_metric(x, test_data$y)
}) %>%
  pivot_longer(everything(), names_to="model", values_to=metricname) %>%
  knitr::kable()
}
```

```
map_dfr(list(mod_gam=mod_gam, mod_mono_gam=mod_mono_gam), AIC) %>%
  pivot_longer(everything(), names_to="model", values_to="AIC") %>%
  knitr::kable()
```

model	AIC
mod_gam	841.5774
mod_mono_gam	844.6143

```
## test set performance

preds_gam <- predict(mod_gam, newdata=test_data, type="response")
preds_mono_gam <- predict(mod_mono_gam, newdata=test_data,
                          type="response")

myPreds <- list(gam=preds_gam, mono_gam=preds_mono_gam)

perf(myPreds, caTools::colAUC, "ROC-AUC")
```

model	ROC-AUC
gam	0.7122259
mono_gam	0.7115072

```
perf(myPreds, MLmetrics::LogLoss, "LogLoss")
```

model	LogLoss
gam	0.2180729
mono_gam	0.2180345

Generate Pseudo-Training Data

The next step is controversial. I pretend that the scam model is the data generating process. In the actual training data, we observe binary outcomes rather than the log odds. In the pseudo data, I ignore the binary outcomes and "observe" the log odds from the scam model.

The reader should be aware that in the actual training data, `y` is a binary response. In the pseudo data, `y` is log odds from the scam model, which is analogous to `z` in the true data generating process.

```
## generate pseudo-data using Mono GAM

## suppose the scam model is the data generating process (dgp).
## append the log odds for each observation in the training
## data

pseudo_train_data <- train_data

pseudo_train_data$y <- as.numeric(predict(mod_mono_gam, newdata=train_data))

summary(pseudo_train_data)
```

```
##        y                 x1                 x2           x3
##  Min.   :-6.6438   Min.   :-3.31420   Min.   :-3.53202   A:1609
##  1st Qu.:-3.9143   1st Qu.:-0.69239   1st Qu.:-0.64836   B: 198
##  Median :-3.1377   Median : 0.02514   Median :-0.01473   C:  98
##  Mean   :-3.1751   Mean   : 0.01329   Mean   : 0.01956   D:  95
##  3rd Qu.:-2.3674   3rd Qu.: 0.67314   3rd Qu.: 0.70888
##  Max.   :-0.2218   Max.   : 3.78302   Max.   : 3.65809
```

Approximate SCAM with MARS

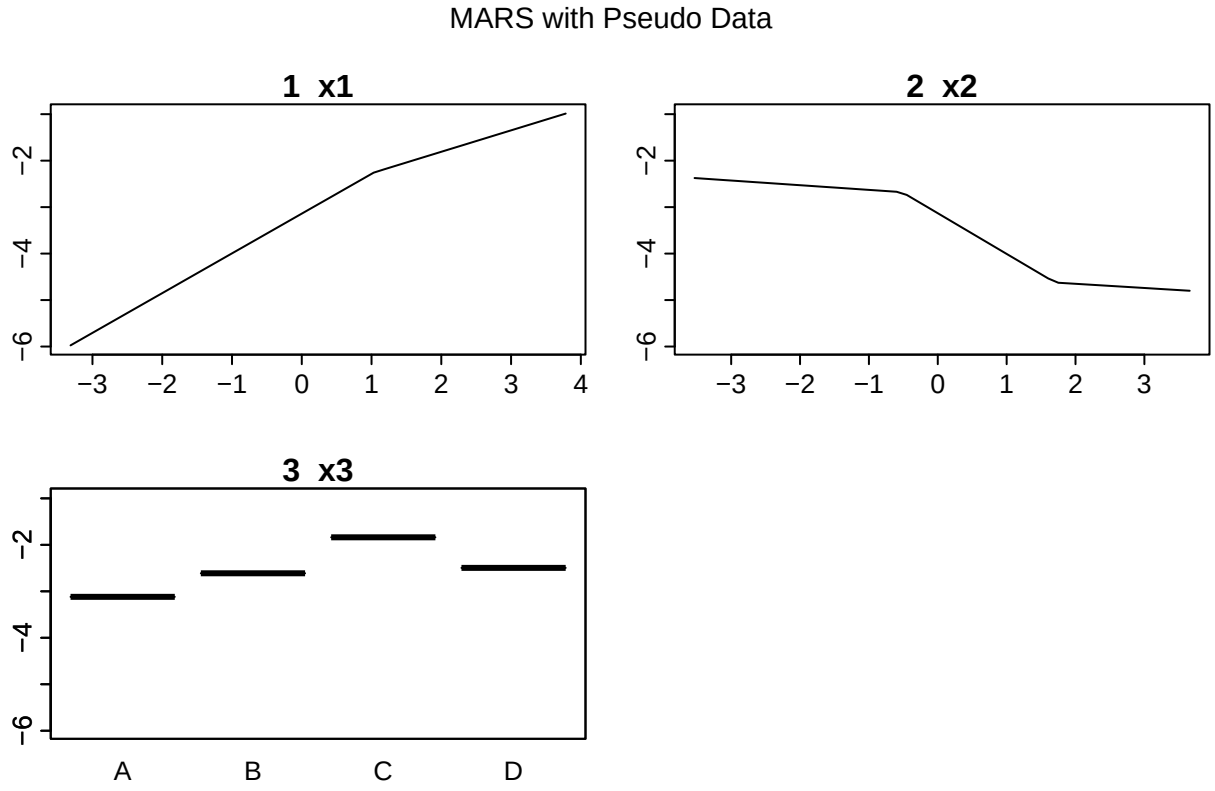
```
## approximate the Mono GAM model with MARS

## unfortunately, both gam and scam models are very difficult to
## implement outside of R due to how the basis functions are defined.
## an alternative is to approximate the scam relationships
## with simpler basis functions like the linear basis (which can be
## easily implemented in Excel) -- the linear basis is also called the
## reLU (rectified linear unit)


mod_logit_pseudo <- earth(y ~ x1 + x2 + x3, data=pseudo_train_data)

summary(mod_logit_pseudo)
```

```
## Call: earth(formula=y~x1+x2+x3, data=pseudo_train_data)
##
##                   coefficients
## (Intercept)         -1.8121035
## x3B                  0.5063307
## x3C                  1.2805724
## x3D                  0.6230057
## h(1.03699-x1)       -0.8552545
## h(x1-1.03699)        0.4611403
## h(-0.518908-x2)      0.1005579
## h(x2- -0.518908)    -0.8751410
## h(x2-1.70494)        0.7850924
##
## Selected 9 of 9 terms, and 5 of 5 predictors
```

```
## Termination condition: RSq changed by less than 0.001 at 9 terms
## Importance: x1, x2, x3C, x3B, x3D
## Number of terms at each degree of interaction: 1 8 (additive model)
## GCV 0.003026326    RSS 5.950193    GRSq 0.9974911    RSq 0.9975311
```

```
plotmo(mod_logit_pseudo, caption="MARS with Pseudo Data")
```

```
##  plotmo grid:    x1          x2 x3
##           0.02514023 -0.01473112  A
```



Since the pseudo model is in the log-odds scale, I need to convert the predictions into probabilities to compare against the probability predictions from gam and scam.

```
## test set performance

preds_pseudo <- boot::inv.logit(predict(mod_logit_pseudo, newdata=test_data))

myPreds <- list(gam=preds_gam, mono_gam=preds_mono_gam, pseudo_mono=preds_pseudo)

perf(myPreds, caTools::colAUC, "ROC-AUC")
```

model	ROC-AUC
gam	0.7122259

model	ROC-AUC
mono_gam	0.7115072
pseudo_mono	0.7129432

```
perf(myPreds, MLmetrics::LogLoss, "LogLoss")
```

model	LogLoss
gam	0.2180729
mono_gam	0.2180345
pseudo_mono	0.2177323

Excel-friendly Equation

The equation below predicts the log odds or logit.

```
cat(format(mod_logit_pseudo, style="pmax", use.names=TRUE))
```

```
##   -1.812103
##   + 0.5063307 * x3B
##   +  1.280572 * x3C
##   + 0.6230057 * x3D
##   - 0.8552545 * pmax(0,   1.036994 -         x1)
##   + 0.4611403 * pmax(0,         x1 -   1.036994)
##   + 0.1005579 * pmax(0, -0.5189081 -         x2)
##   -  0.875141 * pmax(0,         x2 - -0.5189081)
##   + 0.7850924 * pmax(0,         x2 -   1.704942)
```

```
## Compare predict() against score_function

as.func <- function(object, digits = 20, use.names = TRUE, ...)
  eval(parse(text=paste(
    "function(x){\n",
    "if(is.vector(x))\n",
    "  x <- matrix(x, nrow = 1, ncol = length(x))\n",
    "x <- model.matrix(delete.response(object$terms),x) \n",
    "with(as.data.frame(x),\n",
    format(object, digits = digits, use.names = use.names, style = "pmax", ...),
    ")\n",
    "}\n", sep = "")))

score_function <- as.func(mod_logit_pseudo)

compare_df <- expand.grid(x1 = seq(-10, 10, 0.1),
           x2 = seq(-10, 10, 0.1),
           x3 = factor(c('A', 'B', 'C', 'D')))

earth_preds <- predict(mod_logit_pseudo, newdata=compare_df)
```

```
score_preds <- score_function(compare_df)

max(abs(earth_preds - score_preds))
```

```
## [1] 3.552714e-15
```

Testing monotonicity

MARS does not guarantee monotonic relationships. Reader should always test the model for violations of monotonicity by feeding extreme predictor values into the model.

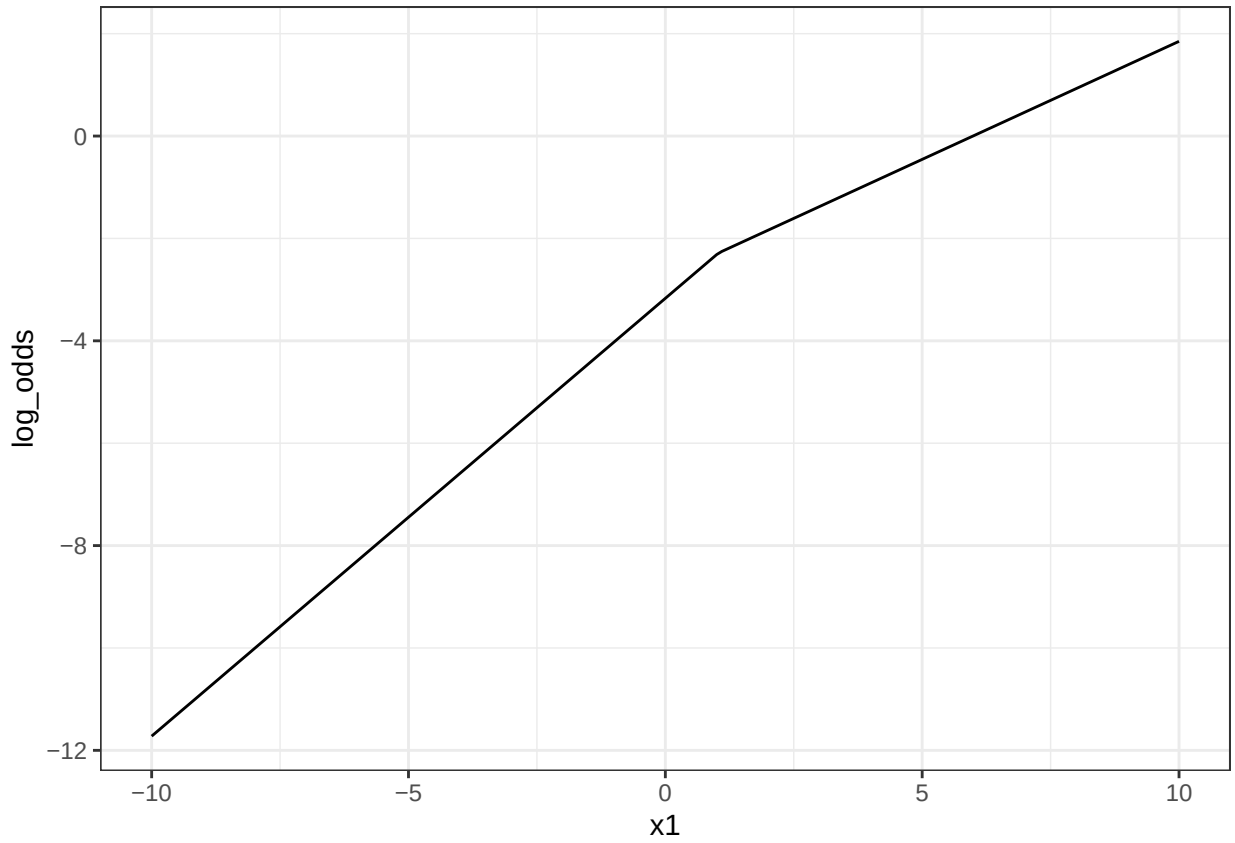
```
mono_check <- function(x, increasing=TRUE){
  if(increasing==TRUE){
    out <- all(x == cummax(x))
  } else {
    out <- all(x==cummin(x))
  }

  return(out)
}

df_x1 <- data.frame(x1 = seq(-10,10, 0.1),
                    x2 = mean(train_data$x2),
                    x3 = factor('A', levels=c('A','B','C','D'))
                    )

df_x2 <- data.frame(x1 = mean(train_data$x1),
                    x2 = seq(-10,10, 0.1),
                    x3 = factor('A', levels=c('A','B','C','D'))
                    )

df_x1$log_odds <- score_function(df_x1)

df_x2$log_odds <- score_function(df_x2)

mono_check(df_x1$log_odds)
```

```
## [1] TRUE
```

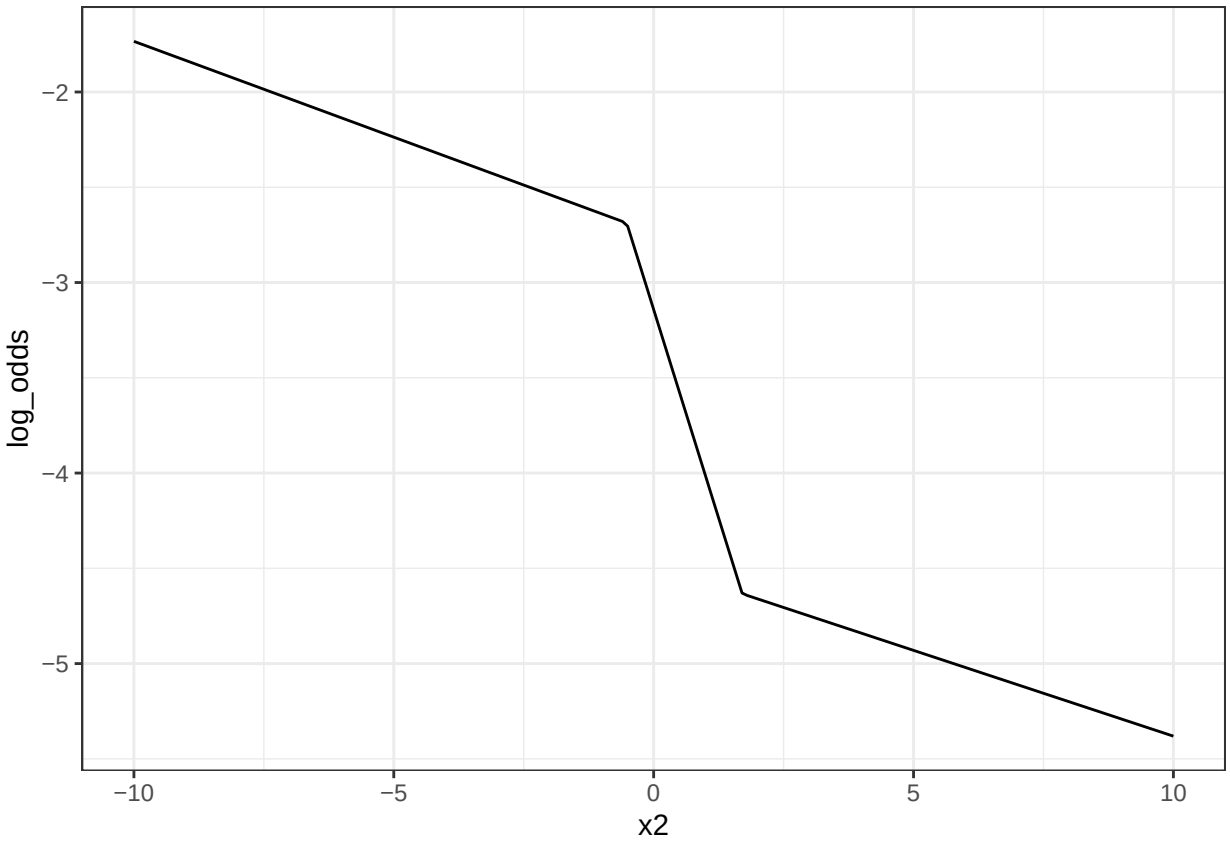
```
mono_check(df_x2$log_odds, FALSE)
```

```
## [1] TRUE
```

```
ggplot(df_x1, aes(x=x1, y=log_odds)) + geom_line() + theme_bw()
```




```
ggplot(df_x2, aes(x=x2, y=log_odds)) + geom_line() + theme_bw()
```



How often does monotonicity fail?

Since MARS does not guarantee monotonicity. I repeated, 60 times, the following steps:

1. Set a new seed value
2. Generate full, train, and test sets
3. Fit a scam model to the training data
4. Generate pseudo data
5. Fit a MARS model to the pseudo data
6. Check shape constraints (x1 should increase with y and x2 should decrease with y)

```
simulate_one <- function(myseed){

  # Set a new seed value

  set.seed(myseed)

  nobs <- 100000

  x1 <- rnorm(nobs)
  x2 <- rnorm(nobs)
  x3 <- sample(c('A', 'B', 'C', 'D'), size=nobs, replace=TRUE,
               prob = c(0.8, 0.1, 0.05, 0.05))
```

```
  # Population log odds is a function of sigmoid features

  z <- -35 + 5 * SSfpl(x1, -2, 2, 0, 0.5) - 5 * SSfpl(x2, -1, 1, 0, 0.1) +
          + 5 * I(x3=='B') + 10 * I(x3=='C') + 10 * I(x3=='D') +
    rnorm(nobs,0,20)

  y <- rbinom(nobs, 1, prob=boot::inv.logit(z))

  # Generate full, train, and test sets
  full_data <- data.frame(y=y, x1=x1, x2=x2, x3=as.factor(x3))

  train_index <- sample.split(full_data$y, SplitRatio = 2/100)
  train_data <- full_data[train_index,] # small
  test_data <- full_data[!train_index,] # large

  # SCAM

  my_mod_mono_gam <- scam(y ~ s(x1, bs="mpi") + s(x2, bs="mpd") + x3,
                       data = train_data, family = binomial)

  # Pseudo data

  pseudo_train_data <- train_data

  pseudo_train_data$y <- as.numeric(predict(my_mod_mono_gam, newdata=train_data))

  summary(pseudo_train_data)

  # MARS

  my_mod_pseudo_mars <- earth(y ~ x1 + x2 + x3, data=pseudo_train_data)


  # Check shape constraints

  df_x1 <- data.frame(x1 = seq(-10,10, 0.1),
                      x2 = mean(train_data$x2),
                      x3 = factor('A', levels=c('A','B','C','D'))
                      )

  df_x2 <- data.frame(x1 = mean(train_data$x1),
                      x2 = seq(-10,10, 0.1),
                      x3 = factor('A', levels=c('A','B','C','D'))
                      )

  df_x1$log_odds <- predict(my_mod_pseudo_mars, newdata=df_x1)

  df_x2$log_odds <- predict(my_mod_pseudo_mars, newdata=df_x2)

  success <- mono_check(df_x1$log_odds) & mono_check(df_x2$log_odds, FALSE)

  return(success)
}
```

```
mono_success <- future_map_int(61:120, simulate_one,
                          .options=furrr_options(seed=NULL,
                                                 packages=c('earth', 'scam')
                                                 )
                          )

future:::ClusterRegistry("stop")


mean(mono_success)
```

```
## [1] 0.7333333
```

MARS successfully approximated the scam model about 73.3333333% of the time.

Why Pseudo Data?

The pseudo data filters out the noise in the latent response (`z`), which drives the binary response (`y`). Although we could directly pass the actual training data into MARS, the noise in the data generating process could cause `earth` to return non-monotonic relationships.

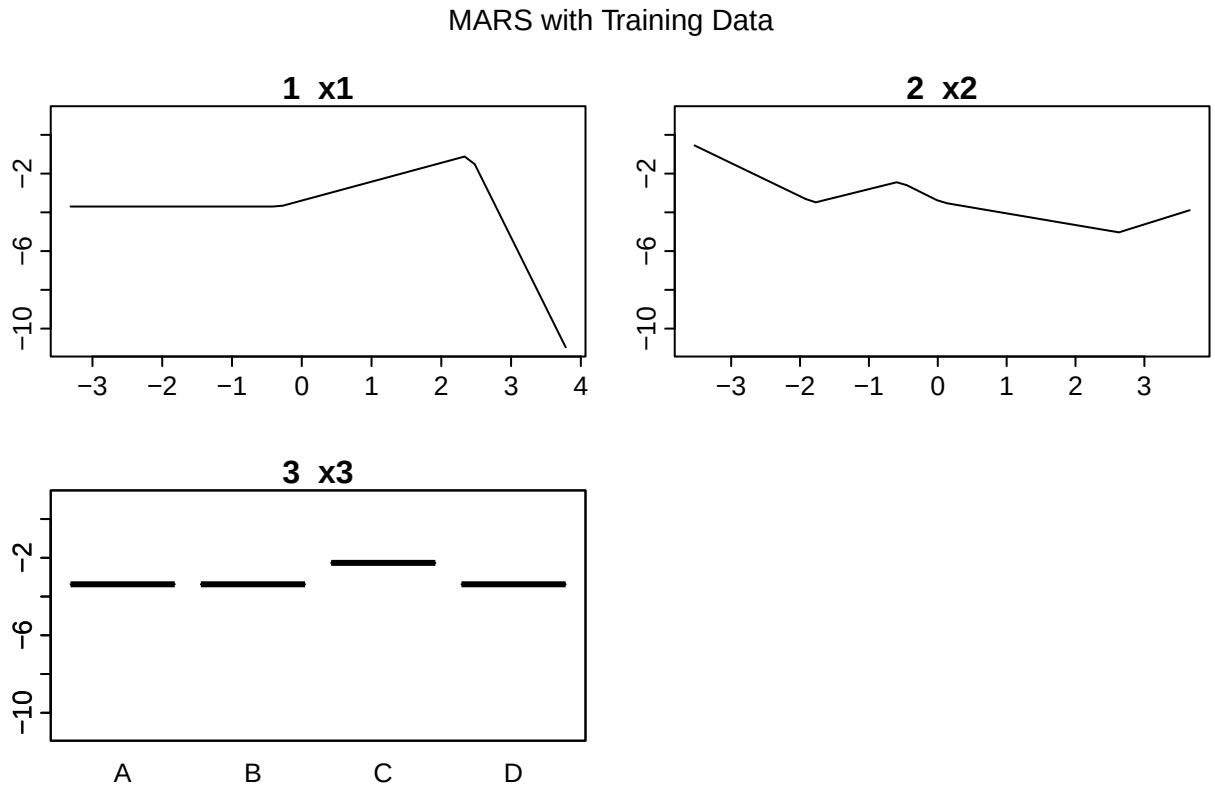
Reader should notice that `y` in the actual training data is a binary outcome.

```
mod_earth <- earth(y ~ x1 + x2 + x3, data=train_data, glm=list(family=binomial))

summary(mod_earth)
```

```
## Call: earth(formula=y~x1+x2+x3, data=train_data, glm=list(family=binomial))
##
## GLM coefficients
##                               y
## (Intercept)      -11.4359941
## x3C                1.1057666
## h(x1- -0.315568)   0.9740516
## h(x1-2.41403)     -8.2227755
## h(x2- -1.80226)    2.5956455
## h(x2- -0.555908)  -2.6511588
## h(x2-0.0490364)    1.1672334
## h(2.6295-x2)       1.7135309
##
## GLM (family binomial, link logit):
##  nulldev   df       dev   df   devratio     AIC iters converged
##  935.167 1999   825.049 1992      0.118     841     6         1
##
## Earth selected 8 of 12 terms, and 3 of 5 predictors
## Termination condition: RSq changed by less than 0.001 at 12 terms
## Importance: x1, x2, x3C, x3B-unused, x3D-unused
## Number of terms at each degree of interaction: 1 7 (additive model)
## Earth GCV 0.05595653    RSS 110.2407    GRSq 0.04596337    RSq 0.05927977
```

```
plotmo(mod_earth, type="link", caption="MARS with Training Data")
```

```
##  plotmo grid:    x1          x2 x3
##          0.02514023 -0.01473112  A
```



Alternative to SCAM: SCAR

Models developed in `scam` return smooth relationships between a response and each predictor. However, the smoothness (and monotonicity) is achieved through the p-spline basis, which is difficult to implement. If smoothness is not required, an alternative approach is to use monotonic step functions to approximate the relationships. The `scar` package achieves this goal.

Unfortunately, the `scar` package does not take data frames. I wrote two helper functions to convert a data frame into a matrix, which is then passed to the scar function.

```
fit_scar <- function(formula, shape=rep("l", d), data, family=gaussian(),
                     weights=rep(1, length(y)), epsilon = 1e-08){

  x <- model.matrix(formula, data)[,-1]

  y <- data[,all.vars(formula)[1]]

  mod <- scar(x, y, shape, family, weights, epsilon)
```

```
  mod$formula <- formula

  return(mod)
}

predict_scar <- function(object, newdata, type = c("link", "response"),
                         rule=1, ...){

  fmla <- delete.response(terms(object$formula))

  # pred_names <- all.vars(fmla)[-1]

  x <- model.matrix(fmla, newdata)[,-1]

  preds <- predict(object, x, type, rule, ...)

  return(preds)
}
```

Now we can fit a scar model with monotonic binning.

```
mod_scar <- fit_scar(y ~ x1 + x2 + x3, shape=c("in", "de", "l", "l", "l"),
                     data=train_data, family=binomial())
```

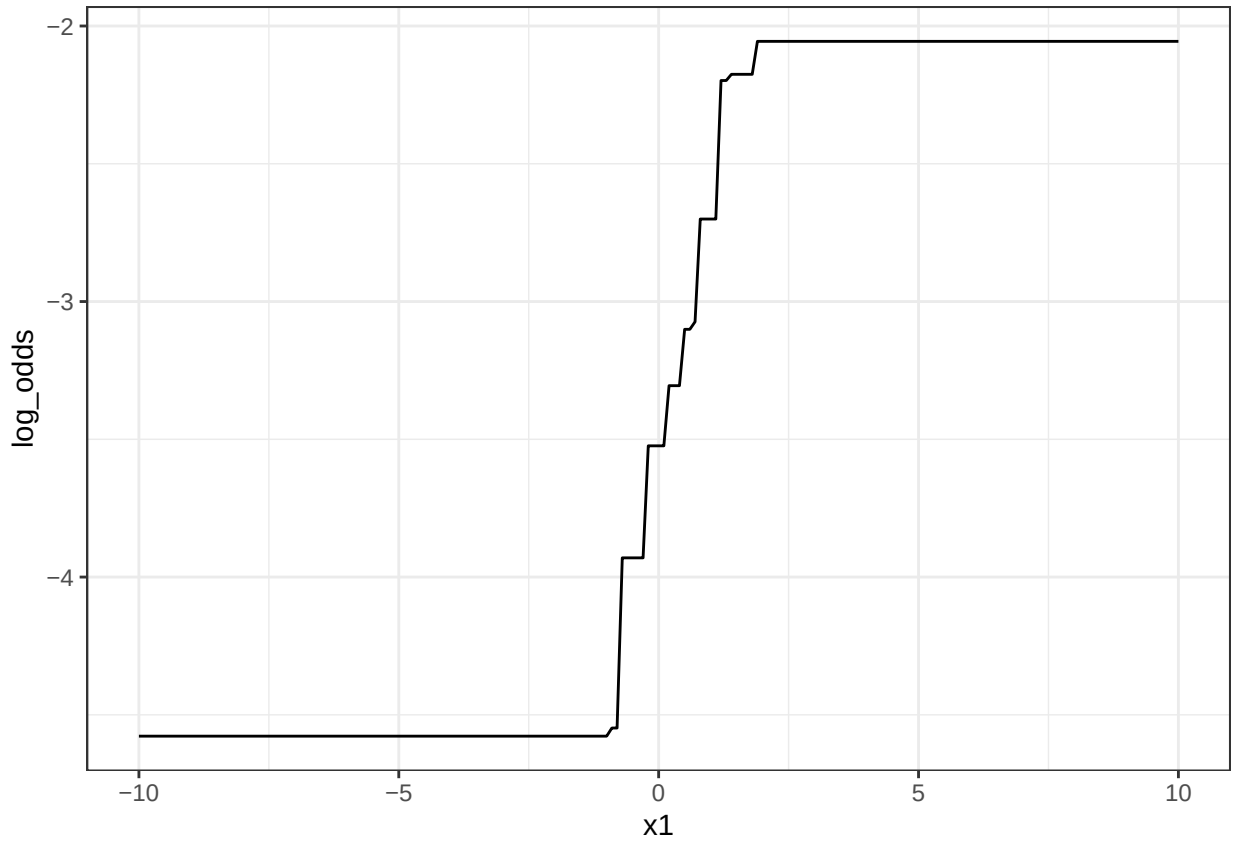
Inside `mod_scar` is an object called `componentfit`. It contains partial fitted values by predictor (column) and observation (row). Each observation in the training set corresponds to a row in `componentfit`. The fitted value for a single observation is the row sum of the partial fitted values.

Given a pair of new predictor values (that are not in the training data), the `predict` function interpolates the partial fitted values from `componentfit`. This may be problematic if `componentfit` is very big. A similar problem arises with generating predictions from k-nearest-neighbors (knn).

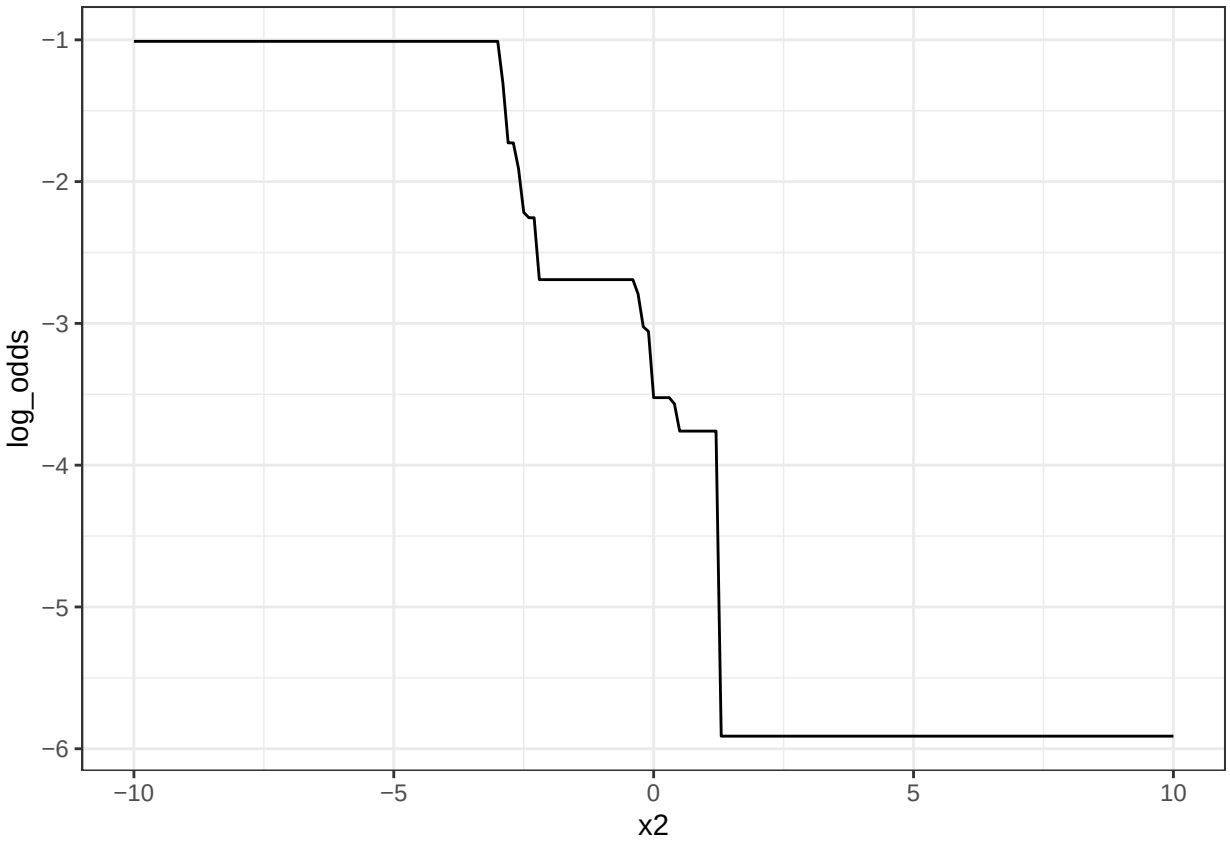
The interpolation of partial fitted values causes the relationship between a response and a predictor to appear "step-like".

Check the partial plots.

```
df_x1$log_odds <- predict_scar(mod_scar, newdata=df_x1)

df_x2$log_odds <- predict_scar(mod_scar, newdata=df_x2)

ggplot(df_x1, aes(x=x1, y=log_odds)) + geom_line() + theme_bw()
```




```
ggplot(df_x2, aes(x=x2, y=log_odds)) + geom_line() + theme_bw()
```



Check test set performance

```
preds_scar <- predict_scar(mod_scar, newdata=test_data, type="response")

myPreds <- list(gam=preds_gam, mono_gam=preds_mono_gam, pseudo_mono=preds_pseudo,
             scar=preds_scar)

perf(myPreds, caTools::colAUC, "ROC-AUC")
```

model	ROC-AUC
gam	0.7122259
mono_gam	0.7115072
pseudo_mono	0.7129432
scar	0.7032823

```
perf(myPreds, MLmetrics::LogLoss, "Log Loss")
```

model	Log Loss
gam	0.2180729

model	Log Loss
mono_gam	0.2180345
pseudo_mono	0.2177323
scar	0.2226342

Approximating SCAR

Similar to the MARS approximation of scam, we could approximate scar with another method (with the help of pseudo data). The pseudo response would be the log-odds from the scar model. The features would be sigmoid functions that are fed into non-linear least squares. Sigmoid functions guarantee monotonicity.

First, some pseudo data!

```
pseudo_train_data_nls <- train_data

pseudo_train_data_nls$y <- predict_scar(mod_scar, newdata=train_data
                                        , type="link")

summary(pseudo_train_data_nls)
```

```
##        y                  x1                 x2           x3
##  Min.   :-6.96499   Min.   :-3.31420   Min.   :-3.53202   A:1609
##  1st Qu.:-3.97509   1st Qu.:-0.69239   1st Qu.:-0.64836   B: 198
##  Median :-3.14439   Median : 0.02514   Median :-0.01473   C:  98
##  Mean   :-3.30188   Mean   : 0.01329   Mean   : 0.01956   D:  95
##  3rd Qu.:-2.39797   3rd Qu.: 0.67314   3rd Qu.: 0.70888
##  Max.   :-0.02539   Max.   : 3.78302   Max.   : 3.65809
```

Second, non-linear least squares and sigmoid features. The classical `nls` function is very sensitive to initial parameter guesses. I have never been successful in using it. I suggest using `nlsLM` from the `minpack.lm` package.

```
mod_pseudo_scar <- nlsLM(y ~ constant + a * SSfpl(x1, A1, B1, xmid1, scal1) +
                           b * SSfpl(x2, A2, B2, xmid2, scal2) +
                           c * I(x3=='B') +
                           d * I(x3=='C') +
                           e * I(x3=='D'),
                         data=pseudo_train_data_nls,
                         start=list(constant=0, a=1, b=1, c=1, d=1, e=1,
                                    A1=-1, B1=1, xmid1=0.5, scal1=0.5,
                                    A2=-1, B2=1, xmid2=0.5, scal2=0.5))
```

```
est <- coef(mod_pseudo_scar)

est_df <- data.frame(Estimate=est)

knitr::kable(est_df)
```

	Estimate
constant	-1.1143412
a	1.3179421
b	0.9694471
c	0.4846551
d	1.3170561
e	0.6501136
A1	-1.1071181
B1	1.1311339
xmid1	0.3076759
scal1	0.7142092
A2	-1.2953769
B2	-5.6418496
xmid2	1.1759558
scal2	0.5813312

We could also bootstrap 95% confidence intervals for each estimate.

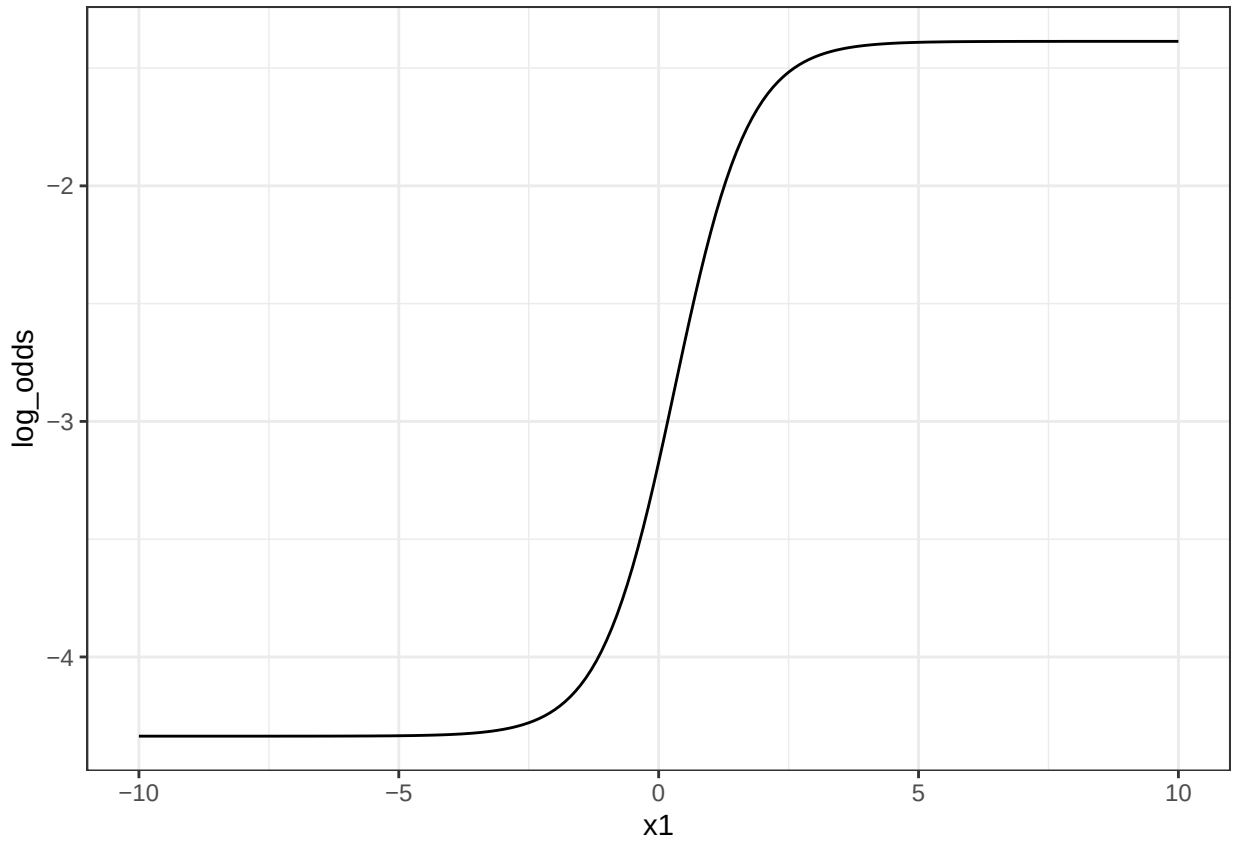
```
boot_psuedo <- Boot(mod_pseudo_scar, R=1000, ncores=6)
```

```
CI_nls <- data.frame(confint(boot_psuedo, type="perc"))

names(CI_nls) <- c("p2.5", "p97.5")

CI_nls$signif <- (CI_nls$p2.5 * CI_nls$p97.5) > 0

knitr::kable(CI_nls)
```

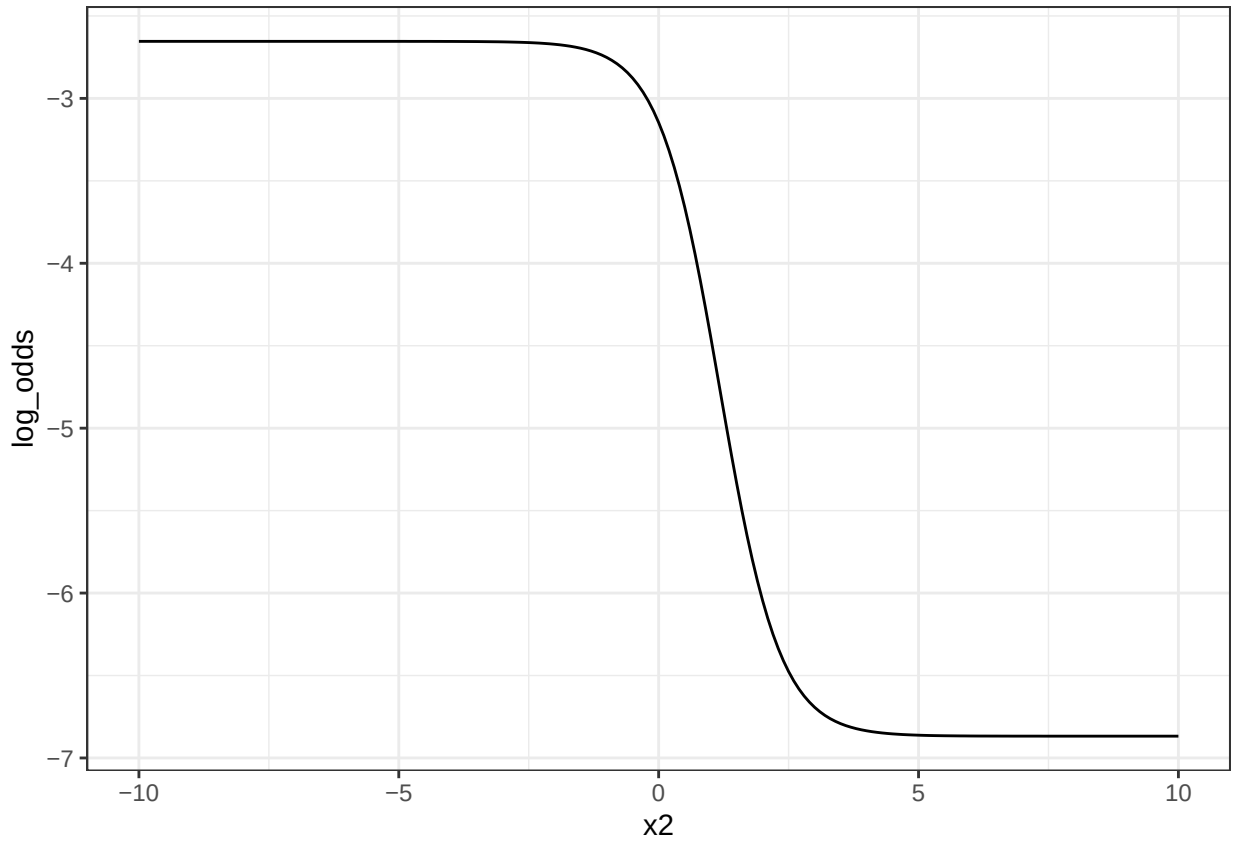
	p2.5	p97.5	signif
constant	-1.5185879	-1.0091381	TRUE
a	1.1410248	1.5684626	TRUE
b	0.9237093	1.1180000	TRUE
c	0.4273741	0.5429882	TRUE
d	1.2456665	1.3908390	TRUE
e	0.5908989	0.7134469	TRUE
A1	-1.3661630	-0.7545919	TRUE
B1	0.8543728	1.5443743	TRUE
xmid1	0.2458566	0.3724398	TRUE
scal1	0.6595468	0.7744543	TRUE
A2	-1.4793416	-0.7718849	TRUE
B2	-5.8894719	-4.8255832	TRUE
xmid2	1.1158966	1.2595466	TRUE
scal2	0.5386111	0.6301844	TRUE

Some plots.

```
df_x1$log_odds <- predict(mod_pseudo_scar, newdata=df_x1)

df_x2$log_odds <- predict(mod_pseudo_scar, newdata=df_x2)

ggplot(df_x1, aes(x=x1, y=log_odds)) + geom_line() + theme_bw()
```




```
ggplot(df_x2, aes(x=x2, y=log_odds)) + geom_line() + theme_bw()
```



Test set performance

```
preds_pseudo_scar <- boot::inv.logit(predict(mod_pseudo_scar, newdata=test_data))

myPreds <- list(gam=preds_gam, mono_gam=preds_mono_gam, pseudo_mono=preds_pseudo,
             scar=preds_scar, pseudo_scar=preds_pseudo_scar)

perf(myPreds, caTools::colAUC, "ROC-AUC")
```

model	ROC-AUC
gam	0.7122259
mono_gam	0.7115072
pseudo_mono	0.7129432
scar	0.7032823
pseudo_scar	0.7061607

```
perf(myPreds, MLmetrics::LogLoss, "Log Loss")
```

model	Log Loss
gam	0.2180729
mono_gam	0.2180345
pseudo_mono	0.2177323

model	Log Loss
scar	0.2226342
pseudo_scar	0.2210625

The approximation of scar actually outperforms the original scar model because the approximation extrapolates using the sigmoid functions that were estimated by non-linear least squares.

Sigmoid functions guarantee monotonicity.

CGAM

SCAM and SCAR do not constrain the relationships between the response and categorical predictors. In our simulated data set, the effects of `C` and `D` should be similar. However, in our training data set, this does not appear to be the case.

An alternative to the `scam` package is `cgam`. One major draw back of this package is that the `predict` function does not allow for extrapolation. Our test data set requires us to extrapolate, so we will not be able to measure the test set performance of the CGAM model. In this particular case, we have no choice but to fit a pseudo model.

First, we convert `x3` from factor to integer.

```
train_data <- train_data %>%
  mutate(x3_num = case_when(x3=='A' ~ 0,
                            x3=='B' ~ 1,
                            x3=='C' ~ 2,
                            TRUE ~ 3))
```

Second, fit a CGAM with a shape constraint on `x3_num`.

```
mod_cgam <- cgam(y ~ s.incr(x1) + s.decr(x2) + incr(x3_num),
                 data = train_data, family = "binomial")

summary(mod_cgam)
```

```
## Call:
## cgam(formula = y ~ s.incr(x1) + s.decr(x2) + incr(x3_num), family = "binomial",
##     data = train_data)
##
## Coefficients:
##             Estimate  StdErr z.value   p.value
## (Intercept)  -3.2362  0.1739 -18.611 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance:  935.1666  on 1999  degrees of freedom
## Residual deviance:  822.5902  on 1980.5  observed degrees of freedom
##
## Approximate significance of constrained components:
```

```
##              edf mixture.of.Beta p.value
## s.incr(x1)     9          0.0330  <2e-16 ***
## s.decr(x2)     6          0.0169  <2e-16 ***
## incr(x3_num)   3          0.0078   2e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## CIC:  0.4174
```

Third, generate pseudo data.

```
pseudo_train_data_cgam <- train_data

pseudo_train_data_cgam$y <- as.numeric(predict(mod_cgam, newData=train_data,
                                               interval="none",
                                      type="link")$object$etahat)

summary(pseudo_train_data_cgam)
```

```
##        y                 x1                 x2           x3
##  Min.   :-16.911   Min.   :-3.31420   Min.   :-3.53202   A:1609
##  1st Qu.: -3.907   1st Qu.:-0.69239   1st Qu.:-0.64836   B: 198
##  Median : -3.164   Median : 0.02514   Median :-0.01473   C:  98
##  Mean   : -3.236   Mean   : 0.01329   Mean   : 0.01956   D:  95
##  3rd Qu.: -2.412   3rd Qu.: 0.67314   3rd Qu.: 0.70888
##  Max.   : -0.401   Max.   : 3.78302   Max.   : 3.65809
##      x3_num
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.3395
##  3rd Qu.:0.0000
##  Max.   :3.0000
```

Next, fit a MARS model on the pseudo data. The threshold was set to 0.01 to get monotonicity. Increasing this threshold raises bias but lowers variance.

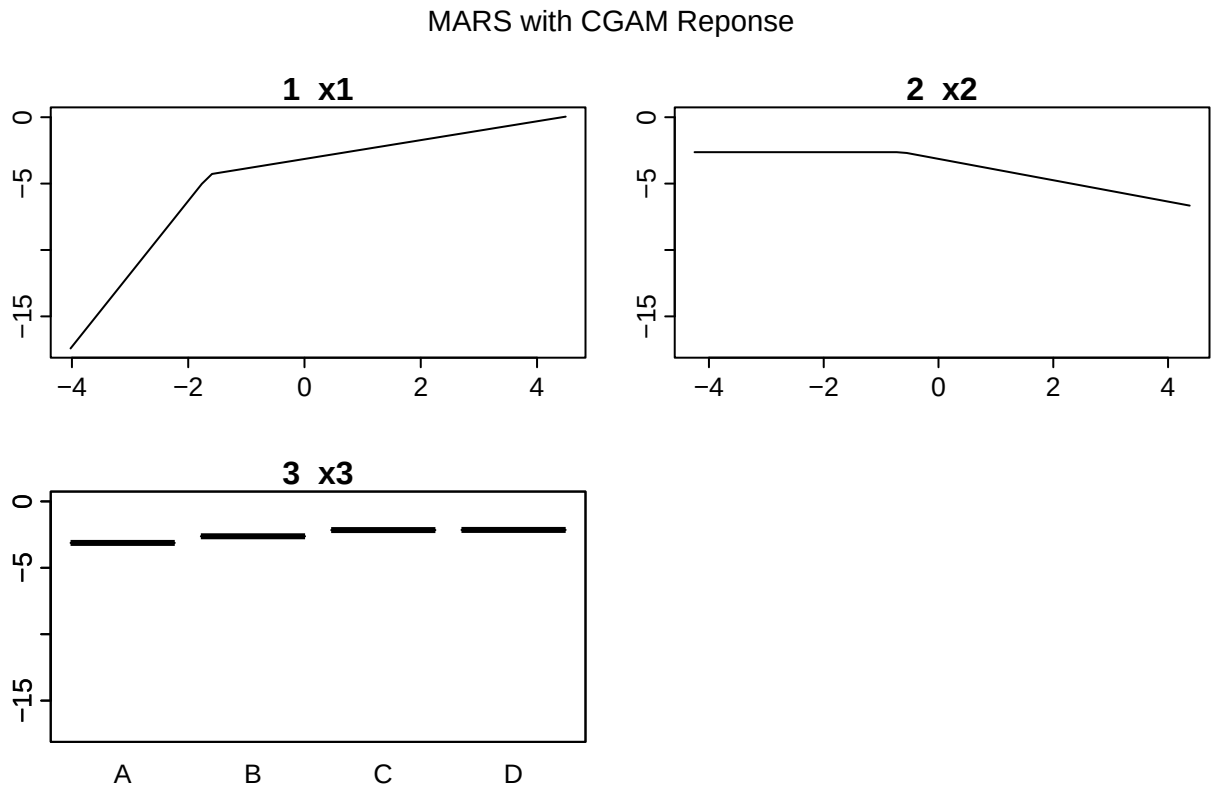
```
mod_pseudo_cgam <- earth(y ~ x1 + x2 + x3, data=pseudo_train_data_cgam,
                         thresh=0.01)

summary(mod_pseudo_cgam)
```

```
## Call: earth(formula=y~x1+x2+x3, data=pseudo_train_data_cgam, thresh=0.01)
##
##                   coefficients
## (Intercept)         -3.8123682
## x3B                  0.4997210
## x3C                  0.9684266
## x3D                  0.9780145
## h(-1.63511-x1)      -5.4866792
## h(x1- -1.63511)      0.7094716
## h(x2- -0.628166)    -0.8035952
##
```

```
## Selected 7 of 8 terms, and 5 of 5 predictors
## Termination condition: RSq changed by less than 0.01 at 8 terms
## Importance: x1, x2, x3C, x3D, x3B
## Number of terms at each degree of interaction: 1 6 (additive model)
## GCV 0.0552045    RSS 108.9783    GRSq 0.9676711    RSq 0.9680581
```

```
plotmo(mod_pseudo_cgam, type="link", caption="MARS with CGAM Reponse",
       extend=0.1)
```

```
##  plotmo grid:    x1          x2 x3
##           0.02514023 -0.01473112  A
```



Check monotonicity.

```
df_x1$log_odds <- predict(mod_pseudo_cgam, newdata=df_x1)

df_x2$log_odds <- predict(mod_pseudo_cgam, newdata=df_x2)

success <- mono_check(df_x1$log_odds) & mono_check(df_x2$log_odds, FALSE)

success
```

```
## [1] TRUE
```

Finally, check test set performance.

```
preds_pseudo_cgam <- boot::inv.logit(predict(mod_pseudo_cgam, newdata=test_data))

myPreds <- list(gam=preds_gam, mono_gam=preds_mono_gam, pseudo_mono=preds_pseudo,
             scar=preds_scar, pseudo_scar=preds_pseudo_scar, pseudo_cgam=preds_pseudo_cgam)

perf(myPreds, caTools::colAUC, "ROC-AUC")
```

model	ROC-AUC
gam	0.7122259
mono_gam	0.7115072
pseudo_mono	0.7129432
scar	0.7032823
pseudo_scar	0.7061607
pseudo_cgam	0.7117960

```
perf(myPreds, MLmetrics::LogLoss, "Log Loss")
```

model	Log Loss
gam	0.2180729
mono_gam	0.2180345
pseudo_mono	0.2177323
scar	0.2226342
pseudo_scar	0.2210625
pseudo_cgam	0.2195729

Conclusion

Imposing shape constraints on data with high noise to signal ratios could greatly reduce variance (wiggliness). The `scam` package is highly effective in developing models with user-defined monotonicity constraints for each predictor. MARS can approximate the scam model with linear basis functions, which are easy to write down on paper and implement in an Excel formula.

The `scar` package also develops models with user-defined monotonicity constraints for each predictor. Non-linear least squares can approximate the scar model with sigmoid functions, which are easy to write down on paper (`?SSfpl` for equation) and implement in an Excel formula.