GAMs and Boosted Stumps

DS 6030 | Fall 2024

GAM-stumps.pdf

Contents

1 Generalized Additive Models (GAM)

In our discussion of Basis Expansions, we covered how the relationship between a single raw predictor x and the outcome could be made more complex with basis expansions.

```
Default = ISLR::Default %>% as_tibble() %>%
  mutate(y = ifelse(default == "Yes", 1L, 0L))
Default
#> # A tibble: 10,000 x 5
#>   default student balance income     y
#>   <fct>   <fct>     <dbl>  <dbl> <int>
#> 1 No      No         730. 44362.     0
#> 2 No      Yes        817. 12106.     0
#> 3 No      No        1074. 31767.     0
#> 4 No      No         529. 35704.     0
#> 5 No      No         786. 38463.     0
#> 6 No      Yes        920.  7492.     0
#> # i 9,994 more rows
p0 = mean(Default$y)
gamma0 = log(p0) - log(1-p0)
```

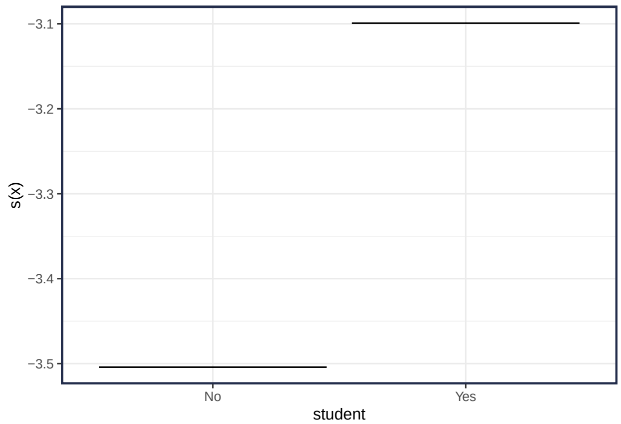
```
library(tidymodels)
library(broom) # for tidy() function
```

```
base_plot_income =
  Default %>%
  ggplot(aes(income)) +
  geom_rug(data=. %>% filter(y==1),
           aes(color=default), sides="t", color = "orange") +
  geom_rug(data=. %>% filter(y==0),
           aes(color=default), sides="b", color = "blue") +
  scale_color_manual(values=c(mgcv = "purple", `B-spline`="red" ,
                              Binning="green", Polynomial="blue",
                              Linear="black"),
                     name="model")
```

- **Example 1: Categorical Predictor One-Hot Encoded**

```
#: One-hot Basis
X1 = model.matrix(~student-1, data=Default)
head(X1, 4)
#>   studentNo studentYes
#> 1         1          0
#> 2         0          1
#> 3         1          0
#> 4         1          0

#: Fit
fit_1 = glm(y ~ student-1, family="binomial", data=Default)
tidy(fit_1)
#> # A tibble: 2 x 5
#>   term       estimate std.error statistic   p.value
#>   <chr>         <dbl>     <dbl>     <dbl>     <dbl>
#> 1 studentNo     -3.50    0.0707     -49.6 0
#> 2 studentYes    -3.10    0.0907     -34.2 8.00e-256
```

```r
#: Plot
Default %>%
  mutate(pred = predict(fit_1, newdata=Default, type="link")) %>%
  distinct(student, pred) %>%
  ggplot(aes(student, pred)) + geom_errorbar(aes(ymin=pred, ymax=pred)) +
  labs(y="s(x)")
```

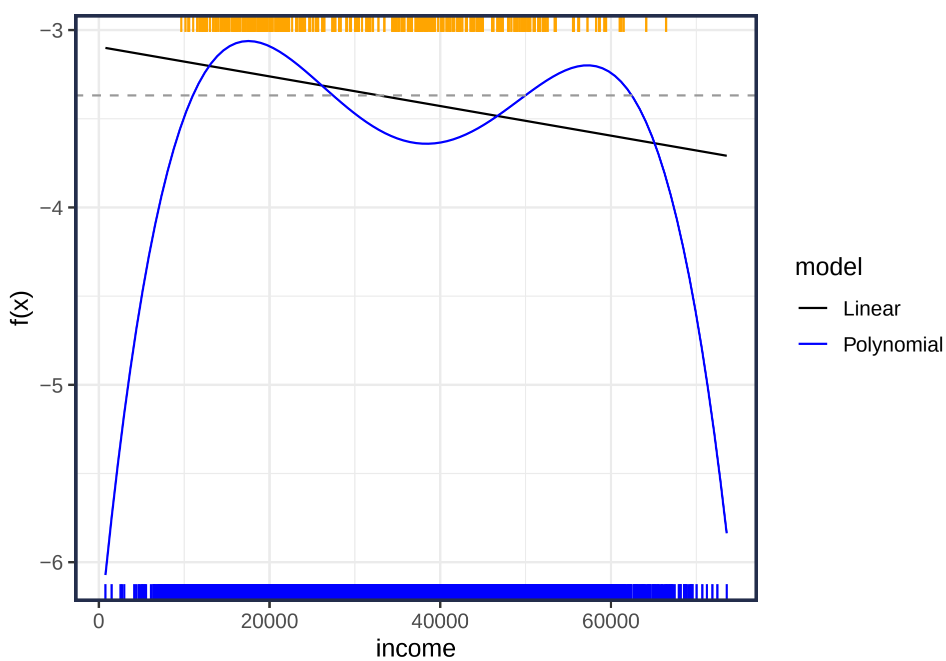


- **Example 2: Continuous Predictor with Polynomial Basis**

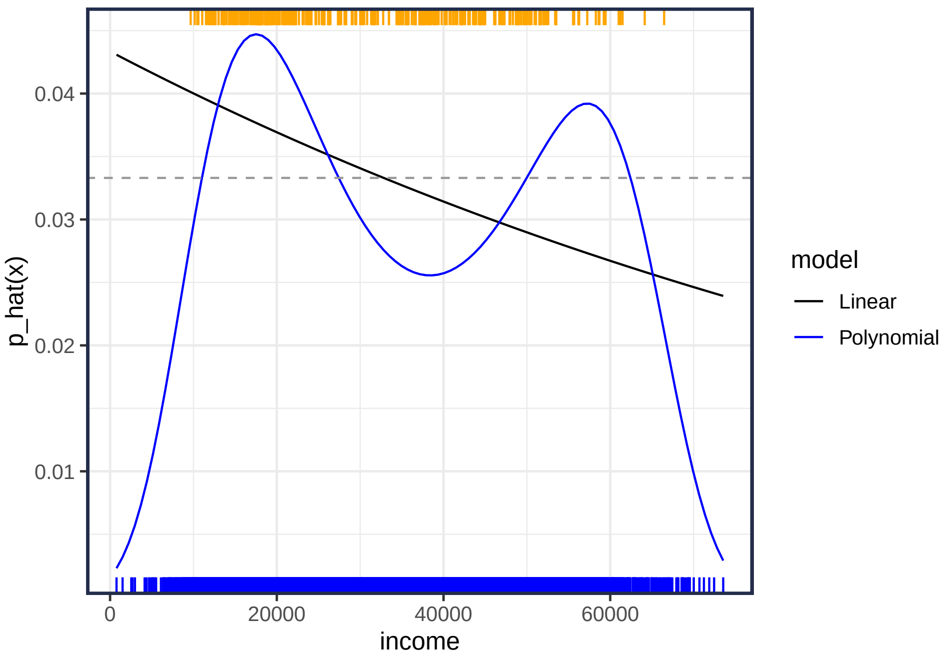
```r
#: Fit linear
fit_lm = glm(y~income, data=Default, family="binomial")
tidy(fit_lm)
#> # A tibble: 2 x 5
#>   term           estimate  std.error statistic  p.value
#>   <chr>             <dbl>      <dbl>     <dbl>    <dbl>
#> 1 (Intercept) -3.09       0.146          -21.2  2.45e-99
#> 2 income      -0.00000835 0.00000421      -1.99 4.71e- 2

#: Polynomial Basis
X2 = model.matrix(y~poly(income, degree=4)-1, data=Default)
head(X2, 4)
#>   poly(income, degree = 4)1 poly(income, degree = 4)2 poly(income, degree = 4)3
#> 1                  0.008132                 -0.003807                -0.0080610
#> 2                 -0.016055                  0.016202                -0.0138049
#> 3                 -0.001312                 -0.009300                 0.0057123
#> 4                  0.001640                 -0.009414                 0.0009502
#>   poly(income, degree = 4)4
#> 1                 0.0006076
#> 2                 0.0052431
#> 3                 0.0063483
#> 4                 0.0083087

#: Polynomial Model (edf=5)
fit_2 = glm(y~poly(income, degree=4), family="binomial", data=Default)
tidy(fit_2)
#> # A tibble: 5 x 5
#>   term                        estimate std.error statistic p.value
#>   <chr>                          <dbl>     <dbl>     <dbl>   <dbl>
#> 1 (Intercept)                    -3.39    0.0570    -59.5   0
#> 2 poly(income, degree = 4)1     -10.4     5.42      -1.93   0.0541
#> 3 poly(income, degree = 4)2       7.47    6.05       1.23   0.217
```

```
#> 4 poly(income, degree = 4)3      4.94     6.38        0.775   0.438
#> 5 poly(income, degree = 4)4    -17.9      7.00       -2.55    0.0106

# equivalent to: glm(y~X2, family="binomial", data=Default)
```

```
base_plot_income +
  geom_function(fun = ~predict(fit_lm, newdata=tibble(income=.)),
                aes(color="Linear")) +
  geom_function(fun = ~predict(fit_2, newdata=tibble(income=.)),
                aes(color="Polynomial")) +
  geom_hline(yintercept = gamma0, linetype = "dashed", color = "grey60") +
  labs(y = "f(x)")
```



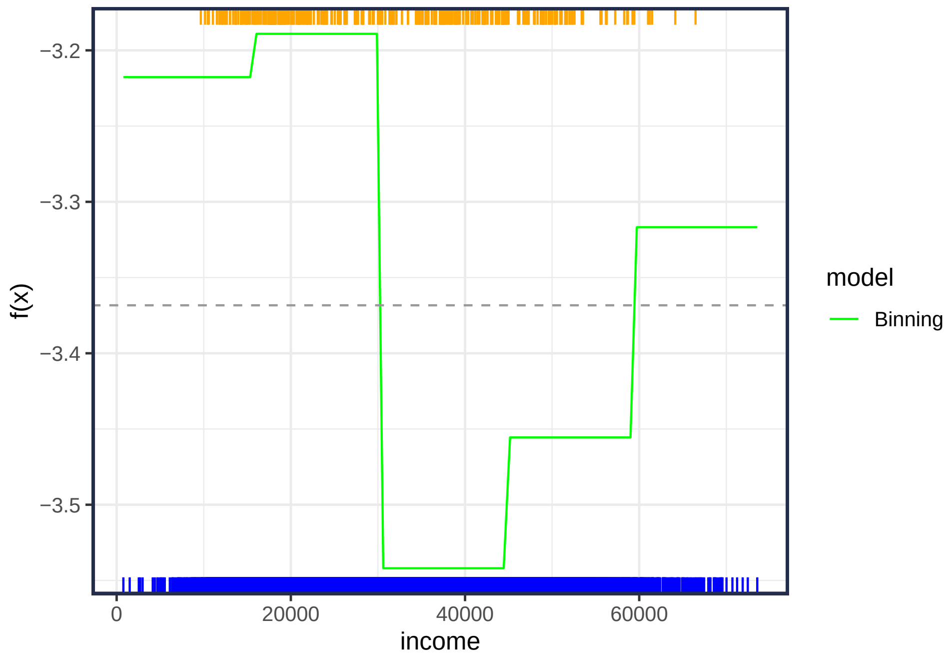

```
base_plot_income +
  geom_function(fun = ~predict(fit_lm, newdata=tibble(income=.),
                              type = "response"),
                aes(color="Linear")) +
  geom_function(fun = ~predict(fit_2, newdata=tibble(income=.),
                              type = "response"),
                aes(color="Polynomial")) +
  geom_hline(yintercept = p0, linetype = "dashed", color = "grey60") +
  labs(y = "p_hat(x)")
```



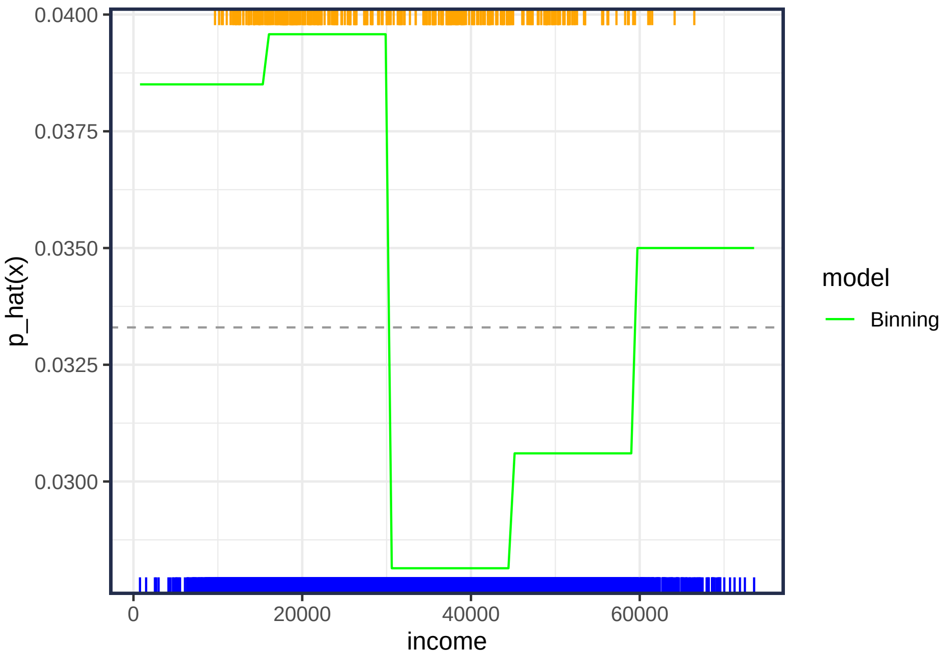
- **Example 3: Continuous Predictor with Binning (Regressograms)**

```
#: Binning Basis
X3 = model.matrix(~cut(income, 5)-1, data=Default)
head(X3, 4)
#>   cut(income, 5)(699,1.53e+04] cut(income, 5)(1.53e+04,2.99e+04]
#> 1                            0                                 0
#> 2                            1                                 0
#> 3                            0                                 0
#> 4                            0                                 0
#>   cut(income, 5)(2.99e+04,4.44e+04] cut(income, 5)(4.44e+04,5.9e+04]
#> 1                                 1                                0
#> 2                                 0                                0
#> 3                                 1                                0
#> 4                                 1                                0
#>   cut(income, 5)(5.9e+04,7.36e+04]
#> 1                                0
#> 2                                0
#> 3                                0
#> 4                                0

#: Binning Model (edf=5)
fit_3 = glm(y~cut(income, 5)-1, data=Default, family="binomial")
tidy(fit_3)
#> # A tibble: 5 x 5
#>   term                              estimate std.error statistic   p.value
#>   <chr>                                <dbl>     <dbl>     <dbl>     <dbl>
#> 1 cut(income, 5)(699,1.53e+04]         -3.22    0.175     -18.4  1.38e- 75
#> 2 cut(income, 5)(1.53e+04,2.99e+04]    -3.19    0.0916    -34.8  2.25e-265
#> 3 cut(income, 5)(2.99e+04,4.44e+04]    -3.54    0.0999    -35.4  4.52e-275
#> 4 cut(income, 5)(4.44e+04,5.9e+04]     -3.46    0.126     -27.4  1.19e-165
#> 5 cut(income, 5)(5.9e+04,7.36e+04]     -3.32    0.385      -8.62 6.67e- 18
# equivalent to:  glm(y~X3-1, family="binomial", data=Default)
```

```r
base_plot_income +
  geom_function(fun = ~predict(fit_3, newdata=tibble(income=.)),
                aes(color="Binning")) +
  geom_hline(yintercept = gamma0, linetype = "dashed", color = "grey60") +
  labs(y = "f(x)")
```



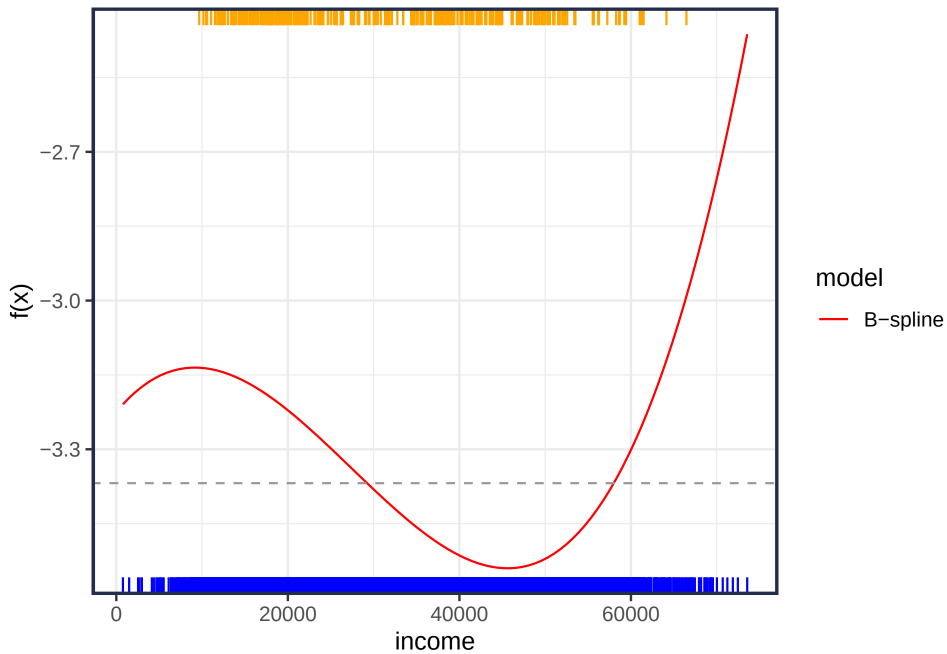

```r
base_plot_income +
  geom_function(fun = ~predict(fit_3, newdata=tibble(income=.),
                               type = "response"),
                aes(color="Binning")) +
  geom_hline(yintercept = p0, linetype = "dashed", color = "grey60") +
  labs(y = "p_hat(x)")
```



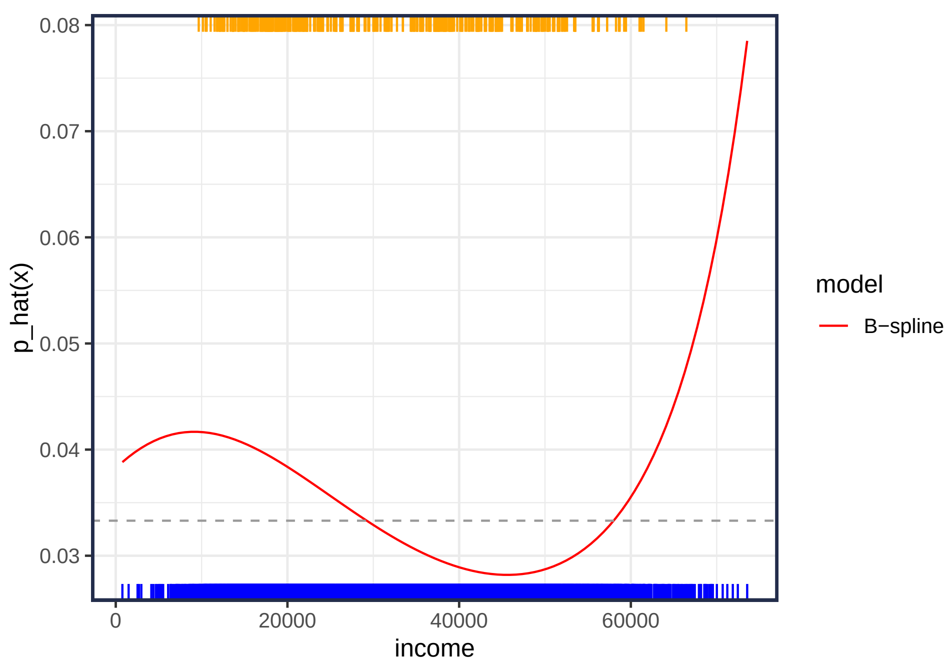
- **Example 4: Continuous Predictor with B-Splines Basis**

```
library(splines) # for bs() function

#: B-spline Basis
X4 = model.matrix(~bs(income, df=4)-1, data=Default)
head(X4, 4)
#>   bs(income, df = 4)1 bs(income, df = 4)2 bs(income, df = 4)3
#> 1              0.1204              0.4351            0.428565
#> 2              0.5755              0.1229            0.008137
#> 3              0.3521              0.4809            0.166404
#> 4              0.2625              0.4994            0.238160
#>   bs(income, df = 4)4
#> 1           1.591e-02
#> 2           0.000e+00
#> 3           0.000e+00
#> 4           2.576e-05

#: Binning Model (edf=5)
fit_4 = glm(y~bs(income, df=3), data=Default, family="binomial")
tidy(fit_4)
#> # A tibble: 4 x 5
#>   term                 estimate std.error statistic      p.value
#>   <chr>                   <dbl>     <dbl>     <dbl>        <dbl>
#> 1 (Intercept)             -3.21     0.580     -5.53 0.0000000323
#> 2 bs(income, df = 3)1     0.457     1.44       0.317 0.752
#> 3 bs(income, df = 3)2    -1.44      0.617     -2.33  0.0196
#> 4 bs(income, df = 3)3     0.747     1.16       0.644 0.519
# equivalent to:  glm(y~X4, family="binomial", data=Default)
```

```
base_plot_income +
  geom_function(fun = ~predict(fit_4, newdata=tibble(income=.)),
                aes(color="B-spline")) +
  geom_hline(yintercept = gamma0, linetype = "dashed", color = "grey60") +
  labs(y = "f(x)")
```



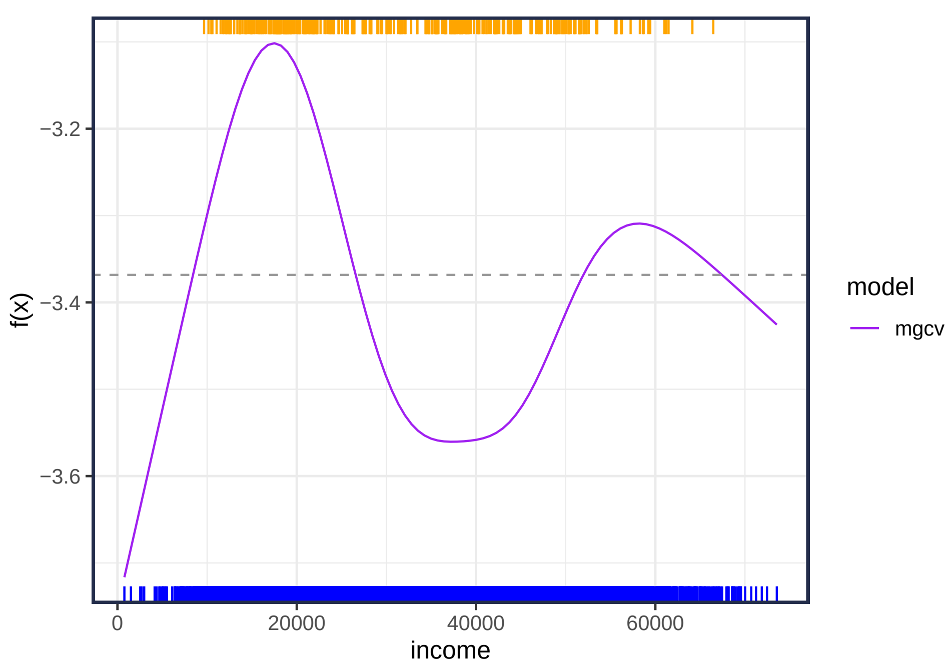

```
base_plot_income +
  geom_function(fun = ~predict(fit_4, newdata=tibble(income=.),
                                type = "response"),
                aes(color="B-spline")) +
  geom_hline(yintercept = p0, linetype = "dashed", color = "grey60") +
  labs(y = "p_hat(x)")
```



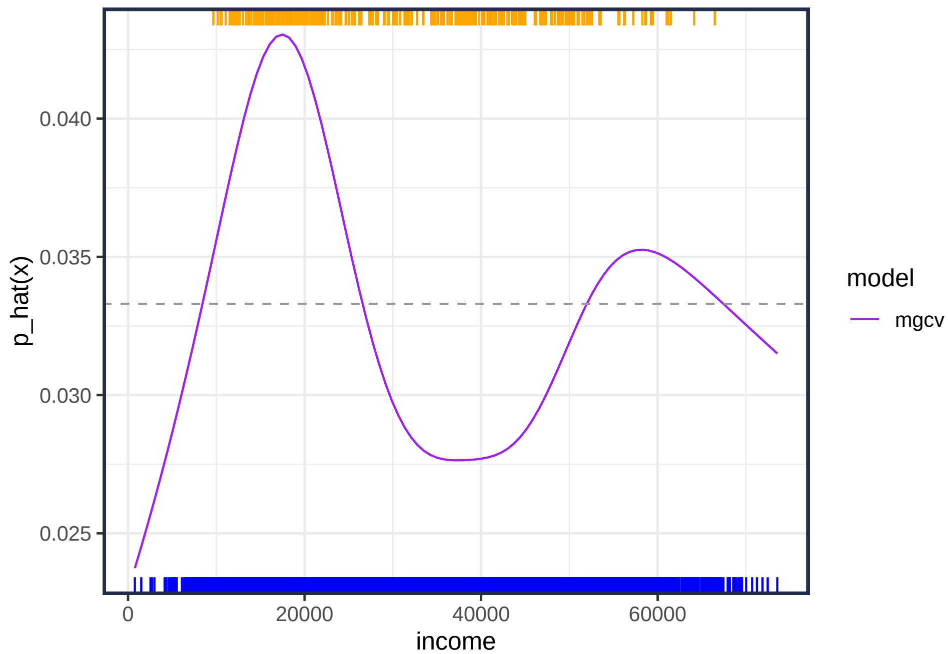
- **Example 5: Continuous Predictor with Penalized Spline**

```r
library(mgcv)
#: Fit penalized spline, it will select best edf
#    specify smooth with s()
fit_5 = gam(y~s(income), data=Default, family="binomial")
summary(fit_5)
#>
#> Family: binomial
#> Link function: logit
#>
#> Formula:
#> y ~ s(income)
#>
#> Parametric coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  -3.3819     0.0564     -60   <2e-16 ***
#> ---
#> 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(income) 4.31   5.37   10.8    0.06 .
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> R-sq.(adj) =  0.00098   Deviance explained = 0.466%
#> UBRE = -0.70823  Scale est. = 1         n = 10000
```

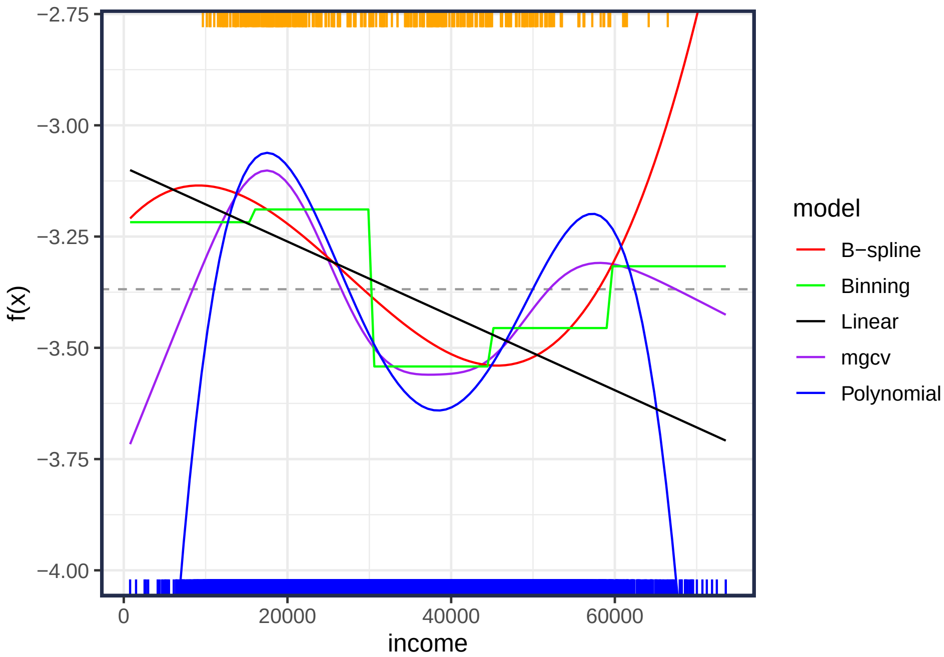
```r
base_plot_income +
  geom_function(fun = ~predict(fit_5, newdata=tibble(income=.)),
                aes(color="mgcv")) +
  geom_hline(yintercept = gamma0, linetype = "dashed", color = "grey60") +
  labs(y = "f(x)")
```



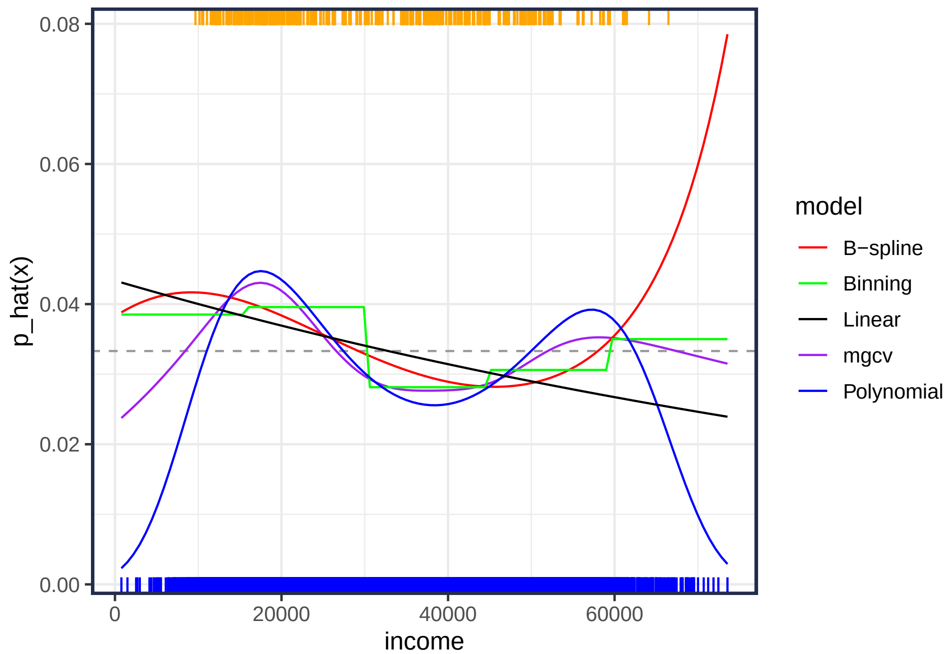
```
base_plot_income +
  geom_function(fun = ~predict(fit_5, newdata=tibble(income=.),
                                       type = "response"),
                aes(color="mgcv")) +
  geom_hline(yintercept = p0, linetype = "dashed", color = "grey60") +
  labs(y = "p_hat(x)")
```




```
#: Plot of Fit
base_plot_income +
  coord_cartesian(ylim = c(-4, -2.8)) +
  geom_hline(yintercept = gamma0, linetype = "dashed", color = "grey60") +
  labs(y="f(x)") +
  geom_function(fun = ~predict(fit_5, newdata=tibble(income=.)), aes(color="mgcv")) +
  geom_function(fun = ~predict(fit_4, newdata=tibble(income=.)), aes(color="B-spline")) +
  geom_function(fun = ~predict(fit_3, newdata=tibble(income=.)), aes(color="Binning")) +
  geom_function(fun = ~predict(fit_2, newdata=tibble(income=.)), aes(color="Polynomial")) +
  geom_function(fun = ~predict(fit_lm, newdata=tibble(income=.)), aes(color="Linear"))
```




```r
#: Plot of Fit
base_plot_income +
  geom_hline(yintercept = p0, linetype = "dashed", color = "grey60") +
  labs(y="p_hat(x)") +
  geom_function(fun = ~predict(fit_5, newdata=tibble(income=.),
                               type = "response"), aes(color="mgcv")) +
  geom_function(fun = ~predict(fit_4, newdata=tibble(income=.),
                               type = "response"), aes(color="B-spline")) +
  geom_function(fun = ~predict(fit_3, newdata=tibble(income=.),
                               type = "response"), aes(color="Binning")) +
  geom_function(fun = ~predict(fit_2, newdata=tibble(income=.),
                               type = "response"), aes(color="Polynomial")) +
  geom_function(fun = ~predict(fit_lm, newdata=tibble(income=.),
                               type = "response"), aes(color="Linear"))
```



1.1 Generalized Additive Models (GAMs)

All of the above models are for a *single* predictor. The extension to multiple predictors is called **Generalized Additive Models (GAMs)**.

Instead of the linear form

$$f(\mathbf{x}) = \beta_0 + \sum_j \beta_j x_j,$$

use non-linear bases for each predictor

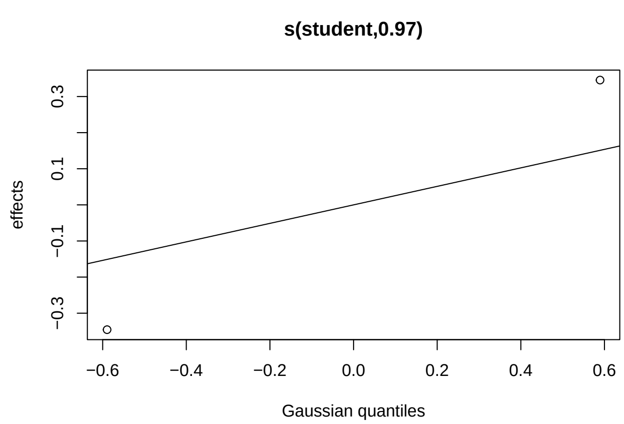
$$f(\mathbf{x}) = \beta_0 + \sum_j s_j(x_j)$$

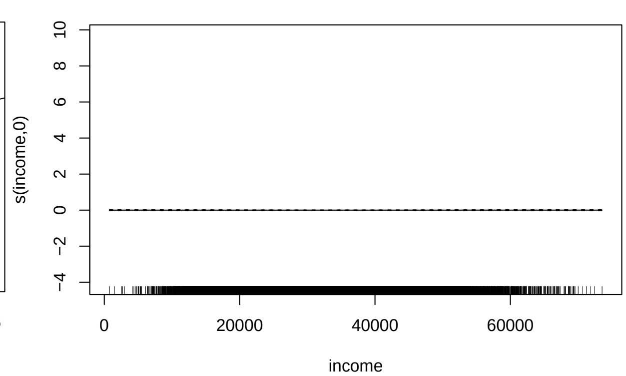
where $s_j(x_j)$ can allow non-linear (e.g., smooth) forms, like B-splines.

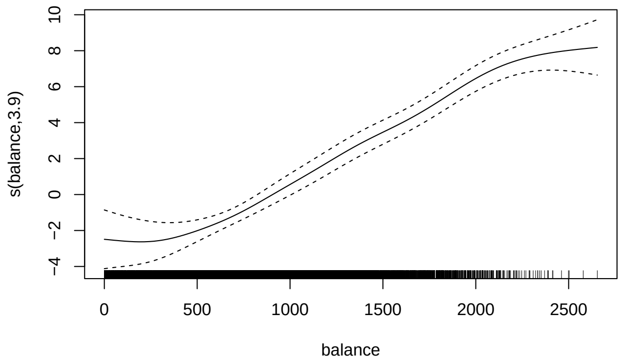
- For binary classification setting: logit $p(\mathbf{x}) = f(\mathbf{x})$
- These are *additive* models because each term adds its contribution, although potentially in a non-linear way
  - But interactions can still be accommodated using `s(x1, x2)` or `s(x1, by=fac)` where `fac` is a factor.
- These are *generalized* models following the GLM notation. You can use different distributions with the `family=` argument
- GAMs retain the interpretability of a linear additive model (linear regression, logistic regression), but can add complexity to predictors where needed
  - Drawback: can be slow, especially for very high dimensional data

- In **R**, the `mgcv` package is excellent for implementing GAM models.
  - It used Generalized Cross-validation to select optimal smoothing for each component
  - It also has a `select=TRUE` argument to further shrink entire components toward 0
  - Can handle low dimension interactions (even factor-continuous)
- See ISL 7.7 or ESL 9.1 for more details
- The **R**package `gratia` makes working with `mgcv` a little easier.

```
library(mgcv)

fit_gam = gam(y ~ s(student, bs="re") + s(income) + s(balance),  # smooth main effects
              select = TRUE,                                  # shrink components toward 0
              family = "binomial",
              data = Default)

summary(fit_gam)
#>
#> Family: binomial
#> Link function: logit
#>
#> Formula:
#> y ~ s(student, bs = "re") + s(income) + s(balance)
#>
#> Parametric coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)   -5.962      0.543     -11   <2e-16 ***
#> ---
#> 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(student) 0.97203      1   22.3 2.4e-06 ***
#> s(income)  0.00314      9    0.0    0.81
#> s(balance) 3.89551      9 2294.9 < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> R-sq.(adj) =  0.339   Deviance explained = 46.3%
#> UBRE = -0.84186  Scale est. = 1         n = 10000
plot(fit_gam)
```







We can make the model more complex by adding interactions, e.g.,

$$\hat{f}(x) = \hat{\beta}_0 + \sum_j \hat{s}_j(x_j) + \sum_j \sum_k \hat{s}_{jk}(x_j, x_k) + \dots$$

```
# complex model with lots of possible interactions
fit_gam_interactions =
  gam(y ~ s(student, bs = "re") + s(income) + s(balance) + # smooth main effects
          s(income, by=student) + s(balance, by=student) + # smooth interaction
          s(income, balance),
      select = TRUE,                      # shrink components toward 0
      family = "binomial",
      data = Default)
summary(fit_gam_interactions)
#>
#> Family: binomial
#> Link function: logit
#>
#> Formula:
#> y ~ s(student, bs = "re") + s(income) + s(balance) + s(income,
#>     by = student) + s(balance, by = student) + s(income, balance)
#>
#> Parametric coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)   -5.606      0.321   -17.4   <2e-16 ***
#> ---
#> 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(student)              0.000448      1    0.0   0.022 *
#> s(income)               0.001130      9    0.0   0.767
#> s(balance)              3.881927      9  638.5 < 2e-16 ***
#> s(income):studentNo     0.001801      9    0.0   0.757
#> s(income):studentYes    4.516272      9   35.8 1.9e-05 ***
#> s(balance):studentNo    0.003674      9    0.0   0.696
#> s(balance):studentYes   0.000822      9    0.0   0.922
#> s(income,balance)       0.001582     27    0.0   0.637
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> R-sq.(adj) =  0.344   Deviance explained = 46.6%
#> UBRE = -0.84222  Scale est. = 1         n = 10000
```

1.2 Estimating $\hat{s}_j(x_j)$ with Backfitting

The smooth terms of a GAM model can be estimating using an iterative approach called *backfitting*.

Algorithm: Backfitting for GAM (Squared Error Loss / Linear Regression)

Model: $\hat{y}(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} \hat{s}_j(x_j)$

1. Start with intercept-only model. All smooth terms set to zero: $s_j(x_j) = 0$.
2. Iterate over all p predictor variables:

   a. Construct *partial residuals* $r_i = y_i - \hat{\beta}_0 - \sum_{k \neq j} \hat{s}_k(x_{ik})$ holding out the jth predictor
   b. Fit jth smoother to residuals: Estimate $\hat{s}(x_j)$ from $\{(r_i, x_{ij})\}_{i=1}^{n}$

3. Repeat many times stopping when converged (i.e., smooth fits no longer changing very much)

Note: There are more details (see ESL 9.1), but this is the main (and simple) idea.

2 GAM fitting with boosted stumps

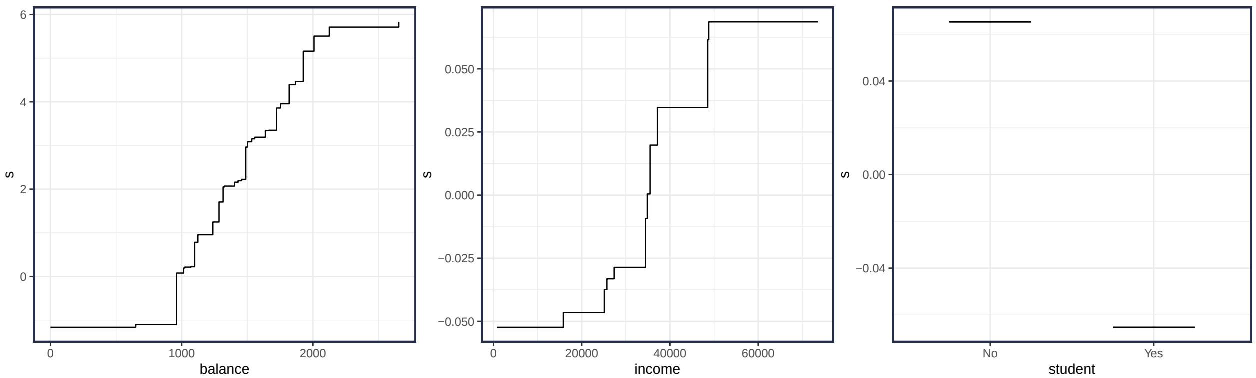
Recall that boosted stumps (i.e., trees with single split and two nodes) will estimate non-linear *main effects* and no interactions. This is exactly what GAM is seeking to do! One difference is that boosting works iteratively to estimate the best functional form, while GAM estimates everything all at once.

```r
library(tidyverse)
library(lightgbm)
library(bonsai)

#: pre-processing specs
rec_lgbm = recipe(default ~ student + balance + income, data = Default)

# Define the lightgbm model specification
lgbm_spec = boost_tree(
  trees = 500,                     # nrounds
  tree_depth = 1,                   # stumps
  learn_rate = 0.01,                # Learning rate
  loss_reduction = 1,               # min_gain_to_split
  sample_size = 0.8,                # Subsample ratio
  mode = "classification"           # For multi-class classification
) %>%
  set_engine("lightgbm", num_threads = 6) %>%
  set_args(bagging_seed = 123) # controls the internal sampling


#: Create XGBoost workflow (combine recipe with model specification)
lgbm_wf = workflow(preprocessor = rec_lgbm, spec = lgbm_spec)


# Fit the model
lgbm_fit = fit(lgbm_wf, data = Default)

# Make predictions on the training data
predict(lgbm_fit, Default, type = "prob") %>% head()
#> # A tibble: 6 x 2
#>   .pred_No .pred_Yes
#>      <dbl>     <dbl>
#> 1    0.998   0.00230
#> 2    0.998   0.00185
#> 3    0.992   0.00808
#> 4    0.998   0.00213
#> 5    0.998   0.00230
#> 6    0.998   0.00185
```

```r
lgbm_fit %>% extract_fit_engine() %>% lgb.importance()
#> # A tibble: 3 x 4
#>   Feature    Gain  Cover Frequency
#>   <chr>     <dbl>  <dbl>     <dbl>
#> 1 balance 0.990   0.900       0.9
#> 2 student 0.00538 0.0539      0.054
#> 3 income  0.00424 0.0460      0.046
```

```r
# Using objects outside of function: lgbm_wf, lgbm_fit
plot_lgbm_smooth <- function(var, n_grid = 200, model = lgbm_fit){
  # get training data
  data_train = model$pre$mold$predictors
  # get quantiles of training data
  x = data_train %>% pull({{var}})
  if(is.numeric(x)) xx = quantile(x, probs = seq(0, 1, length = n_grid), na.rm=TRUE)
  else xx = unique(x)
  n_grid = min(length(xx), n_grid)
  # make prediction data frame using 1st training data observation values
  data_pred =
    slice_sample(data_train[1,], n = n_grid, replace = TRUE) %>%
    mutate(
      {{var}} := xx
    )
  # add predictions to the data (subtract gamma0)
  data_plt =
    data_pred %>%
    mutate(
      s = predict(model, data_pred, type = "raw"),
      s = s - mean(s)
    )
  # output plot
  plt = ggplot(data_plt, aes(.data[[var]], s))
  if(is.numeric(x)) plt + geom_step()
  else plt + geom_errorbar(aes(width = 1/2, ymin=s, ymax=s) )
}
```

```
plot_lgbm_smooth("balance")
plot_lgbm_smooth("income")
plot_lgbm_smooth("student")
```



Now let's refit, but using many more trees:

```
lgbm_fit2 = lgbm_wf %>%
  update_model(extract_spec_parsnip(.) %>% update(trees = 1500)) %>%
  fit(Default)

plot_lgbm_smooth("balance", model = lgbm_fit2)
plot_lgbm_smooth("income", model = lgbm_fit2)
plot_lgbm_smooth("student", model = lgbm_fit2)
```