Supervised Learning

Smoothing splines and GAMs

July 9th, 2021

Kernel regression

Nadaraya-Watson kernel regression

- given training data with explanatory variable x and continuous response y
- *bandwidth* $h > 0$
- and a new point (x_{new}, y_{new}):

$$\hat{y}_{new} = \sum_{i=1}^{n} w_i(x_{new}) \cdot y_i \ ,$$

where

$$w_i(x) = \frac{K_h\left(|x_{new} - x_i|\right)}{\sum_{j=1}^{n} K_h\left(|x_{new} - x_j|\right)} \text{ with } K_h(x) = K(\frac{x}{h})$$

Example of a **linear smoother**

- class of models where predictions are *weighted* sums of the response variable

Local regression

We can fit a linear model **at each point** x_{new} with weights given by kernel function centered on x_{new}

- we can additionally combine this with *polynomial regression*

Local regression of the k^{th} order with kernel function K solves the following:

$$\hat{\beta}(x_{new}) = \arg\min_{\beta}\Big\{\sum_{i} K_h(|x_{new} - x_i|) \cdot (y_i - \sum_{j=0}^{k} x_i^k \cdot \beta_k)^2\Big\}$$

Yes, this means every single observation has its own set of coefficients

Predicted value is then:

$$\hat{y}_{new} = \sum_{j=0}^{k} x_{new}^k \cdot \hat{\beta}_k(x_{new})$$

Smoother predictions than kernel regression but comes at **higher computational cost**

- **LOESS** replaces kernel with k nearest neighbors
  - faster than local regression but discontinuities when neighbors change

Smoothing splines

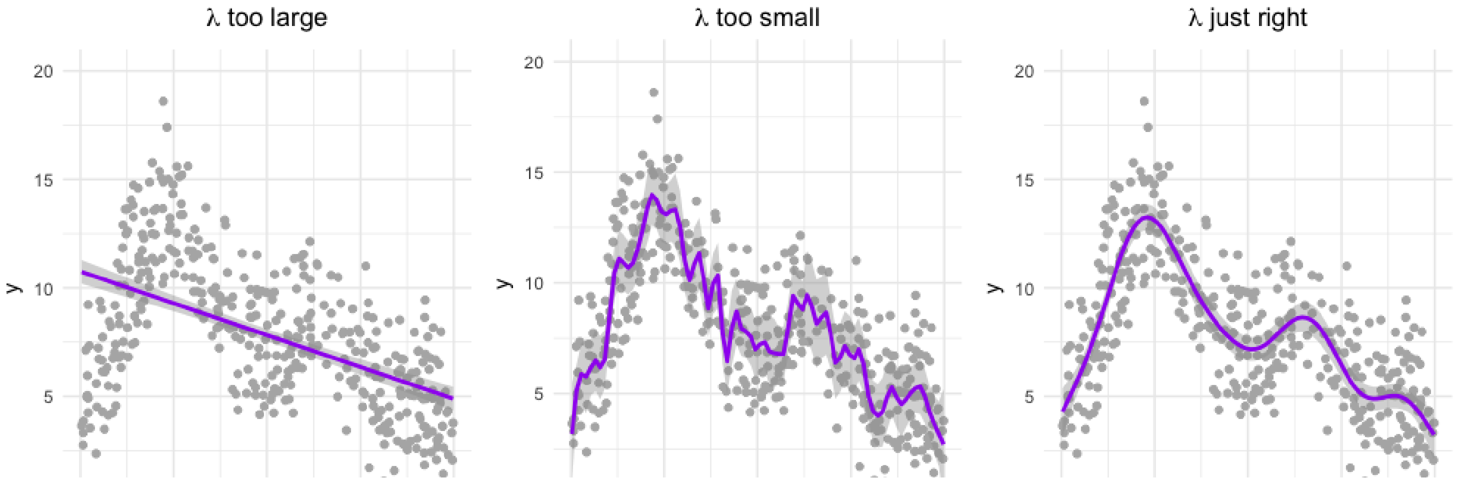
Use **smooth function** $s(x)$ to predict y, control smoothness directly by minimizing the **spline objective function**:

$$\sum_{i=1}^{n}(y_i - s(x_i))^2 + \lambda \int (s''(x))^2 dx$$

$$= \text{fit data} + \text{impose smoothness}$$

$$\Rightarrow \text{model fit} = \text{likelihood} - \lambda \cdot \text{wiggliness}$$

Estimate the **smoothing spline** $\hat{s}(x)$ that **balances the tradeoff between the model fit and wiggliness**



Basis functions

Splines are *piecewise cubic polynomials* with **knots** (boundary points for functions) at every data point

Practical alternative: linear combination of set of **basis functions**

Cubic polynomial example: define four basis functions:

- $B_1(x) = 1$, $B_2(x) = x$, $B_3(x) = x^2$, $B_4(x) = x^3$

where the regression function $r(x)$ is written as:

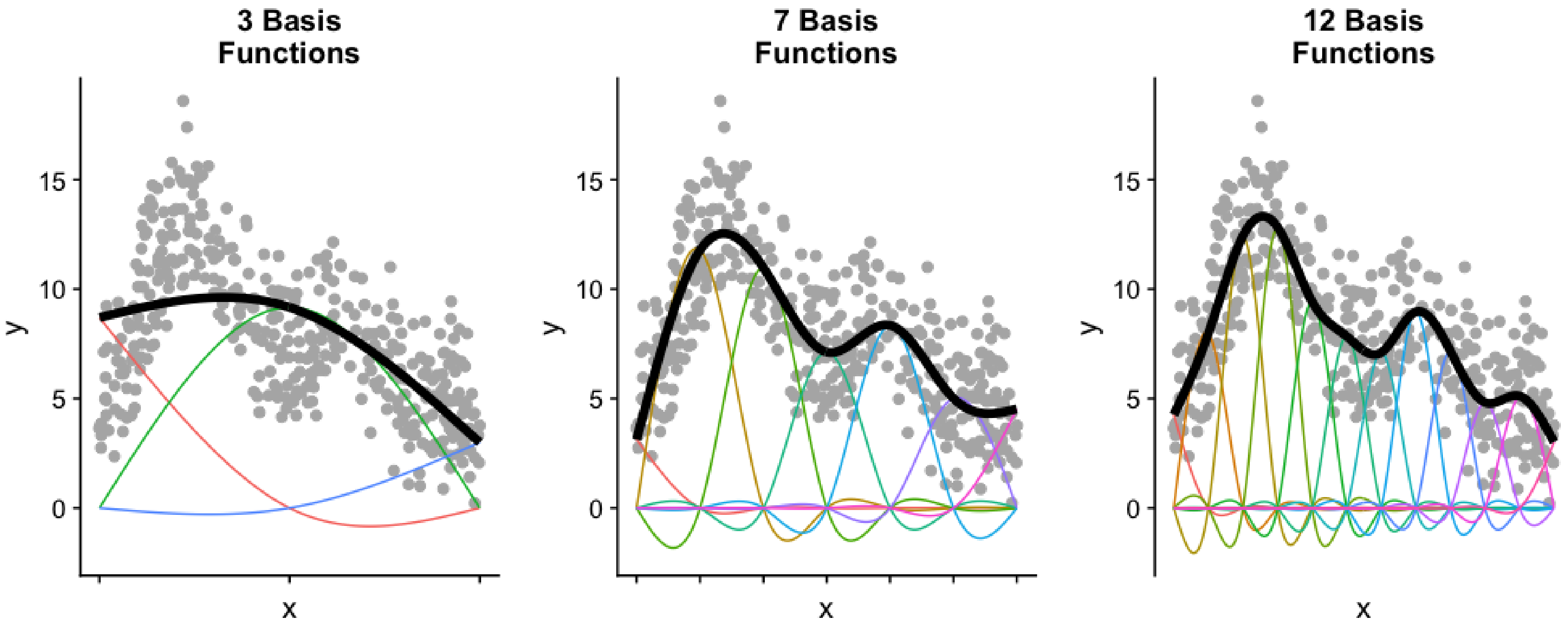
$$r(x) = \sum_j^4 \beta_j B_j(x)$$

- **linear in the transformed variables** $B_1(x), \ldots, B_4(x)$ but it is **nonlinear in** x

We extend this idea for splines *piecewise* using indicator functions so the spline is a weighted sum:

$$s(x) = \sum_j^m \beta_j B_j(x)$$

Number of basis functions is another tuning parameter



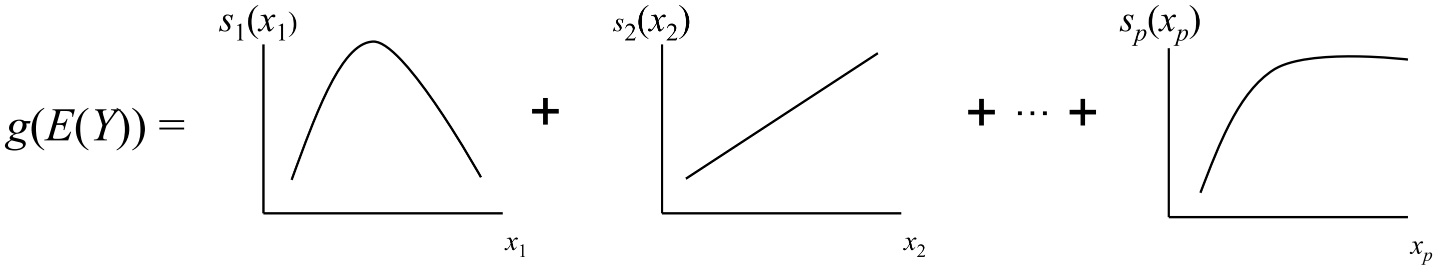
Generalized additive models (GAMs)

GAMs were created by Trevor Hastie and Rob Tibshirani in 1986 with intuitive construction:

- relationships between individual explanatory variables and the response variable are smooth (either linear or nonlinear via basis functions)
- estimate the smooth relationships **simultaneously** to predict the response by just adding them up

Generalized like GLMs where $g()$ is the link function for the expected value of the response $E(Y)$ and **additive** over the p variables:

$$g(E(Y)) = \beta_0 + s_1(x_1) + s_2(x_2) + \cdots + s_p(x_p)$$



- can be a convenient balance between flexibility and interpretability
- you can combine linear and nonlinear terms!

Example: predicting MLB HR probability

Used the `baseballr` package to scrape all batted-balls from 2019 season:

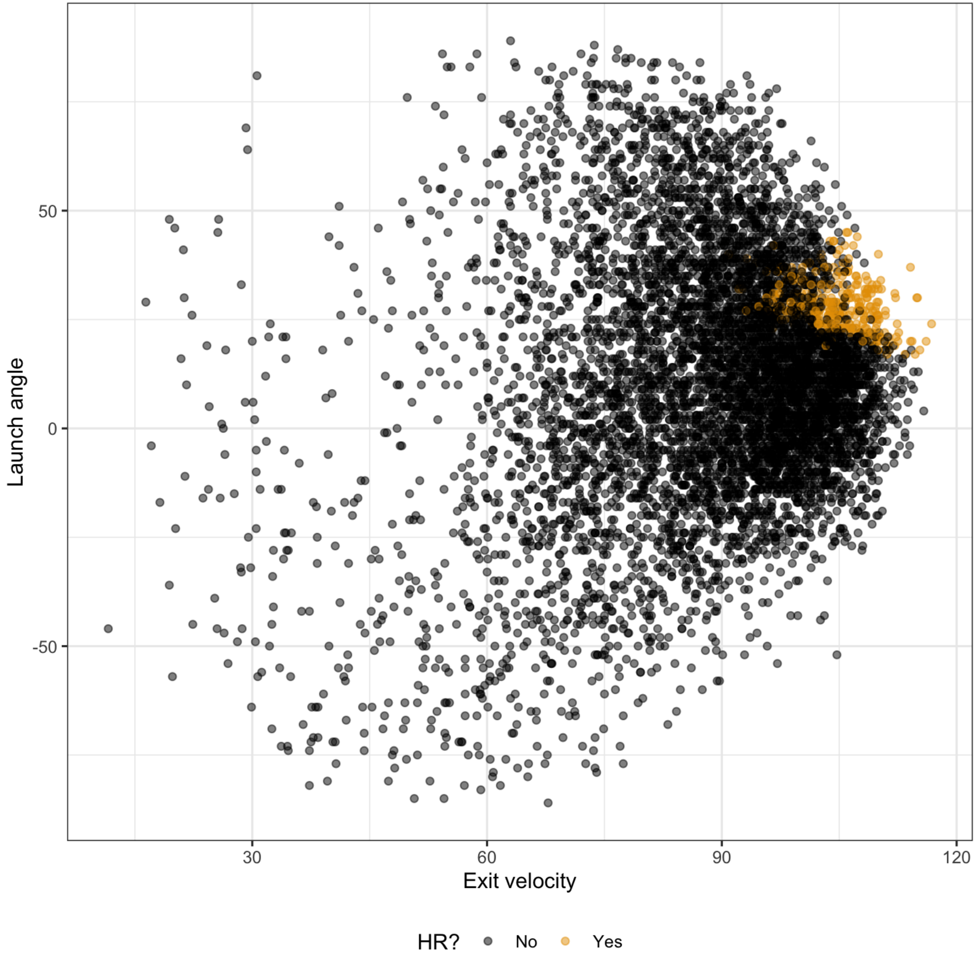
```
library(tidyverse)
batted_ball_data <- read_csv("http://www.stat.cmu.edu/cmsac/sure/2021/materials/data/eda_projects
  mutate(is_hr = as.numeric(events == "home_run")) %>%
  filter(!is.na(launch_angle), !is.na(launch_speed),
         !is.na(is_hr))
head(batted_ball_data)
```

```
## # A tibble: 6 × 32
##   player_name    batter stand events  hc_x  hc_y hit_d…¹ launc…² launc…³ hit_l…⁴
##   <chr>           <dbl> <chr> <chr>  <dbl> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 Ahmed, Nick    605113 R     single  94.3  94.7     272    86.3      19       7
## 2 Herrera, Odúb… 546318 L     field…  97.7  86.7     289    90.7      44       8
## 3 Davis, Jonath… 641505 R     field… 121.  107.      233    87        52       8
## 4 Harrison, Josh 543281 R     field… 116.  148.      159    53        38       6
## 5 Smith, Pavin   656976 L     field…  79.2  75.4     328    99.1      19       7
## 6 Hernández, Te… 606192 R     single  44.0  78.8     357   104.       22       7
## # … with 22 more variables: bb_type <chr>, barrel <dbl>, pitch_type <chr>,
## #   release_speed <dbl>, effective_speed <dbl>, if_fielding_alignment <chr>,
## #   of_fielding_alignment <chr>, game_date <date>, balls <dbl>, strikes <dbl>,
## #   outs_when_up <dbl>, on_1b <dbl>, on_2b <dbl>, on_3b <dbl>, inning <dbl>,
## #   inning_topbot <chr>, home_score <dbl>, away_score <dbl>,
```

Predict HRs with launch angle and exit velocity?

```
batted_ball_data %>%
  ggplot(aes(x = launch_speed,
             y = launch_angle,
             color = as.factor(is_hr))) +
  geom_point(alpha = 0.5) +
  ggthemes::scale_color_colorblind(labels = c
  labs(x = "Exit velocity",
       y = "Launch angle",
       color = "HR?") +
  theme_bw() +
  theme(legend.position = "bottom")
```

- HRs are relatively rare and confined to one area of this plot



Fitting GAMs with mgcv

First set-up training data

```
set.seed(2004)
batted_ball_data <- batted_ball_data %>%
  mutate(is_train = sample(rep(0:1, length.out = nrow(batted_ball_data))))
```

Next fit the initial function using smooth functions via `s()`:

```
library(mgcv)
init_logit_gam <- gam(is_hr ~ s(launch_speed) + s(launch_angle),
                      data = filter(batted_ball_data, is_train == 1),
                      family = binomial, method = "REML")
```

- Use REML instead of the default for more stable solution

GAM summary

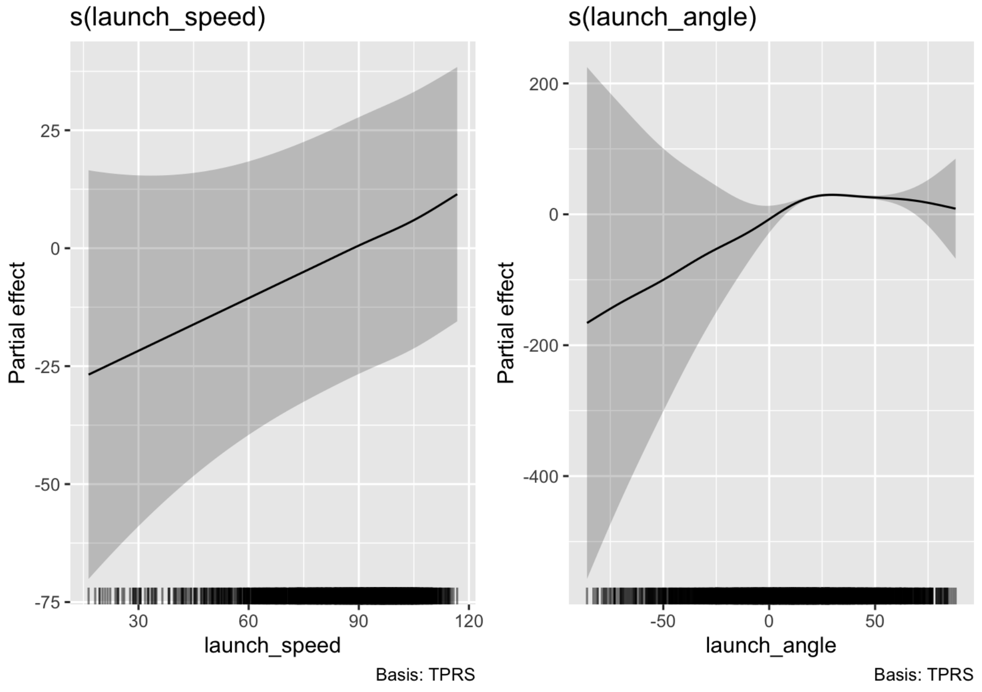
```r
summary(init_logit_gam)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## is_hr ~ s(launch_speed) + s(launch_angle)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -34.14      13.91  -2.455   0.0141 *
## ---
## 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(launch_speed) 1.928  2.388 172.22  <2e-16 ***
## s(launch_angle) 3.640  3.970  92.94  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.551   Deviance explained = 65.8%
## -REML = 234.05  Scale est. = 1         n = 3377
```

Visualizing partial response functions with gratia

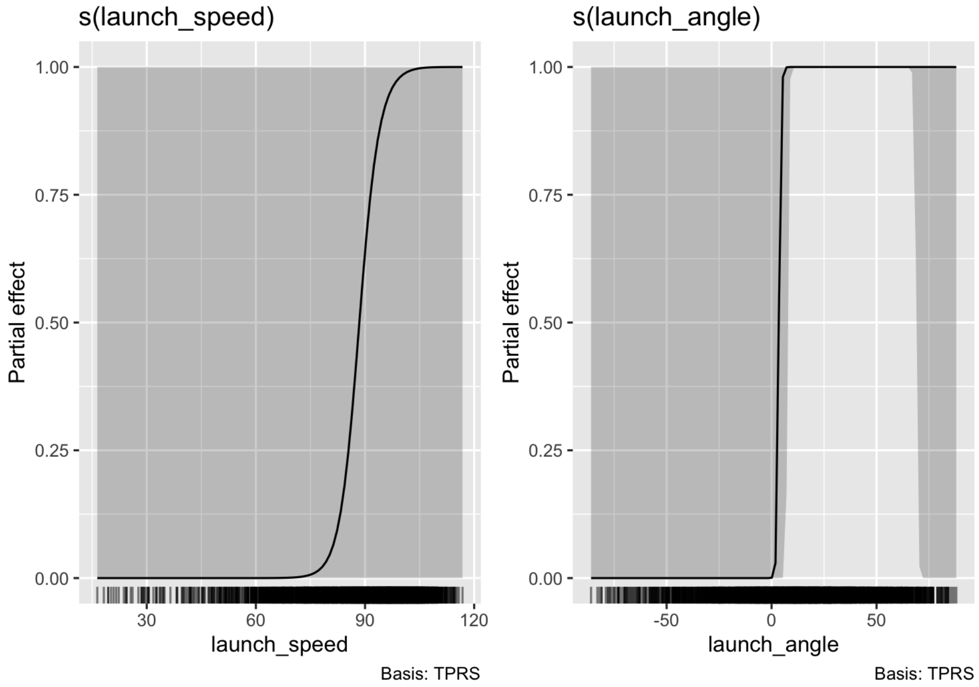
Displays the partial effect of each term in the model ⇒ add up to the overall prediction

```
library(gratia)
draw(init_logit_gam)
```



Convert to probability scale with `plogis` function

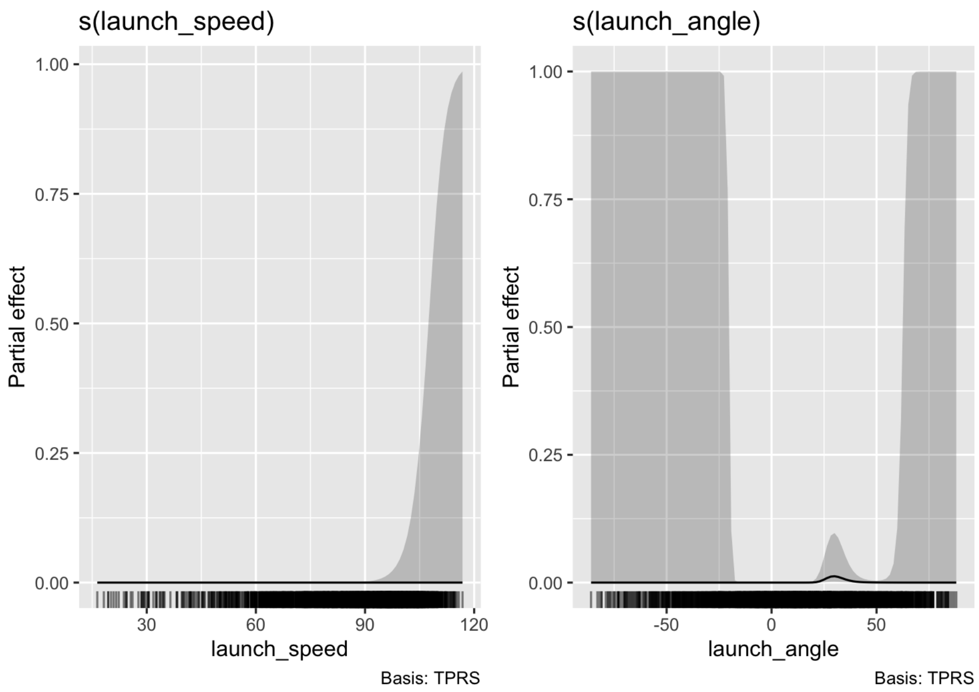
```r
draw(init_logit_gam, fun = plogis)
```



- centered on average value of 0.5 because it's the partial effect without the intercept

Include intercept in plot...

```
draw(init_logit_gam, fun = plogis, constant = coef(init_logit_gam)[1])
```



Intercept reflects relatively rare occurence of HRs!

Model checking for number of basis functions

Use `gam.check()` to see if we need more basis functions based on an approximate test

```
gam.check(init_logit_gam)
```

```
##
## Method: REML   Optimizer: outer newton
## full convergence after 9 iterations.
## Gradient range [-0.0001120962,1.387397e-05]
## (score 234.0517 & scale 1).
## Hessian positive definite, eigenvalue range [0.1473281,0.7723187].
## Model rank =  19 / 19
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                   k'  edf k-index p-value
## s(launch_speed) 9.00 1.93    0.97    0.12
## s(launch_angle) 9.00 3.64    1.03    0.97
```

Check the predictions?

```r
batted_ball_data <- batted_ball_data %>%
  mutate(init_gam_hr_prob =
           as.numeric(predict(init_logit_gam,
                              newdata = batted_ball_data,
                              type = "response")),
         init_gam_hr_class = as.numeric(init_gam_hr_prob >= 0.5))
batted_ball_data %>%
  group_by(is_train) %>%
  summarize(correct = mean(is_hr == init_gam_hr_class))
```

```
## # A tibble: 2 × 2
##   is_train correct
##      <int>   <dbl>
## 1        0   0.974
## 2        1   0.970
```

What about the linear model?

```
init_linear_logit <- glm(is_hr ~ launch_speed + launch_angle,
                         data = filter(batted_ball_data, is_train == 1),
                         family = binomial)
batted_ball_data <- batted_ball_data %>%
  mutate(init_glm_hr_prob = predict(init_linear_logit,
                                    newdata = batted_ball_data,
                                    type = "response"),
         init_glm_hr_class = as.numeric(init_glm_hr_prob >= 0.5))
batted_ball_data %>%
  group_by(is_train) %>%
  summarize(correct = mean(is_hr == init_glm_hr_class))
```

```
## # A tibble: 2 × 2
##   is_train correct
##      <int>   <dbl>
## 1        0   0.960
## 2        1   0.954
```

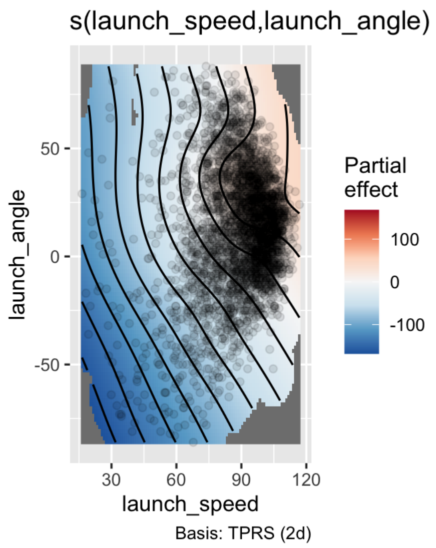
Very few situations in reality where linear regressions perform better than an additive model using smooth functions - especially since smooth functions can just capture linear models...

Continuous interactions as a smooth surface

```
multi_logit_gam <- gam(is_hr ~ s(launch_speed, launch_angle),
                       data = filter(batted_ball_data, is_train == 1),
                       family = binomial)
```

Plot the predicted heatmap:

```
draw(multi_logit_gam)
```



Check the predictions?

```r
batted_ball_data <- batted_ball_data %>%
  mutate(multi_gam_hr_prob =
           as.numeric(predict(multi_logit_gam,
                              newdata = batted_ball_data,
                              type = "response")),
         multi_gam_hr_class = as.numeric(multi_gam_hr_prob >= 0.5))
batted_ball_data %>%
  group_by(is_train) %>%
  summarize(correct = mean(is_hr == multi_gam_hr_class))
```

```
## # A tibble: 2 × 2
##   is_train correct
##      <int>   <dbl>
## 1        0   0.975
## 2        1   0.971
```

- This has one smoothing parameter for the 2D smooth...

Separate interactions from individual terms with tensor smooths

```
tensor_logit_gam <- gam(is_hr ~ s(launch_speed) + s(launch_angle) +
                          ti(launch_speed, launch_angle),
                        data = filter(batted_ball_data, is_train == 1), family = binomial)
```

Check the predictions

```
batted_ball_data %>%
  mutate(tensor_gam_hr_prob =
           as.numeric(predict(tensor_logit_gam, newdata = batted_ball_data, type = "response")),
         tensor_gam_hr_class = as.numeric(tensor_gam_hr_prob >= 0.5)) %>%
  group_by(is_train) %>%
  summarize(correct = mean(is_hr == tensor_gam_hr_class))
```

```
## # A tibble: 2 × 2
##   is_train correct
##      <int>   <dbl>
## 1        0   0.975
## 2        1   0.970
```

More complicated model but yet it does not help!

Recap and useful resources

Dr Gavin Simpson
@ucfagls · Follow

140 char vrsn

1 GAMs are just GLMs
2 GAMs fit wiggly terms
3 use + s(foo) not foo in frmla
4 use method = "REML"
5 gam.check()

2:37 PM · Mar 16, 2017

71 Reply Copy link

Read 5 replies

- GAMs in R by Noam Ross
- `mgcv` course
- Stitch Fix post: GAM: The Predictive Modeling Silver Bullet
- Chapters 7 and 8 of Advanced Data Analysis from an Elementary Point of View by Prof Cosma Shalizi