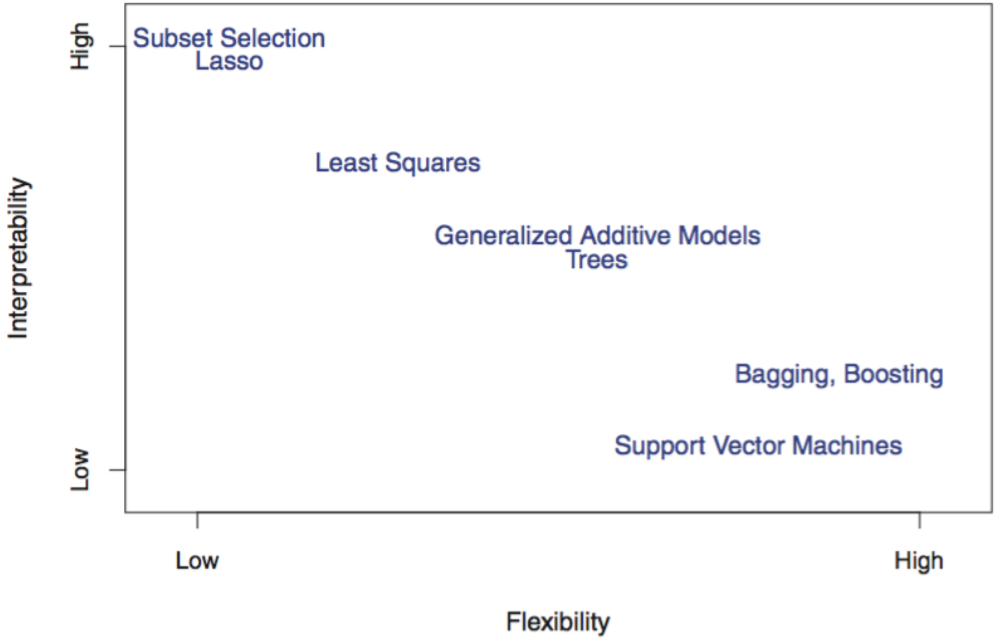
Advanced Topics in Regression

Kernels, Smoothers, and Generalized Additive Models

July 11th, 2023

Model flexibility vs interpretability

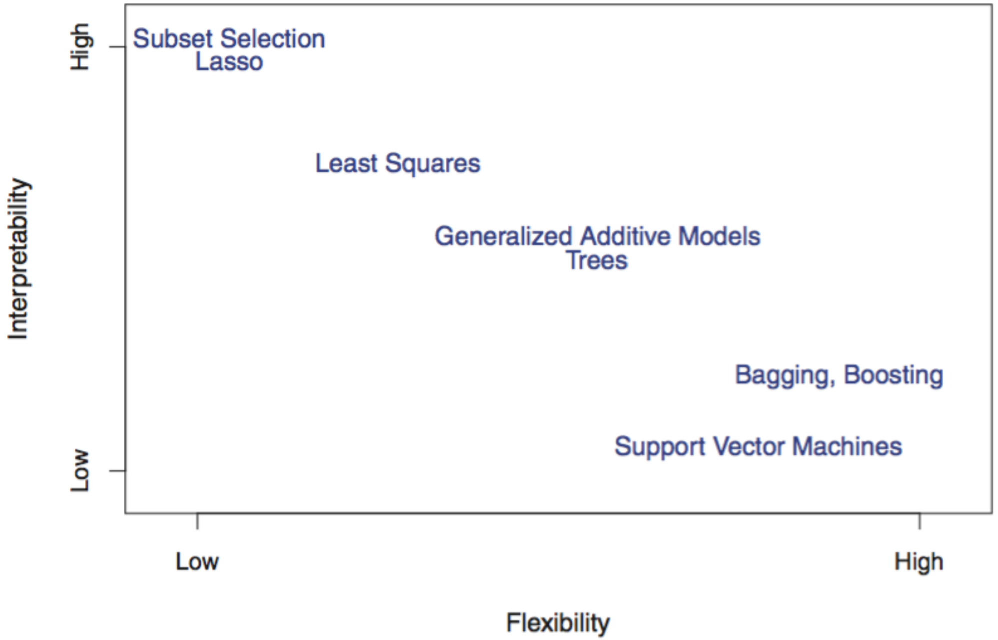
Figure 2.7, Introduction to Statistical Learning with Applications in R (ISLR)



Tradeoff between model's *flexibility* (i.e. how "curvy" it is) and how **interpretable** it is

- Simpler, parametric form of the model $\Rightarrow$ the easier it is to interpret

Model flexibility vs interpretability

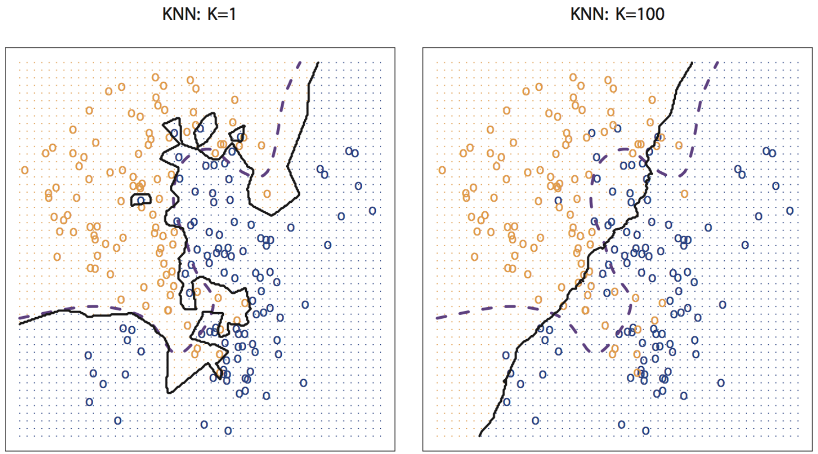


- **Parametric** models, for which we can write down a mathematical expression for $f(X)$ **before observing the data**, *a priori* (e.g. linear regression), **are inherently less flexible**
- **Nonparametric** models, in which $f(X)$ is **estimated from the data** (e.g. kernel regression)

Recall: K Nearest Neighbors (KNN)

- Find the k data points **closest** to an observation x, use these to predict
  - Regression: $\hat{Y}|X = \frac{1}{k}\sum_{i=1}^{k} Y_i$ (*average response*)
  - Classification: $\hat{P}[Y = j|X] = \frac{1}{k}\sum_{i=1}^{k} 1(Y_i = j)$ (*majority vote*)

Determining the optimal value of k requires balancing bias and variance



Averaging with Neighbors?? Kernels!!

A kernel $K(x)$ is a weighting function used in estimators, and technically has only one required property:

- $K(x) \geq 0$ for all x

However, in the manner that kernels are used in statistics, there are two other properties that are usually satisfied:

- $\int_{-\infty}^{\infty} K(x)dx = 1$; and
- $K(-x) = K(x)$ for all x.

In short: **a kernel is a symmetric PDF!**

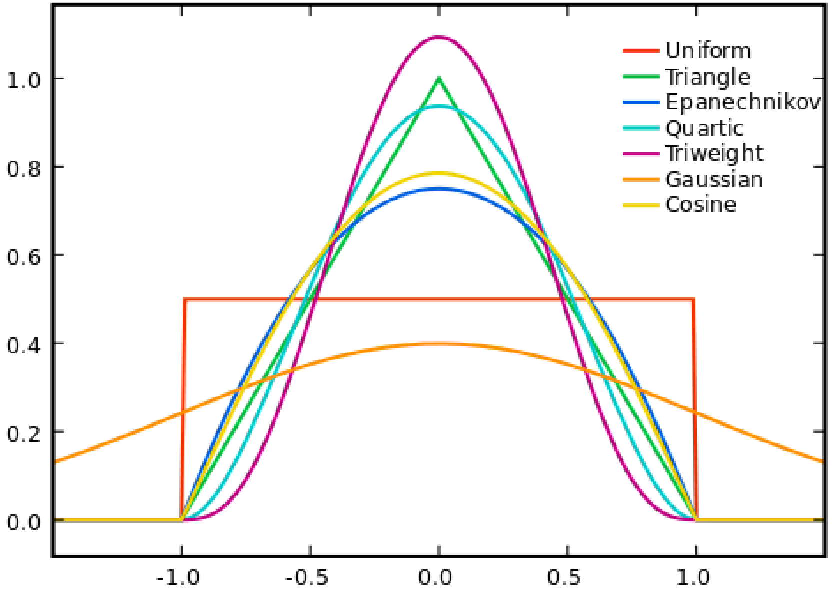
Recall: Kernel density estimation

Goal: estimate the PDF $f(x)$ for all possible values (assuming it is continuous / smooth)

$$\text{Kernel density estimate: } \hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}K_h(x - x_i)$$

- $n =$ sample size, $x =$ new point to estimate $f(x)$ (does NOT have to be in dataset!)
- $h =$ **bandwidth**, analogous to histogram bin width, ensures $\hat{f}(x)$ integrates to 1
- $x_i = i$th observation in dataset
- $K_h(x - x_i)$ is the **Kernel** function, creates **weight** given distance of ith observation from new point
  - as $|x - x_i| \to \infty$ then $K_h(x - x_i) \to 0$, i.e. further apart ith row is from x, smaller the weight
  - as **bandwidth** $h \uparrow$ weights are more evenly spread out (as $h \downarrow$ more concentrated around x)
  - typically use **Gaussian** / Normal kernel: $\propto e^{-(x-x_i)^2/2h^2}$
  - $K_h(x - x_i)$ is large when x_i is close to x

Commonly Used Kernels



A general rule of thumb: the choice of kernel will have little effect on estimation, particularly if the sample size is large! The Gaussian kernel (i.e., a normal PDF) is by far the most common choice, and is the default for `R` functions that utilize kernels.

Kernel regression

We can apply kernels in the regression setting as well as in the density estimation setting!

The classic kernel regression estimator is the **Nadaraya-Watson** estimator:

$$\hat{y}_h(x) = \sum_{i=1}^{n} w_i(x) Y_i \, ,$$

where

$$w_i(x) = \frac{K\left(\frac{x - X_i}{h}\right)}{\sum_{j=1}^{n} K\left(\frac{x - X_j}{h}\right)} \, .$$

Regression estimate is the average of all the *weighted* observed response values;

- Farther x is from observation $\Rightarrow$ less weight that observation has in determining the regression estimate at x

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

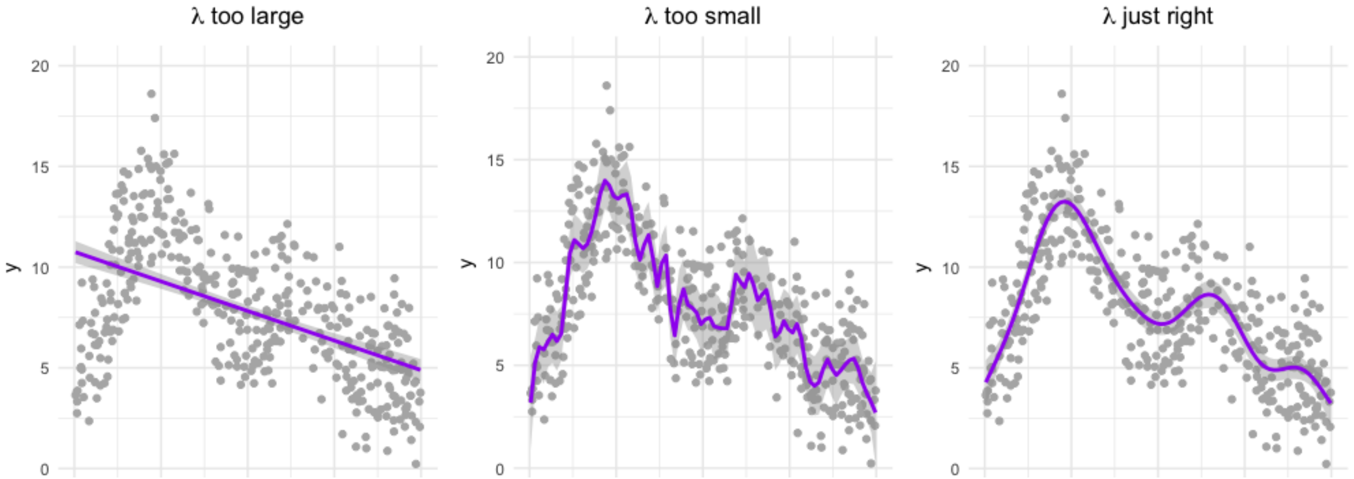
Use **smooth function** $s(x)$ to predict y, control smoothness directly by minimizing the **spline objective function**:

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

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

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

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:

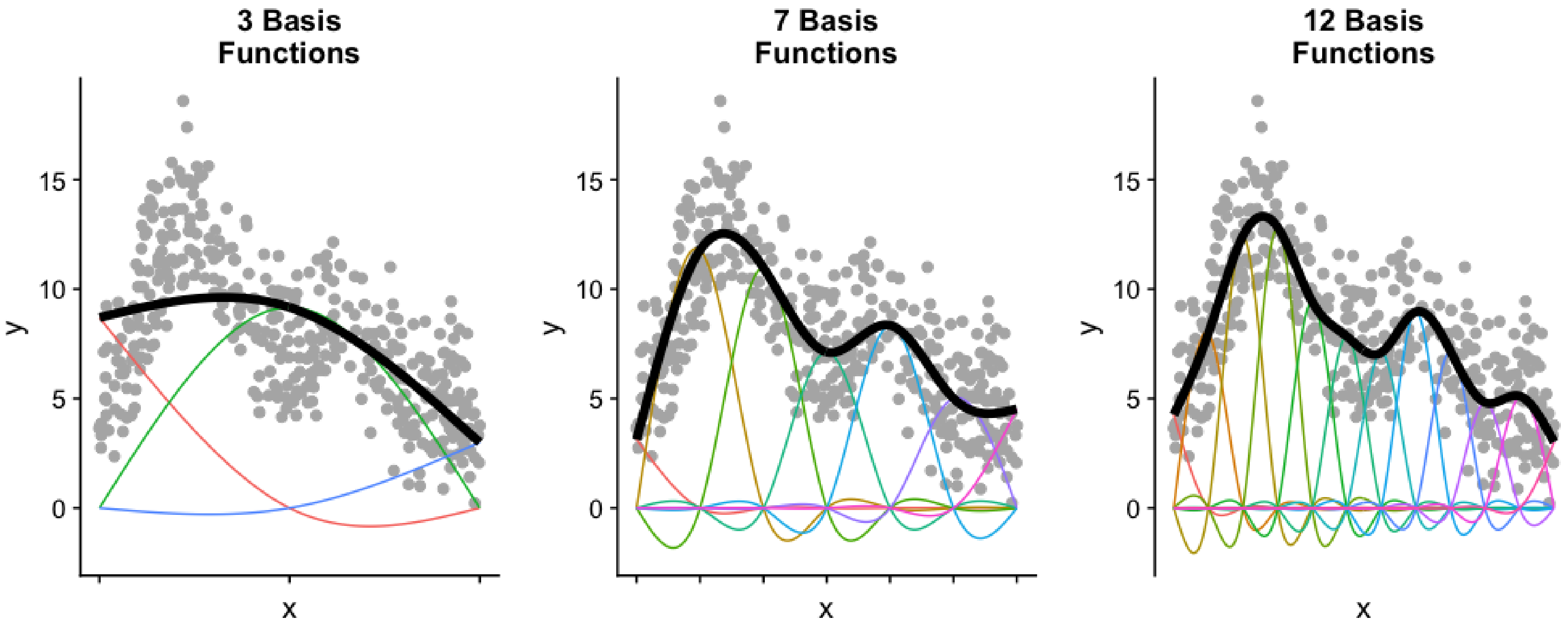
$$r(x) = \sum_j^4 \beta_j B_j(x)$$

- **linear in the transformed variables** $B_1(x), \dots, B_4(x)$ 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



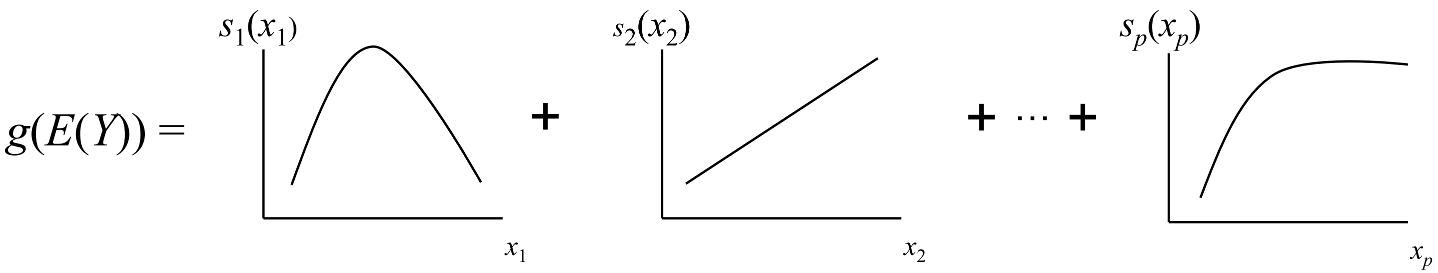
Generalized additive models (GAMs)

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 2022 season:

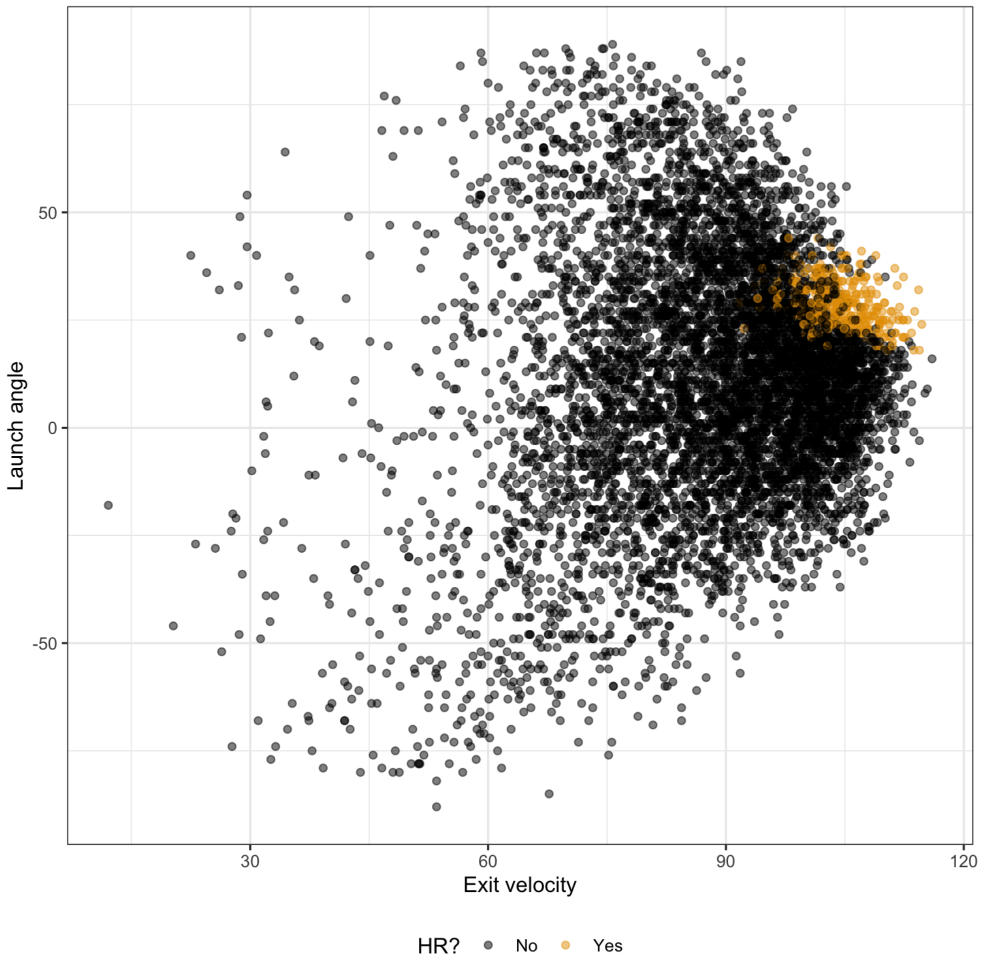
```r
library(tidyverse)
batted_ball_data <- read_csv("https://shorturl.at/moty2") %>%
  mutate(is_hr = as.numeric(events == "home_run")) %>%
  filter(!is.na(launch_angle), !is.na(launch_speed),
         !is.na(is_hr))
head(batted_ball_data)
```

```
## # A tibble: 6 × 32
##   player_name    batter stand events      hc_x  hc_y hit_distance_sc launch_speed
##   <chr>           <dbl> <chr> <chr>      <dbl> <dbl>           <dbl>        <dbl>
## 1 Daza, Yonathan 602074 R     force_out 103.   150.               18         97.4
## 2 Robles, Victor 645302 R     single     58.6 120.              158         80.2
## 3 Hoerner, Nico  663538 R     field_out  99.3 166.               20        101.
## 4 Clemens, Kody  665019 L     field_out 126.   191.              165         84
## 5 Rosario, Amed  642708 R     field_out  97.4 170.                9         94.3
## 6 Castro, Willi  650489 L     sac_fly   178.    58.9             369         96
## # ℹ 24 more variables: launch_angle <dbl>, hit_location <dbl>, 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>,
```

Predict HRs with launch angle and exit velocity?

```r
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("No", "Yes")) +
  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

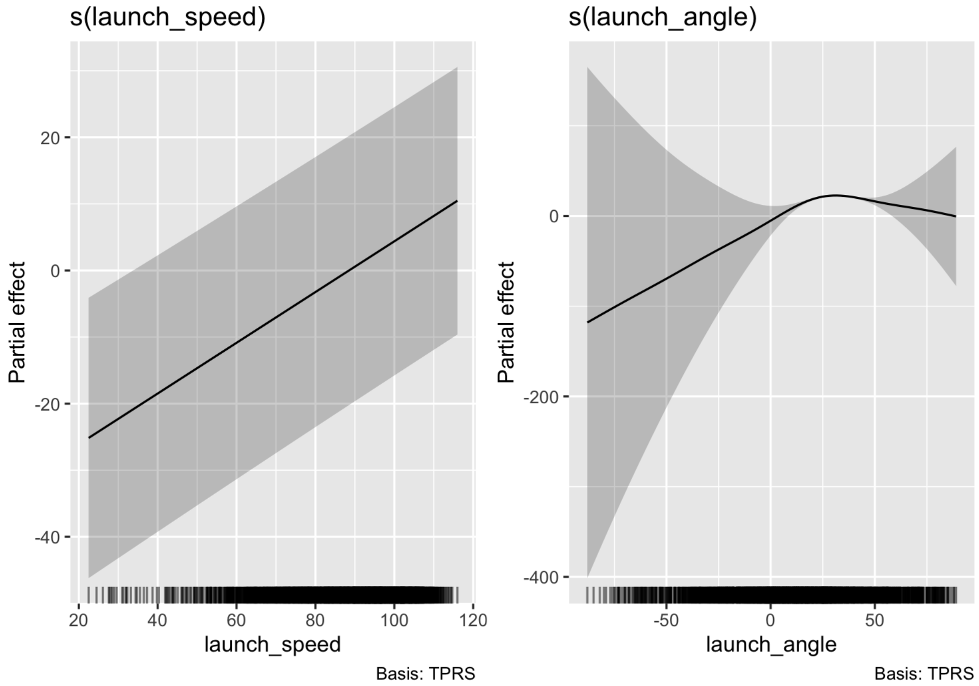
```
summary(init_logit_gam)
```

```
## 
## Family: binomial 
## Link function: logit 
## 
## Formula:
## is_hr ~ s(launch_speed) + s(launch_angle)
## 
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)   
## (Intercept)   -26.96      10.31  -2.614  0.00895 **
## ---
## 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) 1.000  1.000  151.5  <2e-16 ***
## s(launch_angle) 2.962  3.305  112.0  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## R-sq.(adj) =  0.588   Deviance explained = 68.3%
## -REML = 231.49  Scale est. = 1         n = 3517
```

Visualizing partial response functions with gratia

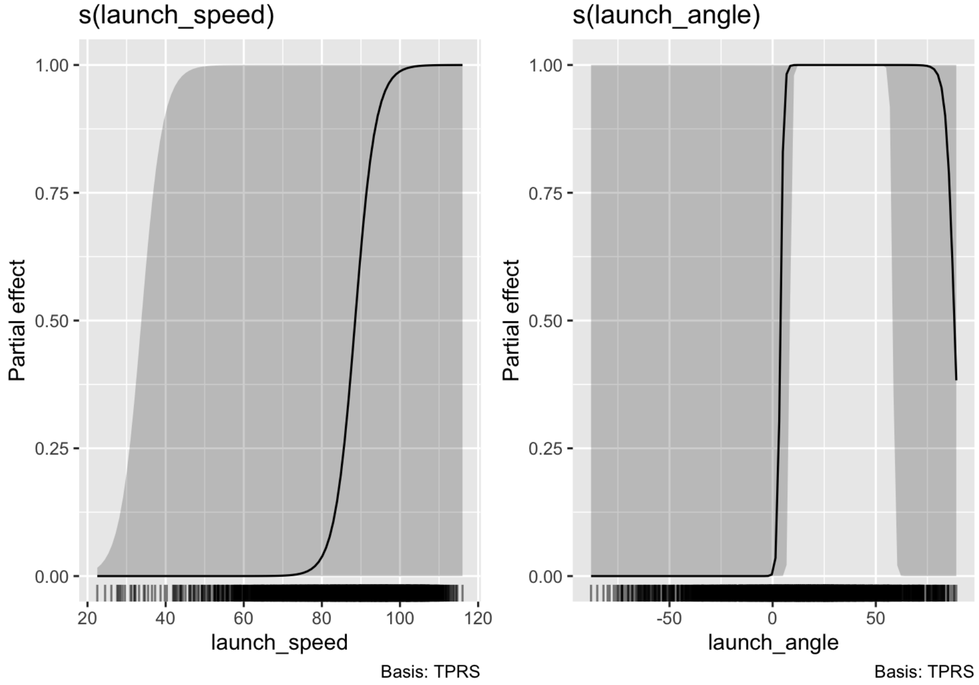
Displays the partial effect of each term in the model ⇒ add up to the overall prediction

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



Convert to probability scale with `plogis` function

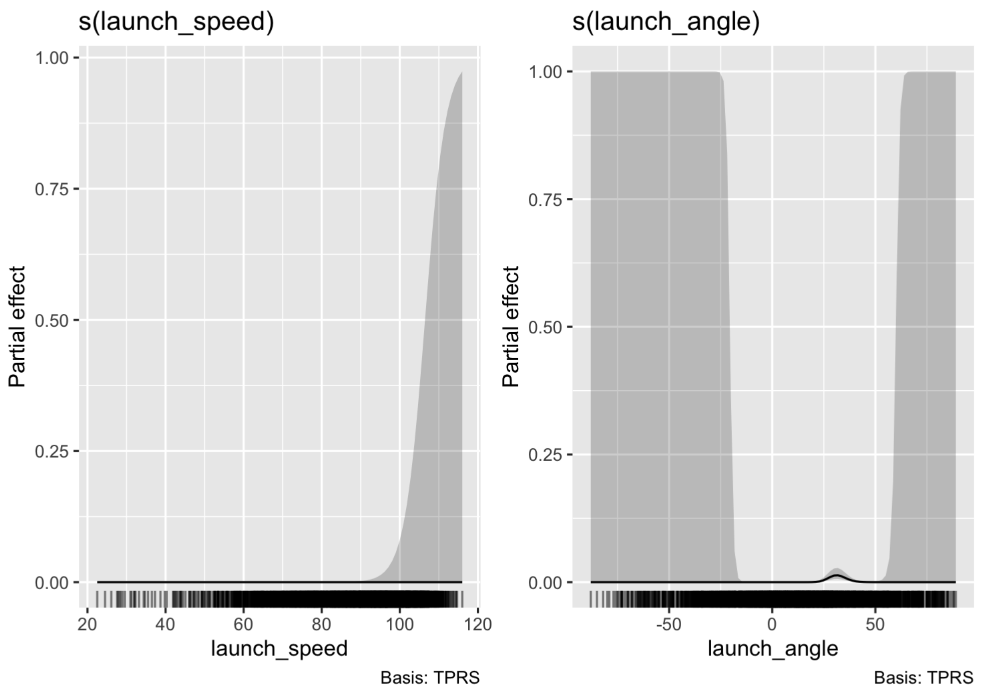
```
draw(init_logit_gam, fun = plogis)
```



- 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 11 iterations.
## Gradient range [-5.632542e-05,-2.964163e-06]
## (score 231.4864 & scale 1).
## Hessian positive definite, eigenvalue range [5.631851e-05,0.8679399].
## 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 1.00    1.05    1.00
## s(launch_angle) 9.00 2.96    0.97    0.08 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Check the predictions?

```
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   0.977
## 2        1   0.972
```

What about the linear model?

```r
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   0.951
```

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...

Some useful resources

- 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
    - I strongly recommend you download this book, and you will refer back to it for the rest of your life