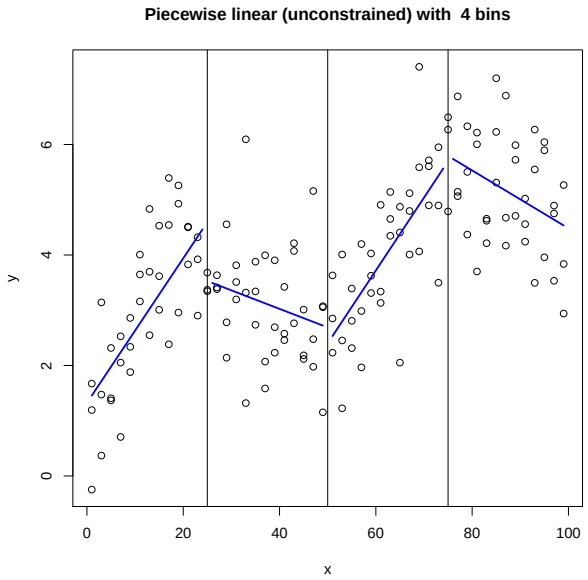
Machine Learning II
Week 3 Lecture – 23rd October 2019
Non-linear regression methods II

Contents

Local Regression

Last Week: One of the non-linear function prediction methods was to piecewise linear regression. A regression line is fitted to each intervall



$$y_i = f(x_i) = f_0(x_i)C_0(x_i) + f_0(x_i)C_0(x_i) + \cdots f_K(x_i)C_K(x_i) + \epsilon_i$$

where each $f_k(x_i) = \beta_{0k} + \beta_{1k}x$ is a linear function

We quickly moved on to adding constraints such as the piecewise function should be continuous and smooth.

This week we will take this idea in another direction: Instead of four disjoint intervals each with a quarter of the points, we move a *window* containing a quarter of the data points through the range of x.

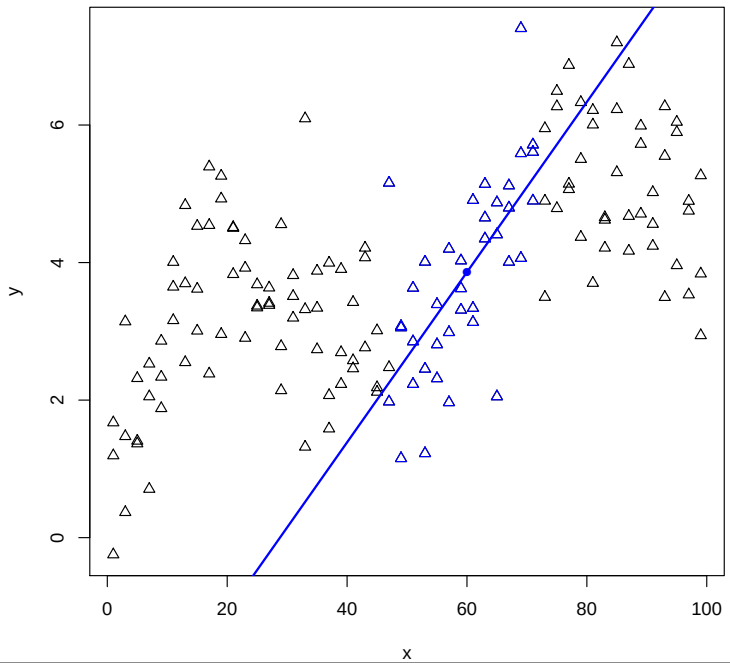
To estimate $f(x_0)$, instead of finding the disjoint interval which contains x_0, we take the $\frac{1}{4}n$ points nearest to x_0.

Because we are using $\frac{1}{4}n$ points, the *span* of the window is equal to $\frac{1}{4}$.

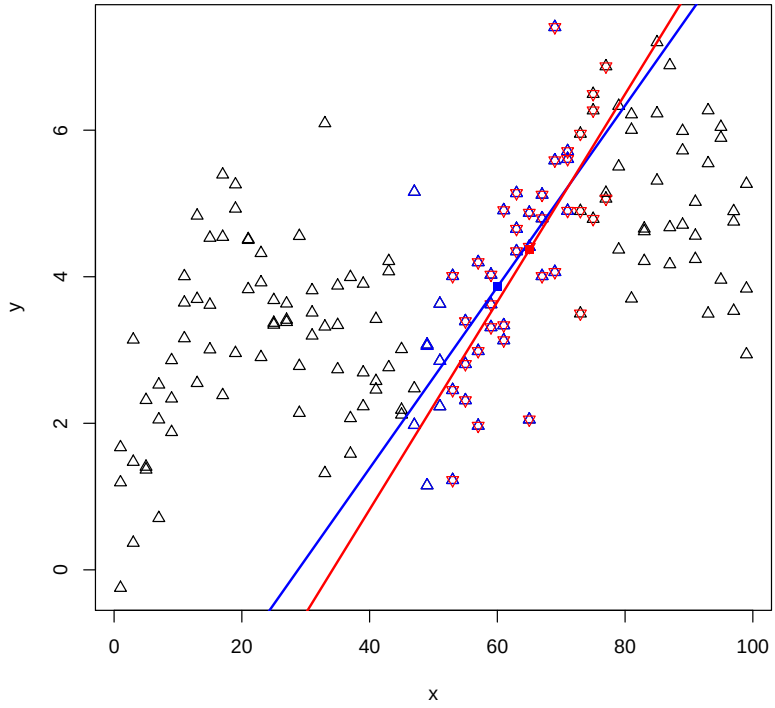
Example: $x_0 = 60$.
The regression line using the $\frac{1}{4}n = 38$ points nearest to x_0 is

$$f(x) = -3.6 + 0.12x$$

giving $f(x_0) = f(60) = 3.86$

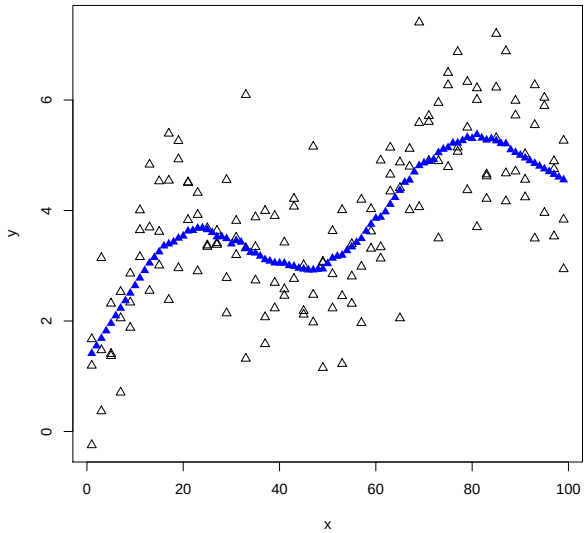


To predict $f(65)$ we use the 38 nearest points to $x_0 = 65$ which includes 31 of the blue points on the previous slide.



Blue: $x_0 = 60$ Red: $x_0 = 65$

We then repeat for all the points on a grid and plot them to result in the predictor function $f(x)$.



Note that it is not necessary that x_0 is an x-value in our data, we just need to identify the values nearest to x_0

The local regression algorithm, called *loess* (locally estimated least squares) or *lowess*, has a couple of important difference to the above explanation.

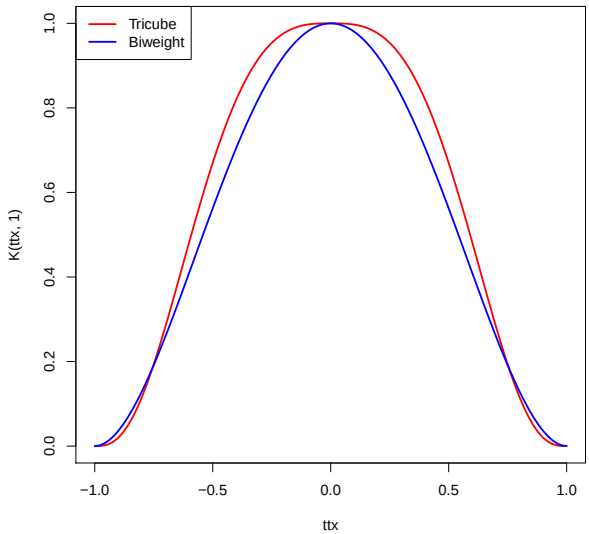
In the above example we computed the regression line giving equal weight to all points in the window.

Loess uses weighted regression, so that the points close to x_0 have a large influence on f.
Points further away, but still within the window, have less influence.
Points outside the window have zero influence.

The weight function is defined using the distance of each x_i from x_0.
The function used to calculate the weights is called a Kernel $K(\delta)$.
In loess $\delta = x_i - x_0$.
There are lots of possibilities for the Kernel. They are all symmetric about the y axis, have a maximum at δ=0 and are monotonic decreasing in $|\delta|$. There is a computational advantage when $K(\delta) = 0$ when δ is outside the window.

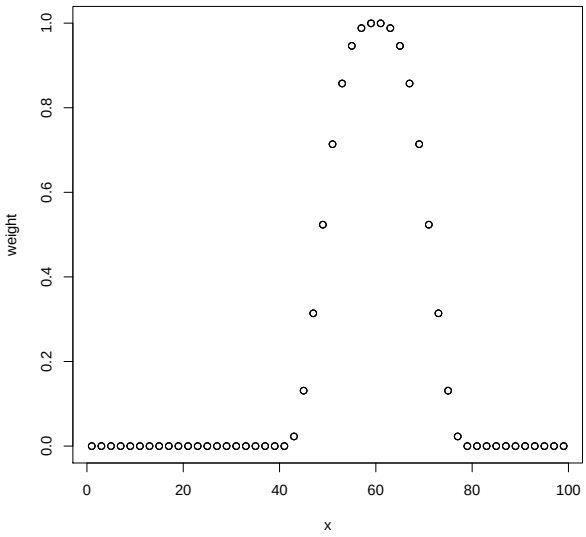
The two most common are Kernal functions for loess fitting are
Biweight $K(\delta) = (1 - \delta^2)^2$ for $|\delta| \leqslant 1$ and
Tricube $K(\delta) = (1 - |\delta|^3)^3$ for $|\delta| \leqslant 1$ (the default in R)

The unweighted example corresponds to using a *uniform* or *boxcar* kernel.
$K(\delta) = 1$ for $|\delta| \leqslant 1$



For a given x_0 the weight of the i-th point is $w_i = K\left(\frac{x_i - x_0}{s}\right)$ where s is the distance of the furthest x_i still inside the window.

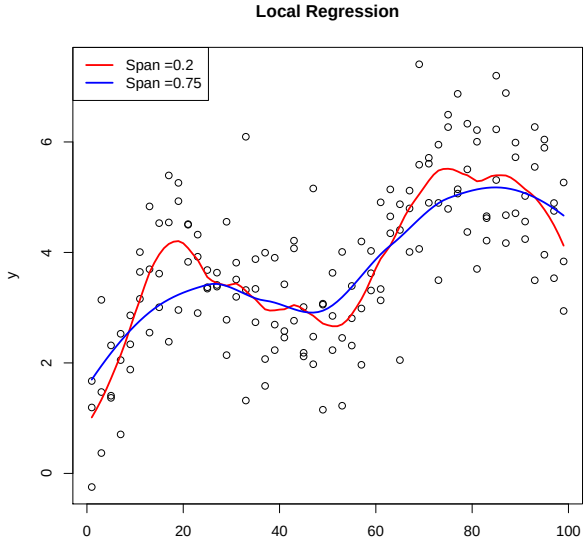
As the weight is more localised than with the uniform kernel it makes sense to use a wider window. Here are the weights for each data point with a span of 0.4 and $x_0 = 60$



Weighted least squares

The principle is the same as least squares estimation, except that each residual is multiplied by the weight. Minimise $RSS = \sum_{i=1}^{n} w_i(y_i - f_\beta(x_i))^2$

The other difference in the loess implementation is that usually the regression is a locally quadratic regression instead of locally linear regression. The `loess` function in R takes an argument `degree=` to specify locally constant, linear or quadratic.



Comments on local regression

Again we need to choose choose the level of smoothing. This is done by defining the span. As the span is (almost) always within the range $[0, 1]$ we have an reasonable starting point for the parameter unlike some of the other methods last week. The default in R is 0.75.

Loess is a good all-round method which can be used to gain a first impression of the relationship between x and y. In R some plot methods add a loess curve by default, or make it easy to specify as an option. Example: The diagnostic plots for an R object of class `lm` add a red loess curve as a visual guide

Generalised additive Models

A multiple linear regression model has the form.

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \epsilon_i.$$

Non-linear equivalent is the generalised additive Model (GAM)

$$Y_i = \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \ldots + f_p(x_{ip}) + \epsilon_i.$$

Note: there is another class of statistical models called generalised linear models (GLMs), which include logistic regression. These are not the same!

There is no constraint that each f_j has to be the same type of function.
So f_1 could be a quadratic function, f_2 a smoothing spline function and f_3 a loess function.

As with linear models the predictor variables can be either numeric or factor variables.

How do we estimate each of the functions in the GAM?
$f_1(x_{i1})$ is fitted using univariate spline smoothing*
To now estimate $f_2(x_{i2})$ we need to subtract out the estimate of $f_1(x_{i1})$ before using spline smoothing.
To estimate $f_3(x_{i3})$ we need to subtract out the estimate of $f_1(x_{i1})$ and $f_2(x_{i2})$ before using spline smoothing.

We define the *k-th partial residuals* for a GAM model, which are the values obtained, once all but the k-th component function has been removed:

$$\eta_{ik} = y_i - \beta_0 - f_1(x_{i1}) - \ldots - f_{(k-1)}(x_{i(k-1)}) - f_{(k+1)}(x_{i(k+1)}) - \ldots f_p(x_{ip})$$

for $i = 1, \ldots, n$.

The method for obtaining the non-parametric functions f_k is called *backfitting*. An insight to the method is below, but the technical details are omitted.

*We will assume for convenience that each component function is a smoothing spline, but the algorithm easily generalises to any regression method.

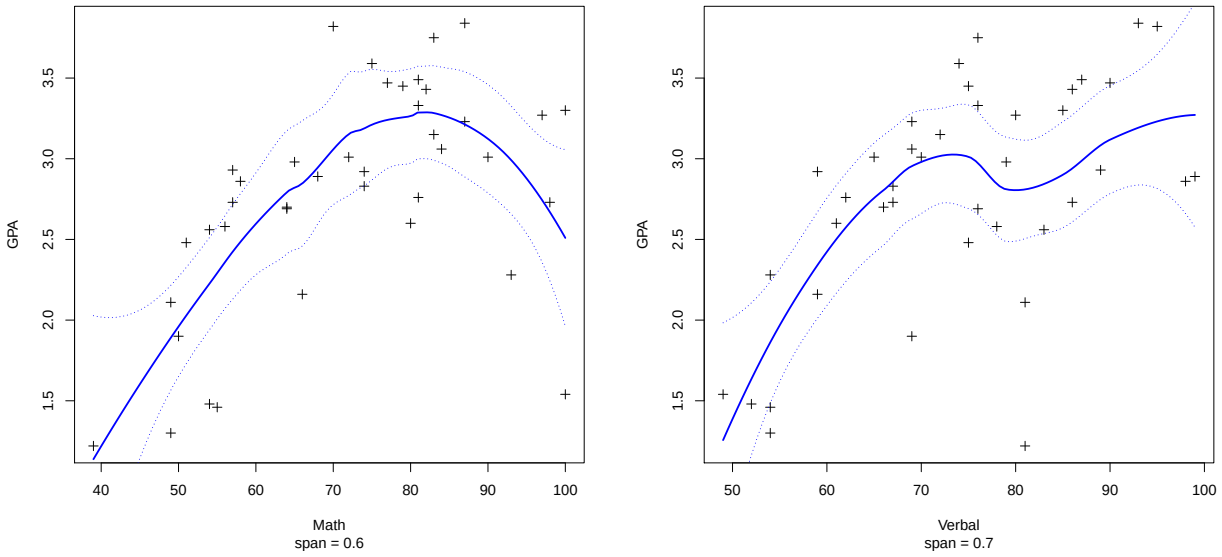
Backfitting algorithm for GAMs

1) Set the constant parameter $\widehat{\beta}_0 = \overline{y}$ this remains fixed throughout the algorithm.
2) Initialise all the component functions $f_k(x) = 0$
3) Iterate over $k \in \{1, \ldots, p\}$
   a) Obtain the k-th partial residuals η_{ik}.
   b) Obtain the spline smoothing function $f_k(x)$ using the η_{ik} partial residuals as the observed values.
4) Repeat Step 3 until convergence, i.e. the change in each of the components $f_k(x)$ is less than a specified tolerance.

Example

The GPA (general point average) for matriculating freshmen in US universities is the grade attained when completing the 1st year at university. GPA is a continuous number on a scale from 1 (bad) to 4 (very good). The data set `Freshmen` contains the GPA score along with the percentage scores for a mathematics test and a verbal test, as part of their entrance exam.

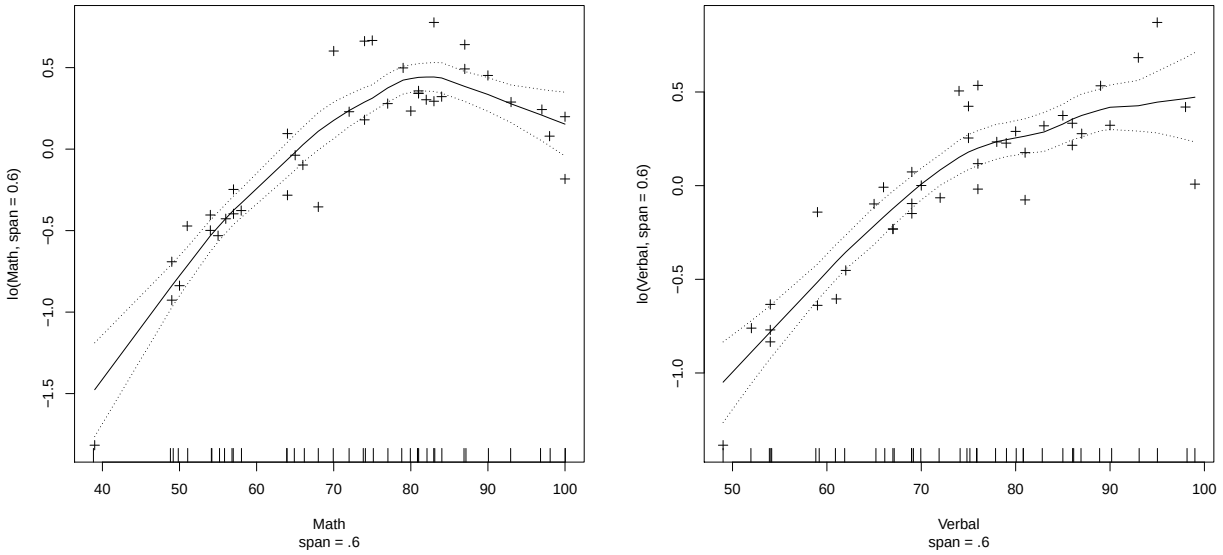
Loess curves



The local regression *GPA dependent on math score* ignoring the verbal score seems to be non-linear, as does *GPA dependent on verbal score* ignoring the math score.

GAM with loess curve for Math and Verbal

```
gam.mod=gam(GPA~lo(Math,span=0.6)+lo(Verbal,span=0.6),
        data=Freshmen)
```

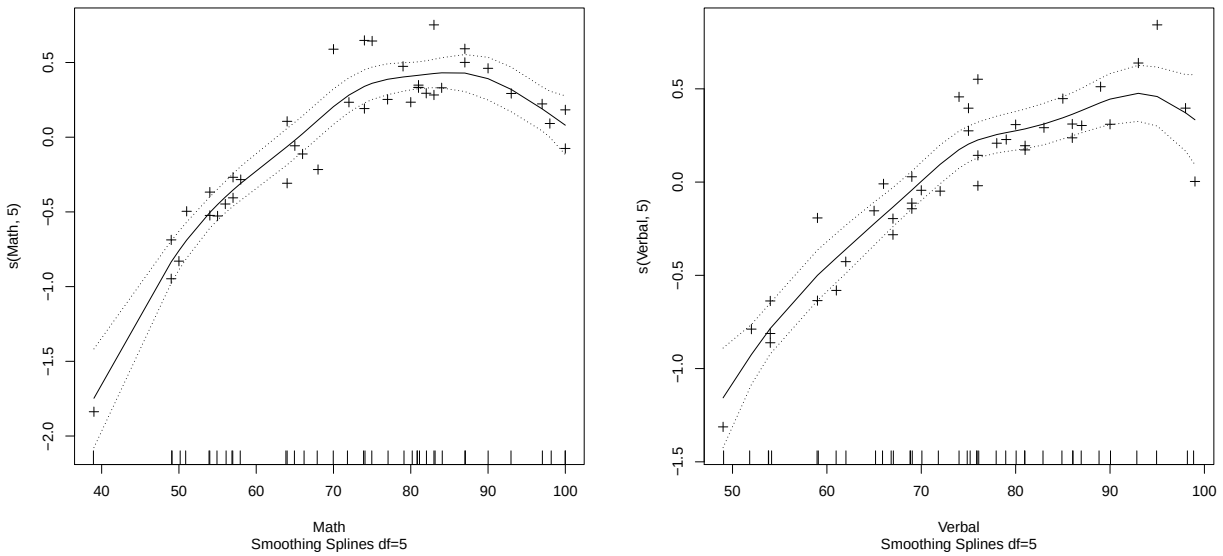


The dip in predicted GPA for verbal scores around 80 has disappeared when Math and Verbal are simultaneously fitted.

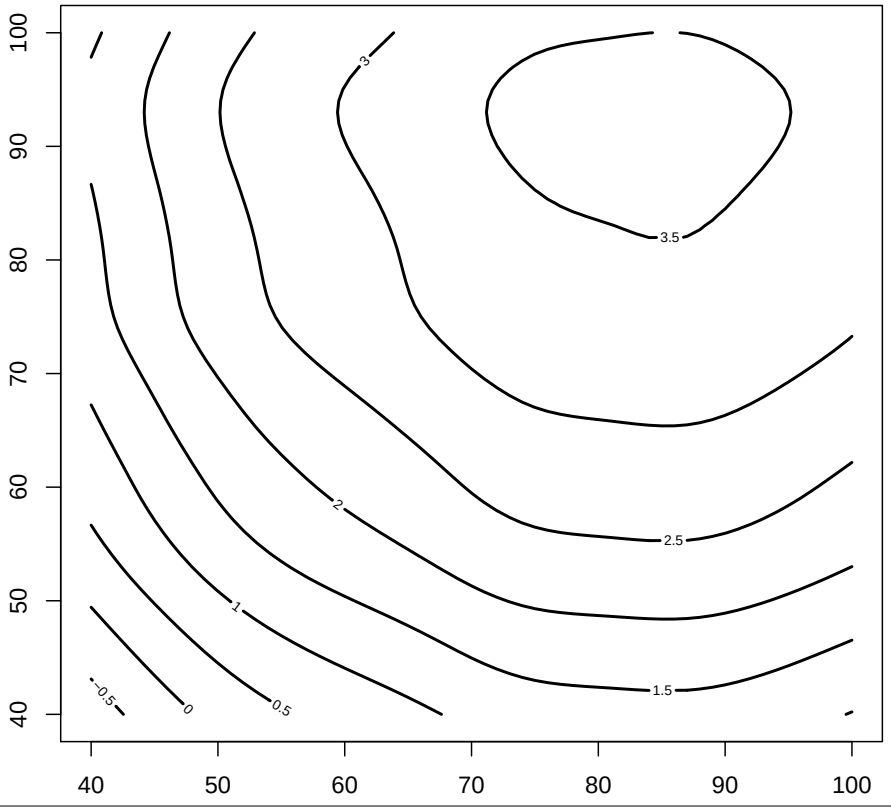
NB: the points plotted on the *y*-axis are the partial residuals and not GPA scores.

GAM with spline smoothing curve for Math and Verbal

```
gam.mod=gam(GPA~s(Math,5)+s(Verbal,5),data=Freshmen)
```



Contour plot of the predicted values for spline smoothing.



You can specify different combinations such as spline smoothing for Math and a quadratic polynomial for verbal scores

```
gam.mod=gam(GPA~s(Math,5)+poly(Verbal,2),data=Freshmen)
```

Comment: James et al. and other sources say when using spline bases (B-Splines or Natural splines), then `lm` should be used. This is not necessary, `gam` can also be used. The command

```
gam.mod=gam(GPA~ns(Math,5)+bs(Verbal,5),data=Freshmen)
```

works without error and with the same output as with `lm`.

Comments

- GAMs allow us to fit a non-linear predictor to each variable. We will find non-linear relationships that standard multiple linear regression will miss. We do not need to manually try out many different transformations on each variable individually.
- The non-linear fits can potentially make more accurate predictions for the response Y .
- Because the model is additive, we can examine the effect of each X_j on Y individually while holding all of the other variables fixed.
- The smoothness of the function f_j for the variable X_j can be controlled independently for each variable.
- A GAM is restricted to be additive. Important interactions might be missed. We can manually add interaction terms to the GAM model by including additional predictors for $X_j \cdot X_k$.