Generalized Additive Models

Nonparametric methods like kernel regression and smoothing splines become difficult to use and do not work well for p greater than about 3.

For tractability, we need to restrict the ways in which the various predictors interact.

A natural assumption is that their effects are **additive**.

For regression, a **generalized additive model**, or **GAM**, has the form

$$\mathrm{E}(y \mid \boldsymbol{x}) = \alpha + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p), \tag{1}$$

where the f_j are smooth nonparametric functions.

The $f_j(x_j)$ could be either kernel regressions or smoothing splines, but they could also be much simpler things, such as dummy variables.

Thus a GAM lets us combine fixed effect dummies with nonlinear function(s) for the variable(s) of primary interest.

The easiest approach is simply to replace each of the additive functions by something that is linear in functions of x_j.

These could include:

- polynomials created by `poly()`;
- regression splines created by `bs()`;
- natural cubic splines created by `ns()`;
- step functions created by `cut()`; or
- categorical (dummy) variables created by `factor()`.

Economists like to use dummy variables, often a great many of them.

Combining them with polynomials or splines is fast and easy. We can just use `lm()` or `glm()` together with `factor()`.

We can create the basis functions on the fly, within the estimation command, or use them first to create matrices of basis functions. This may be desirable if we will use them several times.

As an example, let us see how we can model the wage data using natural cubic splines.

The four plots in Figure 14.1 were created using the commands

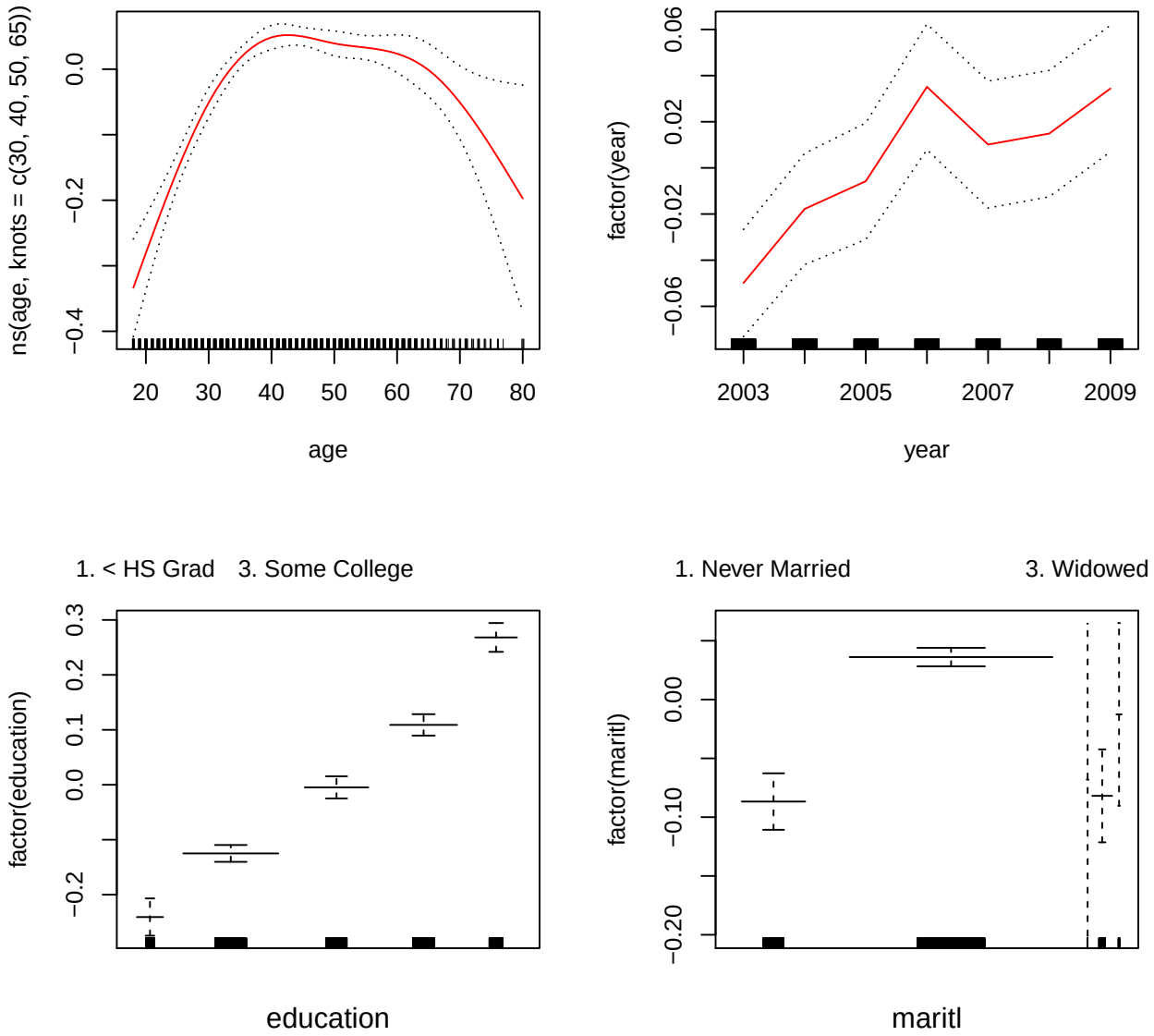
```
sfit <- lm(lw ~ ns(age,knots = c(30,40,50,65))
+ factor(year) + factor(education) + factor(maritl))
par(mfrow=c(2,2))
plot.Gam(sfit, se=TRUE, col='red')
```

Note that I had to use `plot.Gam` instead of `plot.gam`.

This works even though the estimation was done using `lm()` rather than `gam()`, as long as the `gam` library is loaded.

`summary(sfit)` provides estimates, along with (ordinary) standard errors, t statistics, and P values.

Figure 14.1 — Natural Splines + Factors for Log Earnings



Instead of (natural) cubic splines, we can combine dummy variables with kernel regression or smoothing splines. The latter is cheaper and easier in R.

For smoothing splines, we use the `gam()` function from the `gam` library.

The four plots in Figure 14.2 were created using the commands

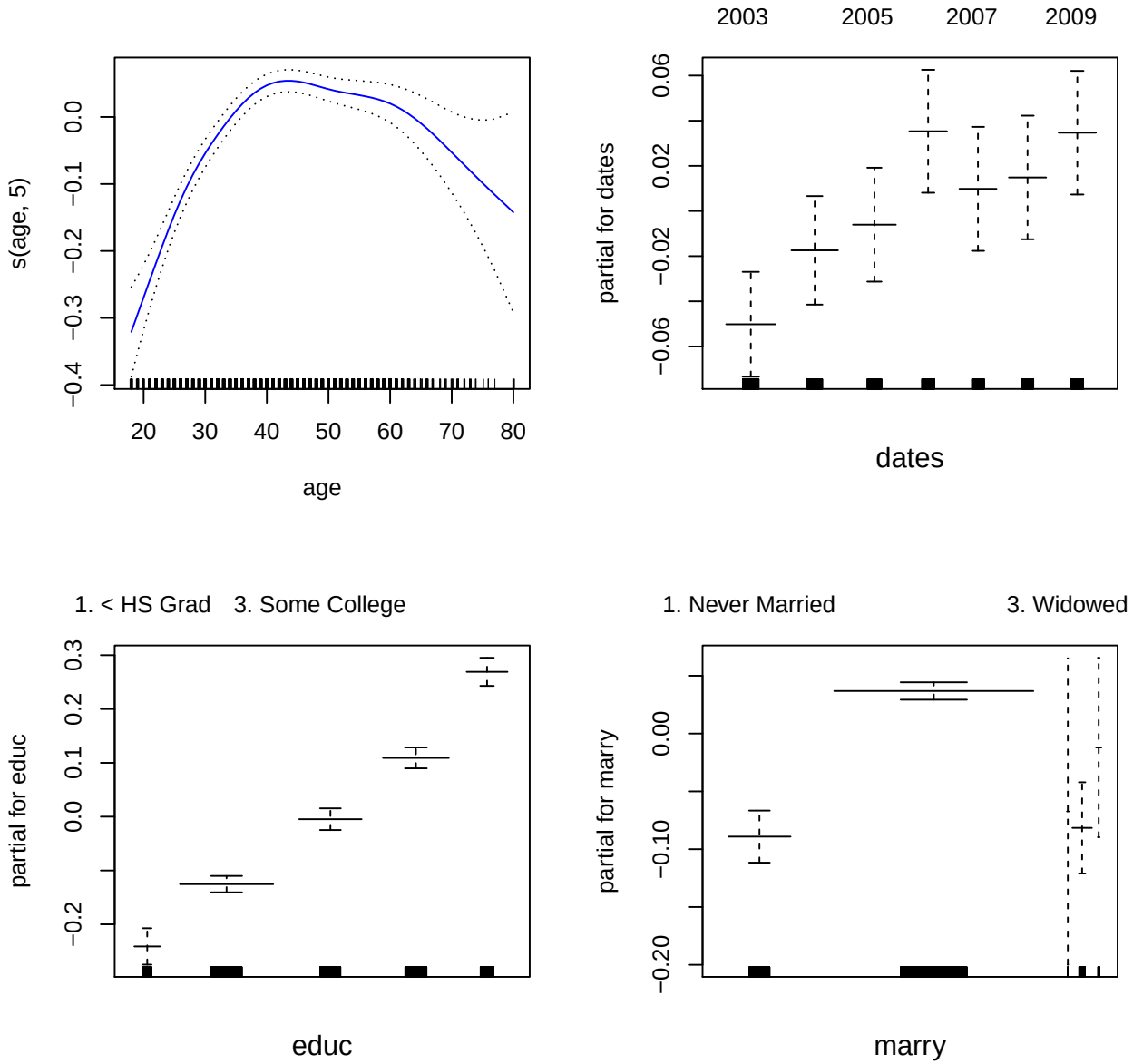
```
gfit <- gam(lw s(age,5)+dates+educ+marry)
par(mfrow=c(2,2))
plot(gfit,se=TRUE, col='blue')
```

This is actually a little easier than using natural cubic splines, because we do not have to choose the number and locations of the knots.

We also do not explicitly have to tell `gam` to treat the other variables as factors, since they all take integer values.

However, we do have to specify a smoothing parameter, which in this case is the "equivalent degrees of freedom" which equals 5. We could also specify `spar` between 0 and 1.

Figure 14.2 — Smoothing Splines + Factors for Log Earnings



Local Regression

There is a short section (7.6) in ISLR/ISLP on **local regression**, which is very similar to kernel regression.

The main difference is that it uses neither the Gaussian nor the Epanechnikov kernel. Instead it uses **tricubic weighting**, which (optionally) sets many of the weights to 0.

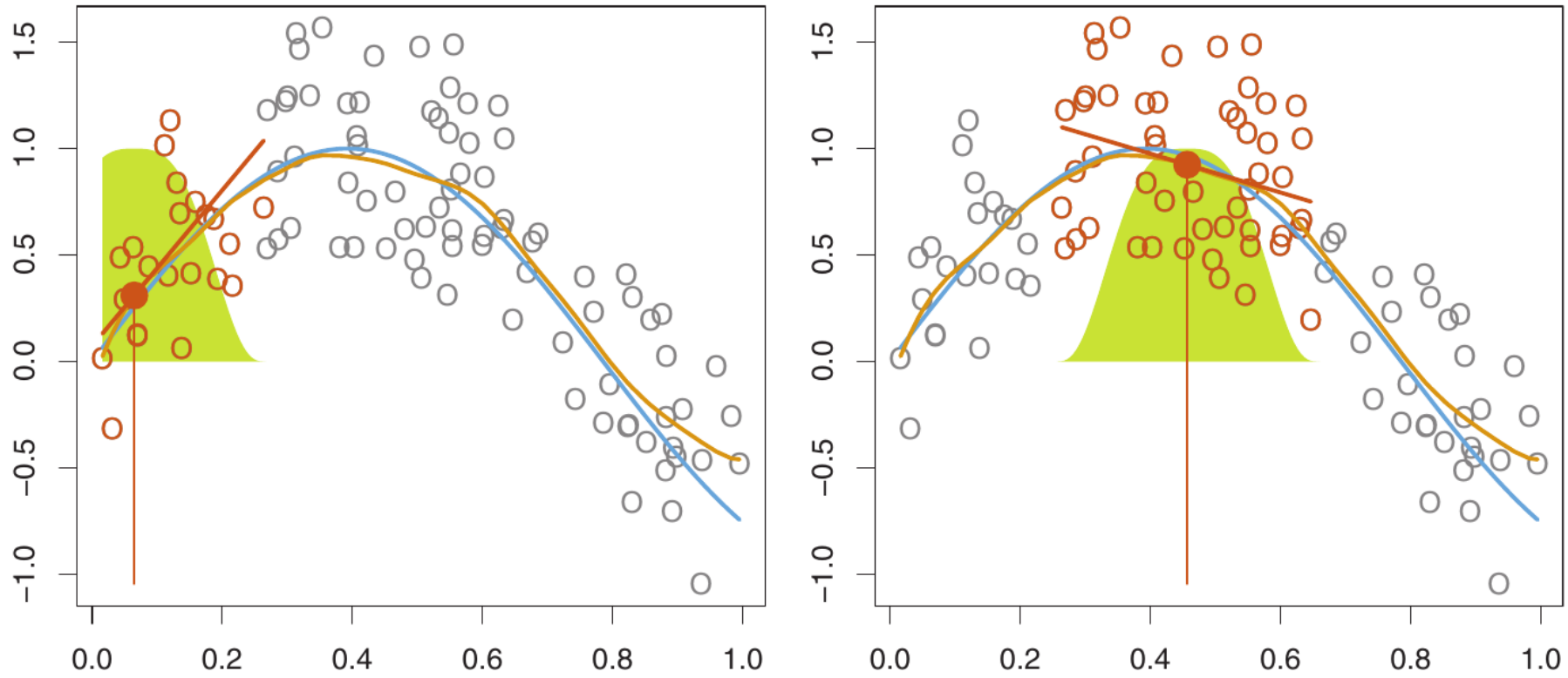
The `loess()` function from the `loess` package performs local regression, either linear or quadratic.

It can also perform locally constant regression, but the documentation says to avoid it. This is good advice!

The `lo()` function can be used to include local regression terms within a GAM, instead of smoothing splines or regression splines.

The `lo()` function can take two or variables as arguments, so that a model can be additive in most variables but fully nonlinear in a few.

Figure 14.3 — Local Regression



The coloured circles are the observations that get positive weight in each of the local regressions. The solid dot is the fit $\hat{f}(x_0)$. The orange curve shows all the fitted values as x_0 is varied.

Estimating GAMs

When everything we are regressing on can be computed in advance, we can just estimate a regression GAM by OLS.

The r.h.s. variables might include continuous regressors, categorical regressors, or basis functions like polynomials, step functions, and regression splines.

All of these can easily be handled by the `lm()` function.

If the output is binary, we can replace (1) by

$$\log\left(\frac{p(\boldsymbol{x})}{1-p(\boldsymbol{x})}\right) = \alpha + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p), \tag{2}$$

which can be estimated by maximum likelihood using `glm()`.

In the parametric case, each of the $f_j(x_j)$ implicitly includes one or more coefficients that can be estimated linearly.

In many cases, especially for preliminary investigations, a parametric version of (1) or (2) is sufficient.

But when the $f_j(x_j)$ include smoothing splines or local regression terms, estimation gets trickier. The method of choice is **backfitting**.

Suppose we are estimating a GAM that is partly linear and also involves two one-dimensional smoothing splines.

We could write such a GAM as

$$y_i = \beta_0 + \sum_{j=1}^{J} \beta_j x_{ji} + f_a(x_{ai}) + f_b(x_{bi}) + u_i, \tag{3}$$

where x_a and x_b are just x_{J+1} and x_{J+2}. Of course, GAMs can be much more general than (3).

The model (3) cannot be estimated by OLS, since we cannot turn $f_a(x_{ai})$ and $f_b(x_{bi})$ into sums of coefficients times basis functions.

The model (3) is not identified without imposing some restrictions.

We could, for example, add a constant to $f_a(x_{ai})$, subtract the same constant from $f_b(x_{bi})$, and obtain exactly the same fit.

It is therefore common to impose the constraints that

$$\sum_{i=1}^{n} f_a(x_{ai}) = 0, \quad \text{and} \quad \sum_{i=1}^{n} f_b(x_{bi}) = 0. \tag{4}$$

The backfitting algorithm works as follows:

Pretend that $f_a(x_{ai})$ and $f_b(x_{bi})$ are zero and regress y_i on a constant and the x_{ji}. This yields **partial residuals** $\tilde{u}_i^{(1)} = y_i - \tilde{\beta}_0 - \sum_{j=1}^{J} \tilde{\beta}_j x_{ji}$.

Next, estimate the model

$$\tilde{u}_i^{(1)} = f_a(x_{ai}) \tag{5}$$

using a smoothing-spline procedure to obtain $\tilde{f}_a^{(1)}(x_{ai})$.

This yields partial residuals

$$\tilde{u}_i^{(2)} = \tilde{u}_i^{(1)} - \tilde{f}_a^{(1)}(x_{ai}). \tag{6}$$

Use these to estimate the model

$$\tilde{u}_i^{(2)} = f_b(x_{bi}), \tag{7}$$

so as to obtain partial residuals $\tilde{u}_i^{(3)} = \tilde{u}_i^{(2)} - \tilde{f}_b^{(2)}(x_{bi})$.

- Optionally, we could regress $\tilde{u}_i^{(2)}$ on the x_{ji} before performing the second smoothing spline procedure in (7).
- This would yield a different vector of partial residuals, which could be called $\tilde{u}_i^{(2')}$.
- The next step would be the second smoothing spline procedure, (7), but with $\tilde{u}_i^{(2')}$ instead of $\tilde{u}_i^{(2)}$ as the regressand.

Linear regression is cheap. Doing two linear regressions for each pair of smoothing splines, instead of just one, may yield faster convergence.

Once we have the partial residuals based on both the ordinary regressors and the two smoothing splines, we start all over again.

ESL suggests that it is computationally desirable to renormalize the $\tilde{f}_a(x_{bi})$ and $\tilde{f}_b(x_{bi})$ at each step to have mean zero.

The backfitting procedure stops when changes between successive iterations are sufficiently small.

- Backfitting is very widely applicable. It is not limited to smoothing splines and linear regressions.
- It should work well whenever it involves one-dimensional or perhaps two-dimensional smoothers, and the additivity assumption is a reasonably good one.

We can easily estimate a logistic regression that involves either regression splines or smoothing splines.

This involves nonlinear estimation for every backfit, so it may be costly if the procedure does not converge quickly.

There is a literature in econometrics on **partially linear regression**. The model it concerns is very similar to (3):

$$y_i = \beta_0 + \sum_{j=1}^{J} \beta_j x_{ji} + f(x_{ai}, x_{bi}) + u_i. \tag{8}$$

This model is linear in x_1 through x_J, fully nonparametric in x_a and x_b, and additive between the linear and nonlinear parts.

- The partially linear regression model (8) is just a very particular GAM. The nonlinear part often involves just one input, say $f(x_{ai})$.
- In the econometric literature, the nonlinear part $f(x_{ai}, x_{bi})$ has traditionally been estimated by kernel regression.

For **double machine learning**, $f(\cdot)$ is generally estimated by lasso or random forests, so there can be many inputs.

GAMs may or may not include linear terms. As in (8), the nonlinear terms may depend on two or more predictors.

Because of their computational advantages, it seems much better to use smoothing splines rather than kernel regressions in GAMs.

- GAMs seem to be under-used in applied econometrics.
- They work fine with large numbers of dummy variables (fixed effects), which are almost universal in microeconometrics.
- If we stick to one-dimensional regression splines or smoothing splines, they are not difficult or expensive to estimate.
- Because the model is additive, the effect of each x_j on y can readily be plotted, holding all other variables fixed.
- By default, there will be no interaction terms, so we may miss important interactions.
- But we can add low-dimensional interaction functions if we simply replace, say, $f_1(x_1) + f_2(x_2)$ by $f_a(x_1, x_2)$.

Backfitting, or methods similar to it, can be used in many contexts.

One problem is that, if there are multiple local minima of the objective function, backfitting may get us to the wrong one.

Models with many parameters (e.g. neural networks) typically use a more general procedure called **gradient descent**, specifically **stochastic gradient descent**.

Methods based on gradient descent will be discussed in the context of neural networks.

The basic idea is to take a number of downhill steps whenever we are minimizing something.

If these converge, they should converge to a local minimum.

An Application to Spam Detection

Section 9.1.2 of ESL estimates a GAM where the dependent variable is whether or not an email is spam. They use a logistic GAM.

- There are 4601 observations, which were divided into 3065 training observations and 1536 test observations.
- There are 57 predictors, most of them the percentages of words that match certain given words, like "business" and "free."
- There are also measures of the numbers and lengths of sequences of capital letters.

With so many inputs, it might seem natural to use lasso or elastic net, but that would not allow for nonlinear effects.

No effort was made to calibrate each of 57 smoothing splines.

Instead, they picked the smoothing parameter so that $\text{Tr}(\boldsymbol{S}_j(\lambda_j)) - 1 = 4$ for all j. In other words, edf $= 5$.

They also transformed each input as $\log(x + 0.1)$.

This transformation probably matters a lot, because many (all?) inputs have some 0 values, often a lot of them.

Moreover, they find that there is a lot of nonlinearity near 0.

In principle, we should not need a log transformation when using a smoothing spline, but removing nonlinearity via the transformation may make it easier for the splines to fit well.

In many cases, there is a lot of nonlinearity near 0. So using, say, $\log(x + 1)$ might have made a difference.

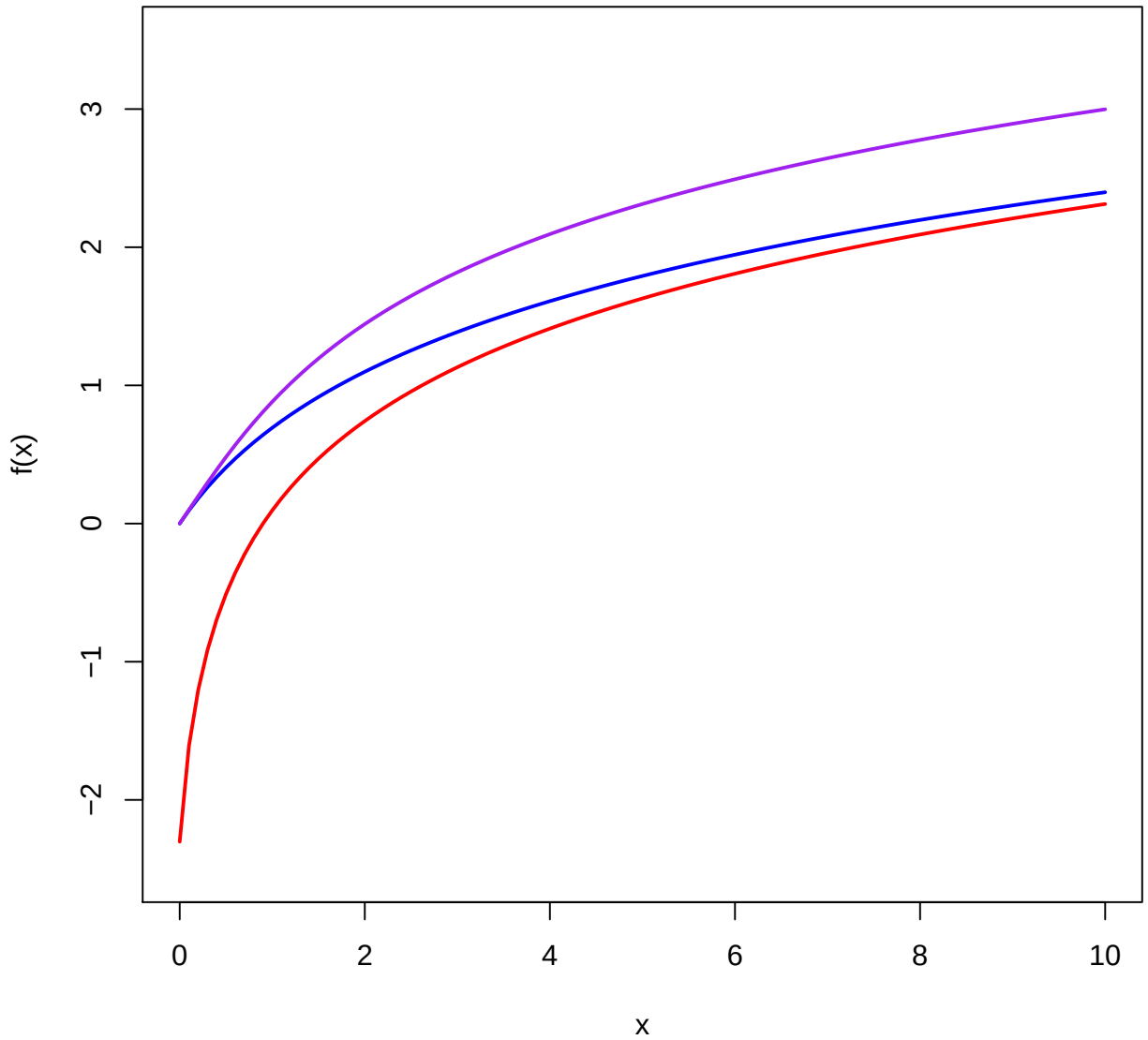
Another possibility is the **inverse hyperbolic sine** function,

$$h(x) = \sinh^{-1}(x) = \log\left(x + (x^2 + 1)^{1/2}\right); \tag{9}$$

see MacKinnon and Magee (1990).

Figure 14.4 shows $\log(x + 0.1)$ in red, $\log(x + 1)$ in blue, and the IHS transformation in purple.

Figure 14.4 — Three Transformations



The estimated $f(x_j)$ functions are shown for 16 inputs in Figure 14.5.

Note that there is no log transformation here.

There is a lot of nonlinearity for some of the inputs.

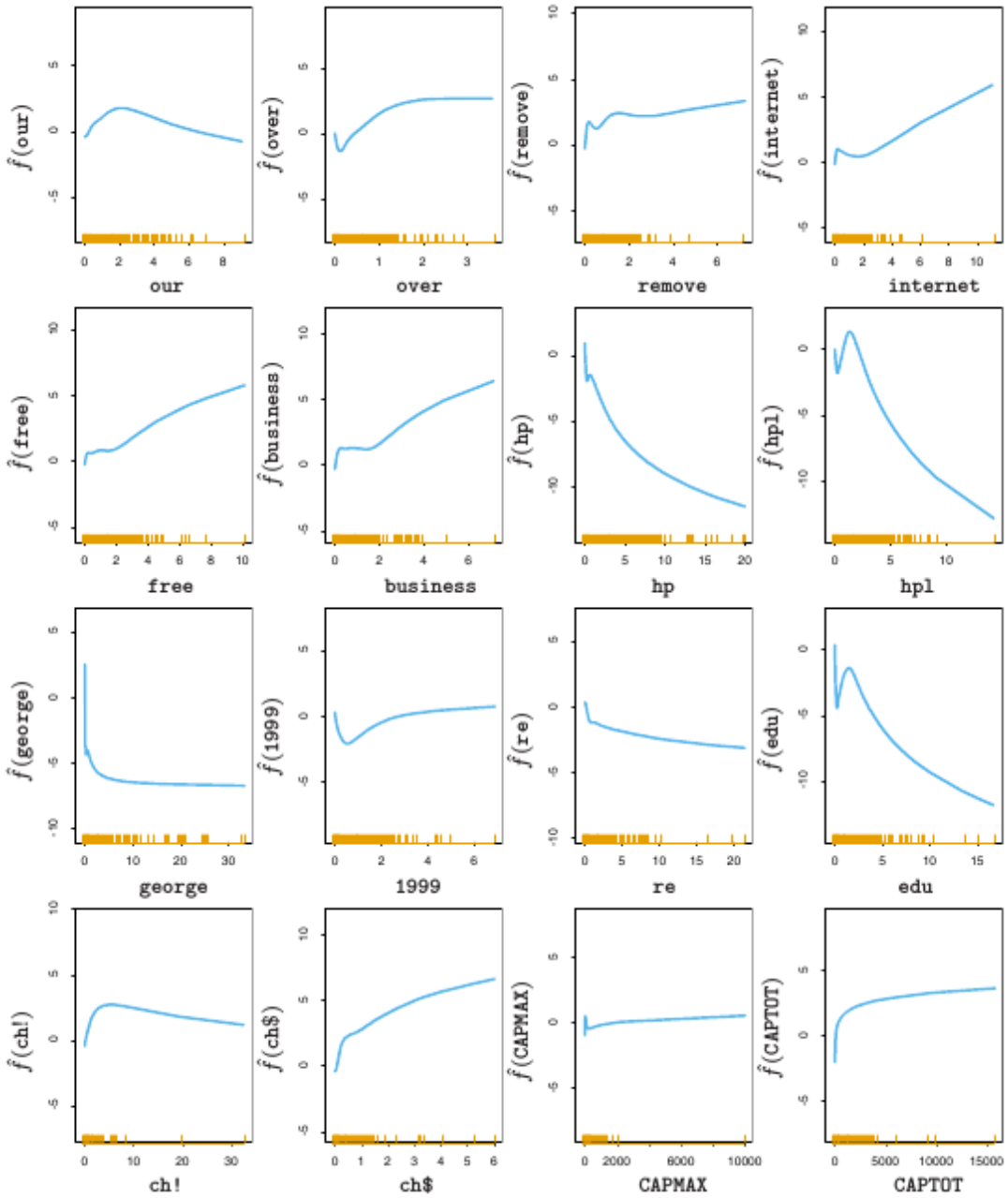
The model works pretty well. The test data confusion matrix is

Table 1: **Confusion Matrix for Spam Example**

Prediction/Outcome	Genuine	Spam
Genuine	58.3	3.0
Spam	2.5	36.3

The total error rate is just 5.5%.

However, $2.5/(58.3 + 2.5) = 4.1\%$ of genuine emails are classified as spam. This can be reduced in various ways, at the cost of classifying more spam as genuine.


$\hat{f}$(our)
our
$\hat{f}$(over)
over
$\hat{f}$(remove)
remove
$\hat{f}$(internet)
internet
$\hat{f}$(free)
free
$\hat{f}$(business)
business
$\hat{f}$(hp)
hp
$\hat{f}$(hpl)
hpl
$\hat{f}$(george)
george
$\hat{f}$(1999)
1999
$\hat{f}$(re)
re
$\hat{f}$(edu)
edu
$\hat{f}$(ch!)
ch!
$\hat{f}$(ch$)
ch$
$\hat{f}$(CAPMAX)
CAPMAX
$\hat{f}$(CAPTOT)
CAPTOT