STA 414/2104

Statistical Methods for Machine Learning and Data Mining

Radford M. Neal, University of Toronto, 2012

Week 2

Modeling Data with Linear Combinations of Basis Functions

A Type of Supervised Learning Problem

We want to model data $(x_1, y_1), \ldots, (x_n, y_n)$, where x_i is a vector of p inputs (predictors) for case i, and y_i is the target (response) variable for case i, which is real-valued.

We are trying to predict y from x, for some future test case, but we are not trying to model the distribution of x.

Suppose also that we don't expect the best predictor for y to be a linear function of x, so ordinary linear regression on the original variables won't work well.

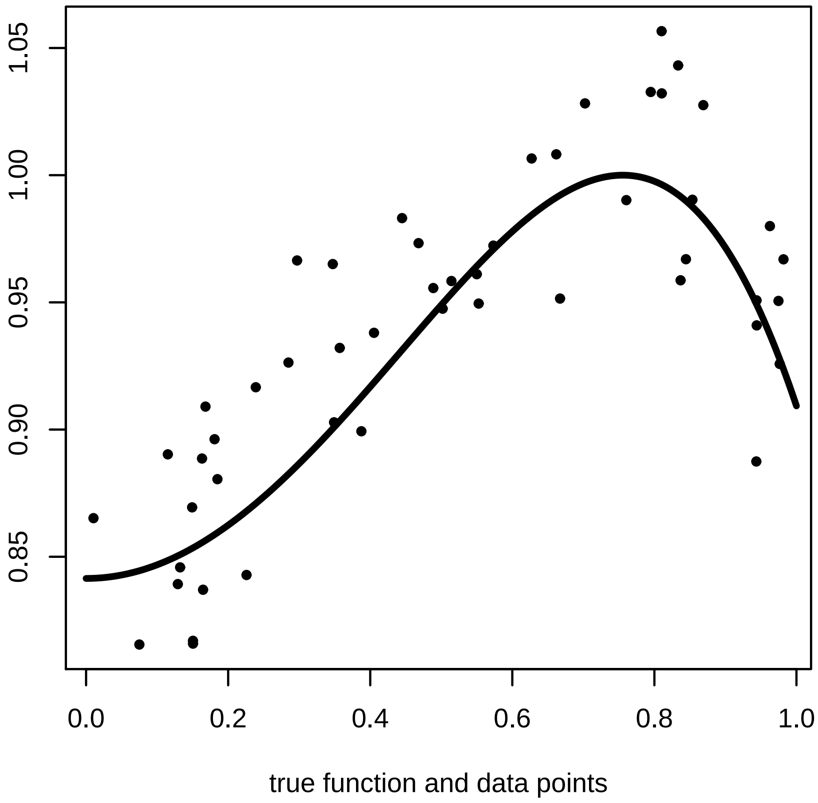
We need to allow for a non-linear function of x, but we don't have any theory that says what form this function should take. What to do?

An Example Problem

As an illustration, we can use the synthetic data set looked at previously — 50 points generated with x uniform from $(0, 1)$ and y set by the formula:

$$y \quad = \quad \sin(1 + x^2) \; + \; \text{noise}$$

where the noise has $N(0, 0.03^2)$ distribution.



The noise-free true function, $\sin(1 + x^2)$, is shown by the line.

This is simpler than real machine learning problems, but lets us look at plots...

Linear Basis Function Models

We earlier looked at fitting this data by least-squares linear regression, using not just x, but also x^2, x^3, etc., up to (say) x^4 as predictors.

This is an example of a *linear basis function model.*

In general, we do linear regression of y on $\phi_1(x)$, $\phi_2(x)$, $\ldots$, $\phi_{m-1}(x)$, where the ϕ_j are *basis functions*, that we have selected to allow for a non-linear function of x.

This gives the following model:

$$\begin{aligned} y &= f(x, \beta) \;+\; \text{noise} \\ f(x, \beta) &= \beta_0 \;+\; \sum_{j=1}^{m-1} \beta_j \phi_j(x) \;=\; \beta^T \phi(x) \end{aligned}$$

where β is the vector of all m regression coefficients (including the intercept, β_0) and $\phi(x)$ is the vector of all basis function values at input x, including $\phi_0(x) = 1$ for the intercept.

Maximum Likelihood Estimation

Suppose we assume that the noise in the regression model is Gaussian (normal) with mean zero and some variance σ^2.

With this assumption, we can write down the *likelihood function* for the parameters β and σ, which is the joint probability density of all the targets in the training set as a function of β and σ:

$$\begin{aligned} L(\beta, \sigma) &= P(y_1, \ldots, y_n | x_1, \ldots, x_n, \beta, \sigma) \\ &= \prod_{i=1}^{n} N(y_i | \beta^T \phi(x_i), \sigma^2) \end{aligned}$$

where $N(y|\mu, \sigma^2)$ is the density for y under a normal distribution with mean μ and variance σ^2.

The maximum likelihood estimates for the parameters are the values of β and σ that maximize this likelihood. Equivalently, they maximize the log likelihood, which, ignoring terms that don't depend on β or σ, is

$$\log L(\beta, \sigma) \;=\; -n \log(\sigma) \;-\; \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \beta^T \phi(x_i))^2$$

Least Squares Estimation

From this log likelihood function, we see that regardless of what σ might be, the maximum likelihood estimate of β is the value that minimizes the sum of squared prediction errors over training cases.

Let's put the values of all basis functions in all training cases into an $n \times m$ matrix Φ, with $\Phi_{ij} = \phi_j(x_i)$. We also put the target values for all training cases into a vector y.

We can now write the sum of squared errors on training cases as

$$||y - \Phi\beta||^2 \;=\; (y - \Phi\beta)^T(y - \Phi\beta)$$

This is minimized for the value of β where its gradient is zero, which is where

$$-2\Phi^T(y - \Phi\beta) = 0$$

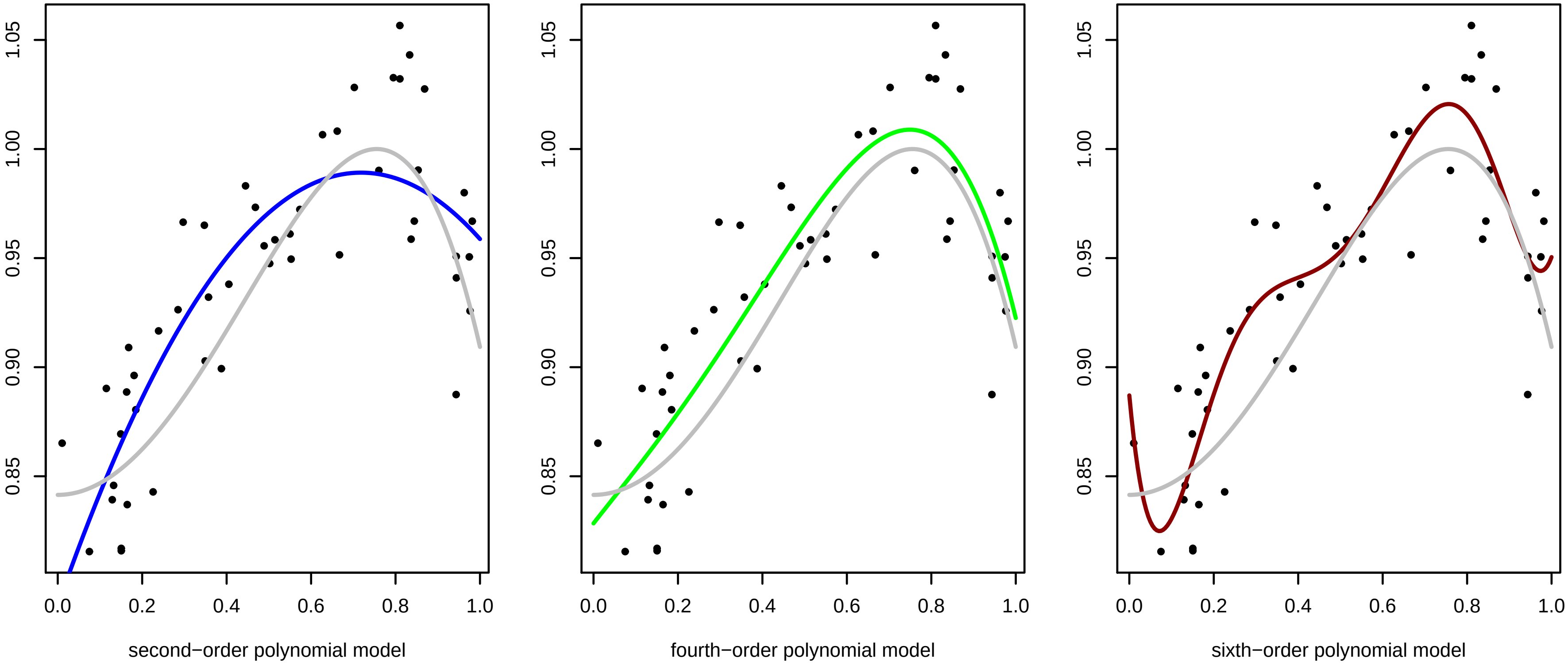
Solving this, the least squares estimate of β is

$$\widehat{\beta} \;=\; (\Phi^T\Phi)^{-1}\Phi^T y$$

This assumes that $\Phi^T\Phi$ is non-singular, so that there is a unique solution. When m is greater than n, this will not be the case!

Results with Polynomial Basis Functions

Recall we looked before at least-squares fits of polynomial models with increasing order, which can be viewed as basis function models with $\phi_j(x) = x^j$.



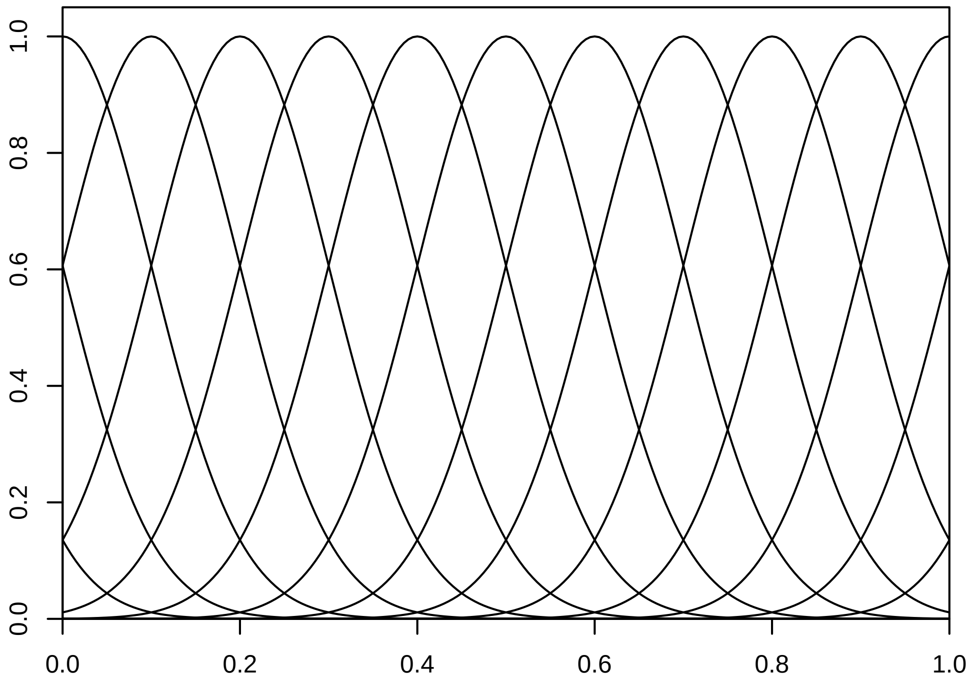
The gray line is the true noise-free function. We see that a second-order fit is too simple, but a sixth-order fit is too complex, producing "overfitting".

Gaussian Basis Functions

Polynomials are *global* basis functions, each affecting the prediction over the whole input space. Often, *local* basis functions are more appropriate. One possibility is to use functions proportional to Gaussian probability densities:

$$\phi_j(x) \;\; = \;\; \exp(-(x-\mu_j)^2/2s^2)$$

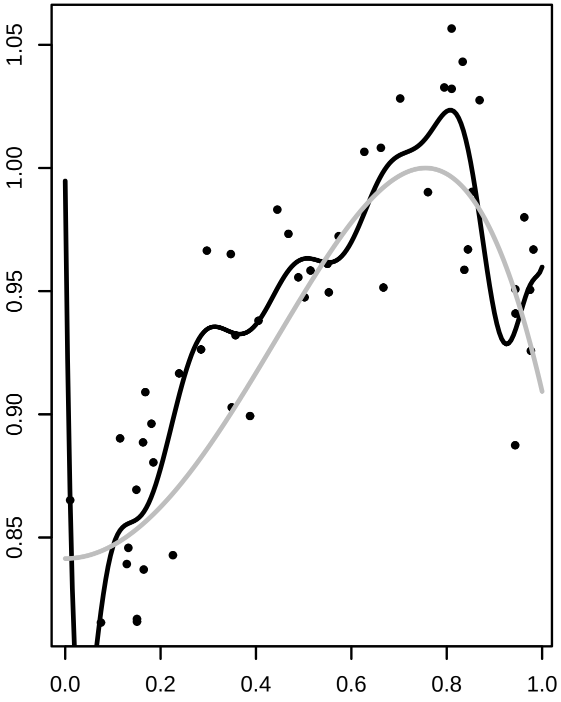
Here are these basis functions for $s = 0.1$, with the μ_j on a grid with spacing s:



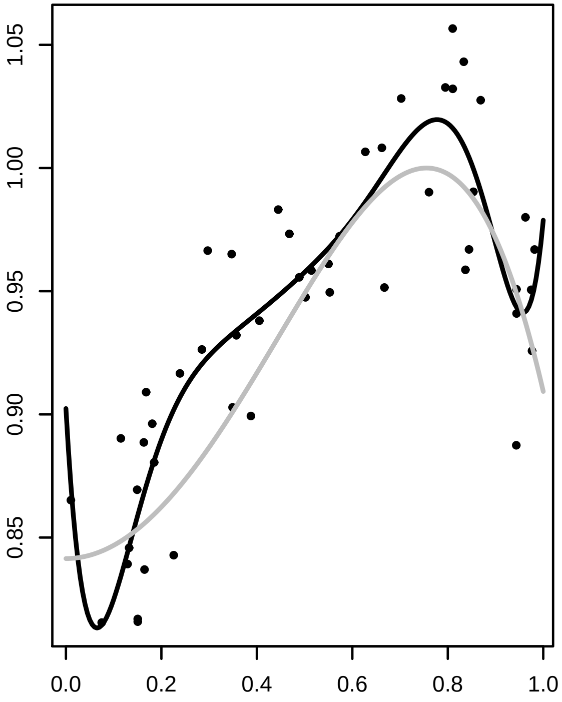
Gaussian basis functions, s = 0.1

Results with Gaussian Basis Functions

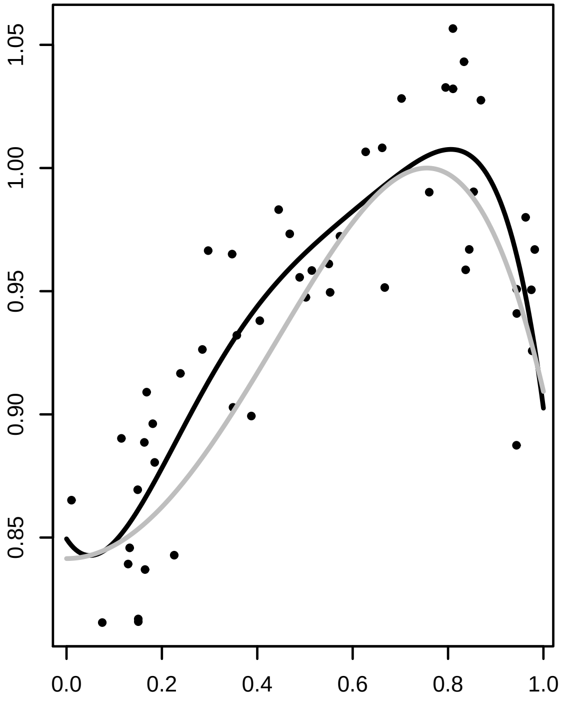
Here are the results using Gaussian basis functions (plus $\phi_0(x) = 1$) on the example dataset, with varying width (and spacing) s:



Gaussian basis function fit, s = 0.1



Gaussian basis function fit, s = 0.5



Gaussian basis function fit, s = 2.5

The estimated values for the β_j are not what you might guess. For the middle model above, they are as follows:

6856.5 -3544.1 -2473.7 -2859.8 -2637.7 -2861.5 -2468.0 -3558.4

Regularized Linear Basis Function Models

Maximum Penalized Likelihood Estimation

We can try to avoid the poor results of maximum likelihood when there are many parameters by adding a *penalty* to the log likelihood, that favours non-extreme values for the parameters. This procedure is also called *regularization*

For regression with Gaussian noise, we minimize the sum of squared errors on training cases plus this penalty.

Quadratic penalties are easiest to implement, as they combine with the squared error to still allow solution by matrix operations. For basis function models, we might use a penalty that encourages all β_j (except β_0) to be close to zero:

$$\lambda \sum_{j=1}^{m-1} \beta_j^2$$

Here, λ controls the strength of the penalty, which we must somehow decide on. (One could include β_0^2 in the penalty, but this usually doesn't make sense, since we usually don't want to bias the intercept to be near zero.)

Solution for Penalized Least Squares

We found the least squares solution before by equating the gradient of the squared error to zero. Now we add the gradient of the penalty function as well, and hence solve

$$2\lambda\beta^* \; - \; 2\Phi^T(y - \Phi w) = 0$$

where β^* is equal to β except that β_0 is zero.

Solving this, the penalized least squares estimate of β is

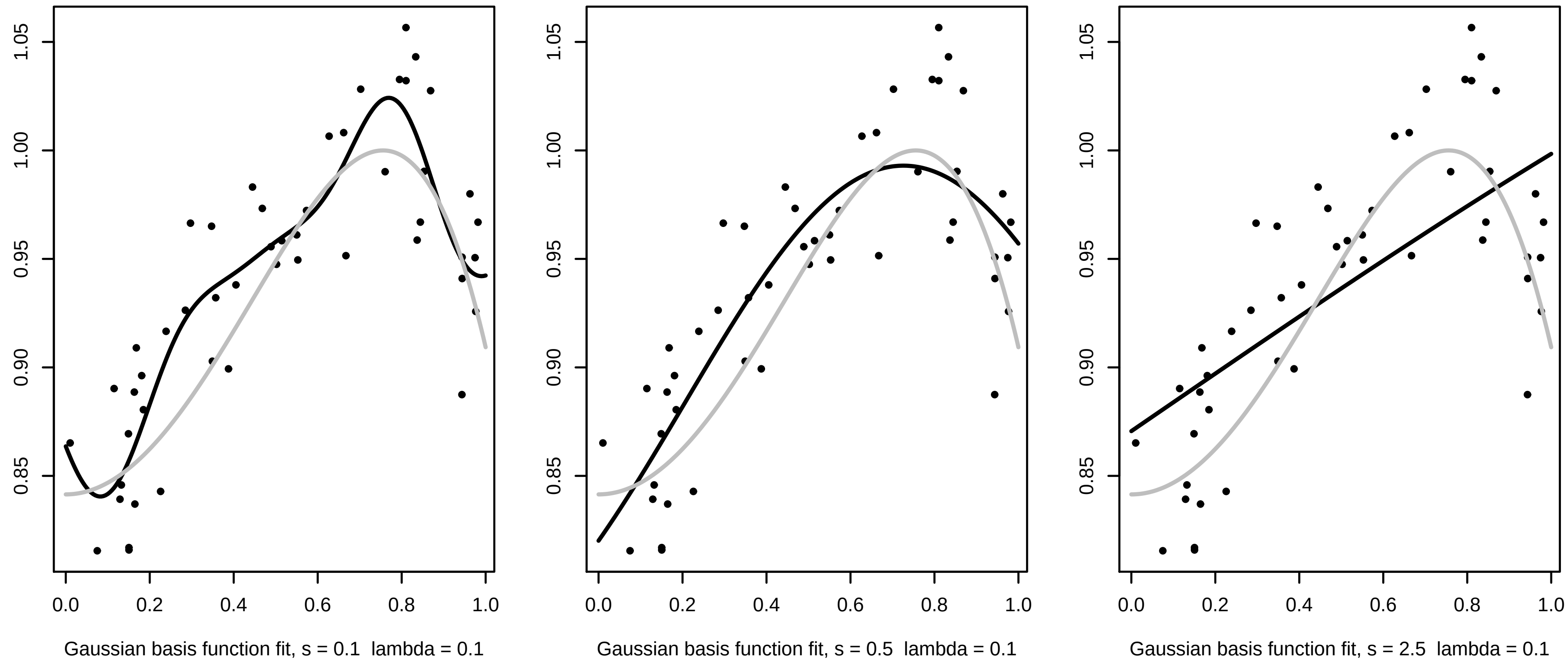
$$\widehat{\beta} \;\; = \;\; (\lambda I^* + \Phi^T\Phi)^{-1}\Phi^T y$$

where I^* is like the identity matrix except that $I^*_{1,1} = 0$.

Note that this estimate will be uniquely defined regardless of how big m and n are, as long as λ is greater than zero.

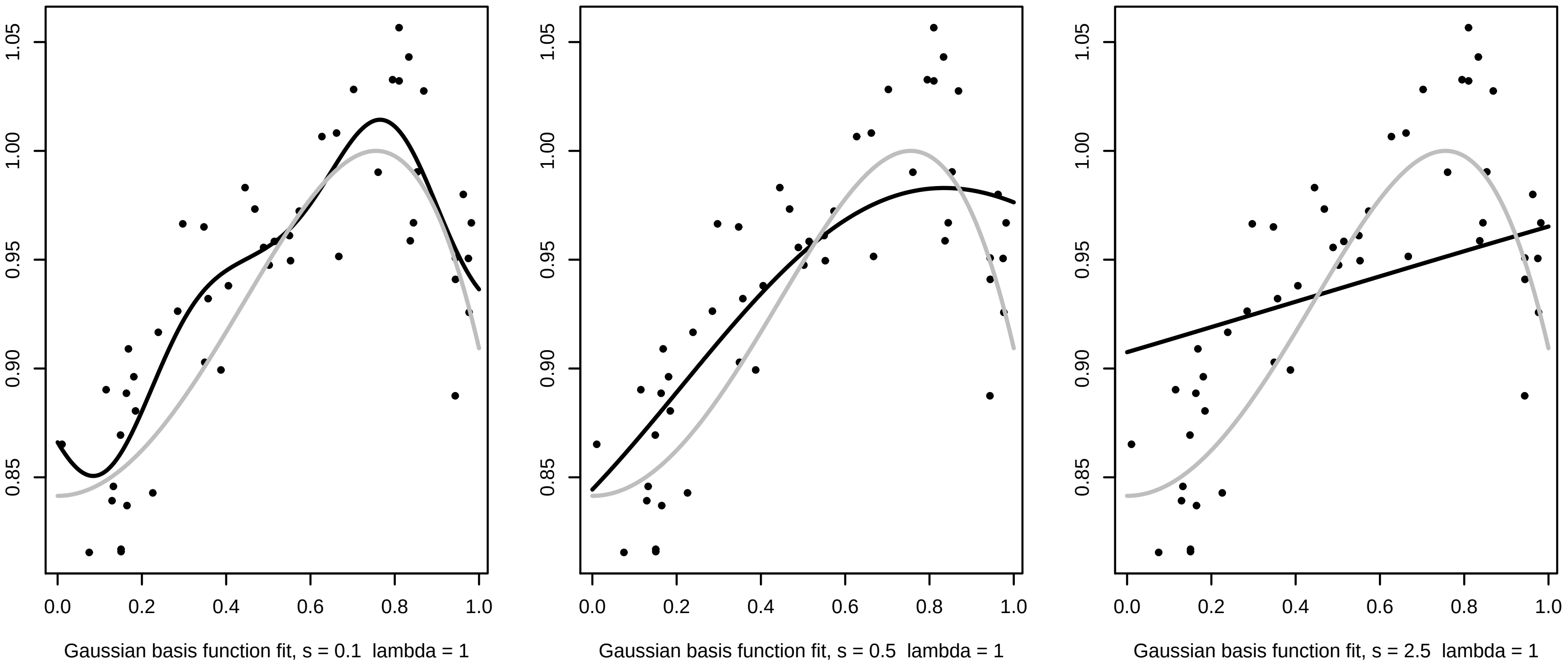
Results with Regularized Gaussian Basis Functions

Here are the results with $\lambda = 0.1$:



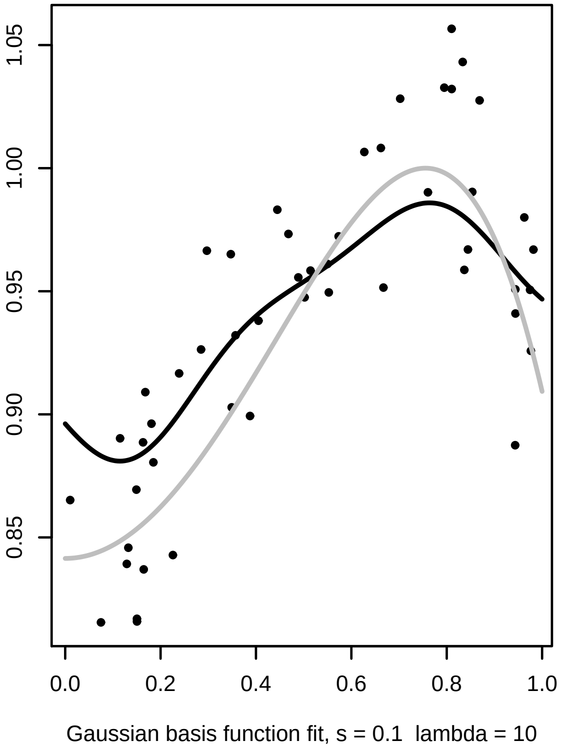
More Results with Regularized Gaussian Basis Functions

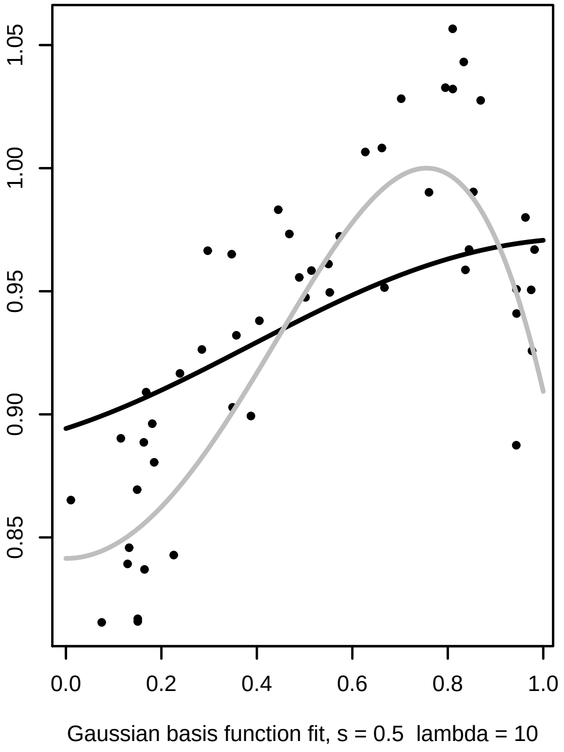
Here are the results with $\lambda = 1$:

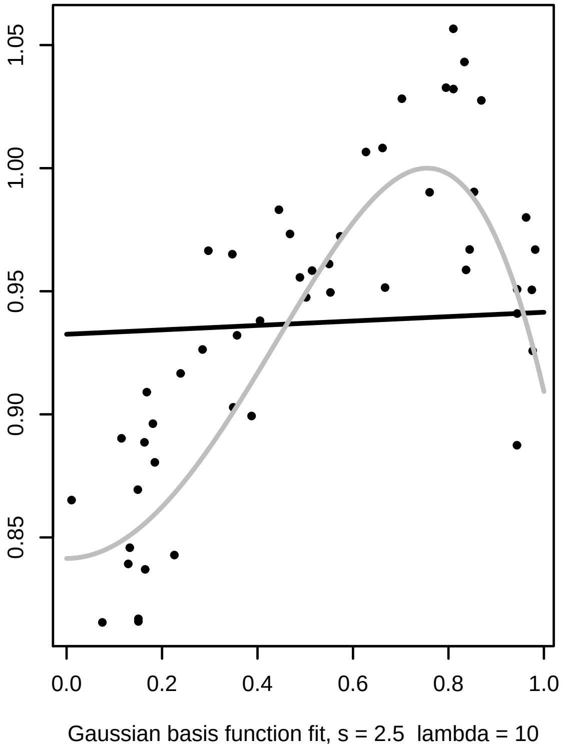


Yet More Results with Regularized Gaussian Basis Functions

Here are the results with $\lambda = 10$:

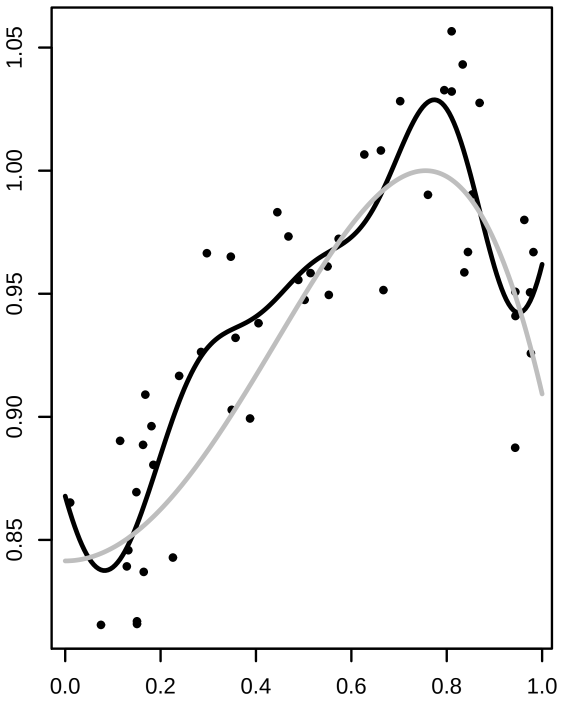




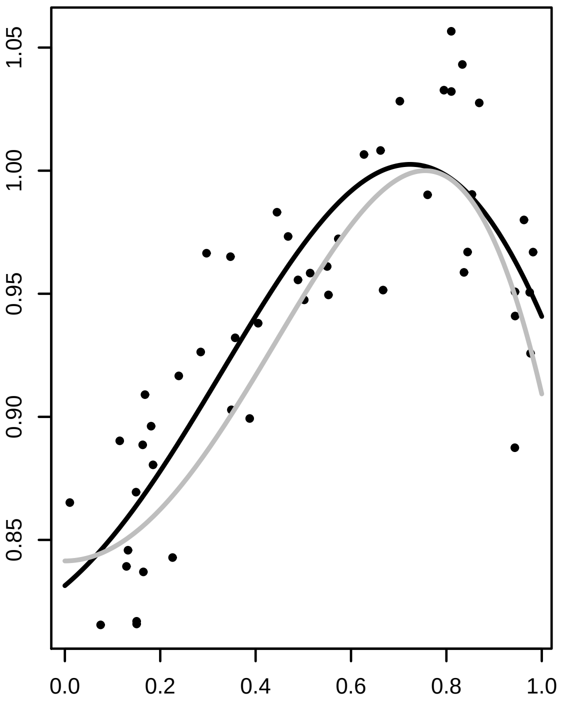


And Yet More Results. . .

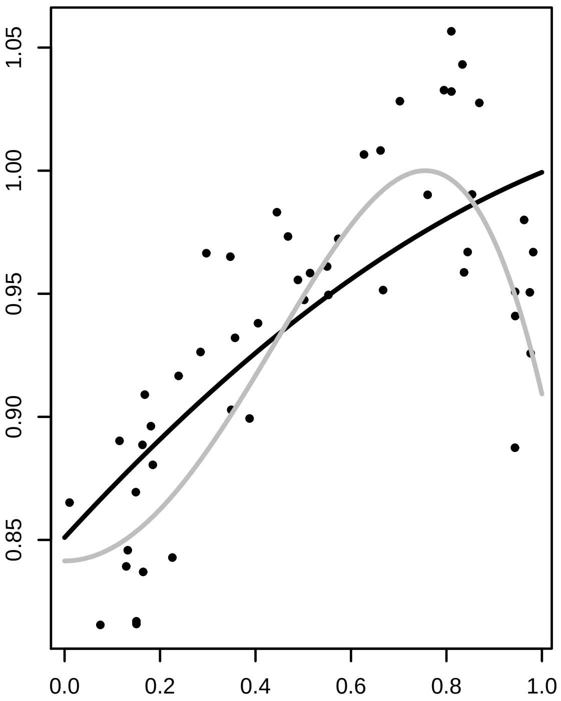
Here are the results with $\lambda = 0.01$:



Gaussian basis function fit, s = 0.1 lambda = 0.01



Gaussian basis function fit, s = 0.5 lambda = 0.01



Gaussian basis function fit, s = 2.5 lambda = 0.01

Conclusions from this Example

We see that we can control overfitting with Gaussian basis functions either by choosing the width of the basis functions, s, to be large, or by using a positive penalty, λ.

It seems that we may need to adjust *both* s and λ to get the best results.

How can we make such adjustments?

In real problems, we can't look at how the results match the true function, as we did here!

Choosing Parameters of Learning Methods by Cross-Validation

Using a Set of Validation Cases

In supervised learning, we try to learn the relationship of targets to inputs from a set of *training cases*, where both are known.

We then use what we have learned to predict the target for some *test case*, where only the inputs are known. We want to use a variaton of our learning method that does as well as possible at these predictions.

One way:

1. Randomly divide the training set into an *estimation set* and a *validation set.*
2. Try out some variations on the method(s) — eg, different basis function widths, different penalty magnitudes, different number of nearest neighbors — pretending that only the data in the estimation set is available for training.
3. See how well each variation does at predicting cases in the validation set.
4. Use the variaton with the best average performance on the validation set to make the prediction for the test case, using all training cases (in both estimation and validation sets).

Choice of Loss Function for Validation

To use this validation procedure, we need some measure of predictive performance to average over validation cases. This can take the form of a *loss function*, ℓ, which takes the actual target, y, and a prediction for the target as argments.

For a point prediction, $\widehat{y}$, two loss functions are commonly used:

$$\begin{aligned} \textbf{Squared error loss:} \quad \ell(y, \widehat{y}) &= (y - \widehat{y})^2. \\ \textbf{Absolute error loss:} \quad \ell(y, \widehat{y}) &= |y - \widehat{y}|. \end{aligned}$$

When the prediction is a distribution, $\widehat{P}$, for y, one can use

$$\textbf{log probability loss:} \quad \ell(y, \widehat{P}) \;=\; -\log \widehat{P}(y).$$

We should choose a loss function that comes close to capturing what we're actually worried about! Squared error is very traditional, however.

Selection of s and λ for the Previous Example

I looked at the square root of the average squared error on validation cases when fitting on the other cases using penalized least squares.

Results using cases 1 to 10 for the validation set:

```
         lambda=0.001 lambda=0.01 lambda=0.1 lambda=1 lambda=10
  s=0.02       0.0869      0.0722     0.0476   0.0413    0.0475
  s=0.1      ->0.0323    ->0.0323     0.0330   0.0342    0.0400
  s=0.5        0.0352      0.0364     0.0389   0.0443    0.0514
  s=2.5        0.0435      0.0506     0.0518   0.0553    0.0613
```

Note: The cases here are in random order. If they aren't one shouldn't just take the first 10 as the validation set! A random subset is needed.

It looks like $s = 0.1$ and $\lambda = 0.01$ or $\lambda = 0.001$ is best.

But what if we had used a different validation set?

Selections from Five Validation Sets

Here the selections for s and λ using five validation sets, partitioning the 50 cases:

```
Using cases 1 to 10 as the validation set:
       lambda=0.001 lambda=0.01 lambda=0.1 lambda=1 lambda=10
s=0.02       0.0869      0.0722     0.0476   0.0413    0.0475
s=0.1      ->0.0323    ->0.0323     0.0330   0.0342    0.0400
s=0.5        0.0352      0.0364     0.0389   0.0443    0.0514
s=2.5        0.0435      0.0506     0.0518   0.0553    0.0613

Using cases 11 to 20 as the validation set:
       lambda=0.001 lambda=0.01 lambda=0.1 lambda=1 lambda=10
s=0.02       0.1285      0.0520     0.0491   0.0522    0.0544
s=0.1        0.0401      0.0359     0.0304   0.0281    0.0340
s=0.5        0.0273    ->0.0269     0.0273   0.0330    0.0442
s=2.5        0.0303      0.0382     0.0409   0.0491    0.0573

Using cases 21 to 30 as the validation set:
       lambda=0.001 lambda=0.01 lambda=0.1 lambda=1 lambda=10
s=0.02       0.0409      0.0426     0.0378   0.0344    0.0439
s=0.1        0.0342      0.0331     0.0322   0.0310    0.0358
s=0.5        0.0304      0.0285   ->0.0278   0.0320    0.0457
s=2.5        0.0294      0.0353     0.0394   0.0513    0.0605

Using cases 31 to 40 as the validation set:
       lambda=0.001 lambda=0.01 lambda=0.1 lambda=1 lambda=10
s=0.02       0.0558      0.0418     0.0444   0.0452    0.0577
s=0.1        0.0523      0.0404     0.0360 ->0.0347    0.0468
s=0.5        0.0392      0.0383     0.0397   0.0424    0.0587
s=2.5        0.0413      0.0446     0.0499   0.0651    0.0749

Using cases 41 to 50 as the validation set:
       lambda=0.001 lambda=0.01 lambda=0.1 lambda=1 lambda=10
s=0.02       0.0688      0.0363     0.0287 ->0.0239    0.0360
s=0.1        0.0305      0.0310     0.0309   0.0299    0.0322
s=0.5        0.0327      0.0328     0.0334   0.0349    0.0384
s=2.5        0.0354      0.0414     0.0405   0.0422    0.0504
```

S-Fold Cross Validation

The variablility in selection of s and λ from the random choice of a subset of validation cases can be reduced by averaging the squared error over S validation sets, that partition the whole training set.

Here are the square roots of the average validation squared error over the $S = 5$ validation sets from the previous slide:

```
        lambda=0.001 lambda=0.01 lambda=0.1 lambda=1 lambda=10
s=0.02        0.0820      0.0506     0.0422   0.0406    0.0485
s=0.1         0.0387      0.0347     0.0326 ->0.0317    0.0381
s=0.5         0.0332      0.0329     0.0338   0.0377    0.0482
s=2.5         0.0364      0.0423     0.0448   0.0531    0.0614
```

Does the selection of $s = 0.1$ and $\lambda = 1$ seem reasonable given the data points? If you know the true function, does it seem to actually be the best choice?

Other Penalty Functions

A Penalty Function Allowing Some Big Parameters

Penalty functions other than $\lambda \sum\limits_{j=1}^{m-1} \beta_j^2$ are sometimes used.

One possibility is a penalty of $\lambda \sum\limits_{j=1}^{m-1} \log(1+\beta_j^2)$.

This penalizes β_j when it is near zero (where $\log(1+\beta_j^2) \approx \beta_j^2$), but not much when $|\beta_j|$ is big. This may be good if you expect a few β_j to be much bigger than the rest, and don't want to shrink them towards zero.

But the maximum penalized likelihood estimate with this penalty may not be unique.

The Lasso Penalty

The *lasso* penalty is $\lambda \sum_{j=1}^{m-1} |\beta_j|$.

Finding the maximum penalized likelihood estimate for this penalty can't be done with simple matrix operations, but it can be done efficiently. There is a unique solution.

(These are both consequences of the fact that the penalty plus log likelihood is still a convex function.)

The Lasso Produces Sparse Estimates

The lasso penalty has the property that the penalized maximum likelihood estimate often has some $\widehat{\beta}_j$ that are exactly zero. This doesn't happen when the penalty involves β_j^2.

Why? When β_j is close to zero, the derivative of β_j^2 is also close to zero — so the likelihood will dominate (and in general doesn't favour β_j being exactly zero). But even when β_j is close to zero, the derivative of $|\beta_j|$ is ± 1, so the penalty can drive β_j to be exactly zero.

Is this desirable? It may be if

- You believe that many of the true β_j are exactly zero.
- You prefer many β_j to be exactly zero so the result is easier to interpret.
- You want many β_j to be exactly zero to save computation time later, or to save measuring the corresponding x_j for new test cases.

But often you don't believe that any β_j are exactly zero, in which case setting them to zero may degrade predictive performance.