Machine Learning I

Lecture 18: Smoothing Methods

Nathaniel Bade

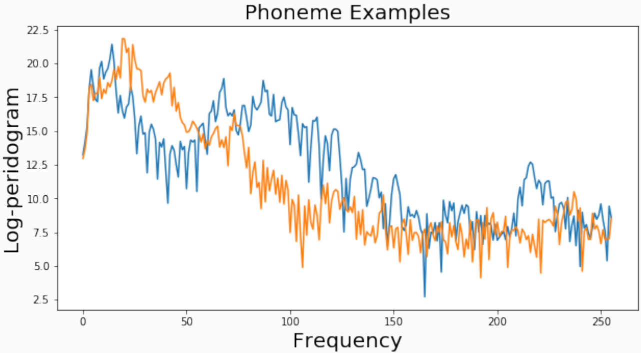
Northeastern University Department of Mathematics

Table of contents

1. Introduction
2. Introduction
3. Piecewise Polynomials and Splines
4. Endpoint Selection and Smoothing Splines
5. Multidimensional Splines
6. Kernel Smoothing
7. Other Bases

Introduction

Introduction to Smoothing

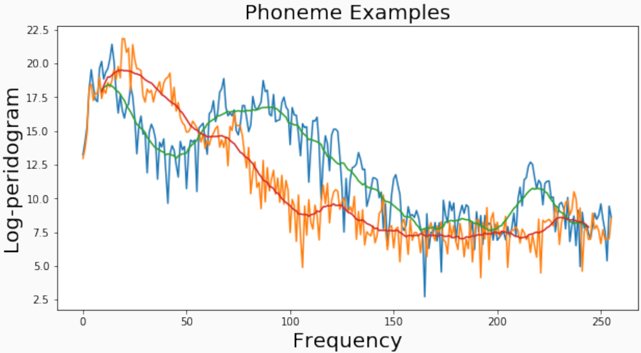


A common challenge in data analysis is trying to extract meaning from noisy data. Whether an end in itself or a step in the data processing pipeline, **smoothing methods** try to extract low variance information from high variance data.

Smoothing in some sense applies unsupervised learning methods to supervised learning data in that we attempt to fit features to the joint space $\mathcal{X} \times \mathcal{Y}$.

Introduction

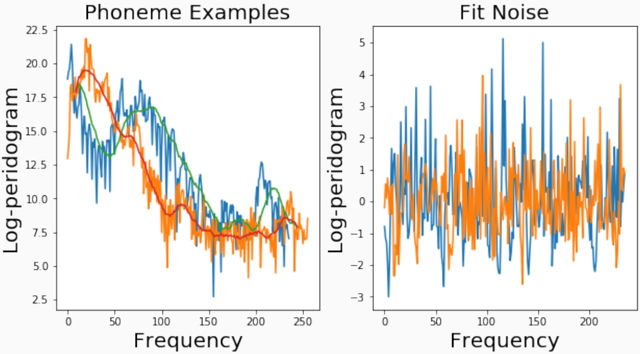
Introduction to Smoothing



A common challenge in data analysis is trying to extract meaning from noisy data. Whether an end in itself or a step in the data processing pipeline, **smoothing methods** try to extract low variance information from high variance data.

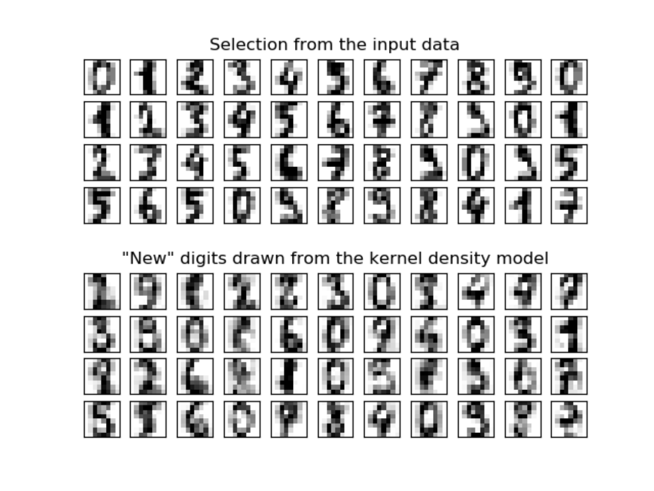
Smoothing in some sense applies unsupervised learning methods to supervised learning data in that we attempt to fit features to the joint space $\mathcal{X} \times \mathcal{Y}$.

Introduction to Smoothing



Smoothing is used in preprocessing and feature engineering to construct a useful feature out of a noisy one. It can also be used to split deterministic and Baysean information when the train data spans the entire range $\mathcal{X}$.

Introduction to Smoothing


Selection from the input data
"New" digits drawn from the kernel density model

Finally, smoothing methods can be used in data generation by fitting a smoothing surface to a large dataset, modeling the noise, and producing new data as a sum of the smoothed features and sampled noise.

Smoothing Procedures

There are four main views on smoothing mechanisms:

Piecewise smoothing cuts the domain into regions and models each region separately.

Basis smoothing attempts to write the data as a weighted sum of basis functions. This will be a unifying theme throughout this lecture.

Kernel smoothing passes a window over over the data, taking a weighted average of data values near a point. Note: this usage of kernel is not the same as in the kernel method from before.

Dimensional reduction/expansion attempts to smooth data by passing through a compression or dimensional reduction filter and then projects it back upwards.

We will assume unless otherwise stated that X is one dimensional.

Piecewise Polynomials and Splines

Piecewise Constant Fitting

A piecewise polynomial function $f(X)$ is built by dividing the domain into K disjoint regions $R_k = [\xi_{k-1}, \xi_k]$ and representing f by a separate polynomial on each region.

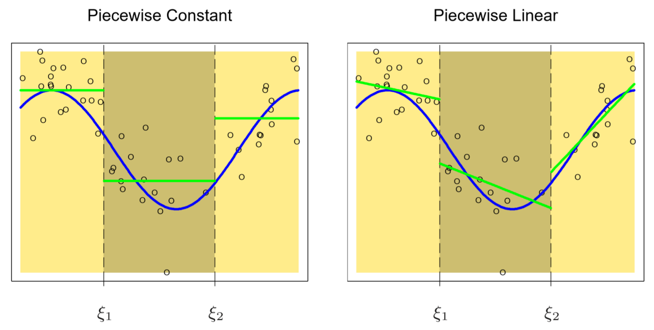
First, lets consider piecewise constant functions $h_k = \mathbb{1}_{R_k}$. The least squares estimate of the model

$$f(X) = \sum_{k=1}^{K} \beta_k h_k(X)$$

can be solved by setting $\beta_k = \bar{Y}_k$, the sample mean for each region R_k.

We can think of these functions as forming a set of **basis** functions that we express the data in terms of.

Piecewise Linear Fitting

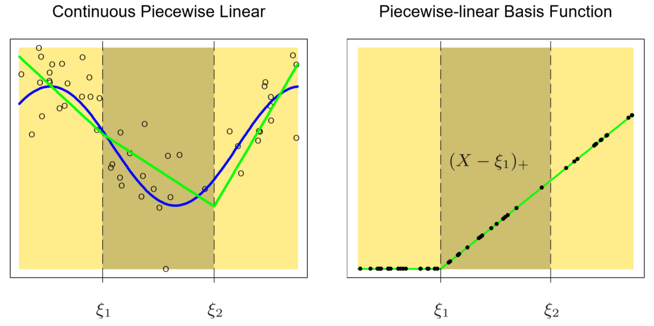


For a piecewise linear fitting, we need K additional functions in our basis $g_k = \mathbb{1}_{R_k} X$. Fitting

$$f(X) = \sum_{k=1}^{K} \beta_k h_k(X) + \gamma_k g_k(X)$$

amounts to performing K linear fits over each region R_k.

Piecewise Linear Fitting

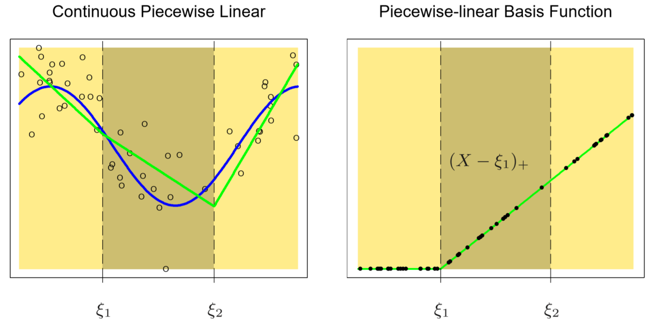


Except in special cases we want to impose some boundary conditions on the regions R_k. For linear functions the most obvious boundary condition is simple continuity. This leads to $K-1$ conditions

$$\beta_k + \gamma_k \xi_k = \beta_{k+1} + \gamma_{k+1} \xi_{k+1} \,.$$

We then have $2K$ parameters and $K-1$ conditions, leaving only $K+1$ basis functions.

Piecewise Linear Basis

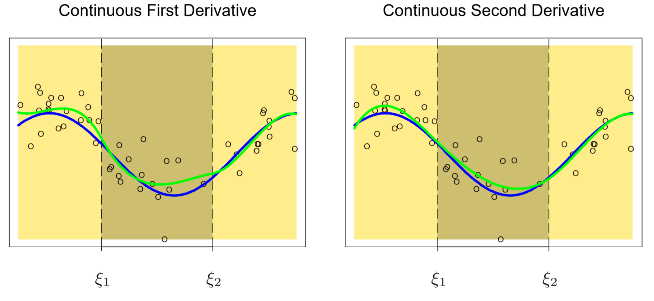


Lets define a basis that better incorporates these conditions. Let $h_0(X) = 1$, $h_1(X) = X$ and

$$h_{k+1}(X) = (X - \xi_k)_+ \,,$$

where $(X - \xi_k)_+$ denotes the ReLUs function as displayed on the right. We can fit sequentially, first fitting h_0 and h_1 to the whole dataset, then h_ℓ in order.

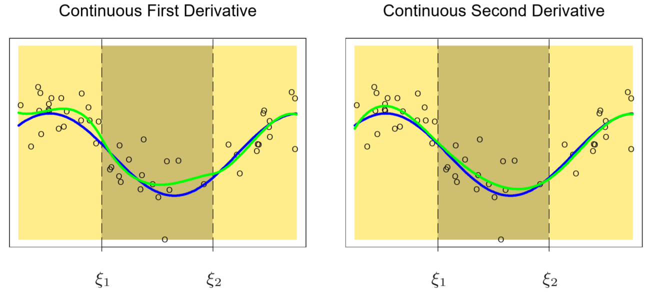
Cubic Splines



Moving beyond linear functions, we can enforce continuity on the higher order derivatives. One the left we see a cubic fit with continuous first derivatives and on the right we also impose continuous second derivatives.

A piecewise cubic with continuous second derivatives at the endpoints ξ is called a **cubic spline**.

Cubic Splines

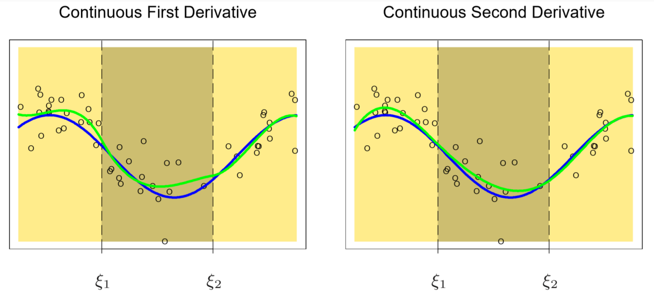


Using a basis of locally cubic function and enforcing the continuity of the second derivative at the points ξ_k lead to the following basis: $h_0(X) = 1$, $h_1(X) = X$ $h_2(X) = X^2$, $h_3(X) = X^3$, and

$$h_{k+3} = (X - \xi_k)_+^3 .$$

Note that there are 4K parameters, but we have imposed 3(K-1) conditions.

Higher Order Splines



More generally, an order M spline is a piecewise continuous polynomial with continuous derivatives up to order $M-2$. It can be written in a basis of $K+M$ functions

$$h_k = X^k \,, k = 0, \ldots, M-1 \qquad h_{M+\ell-1} = (X - \xi_\ell)^{M-1} \,, \ell = 1, \ldots, K \,.$$

A cubic spline is a order $M = 4$ spline.

Other Bases

The cubic spline basis we've described should only be trusted over the regions R_k, since outside these regions the boundary cubics quickly go to infinity.

If data needs to be evaluate outside this fitting region, one may additionally impose linearity at the boundaries. This new basis has four fewer degrees of freedom and can be expressed by the basis $h_0 = 1$, $h_1(X) = X$,

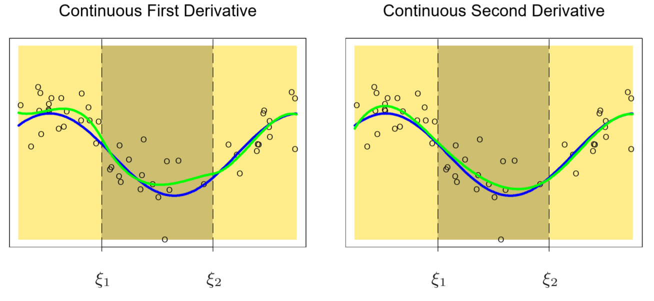
$$N_{k+1}(X) = d_k(X) - d_{K-1}(X)$$

where

$$d_k(X) = \frac{(X - \xi_k)^3_+ - (X - \xi_K)^3_+}{\xi_K - \xi_k} .$$

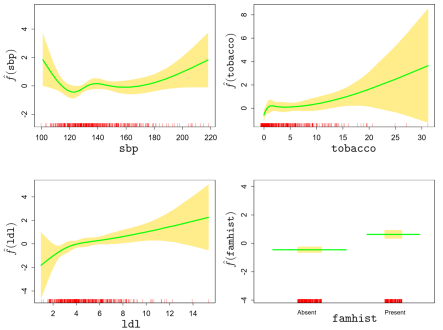
This is called the basis of **natural cubic splines**.

Other Bases



Cubic splines are a decent smoothing basis and there is seldom a good reason to go to higher order. However, for a large number of regions it can become computationally inefficient. In this case, there are other spline bases such as **B-splines** which use the Harr basis to define more computationally efficient spline fittings.

Example: Non-Mixed Features



Lets consider again the heart disease dataset above. We can fit the logit probability

$$\text{logit}[\Pr(1|X)] = \theta_0 + h_1(X_1)^T\theta_1 + \ldots + h_p(X_p)^T\theta_p\,,$$

where h_j is a vector of spline basis functions for each feature and θ_j are a vector of β's for each set of splines.

Example: Non-Mixed Features

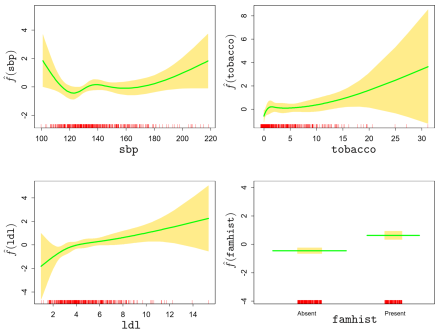
Combining all of the h_j into one vector $h(X)$ we can write

$$\text{logit}[\Pr(1|X)] = \theta_0 + h_1(X_1)^T\theta_1 + \ldots + h_p(X_p)^T\theta_p = h(X)^T\theta\,.$$

The model has $df_j = \sum_{j=1}^{p} \mathrm{df}_j$ degrees of freedom, where df_j is the number of basis elements for each variable. Evaluating each basis function at each X_i results in a $N \times \mathrm{df}$ matrix $\mathbf{H}$.

At this point, the problem is like any other and we can proceed to fit it using Newtons method as we did for the linear logistic model.

Example: Non-Mixed Features



We can also use standard linear methods like backward subset selection, dropping basis elements according to some metric. In the above, features have been dropped to jointly minimize the training error and the number of degrees of freedom.

Endpoint Selection and Smoothing Splines

Smoothing Splines

In the previous section, our fitting required selecting the endpoints by some procedure that was not specified. While these can be set by hand, spaced linearly according to density and degrees of freedom, or fit as hyperparamters, we want to discuss a way to avoid the problem altogether.

To do so, we minimize

$$\mathrm{RSS}(f, \lambda) = \sum_{i=1}^{N} \{y_i - f(x_i)\}^2 + \lambda \int \{f''(t)\}^2 \, dt$$

over the (Sobolev) space of functions with finite total integral $\int \{f''(t)\}^2 \, dt$. The parameter λ define the smoothness, with $\lambda \to 0$ being any function in the space and $\lambda = \infty$ forcing $f(X)$ linear.

Smoothing Splines

It can be shown that

$$\mathrm{RSS}(f, \lambda) = \sum_{i=1}^{N} \{y_i - f(x_i)\}^2 + \lambda \int \{f''(t)\}^2 \, dt$$

has a unique, finite dimensional minimizer given by a natural cubic splines

$$f(X) = \sum_{j=1}^{N} N_j(X)\theta_j \,,$$

with regions defined to have endpoint at the datapoints x_i. The penalty λ becomes a dampening of the coefficients, which reduces the potential overfitting.

Smoothing Splines

Writing

$$\Omega_{jk} = \int N_j''(t) N_k''(t)\, dt \qquad \{\mathbf{N}\}_{ij} = N_j(X_i)\,,$$

the loss

$$\mathrm{RSS}(f,\lambda) = \sum_{i=1}^{N} \{y_i - f(x_i)\}^2 + \lambda \int \{f''(t)\}^2\, dt$$

can be written as

$$\mathrm{RSS}(f,\lambda) = (\mathbf{y} - \mathbf{N}\theta)^T (\mathbf{y} - \mathbf{N}\theta) + \lambda \theta^T \Omega \theta\,.$$

It can then be solved for

$$\hat{\theta} = (\mathbf{N}^T \mathbf{N} + \lambda\Omega)^{-1} \mathbf{N}\mathbf{y}\,.$$

To be clear, the basis $N_i(x)$ is known explicitly, so this is just a generalized ridge regression problem.

Degrees of Freedom for Smoothing Splines

As a function of λ, a smoothing spline is a **linear smoother** since the parameters are linear functions of y_i. Indeed, after fitting

$$\hat{f} = \mathbf{N}(\mathbf{N}^T\mathbf{N} + \lambda\Omega)^{-1}\mathbf{N}\mathbf{y} = \mathbf{S}_\lambda \mathbf{y}\,,$$

where $\mathbf{S}_\lambda$ is a linear operator known as the **smoother matrix**.

If we fit cubic splines with $M < N$ endpoints ξ as in last section, we can form a similar $N \times M$ projection matrix $\mathbf{B}_\xi$ and solve the least squares problem by

$$\hat{f} = \mathbf{B}_\xi(\mathbf{B}_\xi^T\mathbf{B}_\xi)^{-1}\mathbf{B}_\xi^T\mathbf{y} = \mathbf{H}_\xi\mathbf{y}\,.$$

Degrees of Freedom for Smoothing Splines

Both $\mathbf{S}_\lambda$ and $\mathbf{H}_\xi$ are symmetric and positive definite. In addition, $\mathbf{H}_\xi^2 = \mathbf{H}_\xi$ is idempotent, while $\mathbf{S}_\lambda^2 \preceq \mathbf{S}_\lambda$, that is $\mathbf{S}_\lambda^2$ exceed itself by a positive semidefinite matrix.

The trace of an idempotent is its rank so for generic ξ, $M = \mathrm{tr}(\mathbf{H}_\xi)$ gives then number of basis functions or degrees of freedom. By analogy, we define the effective degrees of freedom of $\mathbf{S}_\lambda$ to be $\mathrm{df}_\lambda = \mathrm{tr}(\mathbf{S}_\lambda)$.

We will say a few words to justify this, and describe the smoothing behavior.

Degrees of Freedom for Smoothing Splines

Notice that

$$\mathbf{S}_\lambda = \mathbf{N}(\mathbf{N}^T\mathbf{N} + \lambda\Omega)^{-1}\mathbf{N} = (1 + \lambda\mathbf{K})^{-1}$$

so $\hat{f} = \mathbf{S}_\lambda \mathbf{y}$ minimizes

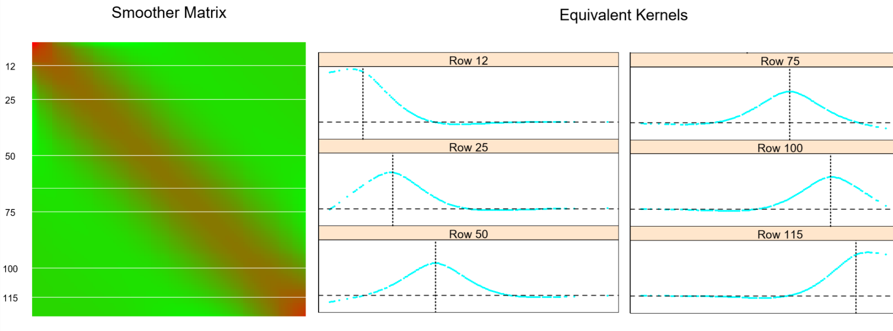
$$(\mathbf{y} - f)^T(\mathbf{y} - f) + \lambda f^T\mathbf{K}f\,.$$

If the penalty matrix $\mathbf{K}$ has eigenvalues d_k, we can decompose $\mathbf{S}_\lambda\mathbf{y}$ as

$$\mathbf{S}_\lambda\mathbf{y} = \sum_{i=1}^{N} \frac{1}{1 + \lambda d_k}\mathbf{u}_i\langle\mathbf{u}_i, \mathbf{y}\rangle\,.$$

Note, $\mathbf{S}_\lambda$ has the same Eigenvectors regardless of λ, all λ is doing is dampening them proportional to the eigenvalues d_i. In addition, as $\lambda \to \infty$, df decreases from N to 2, since $\mathbf{K}$ can be shown to have 2 zero eigenvalues.

Degrees of Freedom for Smoothing Splines



Here we see a smoothing matrix $\mathbf{S}_\lambda$ ordered so that $x_i < x_{i+1}$. The expressions on the matrix correspond to points projected onto the basis and imply that smoothing splines are a local method.

Multidimensional Splines

Multidimensional Splines

The multivariate case follows directly from the one variable case. For example, in $\mathbb{R}^2$, if h_{1i} is a spline basis for X_1, and h_{2j} form a spline basis for X_2, we can form the tensor product basis

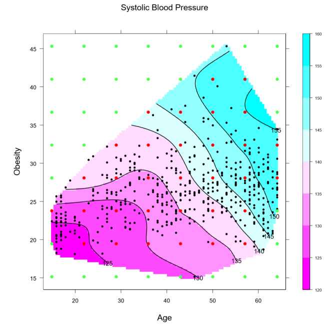
$$g_{ij}(X) = g_{ij}(X_1, X_2) = h_{1i}(X_1)h_{2j}(X_2)\,.$$

We can then fit

$$g(X) = \sum_{i=1}^{M_1}\sum_{j=1}^{M_2} g_{ij}(X)^T\theta_{ij} = g(X)^T\theta\,,$$

for a suitably reshaped $g(X)$ and θ.

Multidimensional Splines



In practice, the only difficulty is picking the lattice. For a maximum number of degrees of freedom M, one can separate the domain into a lattice of M points and then throw away all points outside the convex hull of the dataset.

Multidimensional Smoothing Splines

For smoothing splines, we now need to minimize a function of the form

$$RSS(\theta) = \sum_{i=1}^{N} (y_i - f(x_i))^2 + \lambda \int_{\mathbb{R}^2} \left(\frac{\partial}{\partial x_1} + \frac{\partial}{\partial x_2} \right)^2 f(x)\, dx_1\, dx_2 \,.$$

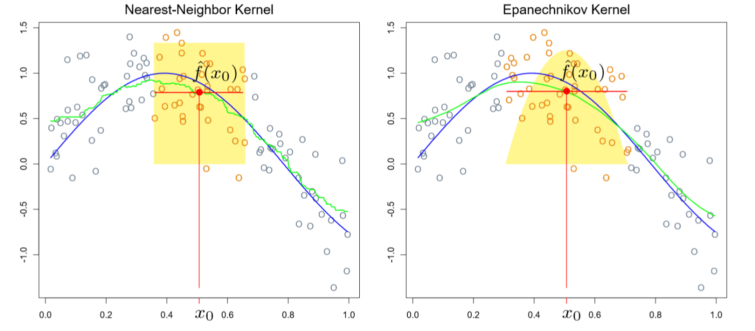
The solutions are smooth 2d surfaces known as **thin plate splines**, and take the form of radial basis functions

$$h_i(X) = ||X - X_i||^2 \log ||X - X_i|| \,.$$

As before, for $\lambda = 0$ the solution approaches an interpolating function and for $\lambda \to \infty$, the solution approaches the least squares plane.

Kernel Smoothing

Nearest Neighbor Kernel Smoothing

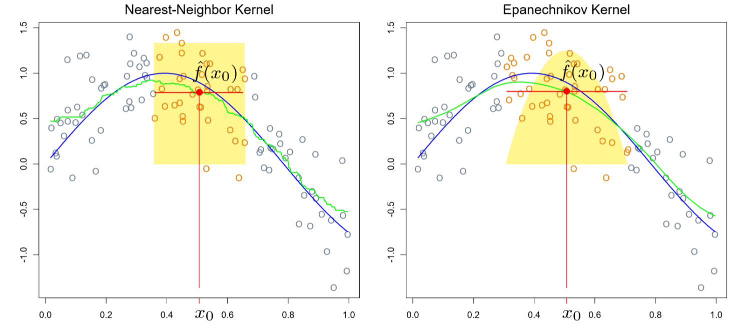


Another way to smooth functions locally is to apply a filter to each point. The simplest kernel smoothing filter is to simply compute the average value of k nearest training points $N_k(x)$:

$$\hat{f}(X) = \text{Avg}(y_i | x_i \in N_k(x))$$

However, this results in the bumpy discontinuous curve on the left.

Nearest Neighbor Kernel Smoothing

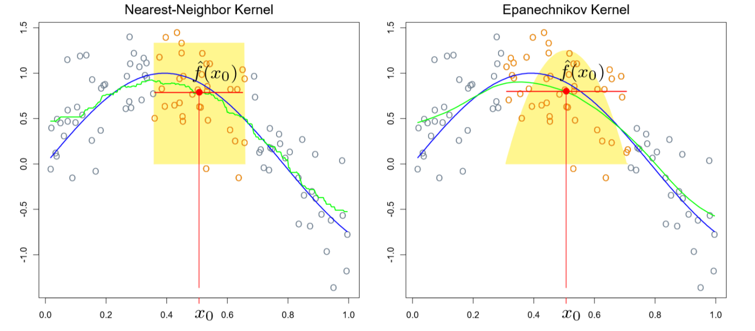


A more sophisticated smoothing involves weighting the points so that further points contribute less. For example, we take the average

$$\hat{f}(x) = \frac{\sum_{i=1}^{N} K_\lambda(x, x_i) y_i}{\sum_{i=1}^{N} K_\lambda(x, x_i)}, \quad K_\lambda(x, x_i) = D\left(\frac{|x - x_0|}{\lambda}\right)$$

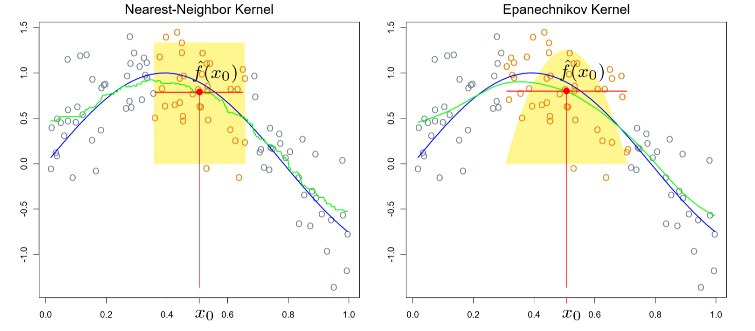
where $D(t) = \frac{3}{4}(1 - t^2)$ for $|t| \leq 1$ is the Epanechnikov kernel.

Nearest Neighbor Kernel Smoothing



The Epanechnikov kernel fits a continuous function to the data. The **bandwidth** λ determines the window size and can be constant, depend on k-neighborhood size $\lambda = h_\lambda(X) = |X - X_{[k]}|$, or vary according to other considerations.

Nearest Neighbor Kernel Smoothing



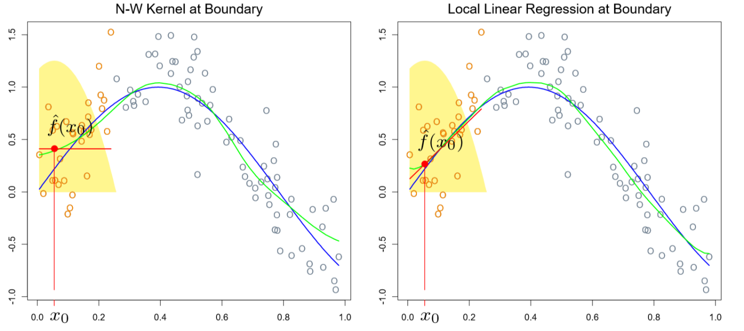
In general, a kernel has finite support $K(x, x_i) = 0$ for $|x - x_i| \geq 1$, symmetry $K(-x, -x_i) = K(x, x_i)$ and positivity $K(x, x_i) > 0$ for $|x - x_i| < 1$. All of the following results will hold for such kernels.

Practical concerns

There are a couple of consideration in practice:

- For fixed λ, large λ implies low variance since we're averaging over more data points but higher bias.
- For density based λ, the bias tends to be constant but the variance is inverse proportional to the local density.
- The neighborhood of the boundary tends to contain fewer points, and so our estimates will be less accurate there.
- We have to select a kernel. The Epanechnikov kernel can be replaced by a Gaussian kernel for example, giving slightly different fitting.

Locally Linear Regression



The boundary value problem, and indeed some internal variance, can be solved by replacing the pointwise estimate with a linear estimate called **local linear regression**.

Locally Weighted Regression

Local linear regression attempts to solve a separate weighted least squared problem at each target point x_0. The local loss is a function

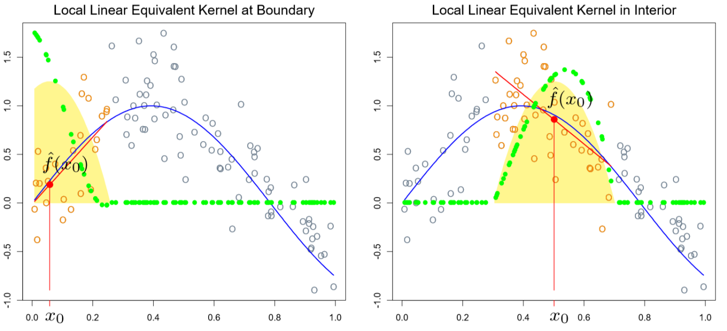
$$RSS = \sum_{i=1}^{N} K_\lambda(x_0, x_i)[y_i - \alpha_{x_0} - \beta_{x_0} x_i]^2 \,,$$

where α_{x_0} and β_{x_0} are constants. The fit is then $\hat{f}(x_0) = \hat{\alpha}_{x_0} + \hat{\beta}_{x_0} x_0$, and is uniquely determined for the point x_0.

Letting $\mathbf{X} = (1, x_i)$ be the matrix with a column of 1's and a column of data points x_i as usual and let $\mathbf{W}(x_0)$ be diagonal with entries $K_\lambda(x_0, x_i)$. Then

$$\hat{f}(x_0) = [1, x_0]^T (\mathbf{X}^T \mathbf{W}(x_0) \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W}(x_0) \mathbf{y} = \sum_{i=1}^{N} \ell_i(x_0) y_i \,.$$

Locally Linear Regression



In the figure above, we see the weights $\ell_i(x_0)$ for different x_0 as we move through the dataset.

Kernel Carpentry

Assume for a moment that the labels are generated by $y_i = f(x_i) + \epsilon$, where f is twice differentiable. It turns out the local linear regression exactly matches the function $f(x)$ up to first order. With a bit of work one can show that

$$\sum_i \ell_i(x_0) = 1\,, \qquad \sum_i^N (x_i - x_0)\ell_i(x_0) = 0$$

provided K is a kernel in the sense defined before. Expanding f around x_0,

$$\hat{f}(x_0) = f(x_0)\sum_{i=1}^N \ell_i(x_0) + f'(x_0)\sum_{i=1}^N \ell_i(x_0)(x_i - x_0) + \text{h.o.t.}\,.$$

But the conditions above mean that

$$\hat{f}(x_0) = f(x_0) + \frac{1}{2}f''(x_0)\sum_{i=1}^N \ell_i(x_0)(x_i - x_0)^2 + R\,,$$

so up to linear order the the local regression exactly matches the true function.

Kernel Carpentry

The result

$$\hat{f}(x_0) = f(x_0) + \frac{1}{2} f''(x_0) \sum_{i=1}^{N} \ell_i(x_0)(x_i - x_0)^2 \,,$$

can be rewritten

$$\text{Bias}[\hat{f}(x_0)] = \hat{f}(x_0) - f(x_0) = \frac{1}{2} f''(x_0) \sum_{i=1}^{N} \ell_i(x_0)(x_i - x_0)^2 + R \,,$$

and we say the local bias is of quadratic order. This is the same as saying that locally we have fit the function exactly up to linear order. This result is known as **automatic kernel carpentry**.

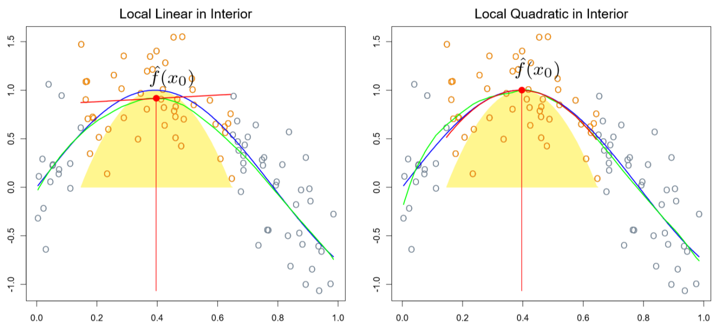
Kernel Carpentry

Finally, it can be shown that

$$\text{Bias}[\hat{f}(x_0)] = O(\lambda^2)\,, \quad \text{Var}[\hat{f}(x_0)] = O_p\left(\frac{1}{nh^p}\right)$$

So we explicitly see that for a higher bandwidth, the local bias increases while the local variance decreases, and visa versa. Here p is the number of features.

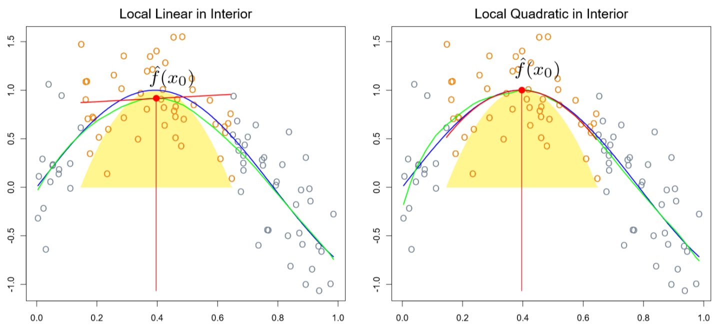
Local Polynomial Regression



Higher degree polynomials account for higher degrees of bias, but they of course come with a tradeoff in variance. The local polynomial problem is to minimize

$$RSS = \sum_{i=1}^{N} K(x_0, x_i) \left(y_i - \alpha_{x_0} - \sum_{j=1}^{d} \beta_{x_0,j} x_i^j \right) .$$

Local Polynomial Regression



In the figure above, we see that local linear fits exhibit their bias in regions of high curvature, while degree 2 polynomials can correct this. In general it is very nongeneric to have a point that is dominated by behavior of order higher than 2, so we can usually stop at local quadratic regression.

Selecting the Kernel Width

The bias-variance tradeoff is controlled by the bandwidth λ, where λ acts as the cutoff for the Epanechnikov kernel, the standard deviation for the Gaussian kernel, or number of neighbors for the k nearest neighbors kernel.

Local regression smoothers are linear estimates, and so can be expressed by the smoother matrix $\hat{f} = \mathbf{S}_\lambda \mathbf{y}$, where $\{\mathbf{S}_\lambda\}_{ij} = \ell_i(x_j)$. The effective number of degrees of freedom can then be calculated by

$$\mathrm{df}_\lambda = \mathrm{tr}(\mathbf{S}_\lambda)\,.$$

Kernels in Higher Dimensions

For higher dimensional kernel fitting, we again want to fit a dataset to a set of degree d polynomials. For example, for two features (X_1, X_2), let $b(X) = (1, X_1, X_2, X_1^2, X_1X_2, X_2^2)$ and consider minimizing

$$RSS = \sum_{i=1}^{N} K_\lambda(x_0, x_i)(y_i - b^T(x_i)\beta_{x_0})$$

In the kernel, we typically replace evenness with radial symmetry, such as in the radial Epanechnikov function

$$K_\lambda(x_0, x) = D\left(\frac{||x - x_0||}{\lambda}\right).$$

The main issue for high dimensional smoothing is curse of dimensionally, which here means that an increasing percentage of the points will be boundary points. Even local linear regression becomes useless in dimensions much above 3, but higher order polynomial techniques can still be effective.

Other Bases

Other Bases

There are a few other bases we should mention.

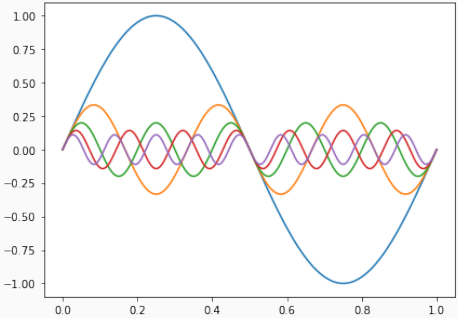
Fourier Basis: The canonical basis for wavelike data, separates waves into sums of frequencies.

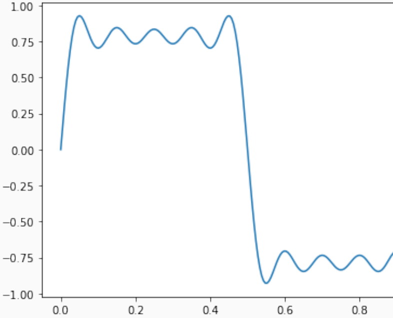
Wavelet Bases: Any of a family of bases that consist of “waves” of a finite length. Unlike the Fourier basis this gives them both frequency and locality.

Haar Basis: A computationally efficient wavelet basis composed of piecewise step functions that cover a partition of the domain.

BSpline Basis: Polynomial combinations of the Haar basis functions that provide a differenitable spline basis. The Bspline basis is what is typically used in actual spline computation.

Fourier Basis



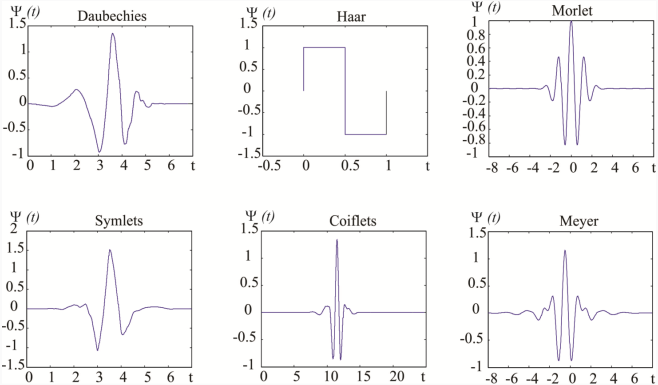


The Fourier basis transforms repeating waves into sums of sine and cosine functions at different frequencies:

$$f(x) = \sum_{n=0}^{\infty} a_n \sin(2\pi n/T) + b_n \cos(2\pi n/T)\,.$$

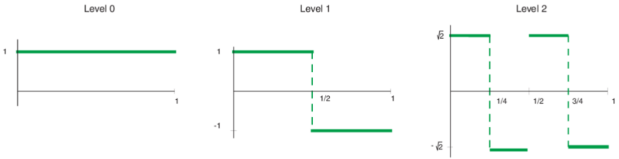
It is important to note that the Fourier basis truncated to a finite number of terms is inherently non-local.

Wavelet Basis



A wavelet basis is a basis of functions that try to capture both frequency and location. There are many adapted to various theoretical and practical uses, and are particularly used in image processing and storage.

Haar Basis

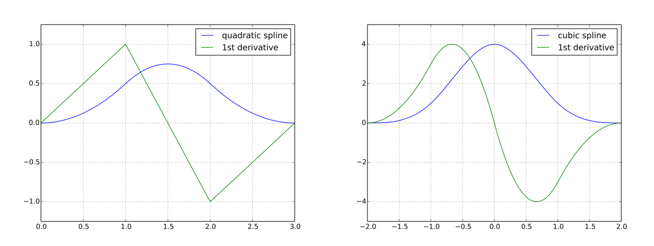


One of the building blocks of computationally efficient wavelet computations is the Haar basis. We construct the basis recursively, starting with a constant function $h_0(x)$ on a bounded domain, for example $R = [0, 1]$. Then,

$$h_1(x) = \begin{cases} 1 & \text{for } x < \frac{1}{2}\,, \\ -1 & \text{for } x \geq \frac{1}{2}\,. \end{cases}$$

h_0 and h_1 are orthogonal. We continue cutting each domain in half.

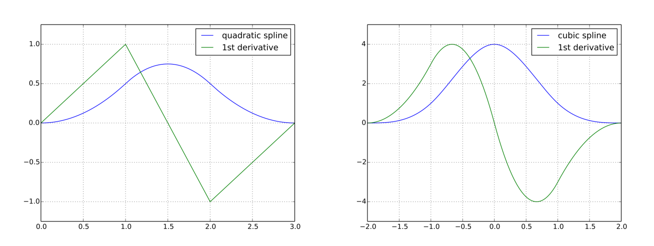
B-Spline Basis



B-splines are defined recursively from a Haar like basis. Letting $B_{i,k}(x) = \mathbb{1}_{R_k}(x)$, the order m spline is

$$B_{i,m}(x) = \frac{x - \xi_i}{\xi_{i+m-1} - \xi_i} B_{i,m-1}(x) + \frac{\xi_{i+m} - x}{\xi_{i+m} - \xi_{i+1}} B_{i+1,m-1}(x)$$

B-Spline Basis



The recursive nature of the construction allows for much faster smoothing by B-splines of order 4 than by normal cubic splines. For a large number N of regions, fitting by cubic splines can be shown to be $O(N^3)$ while under mild sparsity conditions fitting by B-splines is of order $O(N)$.

Reference

The main reference for this lecture is ESLII Chapters 5 and 6.

The following reference summarizes many results for local linear regression: http://courses.ieor.berkeley.edu/ieor265/lecture_notes/ieor265_lec6.pdf

Wavelet Gallery picture taken from https://www.intechopen.com/books/wavelet-transform-and-some-of-its-real-world-applications/empirical-wavelet-transform-based-detection-of-anomalies-in-ulf