MATH 7339 - Machine Learning and Statistical Learning Theory 2 –He Wang

Section Kernel Smoothing methods

1. Kernel Smoothing
2. Locally polynomial regressions
3. Automatic Kernel Carpentry

Introduction - Kernel Smoothing

Kernel Smoothing is *a class of regression techniques that achieve flexibility in estimating function $f(\vec{x})$ over the domain $\mathbb{R}^d$ by fitting a different but simple model separately at each query point $\vec{x}_0$.*

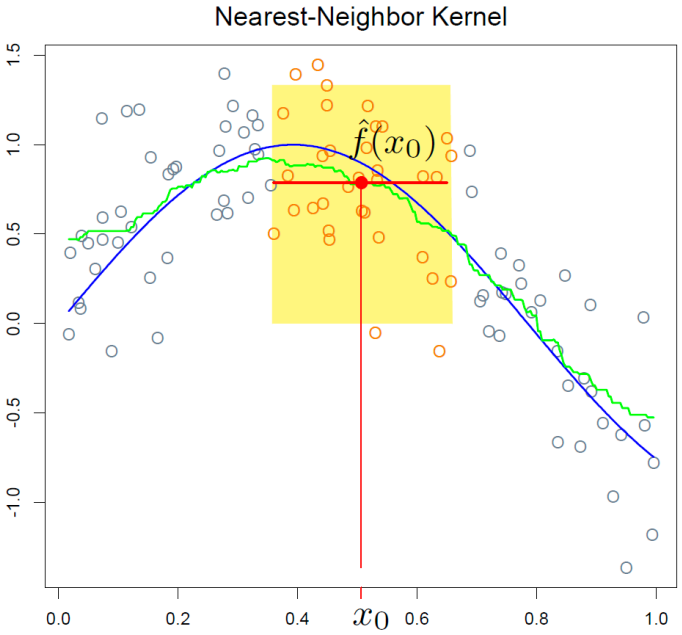
Warning: Not the same inner product kernel method we've learned previously.

➢ Nearest Neighbor Kernel Smoothing

A 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 $N_k(x)$.

$$\hat{f}(\vec{x}) = \text{Ave}\left(y^{(i)} \middle| \vec{x}^{(i)} \in N_k(x)\right)$$

However, this results in the bumpy discontinuous curve.



30-NN running-mean smoother

True function $y = \sin(4x)$ Noise $\epsilon \sim N(0,1/3)$
KNN average
Observations contributing to $\hat{f}(x_0)$

Solid yellow region: the weights assigned to observations

Nearest neighbor method uses a 'uniform kernel' (constant weight for k closest points to $\vec{x}^{(0)}$.

So bandwidth is variable, i.e., the 'width of kernel' depends on how close the k nearest neighbors of $\vec{x}^{(0)}$ are.

Use a kernel function $K(x, y)$ directly in the estimator:

Nadaraya-Watson estimator: uses piecewise constant kernel

$$K_\lambda\left(\vec{x}, \vec{x}^{(i)}\right) = \begin{cases} 1 & \text{if } \left\|\vec{x} - \vec{x}^{(i)}\right\| \le \lambda \\ 0 & \text{others} \end{cases}$$

then if we plug in the weight formula with $w_i^{(0)} = K_\lambda\left(\vec{x}^{(i)}, \vec{x}^{(0)}\right)$, we have

$$\hat{f}(\vec{x}) = \frac{\sum_{i=1}^{N} K_\lambda\left(\vec{x}, \vec{x}^{(i)}\right) y^{(i)}}{\sum_{i=1}^{N} K_\lambda\left(\vec{x}, \vec{x}^{(i)}\right)}$$

which is called Nadaraya-Watson kernel-weighted average.

➢ Epanechnikov Quadratic kernel

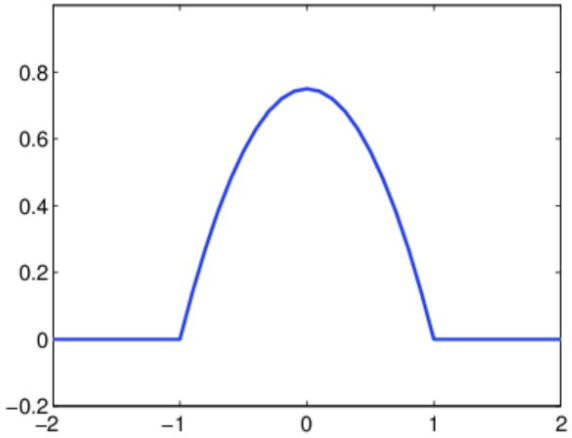
A more sophisticated smoothing involves weighting the points so that further points contribute less. For example, we take the average

$$\hat{f}(\vec{x}) = \frac{\sum_{i=1}^{N} K_\lambda(\vec{x}, \vec{x}^{(i)}) y^{(i)}}{\sum_{i=1}^{N} K_\lambda(\vec{x}, \vec{x}^{(i)})}$$

$$K_\lambda(\vec{x}, \vec{x}^{(i)}) = D\left(\frac{\|\vec{x} - \vec{x}^{(i)}\|}{\lambda}\right)$$

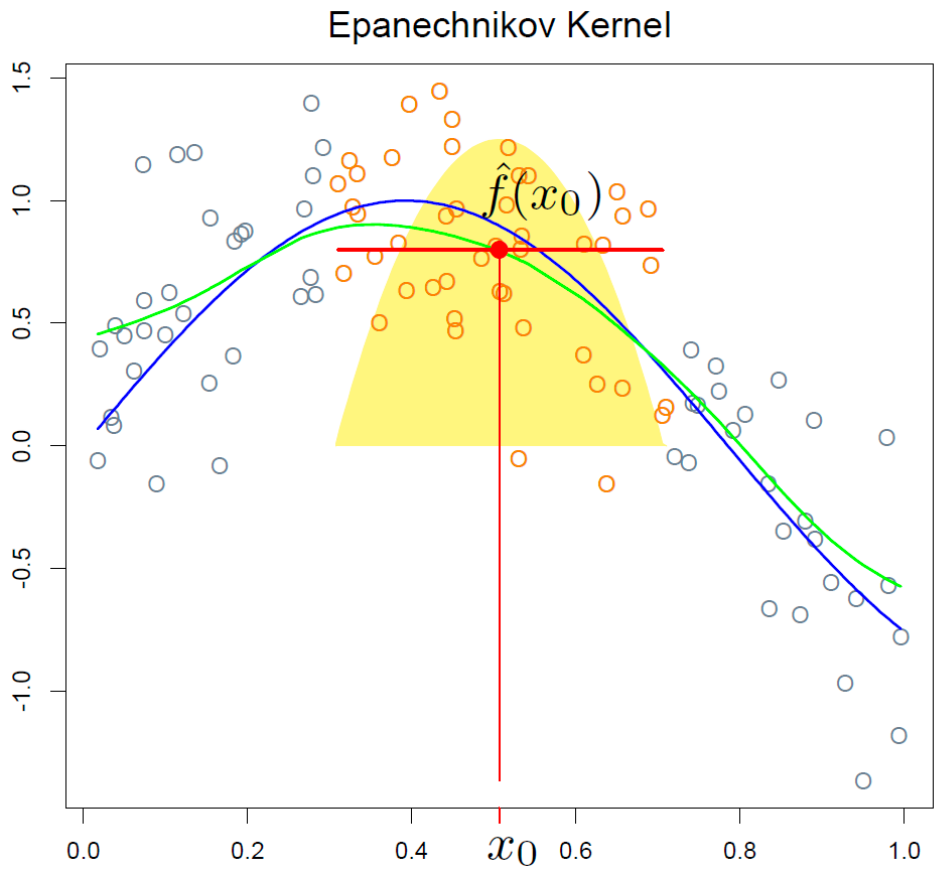
$$D(t) = \frac{3}{4}(1 - t^2)\mathbb{I}(|t| \leq 1)$$

is the Epanechnikov kernel.



The Epanechnikov kernel fits a continuous function to the data. The **band-width** λ determines the window size and can be constant, depend on k-neighborhood size $\lambda = h_\lambda(x)$

or vary according to other considerations.



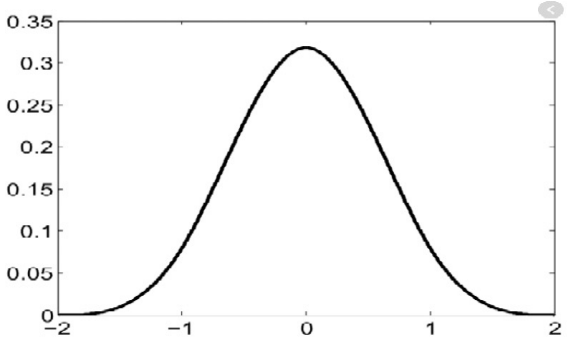
Estimated function is ‘smooth’.

Yellow area indicates the weight assigned to observations in that region.

➢ Kernel Functions

In general, a **kernel** function is $D: \mathbb{R} \to \mathbb{R}$ such that for all $u \in \mathbb{R}$

1. non-negative, $D(u) \geq 0$
2. Symmetry, $D(-u) = D(u)$
3. $\int_{\mathbb{R}} D(u)du = 1$ (it is ok to omit it.)



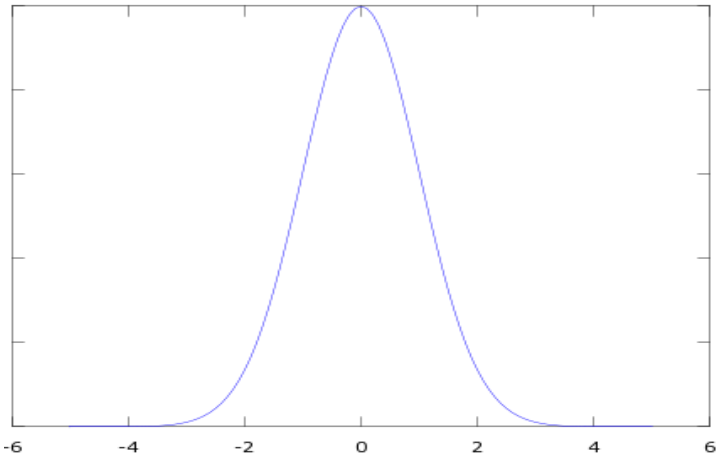
Can also calibrate width λ of the kernel by letting

$$w_i^{(0)} = K_\lambda\left(\vec{x}^{(i)}, \vec{x}^{(0)}\right) = D\left(\frac{\left\|\vec{x}^{(i)} - \vec{x}^{(0)}\right\|}{\lambda}\right)$$

With $\lambda =$ ***kernel bandwidth***

Gaussian kernel

$$D(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\mathrm{u}^2\right)$$



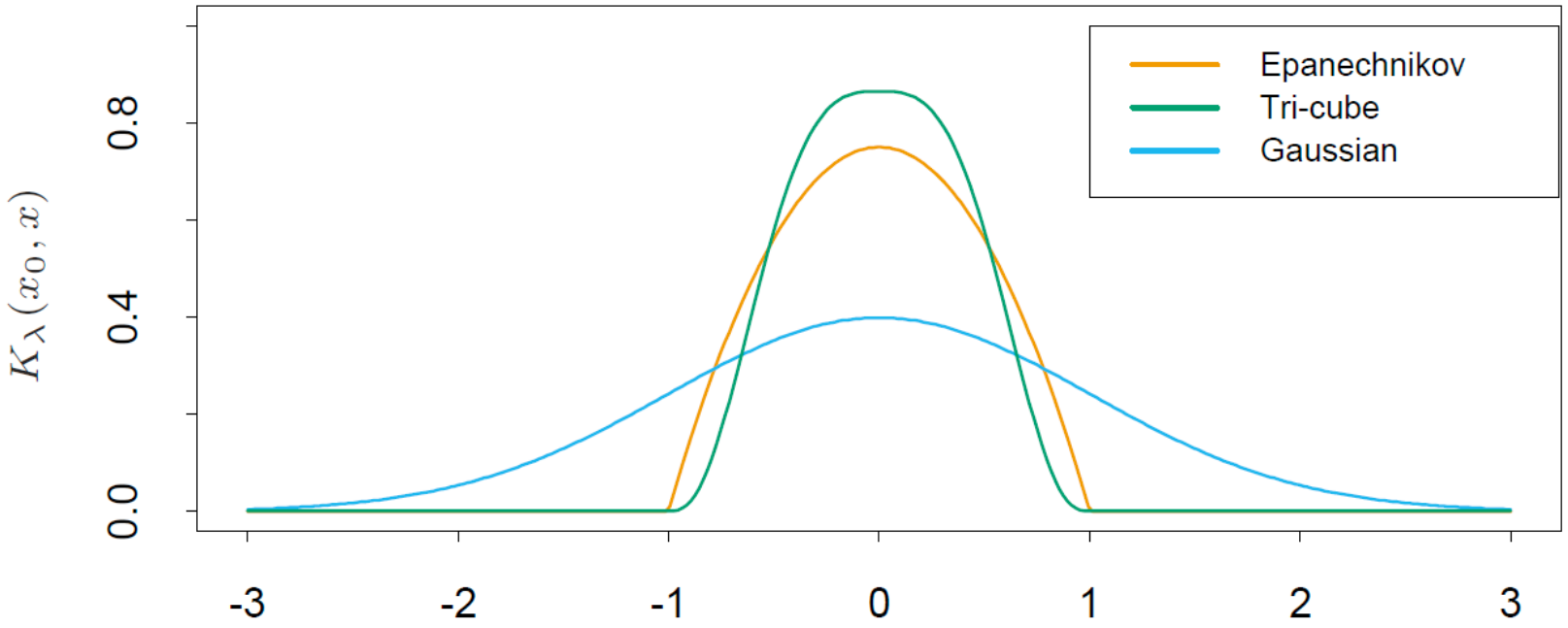
$$w^{(i)} = K_\lambda\left(\vec{x}^{(i)}, \vec{x}^{(0)}\right) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\left\|\vec{x}^{(i)} - \vec{x}\right\|^2}{2\lambda^2}\right)$$

For Gaussian kernel major weight given only to points within $\pm 3\lambda$ of $\vec{x}^{(0)}$.

Tri-cube kernel

$$D(u) = (1 - |t|^3)^3 \, \mathbb{I}(|t| \le 1) = \begin{cases} (1 - |t|^3)^3 & \text{if } |t| \le 1 \\ 0 & \text{otherwise} \end{cases}$$

Compare Popular Kernels



- Epanechnikov: Compact (only local observations have non-zero weight)
- Tri-cube: Compact and differentiable at boundary
- Gaussian density: Non-compact (all observations have non-zero weight)

More general with **adaptive** neighborhood

$$K_\lambda\left(\vec{x}^{(i)}, \vec{x}^{(0)}\right) = D\left(\frac{\left\|\vec{x}^{(i)} - \vec{x}^{(0)}\right\|}{h_\lambda\left(\vec{x}^{(0)}\right)}\right)$$

Here, $h_\lambda\left(\vec{x}^{(0)}\right)$ is a width function indexed by λ that determines the width of the neighborhood at $\vec{x}^{(0)}$

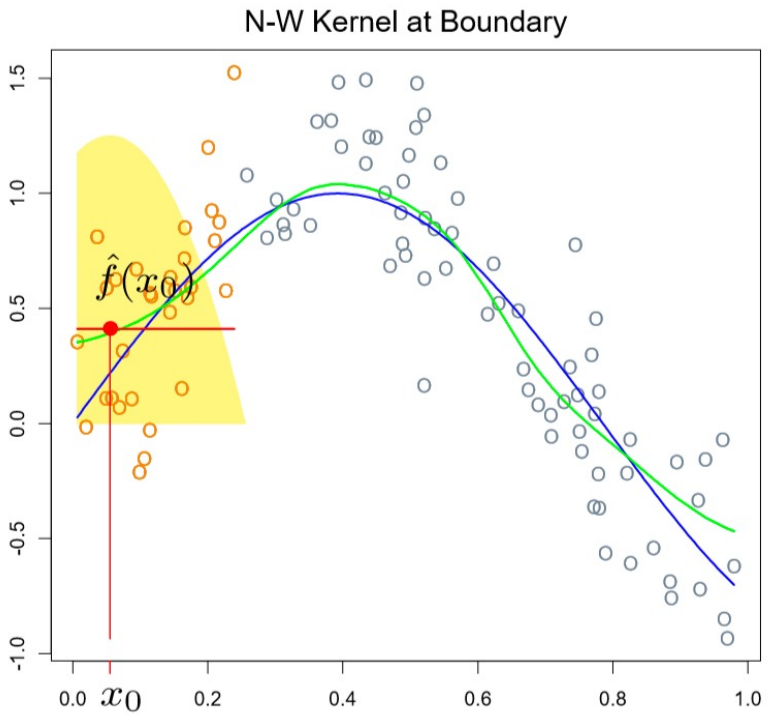
For example, $h_\lambda\left(\vec{x}^{(0)}\right) = \lambda$ is most constant examples.

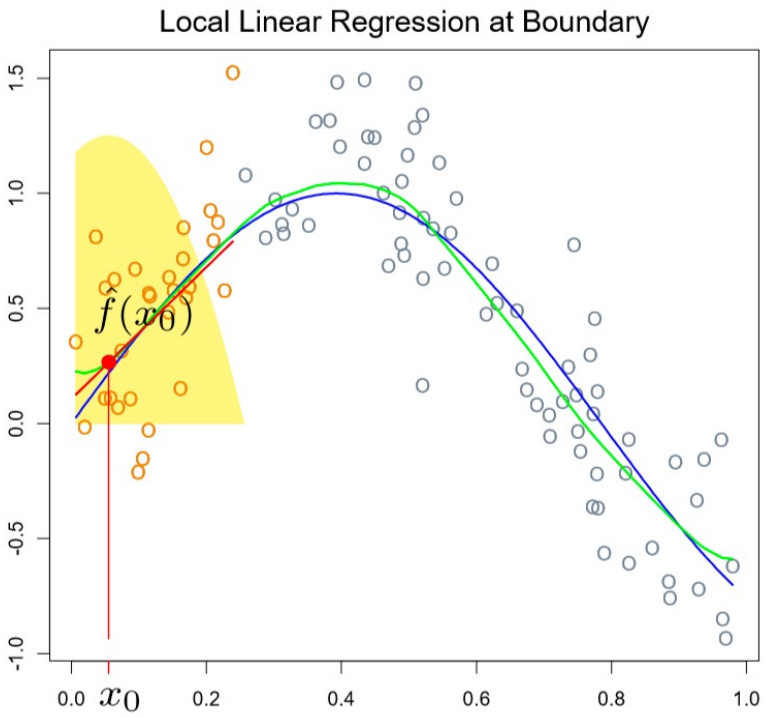
For k-NN, the neighborhood size k replaces λ, and

$$h_\lambda\left(\vec{x}^{(0)}\right) = \left\|\vec{x}^{[k]} - \vec{x}^{(0)}\right\|$$

$\vec{x}^{[k]}$ is the kth closed $\vec{x}^{(i)}$ to $\vec{x}^{(0)}$.

Boundary problems





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**.

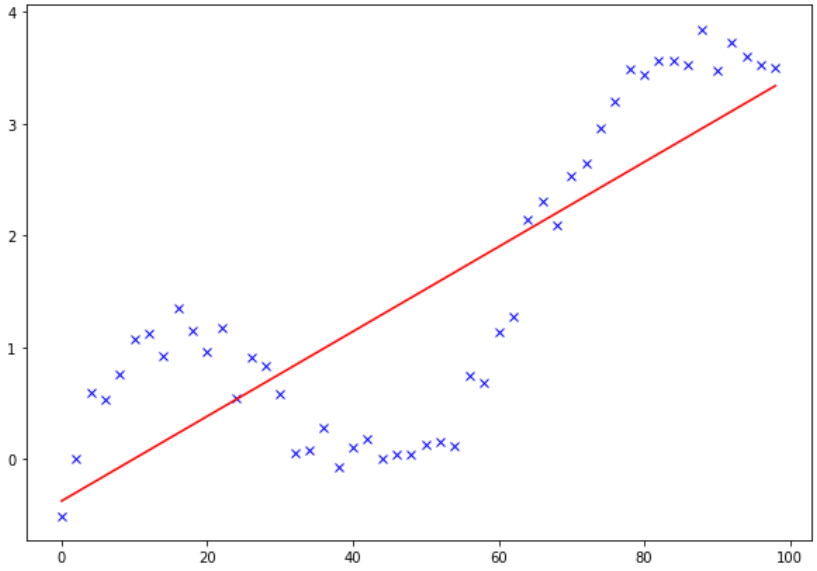
➢ Locally weighted regression

Recall Linear Regression: Find

$$f(\vec{x}) = \vec{\beta}^T \vec{x} = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$$

to minimize the RSS cost function:

$$RSS(\vec{\beta}) = \sum_{i=1}^{n} (f(\vec{x}^{(i)}) - \vec{y}^{(i)})^2$$



Goal: **Fit f locally around certain $\vec{x}_0$.**

Local linear regression attempts to solve a separate weighted least squared problem at each target point $\vec{x}_0$. The **local weighted loss** at $\vec{x}_0$ is a function

$$RSS_W(\vec{\theta}) = \sum_{i=1}^{n} w^{(i)} (f_{\vec{\beta}}(\vec{x}^{(i)}) - \vec{y}^{(i)})^2$$

$$w^{(i)} := K_\lambda\left(\vec{x}^{(i)}, \vec{x}^{(0)}\right) = D\left(\frac{\left\|\vec{x}^{(i)} - \vec{x}^{(0)}\right\|}{\lambda}\right)$$

Denote diagonal **weight matrix** $W = \begin{bmatrix} w^{(1)} & & & \\ & w^{(2)} & & \\ & & \ddots & \\ & & & w^{(n)} \end{bmatrix}$

Matrix notation of the weighted cost:

$$J_W\left(\vec{\beta}\right) = RSS_W\left(\vec{\beta}\right) = \left(X\vec{\beta} - \vec{y}\right)^T W\left(X\vec{\beta} - \vec{y}\right) = \left\|X\vec{\beta} - \vec{y}\right\|_W^2$$

- More general definition of the kernel using Mahalanobis distance:

  Given a positive semidefinte matrix A, we can define **structured kernel**:

$$K_{\lambda,A}\left(\vec{x}^{(i)}, \vec{x}^{(0)}\right) = D\left(\frac{\left\|\vec{x}^{(i)} - \vec{x}^{(0)}\right\|_A}{\lambda}\right) = D\left(\frac{\left(\vec{x}^{(i)} - \vec{x}^{(0)}\right)^T A\left(\vec{x}^{(i)} - \vec{x}^{(0)}\right)}{\lambda}\right)$$

- **Minimize** the new weighted cost function at $\vec{x}_0$

Claim: $\nabla_{\vec{\theta}} J = 2X^T W(X\vec{\beta} - \vec{y})$

So, the **optimization** solution is $\hat{\vec{\beta}} = (X^T WX)^{-1} X^T W\vec{y}$

- **The prediction is**

$$f(\vec{x}^{(0)}) = \vec{x}^{(0)^T}\hat{\vec{\beta}} = \vec{x}^{(0)^T}(X^T WX)^{-1}X^T W\vec{y} = \sum_{i=1}^{N} \ell_i(\vec{x}^{(0)})y^{(i)}$$

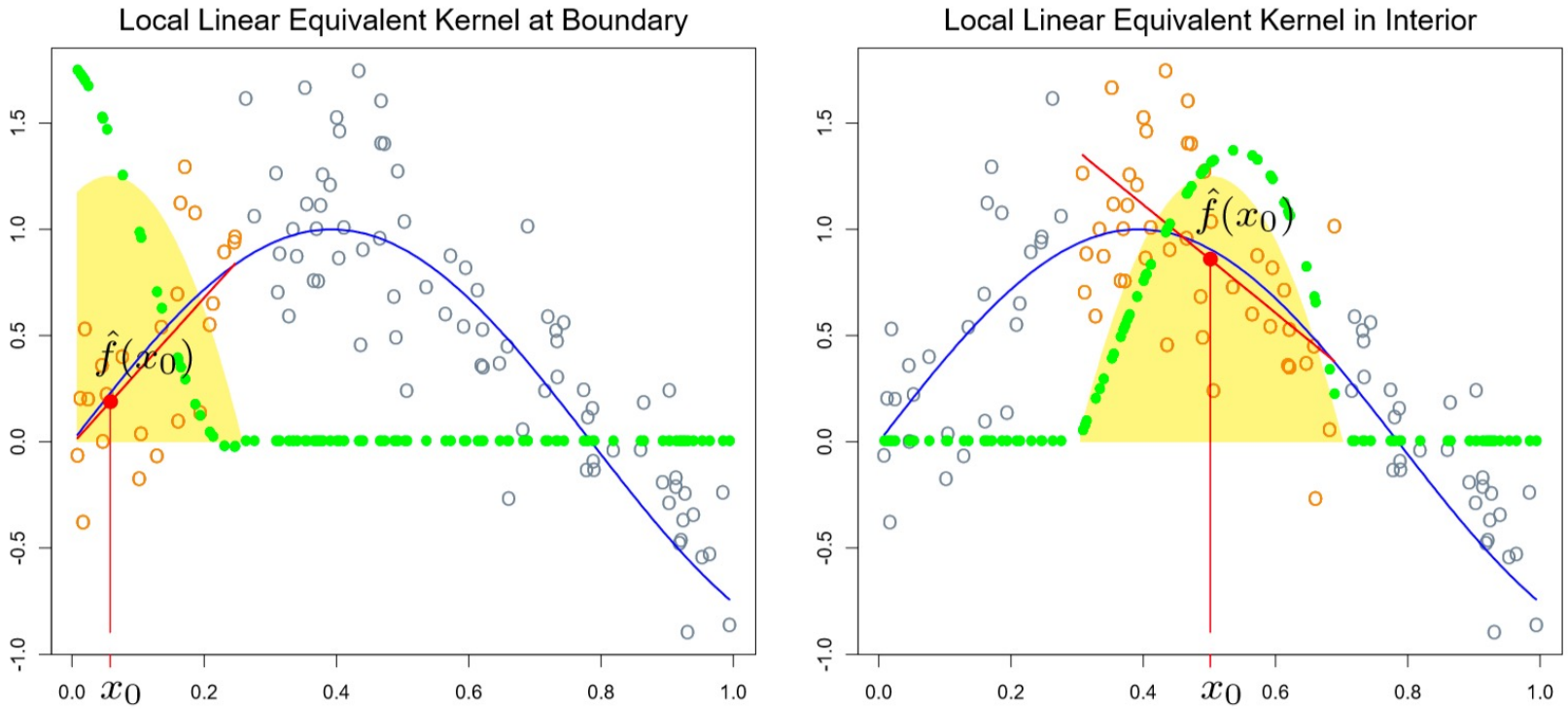
The prediction vector

$$\vec{y} = \begin{bmatrix} \hat{f}(\vec{x}^{(1)}) \\ \hat{f}(\vec{x}^{(2)}) \\ \vdots \\ \hat{f}(\vec{x}^{(N)}) \end{bmatrix} = \begin{bmatrix} \vec{l}(\vec{x}^{(1)})^T \\ \vec{l}(\vec{x}^{(2)})^T \\ \vdots \\ \vec{l}(\vec{x}^{(N)})^T \end{bmatrix} = S_W\vec{y}$$

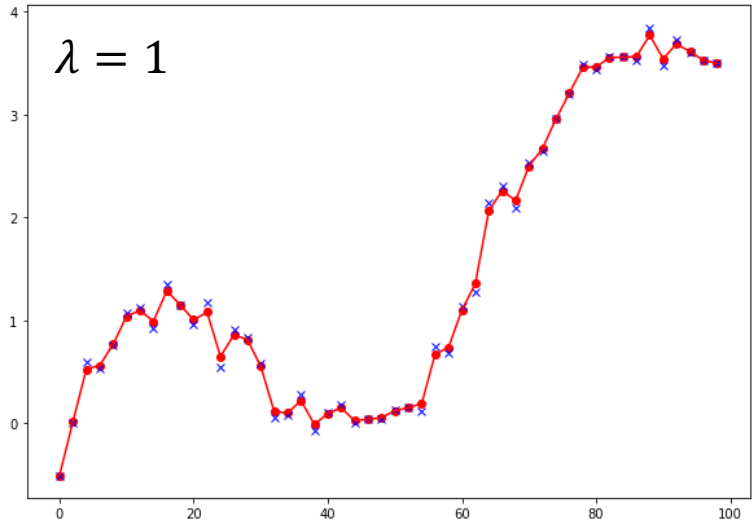
Here $S_W = X(X^T W\mathrm{X})^{-1}X^T W$ is *called the the smoothing matrix.*

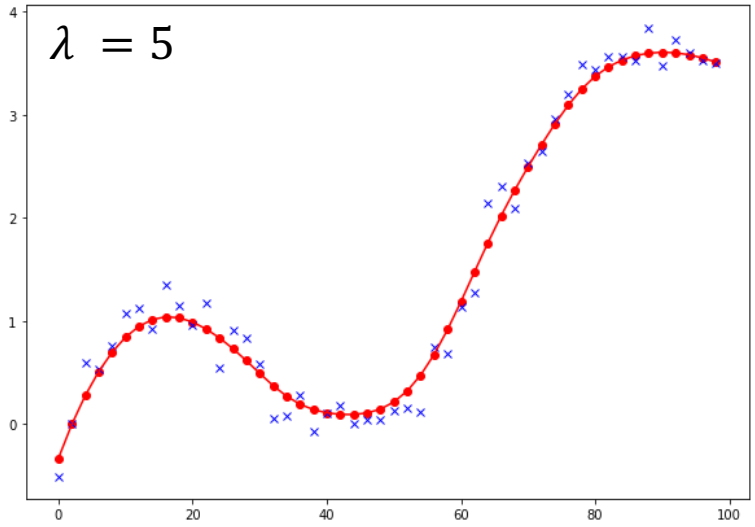
As before we can define the "**effective degrees of freedom**"

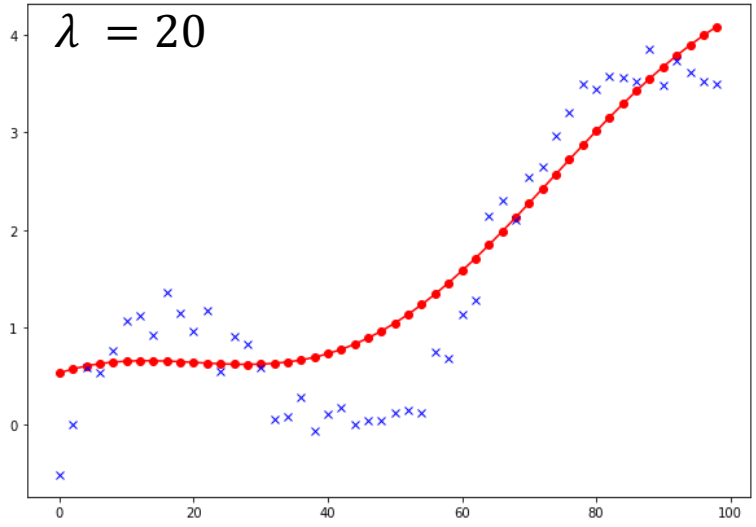
$$\mathrm{df}_\lambda = Tr(S_\lambda)$$

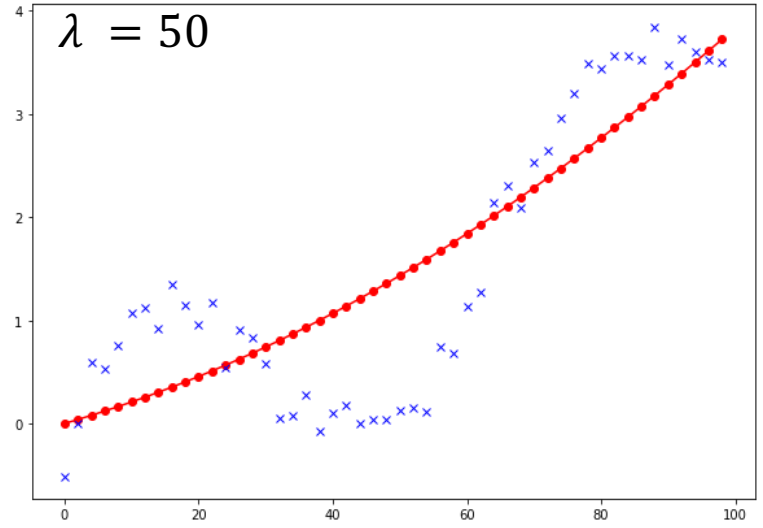


In the figure above, we see the weights $\ell_i(\vec{x}^{(0)})$ (green dots) for different $\vec{x}^{(0)}$ as we move through the dataset.









We need the training data as well as the parameters to make a prediction.

Polynomial regression

We have learned polynomial regression by itself and as a special case of the basis fuctions, we know that the method is the same as linear least squares method by introducing new variables.

Training data: $D = \{(\vec{x}^{(i)}, y^{(i)}) \mid i = 1, \dots n\}$

For example, consider dimension 2 data case. Then we have the new features z_i (or a set of basis functions $N_i(\vec{x})$)

$z_1 = x_1, z_2 = x_2,$

$z_3 = x_1^2, z_4 = x_2^2, z_5 = x_1 x_2,$

$z_6 = x_1^3, \ z_7 = x_2^3, \ z_8 = x_1^2 x_2, \ z_9 = x_1 x_2^2$, …

New Training data: $\widetilde{D} = \{(\vec{z}^{(i)}, y^{(i)}) \mid i = 1, \dots n\}$ and a **new data matrix** Z

Polynomial regression assumption

$$f(\vec{x}) = \vec{\beta}^T \vec{z} = \vec{z}^T \vec{\beta} = \beta_0 + \beta_1 z_1 + \cdots + \beta_m z_m$$

Find $\vec{\beta}$ to minimize $RSS(\vec{\boldsymbol{\beta}}) = \sum_{i=1}^{n} \left(f\left(\vec{x}^{(i)}\right) - y^{(i)}\right)^2 = \left\|Z\vec{\beta} - \vec{y}\right\|^2$

$$\hat{\vec{\beta}} = \text{argmin}\, RSS(\vec{\boldsymbol{\beta}}) = (Z^T Z)^{-1} Z^T \vec{y}$$

The prediction function is

$$f(\vec{x}) = \vec{z}^T \hat{\vec{\beta}} = \vec{z}^T (Z^T Z)^{-1} Z^T \vec{y}$$

Local polynomial regression

For local polynomial regression, consider training set $D = \{(\vec{x}^{(i)}, y^{(i)}) \mid i = 1, \dots n\}$

And the new Training data: $\widetilde{D} = \{(\vec{z}^{(i)}, y^{(i)}) \mid i = 1, \dots n\}$ and a **new data matrix** Z

Fix a **test point** $\vec{x}^{(0)}$ and then we have the test point in new features $\vec{z}^{(0)}$

Polynomial model: $f_{\vec{\beta}}(\vec{x}) = \vec{\beta}^T \vec{z} = \vec{z}^T \vec{\beta} = \beta_0 + \beta_1 z_1 + \cdots + \beta_m z_m$

Local weighted cost

$$RSS_W(\vec{\beta}) = \sum_{i=1}^{n} w^{(i)} \, (f_{\vec{\beta}}(\vec{x}^{(i)}) - \vec{y}^{(i)})^2 = \left\| Z\vec{\beta} - \vec{y} \right\|_W^2$$

$$w^{(i)} := K_\lambda(\vec{x}^{(i)}, \vec{x}^{(0)}) = D\left(\frac{\left\| \vec{x}^{(i)} - \vec{x}^{(0)} \right\|}{\lambda} \right)$$

The **optimization** solution is $\hat{\vec{\beta}} = (Z^T W \mathrm{Z})^{-1} Z^T W \vec{y}$

The prediction is

$$\hat{f}\left(\vec{x}^{(0)}\right) = \vec{z}^{(0)^T} \hat{\vec{\beta}} = \vec{z}^{(0)^T} (Z^T W \mathrm{Z})^{-1} Z^T W \vec{y} = \sum_{i=1}^{N} \ell_i\left(\vec{x}^{(0)}\right) y^{(i)} = \vec{\ell}\left(\vec{x}^{(0)}\right)^T \vec{y}$$

The prediction vector

$$\vec{y} = \begin{bmatrix} \hat{f}\left(\vec{x}^{(1)}\right) \\ \hat{f}\left(\vec{x}^{(2)}\right) \\ \vdots \\ \hat{f}\left(\vec{x}^{(N)}\right) \end{bmatrix} = \begin{bmatrix} \vec{l}\left(\vec{x}^{(1)}\right)^T \\ \vec{l}\left(\vec{x}^{(2)}\right)^T \\ \vdots \\ \vec{l}\left(\vec{x}^{(N)}\right)^T \end{bmatrix} = S_W \vec{y}$$

Here $S_W = Z(Z^T W \mathrm{Z})^{-1} Z^T W$ is *called the the smoothing matrix.*

This smoothing matrix plays the same role as in previous examples (including ridge regression).

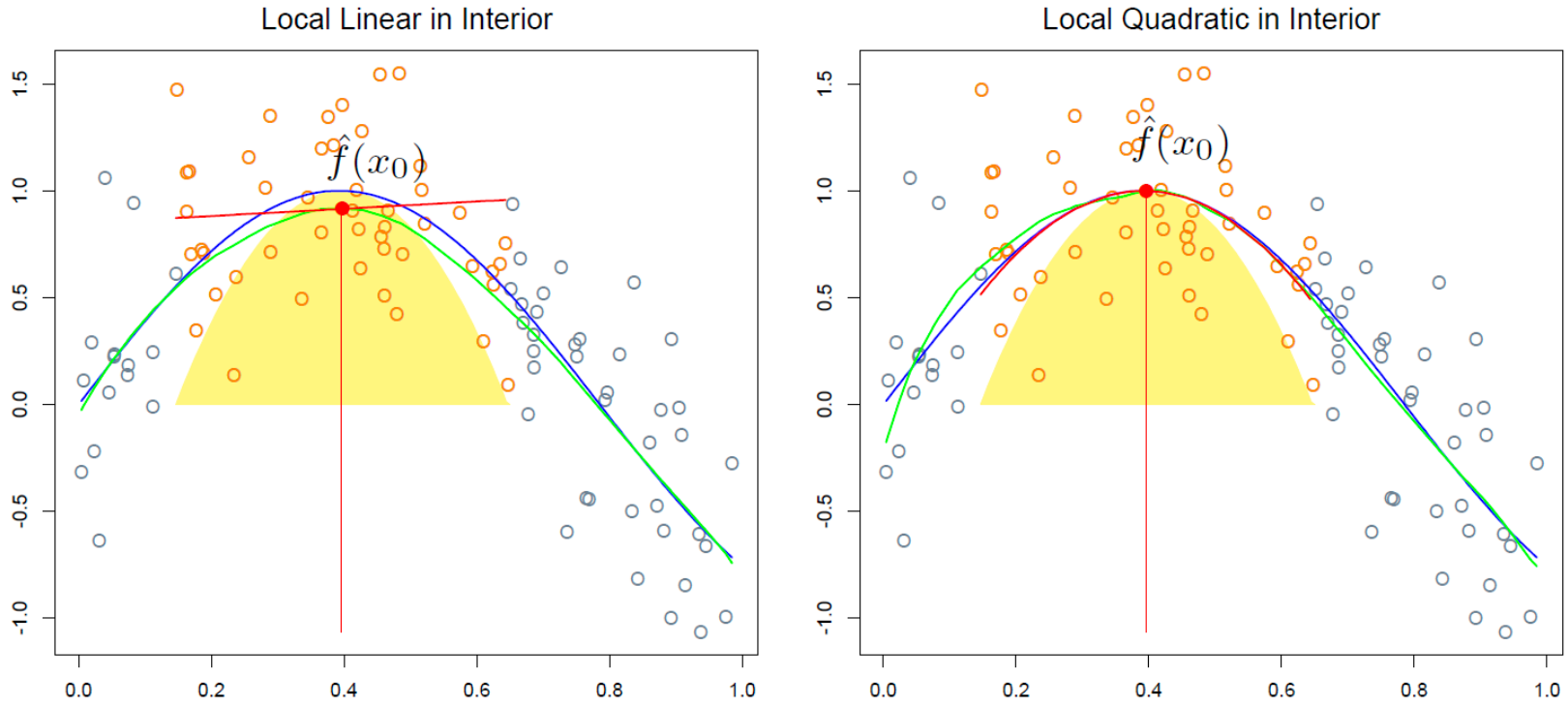
The "effective degrees of freedom"

$$df_\lambda = Tr(S_\lambda)$$

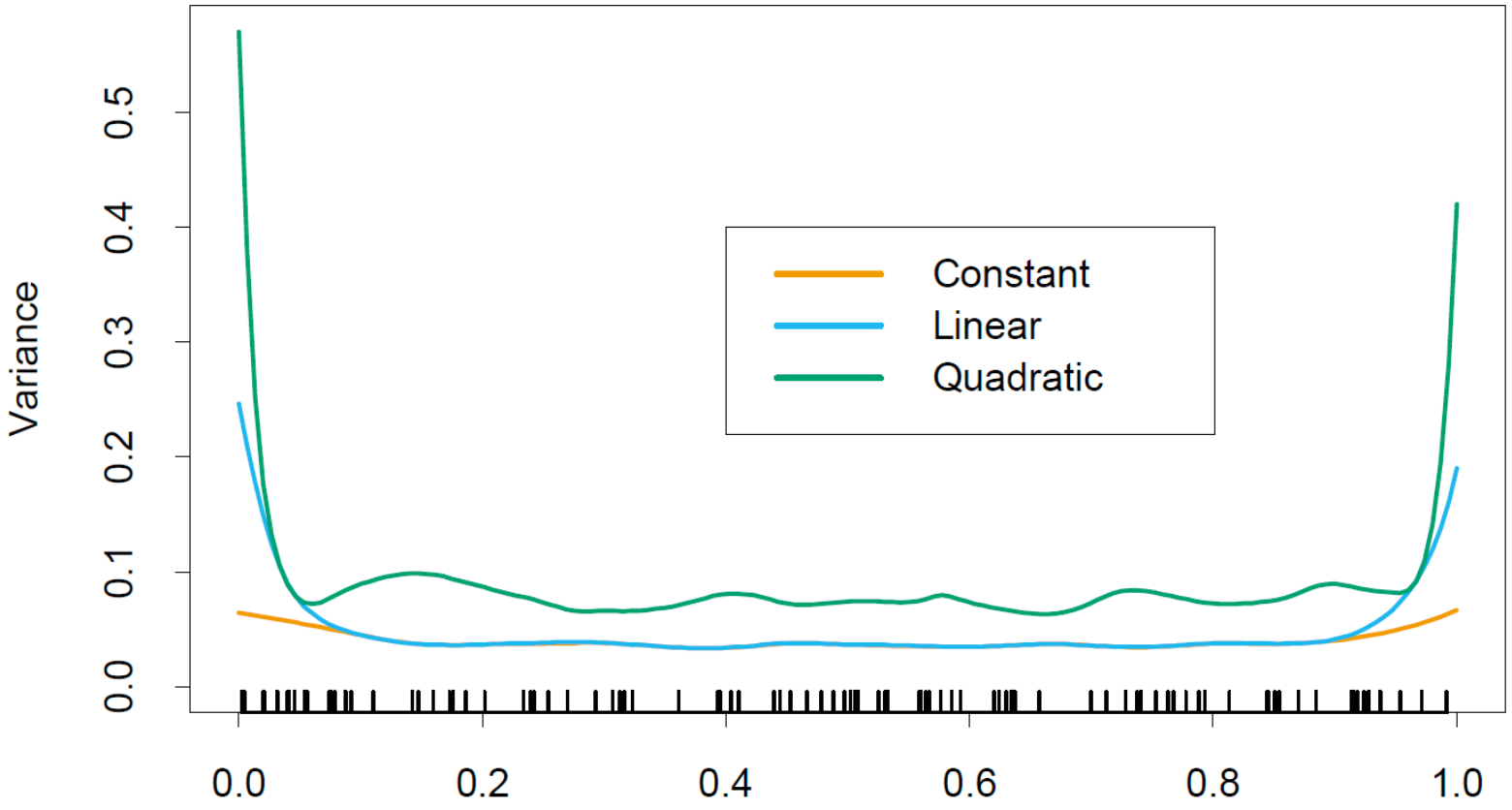
Rules of thumb

Local linear approximation fits (or odd order more generally) is best for fitting near the boundary. (removes bias)

Local quadratic approximation often captures curvature better inside the interval.



Quadratic regression reduces the bias by allowing for curvature.
Higher order regression also increases variance of the estimated function.


Variance
0.0
0.1
0.2
0.3
0.4
0.5
0.0
0.2
0.4
0.6
0.8
1.0
Constant
Linear
Quadratic

➢ Automatic Kernel Carpentry

We will show below that for increasing polynomial degree m, the bias of $\hat{f}(\vec{x}^{(0)})$ decreasing. For simplicity, let us assume the dimension is 1.

We also assume that the data is generated from

$$y = f(x) + \epsilon$$

such that $f(x)$ 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

$$\sum_{i=1}^{N} l_i(x^{(0)}) = 1 \qquad \sum_{i=1}^{N} (x^{(i)} - x^{(0)}) l_i(x^{(0)}) = 0$$

provided K is a kernel in the sense defined before.

Using the Taylor series expansion of $f(x)$ around $x^{(0)}$,

$$f(x) = f\left(x^{(0)}\right) + f'^{\left(x^{(0)}\right)}\left(x - x^{(0)}\right) + \frac{1}{2}f'^{\left(x^{(0)}\right)}\left(x - x^{(0)}\right)^2$$

$$+ \cdots + \frac{1}{m!}f'^{\left(x^{(0)}\right)}\left(x - x^{(0)}\right)^m + O\left[\left(x - x^{(0)}\right)^{m+1}\right]$$

Here, **Big *O* notation** means that: $g(x) = O\left(h(x)\right)\ when\ (x \to a)\ for\ fixed\ a$, if there exists a constant C such that $|g(x)| \leq C \cdot h(x)$ for x sufficiently close to a.

Expand $E\left[\hat{f}\left(x^{(0)}\right)\right]$, and notice that $E\left(y^{(i)}\right) = f\left(x^{(i)}\right)$, so

$$E\left[\hat{f}\left(x^{(0)}\right)\right] = E\left[\sum_{i=1}^{N} \ell_i\left(\vec{x}^{(0)}\right)y^{(i)}\right] = \sum_{i=1}^{N} \ell_i\left(\vec{x}^{(0)}\right)f\left(x^{(i)}\right) = f\left(x^{(0)}\right)\sum_{i=1}^{N} l_i\left(x^{(0)}\right) +$$

$$f'\left(x^{(0)}\right)\sum_{i=1}^{N} l_i\left(x^{(0)}\right)\left(x^{(i)} - x^{(0)}\right) + \frac{1}{2}f''\left(x^{(0)}\right)\sum_{i=1}^{N} l_i\left(x^{(0)}\right)\left(x^{(i)} - x^{(0)}\right)^2 + h.o.t$$

But the conditions (for **local linear regression**) above mean that

$$\hat{f}\left(x^{(0)}\right) = f\left(x^{(0)}\right) + \frac{1}{2} f''\left(x^{(0)}\right) \sum_{i=1}^{N} l_i\left(x^{(0)}\right)\left(x^{(i)} - x^{(0)}\right)^2 + h.o.t.$$

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

$$Bias\left[\hat{f}\left(x^{(0)}\right)\right] = \hat{f}\left(x^{(0)}\right) - f\left(x^{(0)}\right)$$

$$= \frac{1}{2} f''\left(x^{(0)}\right) \sum_{i=1}^{N} l_i\left(x^{(0)}\right)\left(x^{(i)} - x^{(0)}\right)^2 + h.o.t.$$

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. (similarly for high order local polynomial regressions.)

For **m degree local polynomial regressions,**

$$\sum_{i=1}^{N} l_i\left(x^{(0)}\right) = 1 \qquad \sum_{i=1}^{N} \left(x^{(i)} - x^{(0)}\right)^k l_i\left(x^{(0)}\right) = 0$$

For $k = 1,2, \ldots, m$

Also can show that the exact variance at $x^{(0)}$ is

$$Var\left(\hat{f}\left(x^{(0)}\right)\right) = \sigma^2 \left\|\vec{l}\left(x^{(0)}\right)\right\|^2$$

This result is known as **automatic kernel carpentry**.
https://www.jstor.org/stable/2246148

➢ Practical concerns

For fixed λ, **large bandwith** λ implies **low variance** since we're averaging over more data points but **higher bias**. (Bias-variance trade-off)

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.

-- Choice of kernel function less important.
-- Choice of kernel bandwidth, i.e., width, quite important.

Generally we can choose λ adaptively, i.e., by trial and error.

Selecting the bandwidth

Predicted test error

$$C_p = \frac{\sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right)}{N} + 2\sigma^2 Tr(S_\lambda)$$

This is an unbiased estimate of prediction error at the test point $\vec{x}^{(0)}$ known as the C_p **criterion**

Note that here $Tr(S_\lambda)$ is a generalization of the number p of predictors $\vec{x}^{(i)}$ that we had in linear regression, i.e. it's the df.

How to find the best λ?

Can try to minimize C_p above after estimating σ or just use cross-validation.

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.

➢ Density Estimation and Classification

Suppose we wish to estimate the **density** of the RV pair $(\vec{X}, Y)$ with $\vec{X}$ =predictor and Y =outcome or label. How can we do it?

Ignore Y for now since it is treated now exactly like X_i components of $\vec{X}$

Assuming random vector $\vec{X} \in \mathbb{R}^d$, and it has a density function $f(\vec{x})$.

We may want to estimate the density function $f(\vec{x})$ using a training dataset $D = \{\vec{x}^{(i)}\}_{i=1}^{N}$ directly, before trying to do any regression tasks.

We want a 'density estimator' that uses some of the above kernel ideas.

Assume we have a kernel function $K_\lambda\left(\vec{x}^{(0)}, \vec{x}^{(i)}\right)$ (which is large only when $\vec{x}^{(i)}$ close to $\vec{x}^{(0)}$).
Assume the width of the 'bump' (as a function of $\vec{x}^{(i)}$ for fixed $\vec{x}^{(0)}$) is about λ.

Based on this and the training set D, we can form an estimate of the density function $f(\vec{x})$ of X. it is a sum over the training points.

A natural local estimate (bumpy) example is as

$$\hat{f}_\lambda\left(x^{(0)}\right) = \frac{1}{N\lambda}\sum_{i=1}^{N} \mathbb{I}\left(\vec{x}^{(i)} \in \mathcal{N}\left(\vec{x}^{(0)}\right)\right) = \frac{\#\left(\vec{x}^{(i)} \in \mathcal{N}\left(\vec{x}^{(0)}\right)\right)}{N\lambda}$$

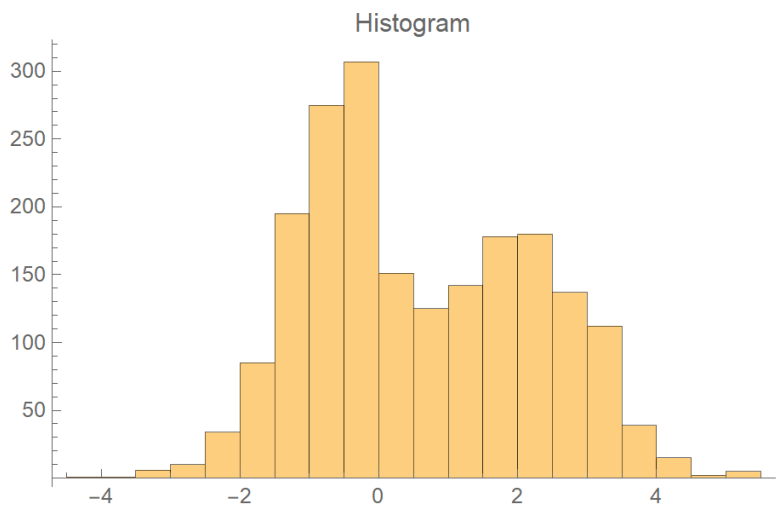
$\mathcal{N}\left(\vec{x}^{(0)}\right)$ is a small metric neighborhood around $\vec{x}^{(0)}$ of width λ

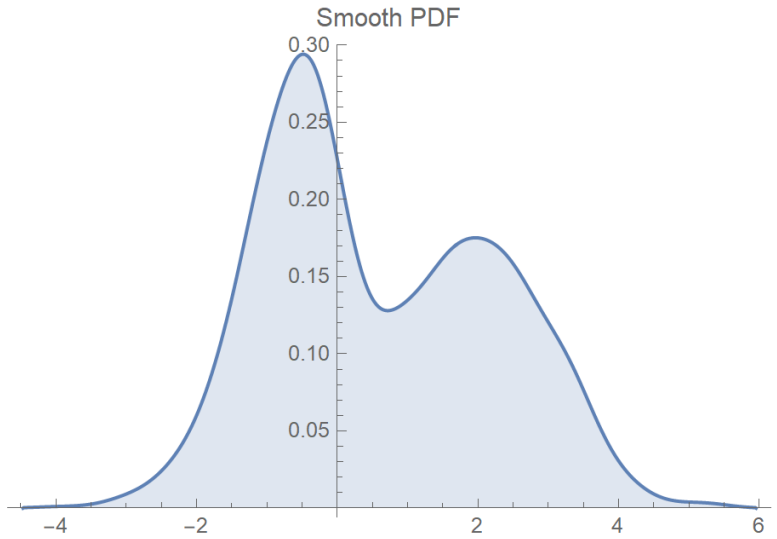
We prefer the smooth Parzen estimator:

$$\hat{f}_\lambda\left(x^{(0)}\right) = \frac{1}{N\lambda}\sum_{i=1}^{N} K_\lambda\left(\vec{x}^{(0)}, \vec{x}^{(i)}\right)$$

Example: Gaussian density estimate:

$$\hat{f}_\lambda\left(x^{(0)}\right) = \frac{1}{N(2\lambda^2\pi)^{\frac{d}{2}}} \sum_{i=1}^{N} e^{-\frac{1}{2}\left(\frac{\left\|\vec{x}^{(0)}-\vec{x}^{(i)}\right\|}{\lambda}\right)^2}$$





Kernel Density Classification

using the usual **Bayes' theorem**, same as in LDA

$$P(Y = j \,|\vec{X} = \vec{x}) = \frac{P\left(\vec{X} = \vec{x}\middle|Y = j\right)P(Y = j)}{\sum_{all\ k} P\left(\vec{X} = \vec{x}\middle|Y = k\right)P(Y = k)}$$

This suggests a simple way to create classifier (if we also have outcome data $y^{(i)}$ in the training set):

$$\hat{P}\left(Y = j\middle|\vec{X} = \vec{x}^{(0)}\right) = \frac{\hat{\pi}_j \hat{f}_j\left(\vec{x}^{(0)}\right)}{\sum_{k=1}^{K} \hat{\pi}_k \hat{f}_k(\vec{x}^{(0)})}$$

Here, $\hat{\pi}_j$ estimate the j-th class prevalence, i.e., $\hat{\pi}_j = \frac{\mathrm{N_j}}{\mathrm{N}}$.

The Naive Bayes Classifier is another example we already learned.

Advantages:

Kernel methods have received much attention in literature and application.

They are useful for visualizations or for finding a parameter or estimating a function of (possibly) less theoretical interest.

[i.e. if the goal is to just to obtain a function or number without worrying about higher 'principles']

They are useful in low dimension, especially 1 dimension, like time series.

Disadvantages:

1. These kernel methods provide less theoretical 'insight'

- don't "explain" anything.
- they "draw" a sophisticated curve.

2. Also don't scale well to high dimension

3. Somewhat more complicated explanations of data than from our other more structured methods

4. Fitting gets done at evaluation time, memory-based methods require in principle little or no training, similar as kNN (https://en.wikipedia.org/wiki/Lazy_learning)

Problems in high dimension:

1. Curse of Dimensionality. 2. More points on boundary. 3.Non-visualizable.

Relationship/difference between **kernel smoothing methods** and **kernel methods** - confused due to **abuse of terminology**.

Kernel Methods:

- Rise from dual representation.
- Inner product of the (usually in higher dimension) feature vectors.
- The advantage of such representations is: we can therefore work directly in terms of of kernels and avoid explicit introduction of the feature vector $\phi(x)$.
- A more general idea containing concepts such as linear kernel regression/classification, kernel PCA, kernel SVM, Gaussian process etc.

Kernel Smoothing Methods

- Basically it specify the methods for deriving more smooth and less biased fitting curves
- The similarity of these two concepts are they share lots of basic kernel function forms such as Gaussian kernel or radial basis function.

[**Hastie**] Chapter 6