Nonparametric Smoothing 1

-Quantifying the World-

Lecturer: Darren Homrighausen, PhD

Local averaging

The set-up (Reminder)

We observe n pairs of data $(X_1, Y_1), \ldots, (X_n, Y_n)$

Let $Z_i = (X_i, Y_i) \in \mathbb{R}^p \times \mathbb{R}$

We'll refer to the training data as $\mathcal{D} = \{Z_1, \ldots, Z_n\}$

Call Y_i the response, while X_i is the feature or covariate

Example: Y_i is whether a threat is detected in an image and the X_{ij} is the value at the j^{th} pixel of an image (p might be $1024^2 = 1048576$)

From linear to nonlinear models

Goal: Develop a prediction function $\hat{f} : \mathbb{R}^p \rightarrow \mathbb{R}$ for predicting Y given an X

Commonly, $\hat{f}(X) = X^\top \beta$

(least squares regression)

This greatly simplifies algorithms, while not sacrificing too much flexibility

However, sometimes directly modeling the nonlinearity is more natural

Prediction via local averaging

Remember: We would like to estimate the regression function:

$$f_*(X) = \mathbb{E}[Y|X]$$

We know how to estimate expectations: if $Y_1, Y_2, \ldots, Y_n$ all have expectation μ, then

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} Y_i$$

is an intuitive estimator of μ

(and a reasonable prediction of a new Y)

However, we have the paired observations

$$(X_1, Y_1), \ldots, (X_n, Y_n)$$

Prediction via local averaging

Similarly, we can estimate $\mathbb{E}[Y|X]$ with our data $(X_1, Y_1), \ldots, (X_n, Y_n)$

$$\hat{f}(X) = \frac{1}{n_X} \sum_{i=1}^{n} Y_i \,\mathrm{Does}(X_i = X)$$

where $n_X = \sum_{i=1}^{n} \mathrm{Does}(X_i = X)$.

Definition of "Does": 1 if the condition is true and 0 if not

(In this case, if $X_i = X$. Note that this is commonly called an "indicator function")

In words: We are taking an average of all the observations Y_i such that $X_i = X$. This is all conditional expectation really is!

Prediction via local averaging

This would work fine, as long as there are a lot of $X_i = X$

However, there generally aren't any $X_i = X$!

Suppose we relax the constraint $\mathrm{Does}(X_i = X)$ a bit and include points that are close enough instead

Again, suppose we have data $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$

$$\hat{f}(X) = \frac{1}{n_X} \sum_{i=1}^{n} Y_i \,\mathrm{Near}(X_i, X)$$

where $n_X = \sum_{i=1}^{n} \mathrm{Near}(X_i, X)$.

Now, $\mathrm{Near}(X_i, X)$ needs to be defined

Prediction via local averaging

$\mathrm{Near}(X_i, X)$ can be defined via

- Nearest neighbors: We can say that X_i is near to X if X_i is one of $X's$ K nearest neighbors
- Distance: We can say that X_i is near to X if the distance between X_i and X is less than some threshold, t
  (Like $d(X_i, X) = \sum_{j=1}^{p}(X_{ij} - X_j)^2$, and $\mathrm{Near}(X_i, X) = \mathrm{Does}(d(X_i, X) < t)$)

Here, t and K quantify nearness

(In fact, they are both tuning parameters)

Prediction via local averaging

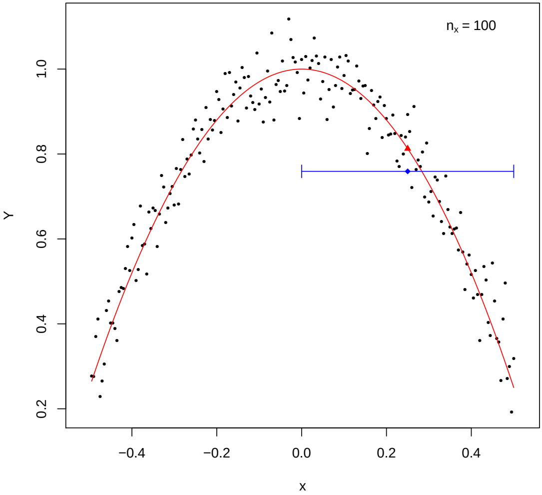


Figure: $t = 0.25$

Prediction via local averaging

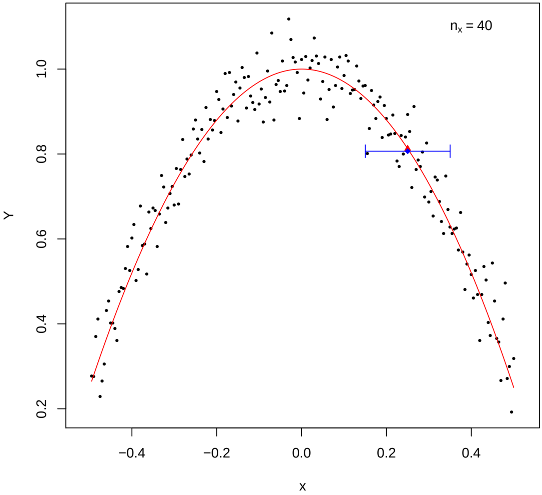


Figure: $t = 0.1$

Prediction via local averaging

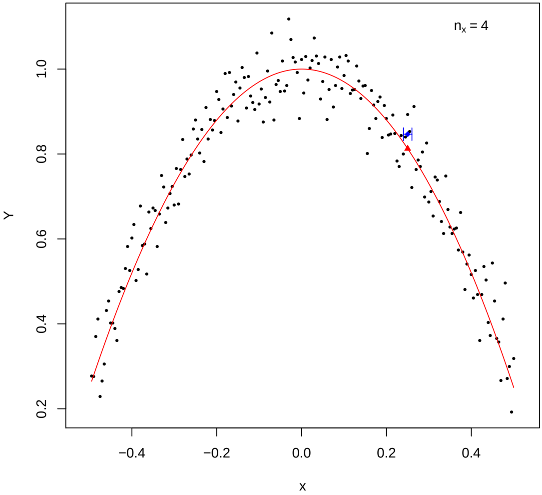


Figure: $t = 0.01$

Prediction via local averaging

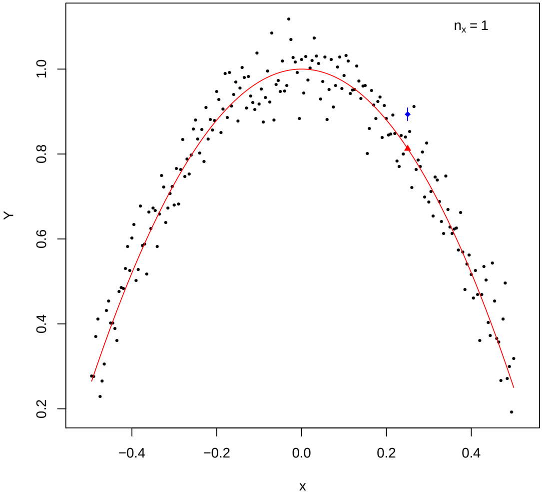
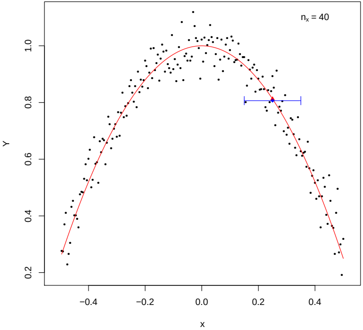


Figure: $t = 0.0001$

Prediction via local averaging



In this case, $t = 0.1$ gets the right amount of nearness. Though it includes some X_i that are $\pm$ 0.1 away from X, we still get a good estimate.

Multiple regression

Reminder: To fit the classic multiple regression, we would do

$$\min_{\beta_0,\beta} \sum_{i=1}^{n} (Y_i - \beta_0 - \beta^\top X_i)^2$$

If X_i is a number (say in time series or the example), then

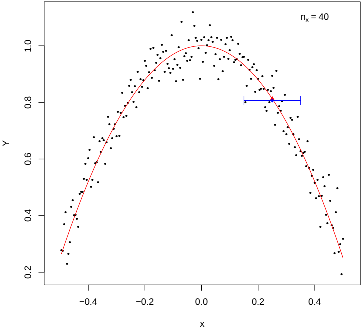
$$\min_{\beta_0,\beta} \sum_{i=1}^{n} (Y_i - \beta_0 - \beta X_i)^2$$

(This is simple linear regression)

We would get predictions like

$$\hat{f}(X) = \hat{\beta}_0 + \hat{\beta} X$$

Multiple regression



Pretty clearly, simple linear regression would not work well here

We could add polynomial terms (a quadratic, say)

But, is there a more flexible way?

Multiple regression to Loess

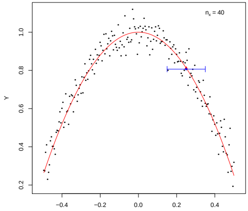
The idea: Take least squares and reweight it

$$\sum_{i=1}^{n}(Y_i - \beta_0 - \beta X_i)^2 = \sum_{i=1}^{n}(Y_i - \beta_0 - \beta X_i)^2 1$$

$$\Rightarrow \sum_{i=1}^{n}(Y_i - \beta_0 - \beta X_i)^2 \mathrm{Near}(X_i, X)$$

(Like $d(X_i, X) = \sum_{j=1}^{p}(X_{ij} - X_j)^2$, and $\mathrm{Near}(X_i, X) = \mathrm{Does}(d(X_i, X) < t^2)$)

That is what we did here (with $\beta = 0$)



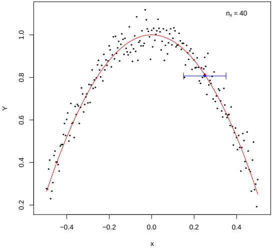
Prediction via local averaging: Loess

From the lectures, the Loess fit looks to minimize:

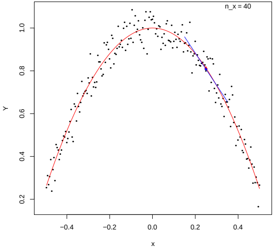
$$\sum_{i=1}^{n}(Y_i - \beta_0 - \beta X_i)^2 \mathrm{Near}(X_i, X) = \sum_{i \in N_K(X)} (Y_i - \beta_0 - \beta X_i)^2 W\left(\frac{|X - X_i|}{\Delta_X}\right)$$

and

- $N_K(X)$ are the indices of the K nearest neighbors to X
- $\Delta_X = \max_{i \in N_K(X)} |X_i - X|$, which plays the role of t

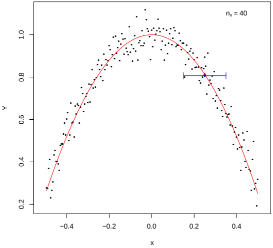


$\beta = 0$

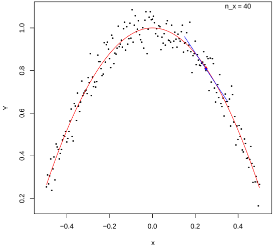


$\beta \neq 0$

Loess

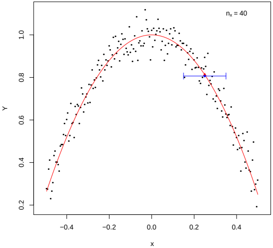


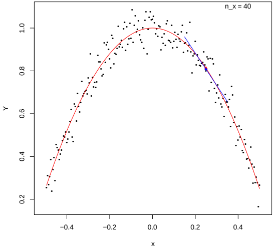
$n_x = 40$
1.0
0.8
0.6
0.4
0.2
Y
−0.4
−0.2
0.0
0.2
0.4
x

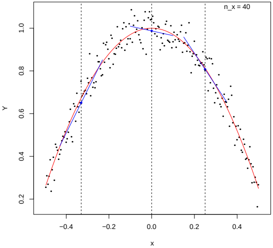


n_x = 40
1.0
0.8
0.6
0.4
0.2
Y
−0.4
−0.2
0.0
0.2
0.4
x

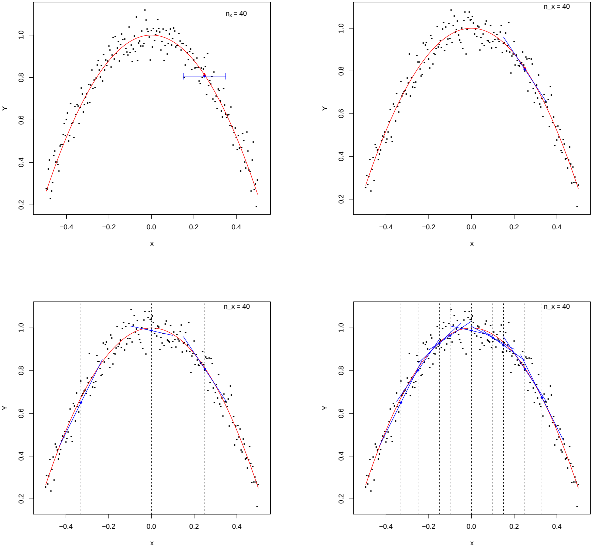
Loess







Loess



Class exercise

Using the `nenana.txt` data and the `unit7inClass.R` code, produce plots of three different methods at a variety of different tuning parameters

- Loess
- Regression Splines: This is like regressing the data on $K < n$ transformations of the features

$$\hat{f}(X) = X^\top \beta \Rightarrow \hat{f}(X) = \Phi(X)^\top \beta = \beta_0 + \sum_{j=1}^{K} \phi_j(X)\beta_j$$

- Smoothing Splines: This is like regressing the data on $K = n$ transformations of the features, but adding a ridge regression penalty