STAT 210
Applied Statistics and Data Analysis
Nonparametric Regression

Joaquin Ortega

Fall 2020

Nonparametric Regression

Nonparametric Regression

So far, in regression, we have assumed that the relation between the output Y and the regressors $\mathbf{X}$ is linear $Y = \mathbf{X}\beta$ and the parameters have been estimated using least squares

In nonparametric regression, we do not assume a predetermined form for the regression function. The shape of the regression function is determined from the data.

We will only review one case of nonparametric regression, kernel regression, linked to the density estimation procedures we studied. Also, we only look at simple regression

Nonparametric Regression

The regression function we want to estimate is the expected value of Y given the value of the regressor $X = x$. We will consider this as a function of x

$$m(x) = E(Y|X = x)$$

To simplify the presentation we will assume that both variables are continuous and have joint density $f(x, y)$. The marginal densities for X and Y are denoted $f_X(x)$ and $f_Y(y)$.

The probability distribution of Y given that $X = x$ is characterized by the conditional density of Y given $X = x$, which is defined as

$$f_{Y|X=x}(y) = \frac{f(x, y)}{f_X(x)}.$$

Nonparametric Regression

Using this conditional density, we can write the regression function as

$$\begin{aligned} m(x) = E(Y|X = x) &= \int y\, f_{Y|X=x}(y)\, dy \\ &= \int y\, \frac{f(x,y)}{f_X(x)} dy \\ &= \frac{1}{f_X(x)} \int y\, f(x,y) dy \qquad (1) \end{aligned}$$

Therefore, given a sample $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$ we may obtain a nonparametric estimate of the regression function m by estimating densities f_X and f and using them in equation (1).

Nonparametric Regression

For the bivariate density $f(x, y)$ we consider the kernel density estimator based on the product of univariate kernels with bandwidths $\mathbf{h} = (h_1, h_2)$.

$$\hat{f}(x, y; \mathbf{h}) = \frac{1}{n} \sum_{i=1}^{n} K_{h_1}(x - x_i) K_{h_2}(y - y_i) \tag{2}$$

If we use the same bandwidth for the univariate density estimation

$$\hat{f}_X(x; h_1) = \frac{1}{n} \sum_{j=1}^{n} K_{h_1}(x - x_j) \tag{3}$$

Nonparametric Regression

Replacing these equation in (1) we have

$$\begin{aligned}
m(x) &= \frac{1}{\hat{f}_X(x;h_1)} \int y \hat{f}(x,y;\mathbf{h}) dy \\
&= \frac{\int y \frac{1}{n}\sum_{i=1}^{n} K_{h_1}(x-x_i) K_{h_2}(y-y_i) dy}{\frac{1}{n}\sum_{j=1}^{n} K_{h_1}(x-x_j)} \\
&= \frac{\frac{1}{n}\sum_{i=1}^{n} K_{h_1}(x-x_i) \int y K_{h_2}(y-y_i) dy}{\frac{1}{n}\sum_{j=1}^{n} K_{h_1}(x-x_j)} \\
&= \frac{\frac{1}{n}\sum_{i=1}^{n} K_{h_1}(x-x_i) y_i}{\frac{1}{n}\sum_{j=1}^{n} K_{h_1}(x-x_j)} \\
&= \sum_{i=1}^{n} \frac{K_{h_1}(x-x_i)}{\sum_{j=1}^{n} K_{h_1}(x-x_j)} y_i
\end{aligned}$$

This is known as the Nadaraya-Watson estimator for the regression function.

Nonparametric Regression

Therefore

$$\begin{aligned}\widehat{m}(x;0,h) &= \sum_{i=1}^{n} \frac{K_{h_1}(x-x_i)}{\sum_{j=1}^{n} K_{h_1}(x-x_j)} y_i \\ &= \sum_{j=1}^{n} W_i^0(x) y_i \qquad (4)\end{aligned}$$

where

$$W_i^0(x) = \frac{K_{h_1}(x-x_i)}{\sum_{j=1}^{n} K_{h_1}(x-x_j)}$$

Thus, the Nadaraya - Watson estimator is a weighted average of the response values $y_1, \ldots, y_n$ using weights $\{W_i^0(x)\}$, which depend upon the evaluation point.

In this sense the estimator is a local average of $y_1, \ldots, y_n$ around $X = x$.

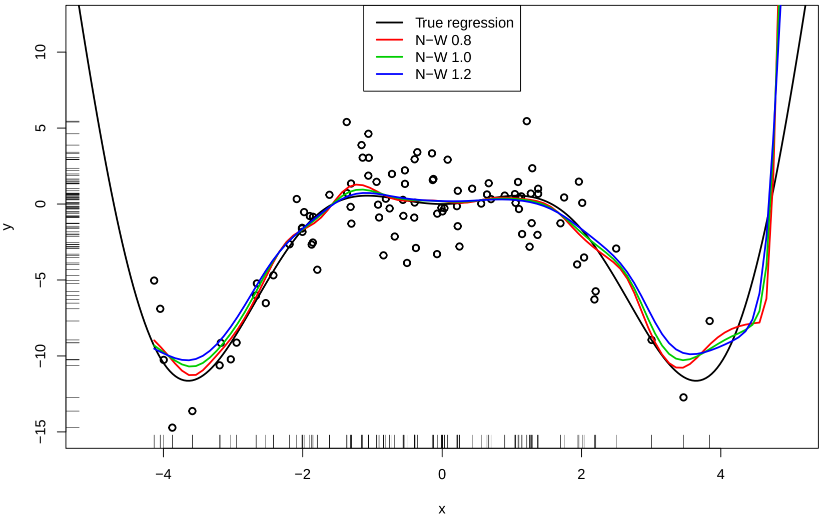
Nonparametric Regression

In R this nonparametric estimator of the regression function is implemented in the function `ksmooth()` with the option `kernel = 'normal'`.

Let's see first a simulated example where the regression function is known.

```r
n <- 100
m <- function(x) x^2 * cos(x)
x <- rnorm(n, sd=2); epsilon <- rnorm(n, sd=2)
y <- m(x) + epsilon
x_points <- seq(-10, 10, l=500)
```

Nonparametric Regression



Local Polynomial Estimators

Local Polynomial Estimators

The Nadaraya -Watson is a particular case of a broader class of nonparametric estimators, known as the *local polynomial estimators.*

We want to find an estimator $\widehat{m}$ of m that minimizes the residual sum of squares

$$\sum_{i=1}^{n}(y_i - \widehat{m}(x_i))^2 \tag{5}$$

without assuming a particular form for the true m.

In (simple) linear regression we assumed that $m(x) = \beta_0 + \beta_1 x$, so the model is parametrized by $\boldsymbol{\beta} = (\beta_0, \beta_1)$ and the minimization of (5) produced the least squares estimators $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \hat{\beta}_1)$

Local Polynomial Estimators

When m has no available parametrization, we can introduce a local parametrization using a p-th order Taylor expansion around the observed points $x_1, \ldots, x_n$.

The degree p of the approximating polynomial is usually one or two.

For x close to x_i

$$m(x) \approx m(x_i) + m'(x_i)(x - x_i) + \frac{1}{2}m''(x_i)(x - x_i)^2 + \\ \cdots + \frac{1}{p!}m^{(p)}(x_i)(x - x_i)^p$$

Local Polynomial Estimators

The derivatives are, of course, unknown, but we can turn this into a regression problem by setting

$$\beta_j = \frac{1}{j!} m^{(j)}, j = 0, \ldots, n$$

and now we want to determine the parameters $(\beta_0, \beta_1, \ldots, \beta_p)$.

Local Polynomial Estimators

The final ingredient is to weigh each point's contribution according to the distance of x_i to x. For that, we use a kernel.

The estimated parameters are

$$\hat{\beta}_h = \arg\min \sum_{i=1}^{n} \left(y_i - \sum_{j=0}^{p} \beta_j (x - x_i)^j \right)^2 K_h(x - x_i)$$

If $p = 0$, this is the Nadaraya-Watson estimator.

In `R`, local polynomials are implemented in the function `locpoly` in the package `KernSmooth`.

Local Polynomial Estimators

The first link is to the chapter on nonparametric regression in Eduardo Garcia's book, Notes for Nonparametric Statistics.

https://bookdown.org/egarpor/NP-UC3M/kre-i-kre.html

The second link is to a Shiny app for nonparametric regression

https://ec2-35-177-34-200.eu-west-2.compute.amazonaws.com/kreg/

loess

Another smoother based on these ideas is `loess` in the base package. The smoothness in loess is determined by the proportion of points in the sample taken into consideration for the local fit around x. The default value for `span` is 0.75.

The contributions are then weighted using a triweight kernel instead of a normal kernel.

This is different from the polynomial regression we have reviewed but is based on the same principles.

We give an example using both functions and polynomials of degree 0 and 1.

The example comes from *Notes for Nonparametric Statistics* by Eduardo García, https://bookdown.org/egarpor/NP-UC3M/

Example

```r
set.seed(123456)
n <- 100
eps <- rnorm(n, sd = 2)
m <- function(x) x^3 * sin(x)
X <- rnorm(n, sd = 1.5)
Y <- m(X) + eps
x_grid <- seq(-10, 10, l = 500)

# KernSmooth::locpoly fits
h <- 0.25
lp0 <- KernSmooth::locpoly(x = X, y = Y, bandwidth = h, degree = 0,
                           range.x = c(-10, 10), gridsize = 500)
lp1 <- KernSmooth::locpoly(x = X, y = Y, bandwidth = h, degree = 1,
                           range.x = c(-10, 10), gridsize = 500)
# Provide the evaluation points by range.x and gridsize

# loess fits
span <- 0.25 # The default span is 0.75, which works very bad here
lo0 <- loess(Y ~ X, degree = 0, span = span)
lo1 <- loess(Y ~ X, degree = 1, span = span)
```

Example

```r
# Prediction at x = 0 and x = 2
# Prediction by locpoly
c(lp1$y[which.min(abs(lp1$x ))],lp1$y[which.min(abs(lp1$x - 2))])
```

```
## [1] 0.9694953 5.4459745
```

```r
# Prediction by loess
predict(lo1, newdata = data.frame(X = c(0,2)))
```

```
## [1] 1.142635 5.379652
```

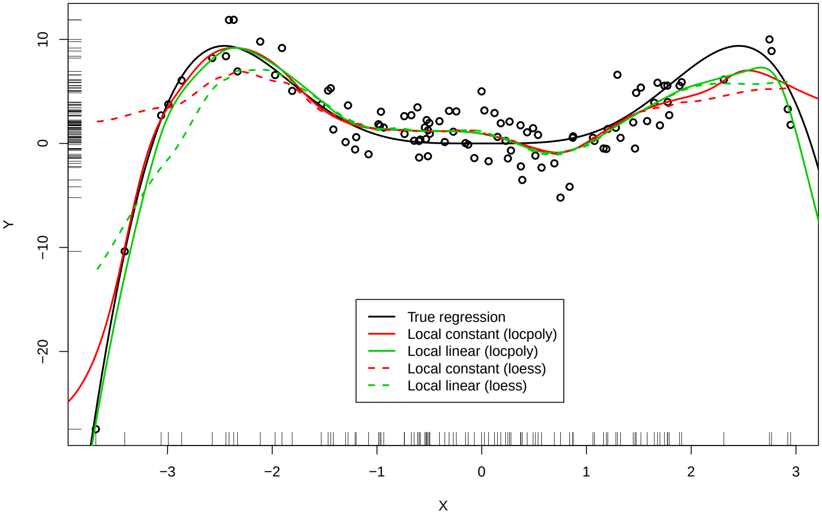
```r
m(c(0,2)) # True regression
```

```
## [1] 0.000000 7.274379
```

Example

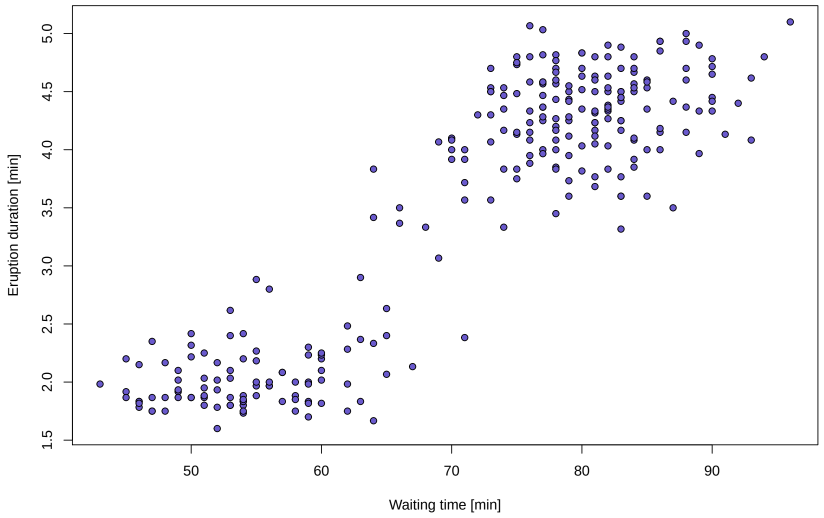
```r
plot(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(x_grid, m(x_grid), col = 1)
lines(lp0$x, lp0$y, col = 2)
lines(lp1$x, lp1$y, col = 3)
lines(x_grid, predict(lo0, newdata = data.frame(X = x_grid)),
      col = 2, lty = 2)
lines(x_grid, predict(lo1, newdata = data.frame(X = x_grid)),
      col = 3, lty = 2)
legend("bottom", legend = c("True regression", "Local constant (locpoly)",
                             "Local linear (locpoly)", "Local constant (loess)",
                             "Local linear (loess)"),
       lwd = 2, col = c(1:3, 2:3), lty = c(rep(1, 3), rep(2, 2)))
```

Example

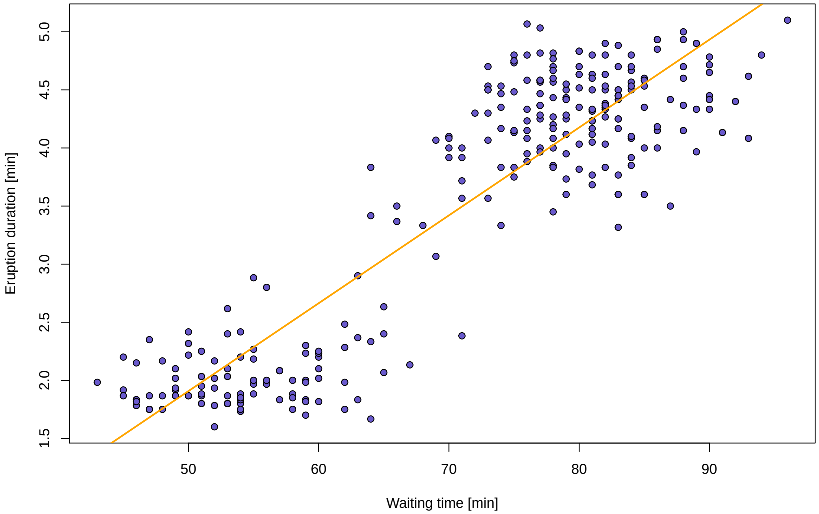


True regression
Local constant (locpoly)
Local linear (locpoly)
Local constant (loess)
Local linear (loess)
Y
X
10
0
−10
−20
−3
−2
−1
0
1
2
3

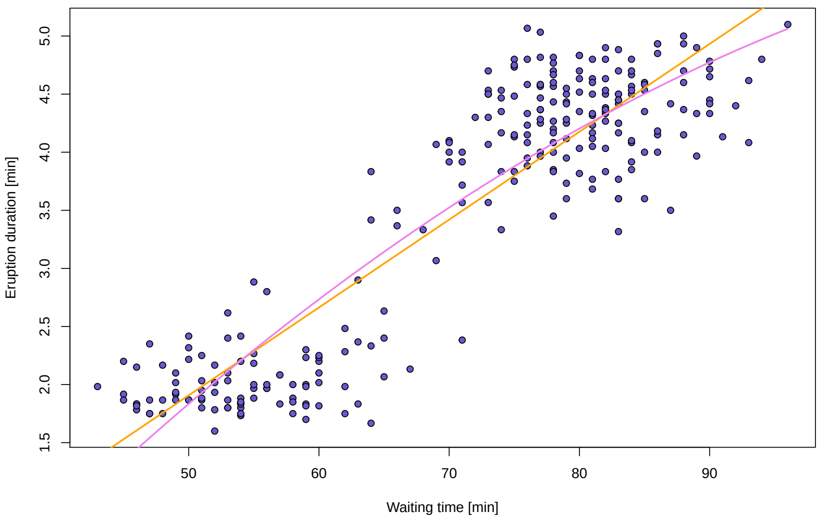
Example: geyser data



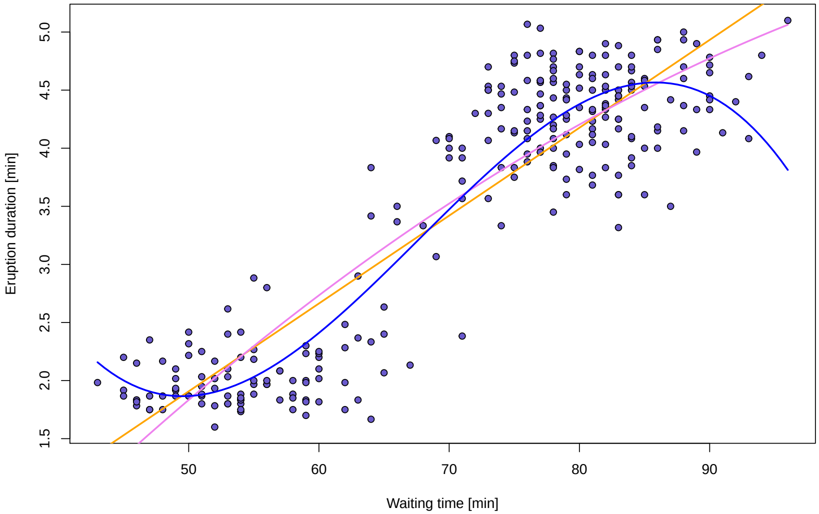
Example: geyser data



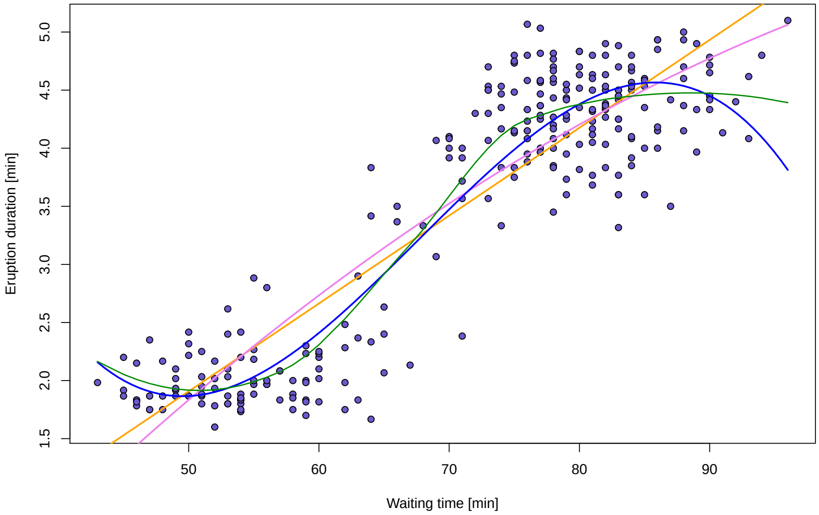
Example: geyser data



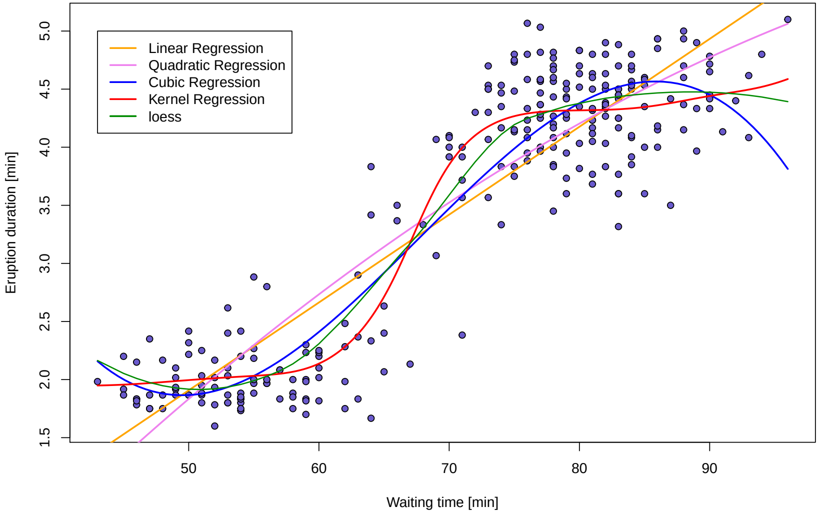
Example: geyser data



Example: geyser data



Example: geyser data


Linear Regression
Quadratic Regression
Cubic Regression
Kernel Regression
loess
Eruption duration [min]
Waiting time [min]