LECTURE 1: INTRODUCTION AND REGRESSION

Text references: Chapter 1 in Shalizi (James et al. Chapter 2)

Introduction to Regression Analysis

Regression analysis is used to answer questions about how one variable depends on the level of one or more other variables. For example, does diet correlate with cholesterol level, and does this relationship depend on other factors, such as age, smoking status, and level of exercise?

Given data $(X_1, Y_1), ..., (X_n, Y_n)$ we have two goals:

Follow-up questions:

Regression and Prediction

Let's start with the simplest regression setting where we want to predict a real-valued variable $Y \in \mathbb{R}$ from nothing at all. For example, suppose I'm taking a particular cholesterol medication, and I want to predict my improvement in blood cholesterol level.

Q: What is the optimal point prediction for Y?

Q: In practice, how would we make this point prediction?

The Regression Function

Prediction with nothing to distinguish one random variable from the next has limited applicability. More often we want to predict Y based on another variable X, called the *predictor*, *covariate*, or *input*. In this context Y is now often called the *response*, *outcome*, or *output*.

Our prediction for Y will be a function of X, denoted $r(X)$. For example, what will be the amount of improvement in blood cholesterol level as a function of the dose X of a particular medication?

Q: What is the "best" prediction of Y given X?

We call $r(x)$ the *regression function*, it's what we use to predict Y from X.

Our statistical model:

Check: What happens when we take expectation conditional on X on both sides?

Note (or disclaimer):

How to estimate the regression function?

From data $(X_1, Y_1), (X_2, Y_2,), \ldots (X_n, Y_n)$, we could estimate the regression function via

$$\widehat{r}(x) = \frac{1}{\#\{i : X_i = x\}} \sum_{i:X_i=x} Y_i \tag{1}$$

But this can work only when X takes discrete values; otherwise for X continuous, at any x, the probability of getting a sample at any particular value of x is zero. Back to cholesterol example: what happens if we didn't observe any person in our study who took exactly $x = 31.5$ mg weekly of the drug? Important concepts: *interpolation, extrapolation, and smoothing*...

Linear regression (revisited). If we're willing to use a *linear* relationship, we can find the best fitting linear function, that is

Taking derivatives, we get

Q: How do we estimate the linear oracle from data? (Mention two approaches)

Method 1:

Method 2:

Let's look at this slightly differently: to make a prediction at an arbitrary point x (center X and Y to simplify the notation), we use

Linear Smoothers

Define "linear smoothers" and go back to previous estimators.

Another common, more flexible linear smoother is the *k-nearest-neighbors regression*:

To use this in practice, we're going to need to choose k, the number of nearest neighbors.

Q: What are the tradeoffs at either end? [See Figures 1 and 2]

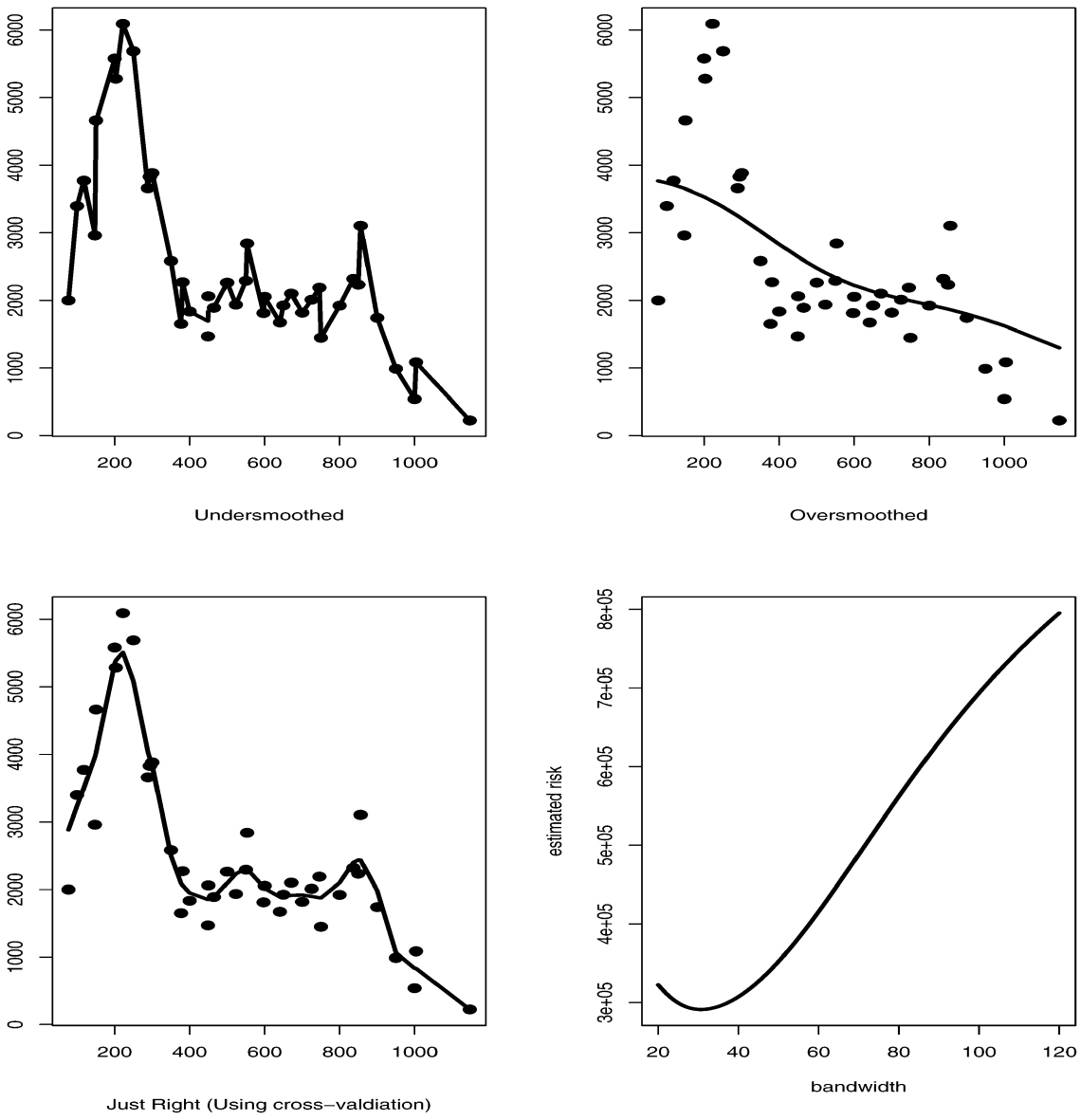
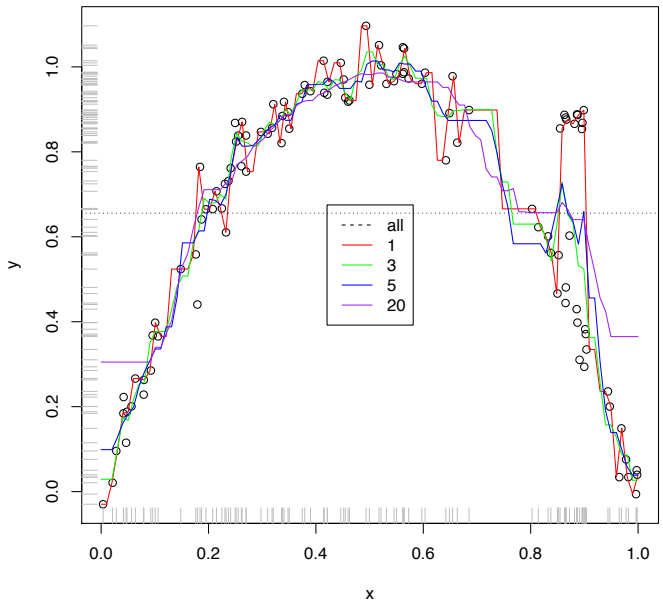


FIGURE 20.8. Regression analysis of the CMB data. The first fit is undersmoothed, the second is oversmoothed, and the third is based on cross-validation. The last panel shows the estimated risk versus the bandwidth of the smoother. The data are from BOOMERaNG, Maxima, and DASI.

Figure 1: From *All of Statistics* by Larry Wasserman




```
library(FNN)
plot.seq <- matrix(seq(from = 0, to = 1, length.out = 100), byrow = TRUE)
lines(plot.seq, knn.reg(train = all.x, test = plot.seq, y = all.y, k = 1)$pred,
    col = "red")
lines(plot.seq, knn.reg(train = all.x, test = plot.seq, y = all.y, k = 3)$pred,
    col = "green")
lines(plot.seq, knn.reg(train = all.x, test = plot.seq, y = all.y, k = 5)$pred,
    col = "blue")
lines(plot.seq, knn.reg(train = all.x, test = plot.seq, y = all.y, k = 20)$pred,
    col = "purple")
legend("center", legend = c("all", "1", "3", "5", "20"), lty = c("dashed", rep("solid",
    4)), col = c("black", "red", "green", "blue", "purple"))
```

FIGURE 1.5: *Points from Figure 1.1 with horizontal dashed line at the mean and the k-nearest-neighbor regression curves for the indicated k. Increasing k smoothes out the regression line, pulling it towards the mean. — The code is repetitive; can you define functions to simplify it?*

Figure 2: From Shalizi

Training and test errors

Q: How are we going to quantify this (overfitting vs underfitting)?

Let's call $(X_1, Y_1), \ldots (X_n, Y_n)$, the sample of data that we used to fit $\widehat{r}$, our **training sample**. What's wrong with looking at how well we do in fitting the training points themselves, i.e., the **training error** or *expected training error*, defined as

Now, suppose that we an independent **test sample** $(X'_1, Y'_1), (X'_2, Y'_2), \ldots (X'_m, Y'_m)$ (following the same distribution as our training sample). We could then look at the **expected test error**, defined as

Note that the expectation here is taken over *all* that is random (both training and test samples). This really does capture what we want, and has the right behavior with k!

Exercise: Summarize how underfitting and overfitting relate to training and test errors. [See R Demo 1]

(Later in Lecture 3 we are going to see that underfitting and overfitting are related to two quantities called *estimation bias* and *estimation variance*, or just bias and variance.)

Smoother Linear Smoothers

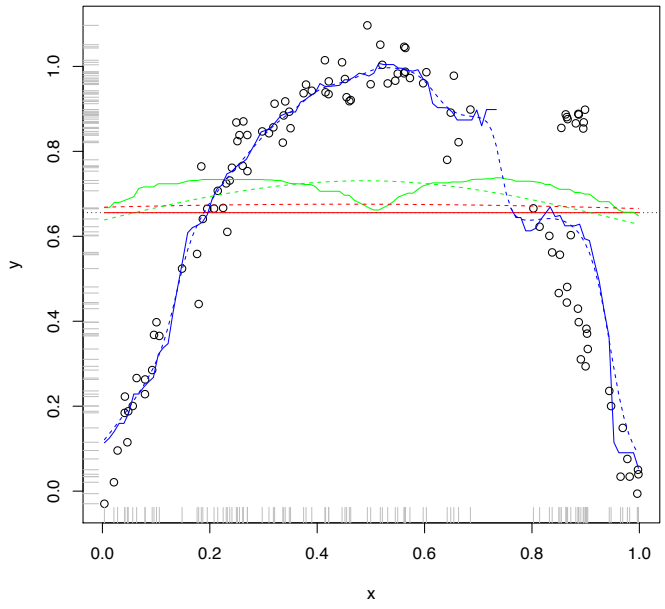
Let's look back at the k-nearest-neighbors regression, the weights $w(x, x_i)$ are either $1/k$ or 0, depending on x_i. This will often produce jagged looking

fits. Why? Are these weights smooth over x?

How about choosing a smoother way of averaging? For example, a *kernel regression* with a *Gaussian kernel*:

The parameter h here is called the **bandwidth**, like k in k-nearest-neighbors regression, it controls the level of adaptivity/flexibility of the fit. One reason to prefer kernel regression with Gaussian kernels is that it can produce smoother (less jagged) looking fits than k-nearest-neighbors or kernel regression with box kernels [See Figure 3]

We are also going to see later in Lecture 4 that these smoother (nonparametric) estimators converge faster toward the true regression function.




```
lines(ksmooth(all.x, all.y, "box", bandwidth = 2), col = "red")
lines(ksmooth(all.x, all.y, "box", bandwidth = 1), col = "green")
lines(ksmooth(all.x, all.y, "box", bandwidth = 0.1), col = "blue")
lines(ksmooth(all.x, all.y, "normal", bandwidth = 2), col = "red", lty = "dashed")
lines(ksmooth(all.x, all.y, "normal", bandwidth = 1), col = "green", lty = "dashed")
lines(ksmooth(all.x, all.y, "normal", bandwidth = 0.1), col = "blue", lty = "dashed")
```

FIGURE 1.6: *Data from Figure 1.1 together with kernel regression lines. Solid colored lines are box-kernel estimates, dashed colored lines Gaussian-kernel estimates. Red, $h = 2$; green, $h = 1$; blue, $h = 0.1$ (per the definition of bandwidth in the* `ksmooth` *function). Note the abrupt jump around $x = 0.75$ in the box-kernel/$h = 0.1$ (solid green) line — with a small bandwidth the box kernel is unable to interpolate smoothly across the break in the training data, while the Gaussian kernel can.*

Figure 3: From Shalizi

R Demo 1

(a) Load data.

Download the file nonlin.Rdata from Canvas, and load it into your R session with `load("nonlin.Rdata")`. You can type `ls()` to see the R objects that have been loaded into memory.

The matrices `xtrain` and `ytrain`, are each 100×50, containing 50 training data sets of x and y points along its columns. That is, the first column of `xtrain` and the first column of `ytrain` make up a training data set of 100 x-y pairs.

For the next bit, we will restrict our attention to just the first training set, i.e., the first columns of `xtrain` and `ytrain`. Plot the x points versus the y points for these data to get an idea of the trend.

(b) Linear regression on the first training set.

Fit a linear regression on the first training set using the function `lm()`. Plot the estimated regression function on top of the training points.

(c) k-nn regression on the first training set.

Similarly to Part (b), now fit a k-nn regression to the training sample. Install the R library `FNN`. Use the function `knn.reg()` to fit a k-nn regression with 3 different values of the number of nearest neighborhoods: $k = 3, 15, 45$. For each value of k, plot the estimated regression function from kernel regression on top of the training points.

(d) Training and test errors as a function of k for the first data set.

Examine training and test errors for the linear regression and for k-nn with $k = 1, \ldots, 60$. What happens as k decreases/increases? Why?

(e) Compute training and test errors averaged over 50 data sets.

Now repeat Part (d) for all 50 pairs of training and test sets. Average these results over the 50 data sets. Plot the results. Compare with the results from Part (d). What do you see? Why?