Principal Components

Ridge, lasso, and elastic net all work with the full set of predictors (or inputs, or features, or regressors).

For lasso and elastic net, some of them end up with zero coefficients. For ridge, they all end up with non-zero coefficients.

When there are many predictors, they often contain closely related information. It seems natural to try to boil these down to a smaller number of predictors.

In the context of macroeconomic prediction, we might have 25 different interest rate series, dozens of series that measure various aspects of economic activity, several stock market indices, and so on.

One way to make data like these manageable is to find their **principal components** and use only a few of those as predictors.

See Stock and Watson (2002a, 2002b) and many other papers.

Recall that the **singular value decomposition** of $\boldsymbol{X}$ is

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top, \tag{1}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are $n \times p$ and $p \times p$ orthogonal matrices, with $\boldsymbol{U}$ and $\boldsymbol{X}$ spanning the same subspace. The columns of $\boldsymbol{V}$ are the eigenvectors $\boldsymbol{v}_m$, $m = 1, \ldots, p$, which are orthogonal.

The eigen decomposition of $\boldsymbol{X}^\top\boldsymbol{X}$ is

$$\boldsymbol{X}^\top\boldsymbol{X} = \boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^\top. \tag{2}$$

Here the diagonal elements of $\boldsymbol{D}$ are the singular values d_m, $m = 1, \ldots, p$, and the squares of the d_m are the eigenvalues λ_m.

The largest eigenvector $\boldsymbol{v}_1$ has the property that $\boldsymbol{z}_1 = \boldsymbol{X}\boldsymbol{v}_1$ has the largest sample variance among all normalized linear combinations of the columns of $\boldsymbol{X}$.

Since $\boldsymbol{z}_1 = \boldsymbol{u}_1 d_1$, this sample variance is

$$\mathrm{Var}(\boldsymbol{X}\boldsymbol{v}_1) = \mathrm{Var}(\boldsymbol{u}_1 d_1) = d_1^2/n = \lambda_1/n. \tag{3}$$

If we rank the eigenvalues from largest to smallest, we can obtain the corresponding **principal components**

$$\boldsymbol{z}_m = \boldsymbol{X}\boldsymbol{v}_m, \quad m = 1, \ldots, p. \tag{4}$$

Each principal component is a linear combination of the columns of $\boldsymbol{X}$. The $\boldsymbol{z}_m$ span the same space as $\boldsymbol{X}$ itself: $\mathcal{S}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_p) = \mathcal{S}(\boldsymbol{X})$.

- The first principal component $\boldsymbol{z}_1$ contains all the variation in the direction in which the columns of $\boldsymbol{X}$ vary the most.
- In a sense, $\boldsymbol{z}_1$ is the single variable that comes as close as possible to explaining all the data.
- The second principal component $\boldsymbol{z}_2$ contains all the remaining variation in the direction, orthogonal to the first, in which the columns of $\boldsymbol{X}$ vary the most.

- The third principal component z_3 contains all the remaining variation in the direction, orthogonal to the first and second, in which the columns of $\boldsymbol{X}$ vary the most. And so on. . . .

The idea of **principal components regression**, or **PCR**, is simply to regress $\boldsymbol{y}$ on a small number of the z_m.

- Use forward selection and choose M, the number of principal components to include, by minimizing a criterion function like AIC or by cross-validation.
- This is easy, because the principal components are ordered.
- Of course, since the principal components depend on how the columns of $\boldsymbol{X}$ are scaled, we should normally rescale the data before we find the principal components.

Finding the z_m is fairly expensive when n is large, but running many PCR regressions is cheap, because the principal components are necessarily orthogonal to each other.

Adding another regressor, say $\boldsymbol{z}_g$, simply means regressing the current residual

$$\hat{\boldsymbol{\epsilon}}_{g-1} \equiv \boldsymbol{y} - \hat{\theta}_0 - \hat{\theta}_1 \boldsymbol{z}_1 - \hat{\theta}_2 \boldsymbol{z}_2 - \ldots - \hat{\theta}_{g-1} \boldsymbol{z}_{g-1} \tag{5}$$

on $\boldsymbol{z}_g$.

Then we set $\hat{\boldsymbol{\epsilon}}_g = \hat{\boldsymbol{\epsilon}}_{g-1} - \hat{\theta}_g \boldsymbol{z}_g$ and keep going in this way until $g = p$ or, more commonly, until we stop with $M << p$.

The raw fit will improve as M increases, but criteria like AIC may get better or worse.

The model we end up with is

$$\boldsymbol{y} = \hat{\theta}_0 + \sum_{m=1}^{M} \hat{\theta}_m \boldsymbol{z}_m + \hat{\boldsymbol{\epsilon}}_m. \tag{6}$$

We could just estimate this by OLS, or we can use a succession of simple regressions like (5).

The coefficients in (6) are not easy to interpret, but this does not matter if we just want to make predictions.

If desired, we can recover the implied β coefficients from the $\hat{\theta}_m$. Because $\boldsymbol{z}_m = \boldsymbol{X}\boldsymbol{v}_m$,

$$\sum_{m=1}^{M} \hat{\theta}_m \boldsymbol{z}_m = \sum_{m=1}^{M} \hat{\theta}_m \boldsymbol{X}\boldsymbol{v}_m = \sum_{m=1}^{M} \hat{\theta}_m \sum_{j=1}^{p} \boldsymbol{x}_j v_{jm} = \sum_{j=1}^{p} \hat{\beta}_j \boldsymbol{x}_j. \tag{7}$$

It follows that

$$\hat{\beta}_j^{\text{pc}} = \sum_{m=1}^{M} \sum_{j=1}^{p} \hat{\theta}_m v_{jm}, \tag{8}$$

where v_{jm} is the j^{th} element of the m^{th} eigenvector.

When M is small, the $\hat{\beta}_j^{\text{pc}}$ are likely to be shrunk relative to the OLS estimates. When $M = p$, they will be equal to the OLS estimates.

There is a close relationship between ridge regression and PCR. Both operate via the principal components of the input matrix.

- In both cases, coefficients are shrunk but not set to zero.
- For ridge, directions with small eigenvalues get shrunk more than ones with large eigenvalues.
- For PCR, directions with small eigenvalues are discarded, while ones with large eigenvalues are retained.

PCR involves **unsupervised learning**. The ordering of the principal components is based entirely on $\boldsymbol{X}$ and not at all on $\boldsymbol{y}$.

I will not discuss **partial least squares**, which does take $\boldsymbol{y}$ into account.

The $\hat{\theta}_m$ are not easy to interpret and may not be reported.

If M were fixed in advance, we could obtain $\widehat{\mathrm{Var}}(\hat{\boldsymbol{\theta}})$ and use it to obtain $\widehat{\mathrm{Var}}(\hat{\boldsymbol{\beta}})$ via (8).

But this will give incorrect results when M is chosen via cross-validation or any criterion function. In principle, we should bootstrap the entire procedure.

Kernel CDF Estimation

ISLR/ISLP does not discuss **kernel smoothing**, **kernel density estimation**, or **kernel regression**. The last of these is conceptually similar to KNN regression.

Since these methods are widely used in economics, I will do so. See Chapter 6 of ESL. Useful books include Racine (2019), Li and Racine (2007), and Henderson and Parmeter (2015).

To begin with, suppose we want to estimate a distribution. The simplest way to estimate a **cumulative distribution function**, or **CDF**, graphically is to use the **empirical distribution function**, or **EDF**.

Suppose we have a sample x_i, $i = 1, \ldots, n$, of realizations of a random variable x. Then the EDF at any point x is

$$\hat{F}(x) \equiv \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(x_i \leq x), \tag{9}$$

where $\mathbb{I}(\cdot)$ is the **indicator function**.

The EDF (9) is not a smooth function of x. It is actually a staircase; see Figure 11.1 below.

The height of each step is $1/n$. The width of each step varies. It is the distance between two adjacent values of the sorted x_i.

If we replace $\mathbb{I}(x_i \leq x)$ in (9) by a smooth function of $x_i - x$, the EDF will be replaced by something smoother.

- Let $K(z)$ be any continuous CDF corresponding to a distribution with mean 0. This function is called a **cumulative kernel**.
- $K(z)$ typically corresponds to a distribution with a density that is symmetric around the origin, such as the standard normal.
- $K(z)$ is usually chosen to have variance 1. Then we introduce a **bandwidth parameter** h as a scaling parameter.

When $K(z)$ is the standard normal distribution, h plays the role of σ, the standard deviation. For very small h, the kernel CDF looks a lot like the EDF. As h increases, it becomes smoother.

The **kernel CDF estimator** is

$$\hat{F}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K\Big(\frac{x_i - x}{h}\Big). \tag{10}$$

This estimator depends on the cumulative kernel $K(\cdot)$ and the bandwidth h, both of which we choose.

- Recall that $K(z) \to 0$ as $z \to -\infty$, and $K(z) \to 1$ as $z \to \infty$.
- As $h \to 0$, a typical term of the summation on the r.h.s. of (10) tends to $\mathbb{I}(x \le x_i)$, so $\hat{F}_h(x)$ tends to the EDF $\hat{F}(x)$ as $h \to 0$.
- Remember that, as $h \to 0$, $(x_i - x)/h$ tends to $+\infty$ or $-\infty$, so $K\big((x_i - x)/h\big)$ tends to 1 or 0.
- At the other extreme, as h becomes large, a typical term of the summation tends to the constant value $K(0)$, which makes $\hat{F}_h(x)$ very much too smooth.
- In the usual case in which $K(z)$ corresponds to a symmetric distribution, $K(0) = 0.5$, so $\hat{F}_h(x)$ tends to 0.5 as $h \to \infty$.

Figure 11.1 — EDFs for Two Sample Sizes

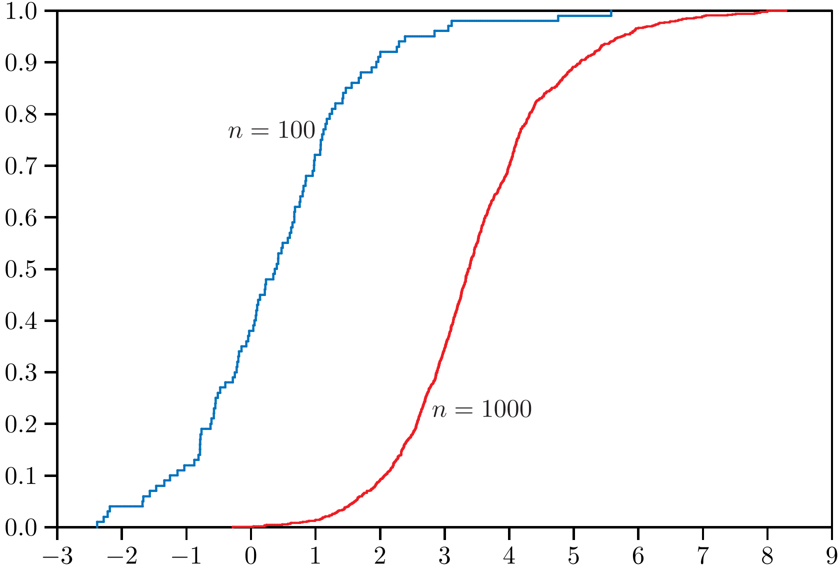


Figure 11.1 shows EDFs for two sample sizes, 100 and 1000.

The data are drawn from continuous distributions that are approximately, but not exactly, normal.

The distributions have different means but are otherwise the same.

- Even for $n = 1000$, the stairstep effect is quite noticeable, especially in the tails.
- In my experience, n needs to be very large indeed for an EDF to look really smooth.

Recall the resampling bootstrap. Formally, it is drawing bootstrap samples from an EDF.

The only possible values of an EDF are the ones in the sample, each of which occurs with probability $1/n$ (the height of each step in the EDF) if there are no repeats.

If a value occurs twice, there will be a step of height $2/n$.

Figure 11.2 — Two Kernel CDFs

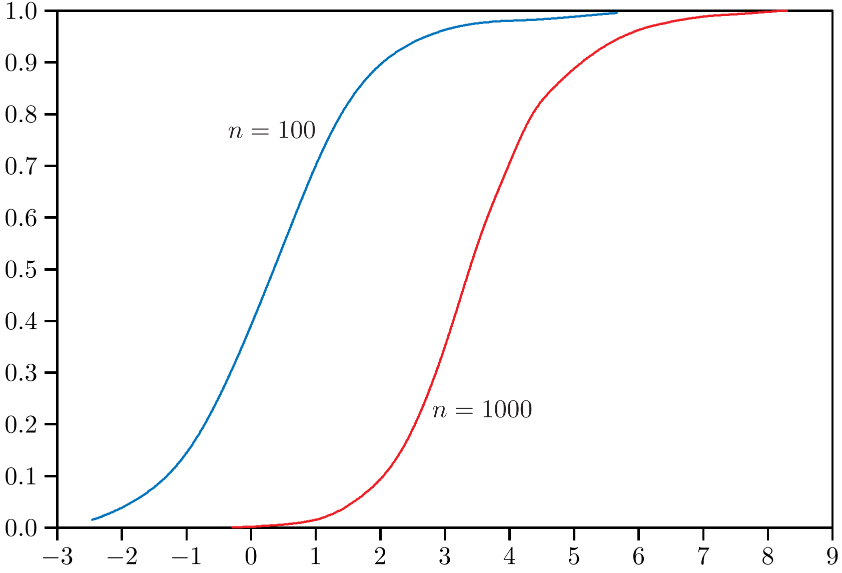


Figure 11.2 shows kernel CDFs for the same samples as Figure 11.1.

These use a Gaussian kernel and are plotted for 101 and 201 points for $n = 100$ and $n = 1000$, respectively.

- The curve for $n = 100$ has a couple of odd features.
- The curve for $n = 1000$ looks extremely smooth.
- The main problem with the kernel CDF for $n = 1000$ is that it completely hides the randomness of the sample.

These figures used $h = 1.587sn^{-1/3}$, where s is the sample standard deviation of the x_i.

The rate of $n^{-1/3}$ is optimal quite generally for CDF estimation, but the factor 1.587 depends on rather special assumptions.

As we shall see, a different (and much slower) rate is optimal for PDF estimation.

Histograms

The traditional way to show a density function graphically is a **histogram**. This would be correct if the data were discrete.

The interval containing the x_i is partitioned into a set of subintervals, or **bins**, by a set of **cut points** z_j, $j = 1, \ldots, M$, with $z_j < z_{j+1}$ for all j, where typically $M << n$. The histogram is discontinuous at the z_j.

Let j be such that $z_j \leq x < z_{j+1}$ for some x. Then the histogram is just the following estimate of the density function at x:

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{\mathbb{I}(z_j \leq x_i < z_{j+1})}{z_{j+1} - z_j}, \text{ for } z_j \leq x < z_{j+1}. \tag{11}$$

The value of the histogram at x is the proportion of the sample points contained in the same bin as x, divided by the length of the bin.

A histogram is extremely dependent on the choice of the number and locations of the cut points, that is, the z_j.

With just two cut points, the histogram would look like a uniform distribution with lower limit z_1 and upper limit z_2.

With a great many cut points, many bins would be empty, and others would contain spikes of various heights.

We want neither too few nor too many bins. To prove anything about asymptotic validity, we would need a rule for increasing the number of bins as $n \to \infty$.

- It is a good idea to make the number of bins odd.
- Usually, the bins are of equal width, but that is not essential.
- It can make sense for bins near the tails to be wider, because there are fewer observations there.
- If we choose to make a tail bin wider than the others, it is essential to divide the number of observations in that bin by the ratio of its length to the length of the others.
- It is the area of each bar that matters, not its height.

Figure 11.3 — Two Histograms for $n = 100$

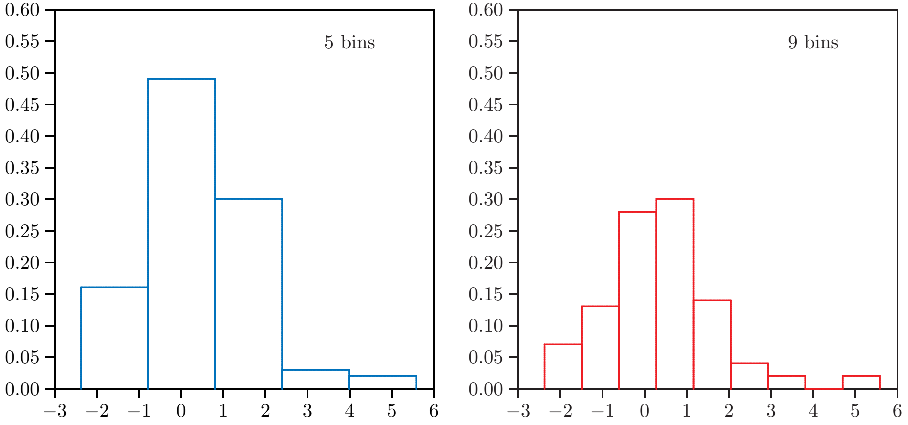
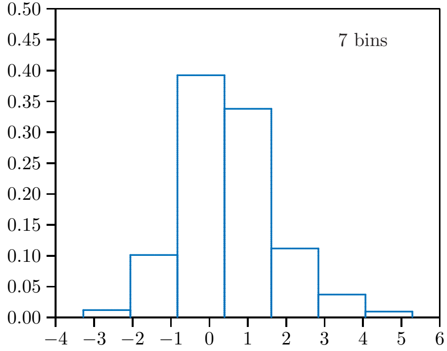


Figure 11.4 — Two Histograms for $n = 1000$



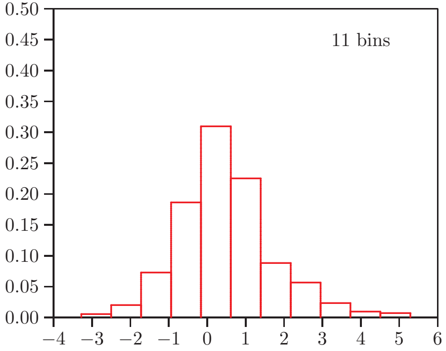


Figure 11.3 shows two histograms for $n = 100$, and Figure 11.4 shows two histograms for $n = 1000$.

- The number of bins we choose to group the data into evidently matters.
- If the number is chosen unwisely, a histogram can be misleading. Imagine using 6 or 8 for a symmetric, unimodal density.
- A larger sample size allows us to use more bins.
- With $n = 100$, 9 bins is probably too many.
- With $n = 1000$, both 7 bins and 11 bins look decent, but they tell somewhat different stories.
- It is probably a good idea to experiment with the number of bins.

Just as we can use a kernel CDF to smooth out an EDF, so can we use a **kernel density** to smooth out a histogram.

Kernel Density Estimation

For density estimation, we define the **kernel function**, often simply called the **kernel**, as $k(z) \equiv K'(z)$. So if $K(z)$ is a CDF, $k(z)$ is a PDF.

The **kernel density estimator** at x is

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} k\Big(\frac{x_i - x}{h}\Big). \tag{12}$$

Notice that we divide by nh rather than just n.

The kernel density estimator (12) depends on the choice of kernel $k(\cdot)$ and bandwidth h.

One popular choice for $k(\cdot)$ is the **Gaussian kernel**, which is just the standard normal density $\phi(\cdot)$.

It gives a positive (although perhaps very small) weight to every point in the sample.

Another commonly used kernel, which has certain optimality properties, is the **Epanechnikov kernel**,

$$k_1(z) = \frac{3(1 - z^2/5)}{4\sqrt{5}} \quad \text{for } |z| < \sqrt{5}, \quad 0 \ \text{ otherwise.} \tag{13}$$

This kernel gives a positive weight only to points for which $|(x_i - x)|/h < \sqrt{5}$.

The big difference between the Epanechnikov and Gaussian kernels is that the former is 0 for $|z| > \sqrt{5}$, while the latter is always positive.

It can be shown that, to highest order, the bias of the kernel estimator is

$$\mathrm{E}\big(\hat{f}_h(x) - f(x)\big) \cong \frac{h^2}{2} f''(x) \kappa_2(k), \tag{14}$$

where $f''(x)$ is the second derivative of the density $f(x)$ and $\kappa_2(k)$ is the second moment of the kernel $k(\cdot)$.

The bias does not depend directly on the sample size. It only depends on n through h, which should become smaller as n increases.

It can also be shown that, to highest order, the variance of $\hat{f}_h(x)$ is

$$\mathrm{E}\Big(\hat{f}_h(x) - \mathrm{E}\big(\hat{f}_h(x)\big)\Big)^2 \cong \frac{1}{nh} f(x). \qquad (15)$$

Notice that this is proportional to $f(x)$.

Choosing h involves a classic bias-variance tradeoff.

- As h becomes larger, the bias (14) increases.
- As h becomes smaller, the variance (15) increases.
- Both the bias and the variance depend directly on x through $f(x)$ or its second derivative.
- Thus the optimal choice of h should actually change with x.
- Ideally, h would be smaller near the mode than in the tails.
- But desirable theoretical properties of (12) only hold for fixed h.

- In a sense, making h larger is like making K larger for KNN estimation.
- If h is chosen optimally for the centre of the distribution, we may see too much bias in the tails.
- If h is chosen optimally for the tails, there is likely to be too much variance near the centre of the distribution

Unlike KNN, where the estimate always depends on K points, $\hat{f}(x)$ now depends on all points, or at least all points that are not a long way away, but with weights that vary.

The number of points is effectively large near the centre of the distribution but small in the tails.

More sophisticated methods that vary h with x, sort of like KNN, exist but will not be discussed.

Epanechnikov kernel is optimal in a certain sense. Loss in efficiency for Gaussian kernel relative to Epanechnikov is roughly 5.13%.

It can be shown that the optimal h is proportional to $n^{-1/5}$.

Since $n^{-1/5}$ decreases very slowly with n, so must the optimal h.

In contrast, since the optimal h for CDF estimation is proportional to $n^{-1/3}$, it decreases much more rapidly.

If we are estimating a normal density using a Gaussian kernel, it turns out that

$$h_{\text{rot}} = 1.059 s n^{-1/5}, \tag{16}$$

where h_{rot} is Silverman's rule-of-thumb bandwidth.

- When a distribution has heavy tails, s is not a very good measure of dispersion.
- A more robust measure is the **inter-quartile range**. For a normal distribution, $\sigma = \text{IQR}/1.349$. Thus we could replace s in the rule-of-thumb bandwidth by IQR/1.349.
- To avoid over-smoothing, we could instead replace it by $\min(s, \text{IQR}/1.349)$.

For the Epanechnikov kernel, the constant that corresponds to 1.059 is 1.049. It is a little bit smaller because of the greater efficiency of estimation based on the Epanechnikov kernel.

- Any rule-of-thumb bandwidth may fail badly if the distribution being estimated differs a lot from the normal.
- There are likely to be severe problems if the distribution is multi-modal.

It is possible to determine h by using cross-validation.

Leave-one-out cross-validation is attractive because sample size matters. It can be inexpensive if implemented correctly.

Figures 11.5 and 11.6 show kernel density estimates for the same data as the histograms in Figures 11.3 and 11.4.

Figure 11.5 — Two Kernel Densities

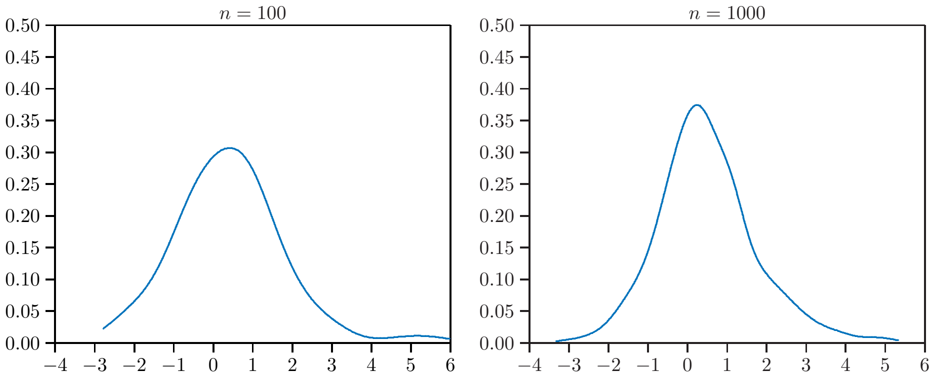
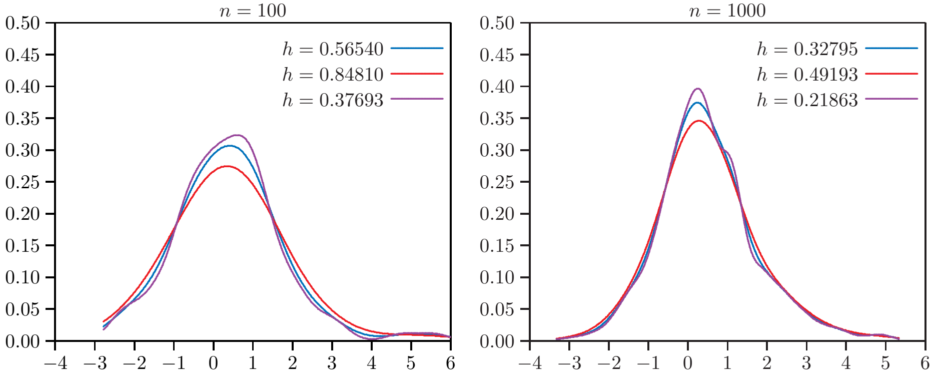


Figure 11.6 — The Effect of *h* on Kernel Density Estimates



The "optimal" density for $n = 100$ looks too smooth, and the density is not high enough at the mode.

When the sample size is small, it is desirable to do a lot of smoothing. As a result, the density estimate tends to look a lot like the kernel.

Choosing h too large or too small has serious consequences.

- Making h too small causes the density to become taller near the mode, but it adds a lot of wiggles.
- Making h too large causes the density to become shorter near the mode, and makes it look too much like the kernel.

`R` contains a function called `density` that can be used to obtain kernel density estimates.

It provides a great many options, and the defaults are not necessarily the best choices. See `help(density)`.

I did not use `density` to create the figures above, but I did use it to create Figure 11.7 below.

The R commands

```
depan1 <- density(xx,kernel=c("epanechnikov"))
depanp75 <- density(xx,adjust=0.75,kernel=c("epanechnikov"))
depan1p5 <- density(xx,adjust=1.5,kernel=c("epanechnikov"))
```

create three different kernel densities. These can be plotted using, e.g., `plot(depan1)`, perhaps with plot options.

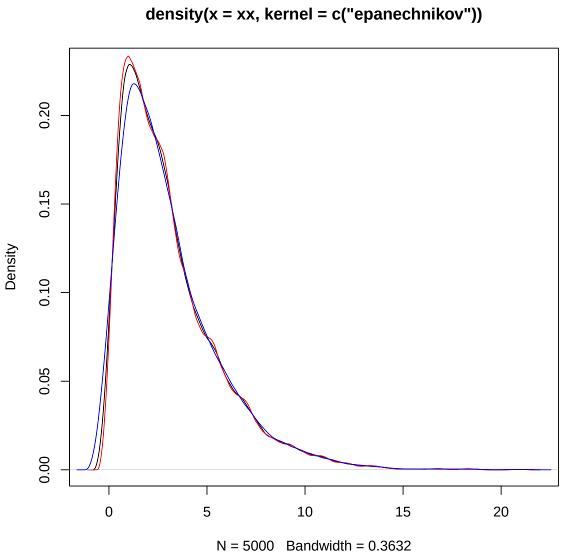
The first one picks the bandwidth in a sensible fashion, and the other two make it smaller or larger.

When all three densities are plotted on the same axes, with the one for the default h in black, the one for the small h in red, and the one for the large h in blue, we obtain Figure 11.7

In this case, the data came from the $\chi^2(3)$ distribution.

To see how well kernel density estimation works, try generating several random samples of different sizes and plotting their densities.

Figure 11.7 — Three Kernel Density Estimates


density(x = xx, kernel = c("epanechnikov"))
Density
N = 5000 Bandwidth = 0.3632