14 Non-Parametric Estimation & Statistical Learning Methods

In most econometric settings, we are interested in relating some outcome variable y_i to some explanatory variables in x_i, be it with the goal of predicting y_i based on x_i, conducting inference on some aspect of the relationship between x_i and y_i,[1] or analyzing the relationship between the two more generally. Thereby, we can always decompose y_i into a part explained by x_i and a part independent of x_i: $y_i = \mathbb{E}[y_i|x_i] + u_i$ with $\mathbb{E}[u_i|x_i] = 0$. Hence, our estimation problem boils down to estimating the function $m(x_i) = \mathbb{E}[y_i|x_i]$. The linear regression model assumes that $m(x_i) = x_i'\beta$ is linear. Starting from an original vector of covariates $\tilde{x}_i$, one can account for nonlinearities by including non-linear transformations of the variables in $\tilde{x}_i$ (e.g. squares or interaction-terms) as additional covariates in x_i. One can also consider nonlinear models like the nonlinear regression (see example in Chapter 6) or the probit model (see Section 4.4). However, rather than assuming a particular, parametric functional form and estimating the parameters that determine its exact shape, one might want to estimate $m(x_i)$ non-parametrically, allowing for more flexible functional forms. This is where machine/statistical learning methods come in handy, because, ultimately, they are functional approximation methods.

This chapter gives an overview of statistical learning methods commonly used in econometrics. After a brief introduction to how nonparametric methods are validated in Section 14.1, Section 14.2 discusses local regressions, Section 14.3 regression trees and Section 14.4 neural networks. A treatment of kernel density estimation – related to local regressions – is given in the Appendix.

[1] This notably includes inference on the causal effect of x_i on y_i as well as inference on some parameter in a structural, parametric model that relating y_i to x_i as motivated by theory. The following discussion does not apply to the former case, but to analyses that are not inherently linked to a parametric model.

14.1 Numerical Approaches to Model Selection

Let

$$y_i = m(x_i) + u_i \ , \quad \mathbb{E}[u_i|x_i] = \mathbb{E}[u_i] = 0 \ . \tag{14.1}$$

The non-parametric methods discussed in the following provide estimators of the function m and therefore $m(x_i) = \mathbb{E}[y_i|x_i]$. The exact estimator produced by a given method depends on so-called tuning parameters λ (e.g. the bandwidth h in local regressions; see Section 14.2), which is why we can write it as $\hat{m}_\lambda$. Model selection is about determining the optimal value of h as well as choosing which method – parametric or not – to apply in the first place.

Ultimately, all models are validated using loss functions, such as the quadratic loss function, which takes the form $L(\theta, \delta) = (\theta - \delta(x))^2$ under a scalar object of interest θ and an estimate (decision) $\delta(x)$ given data x. As discussed in Section 2.1.1, the expectation of this loss function – treating data as random and the object of interest θ as fixed, the so-called frequentist risk – embodies a bias-variance trade-off. Except for causal inference, we typically want to optimally balance this trade-off and obtain an estimator that optimally predicts out-of-sample rather aiming for an unbiased estimator no matter its variance.[2]

Under parametric models, we can rely on model selection criteria like the marginal data density under the Bayesian paradigm or information criteria under the frequentist paradigm (see Section 5.3). These analytical criteria are typically derived with some parametric model in mind, which makes them less suitable in settings where the estimated model deviates significantly from the models implicitly assumed in these analytical criteria. Alternatively, one can do so numerically by setting apart a fraction of available observations to conduct pseudo out-of-sample model validation by determining how well the estimated model predicts these left-out observations. In this regard one distinguishes between a training- and a testing-sample.

In non-parametric settings, analytical model selection criteria are rarely used, even less so if enough data is available. Instead, one divides the sample into a training- and testing sample. Observations in the latter are truly set apart at the start of the estimation procedure and only used in the end to conduct a pseudo out-of-sample analysis and select the best fitting model (method). In contrast to that, the former is typically further divided into a proper

[2]When non-parametric or statistical learning methods are used for causal inference purposes, there are two possible paths. One is to not apply the model selection approaches discussed in this section, but to use methods that yield unbiased or at least asymptotically unbiased estimators and select tuning parameters with the same goal in mind. The other is to apply the following model selection approaches nevertheless, but add a so-called "de-biasing" step.

training sample and a validation sample. This distinction is made in order to select values for tuning parameters for a given method.

One approach to select tuning parameters is Cross-Validation (CV). For example, under k-fold CV, one partitions the data in the training-sample into K equal-sized chunks and predicts y_i by estimating the model using only observations from chunks other than the one where i is in. More precisely, let $\kappa(i) \in 1:K$ denote the chunk to which observation i belongs, and let $\hat{m}_\lambda^{-\kappa(i)}$ denote the estimated function m using observations in all chunks but chunk $\kappa(i)$, which is indexed by some tuning parameter λ. One then chooses λ as

$$\lambda^* = \arg\min_\lambda \ \frac{1}{n}\sum_{i=1}^n L(y_i, m_\lambda^{-\kappa(i)}(x_i)) \ .$$

Typical choices for K are 5 or 10. Taking $K = n$, one obtains the so-called leave-one-out CV, whereby all observations but observation i are used to predict observation i. The larger K, the more precise the estimates, but also the higher the computational burden.[3] One can repeat this sample split several, say R, times, and set

$$\lambda^* = \arg\min_\lambda \ \frac{1}{n}\sum_{i=1}^n \frac{1}{R}\sum_{r=1}^R L(y_i, \hat{m}_\lambda^{-\kappa_r(i)}(x_i)) \ , \tag{14.2}$$

where $\kappa_r(i)$ now indicates the chunk $k \in 1:K$ to which observation i belongs in the rth sample split (assuming that the number of chunks K is unchanged across sample splits r).

Bootstrap methods provide another approach for selecting tuning parameters. Analogously to the use of bootstrapping for approximating the variability of estimators in repeated, finite samples (see Section 7.1), one constructs B samples by drawing with replacement observations from the training sample. One then predicts y_i by using only the estimates from samples B where observation i is left out. More precisely, let $\hat{m}_h^b$ be the estimate of m using sample $b \in 1:B$ and let $\mathcal{B}^{-i}$ be the set of samples b in which observation i is omitted. One then chooses λ as

$$\lambda^* = \arg\min_\lambda \ \frac{1}{n}\sum_{i=1}^n \frac{1}{|\mathcal{B}^{-i}|}\sum_{b\in\mathcal{B}^{-i}} L(y_i, \hat{m}_\lambda^b(x_i)) \ . \tag{14.3}$$

The object being minimized is referred to as the "out-of-bag" loss. Relative to CV, the bootstrap has the added advantage that one can use the boostrap-samples not only to select

[3] One typically takes K not too large not only for reasons of computational efficiency, but also because the higher K, the higher the correlation between the K different leave-one-out samples (because they have more observations in common), which yields highly correlated estimates $\hat{m}_\lambda^{-k}$ across $k = 1:K$ and means that the estimated expectation of the loss function above only slowly converges to the true one.

tuning parameters, but also to estimate the finite sample distribution of $\hat{m}_\lambda(x_0)$ at any point x_0.

The most popular choice of loss function is the L2 loss, $L(y_i, \hat{m}(x_i)) = (y_i - \hat{m}(x_i))^2$, leading to the computation of mean squared errors (MSEs). Under thick tails of y_i, one might consider the L1 loss, $L(y_i, \hat{m}(x_i)) = |y_i - \hat{m}(x_i)|$, leading to the computation of mean absolute errors or mean absolute deviations (MADs).

14.2 Local Regression

Let again

$$y_i = m(x_i) + u_i \ , \quad \mathbb{E}[u_i|x_i] = \mathbb{E}[u_i] = 0 \ .$$

Local or non-parametric regression methods (or averaging methods) estimate $m(x_i)$ at each point x_0 by considering a local weighted average of realizations of y_i around x_0:

$$\hat{m}(x_0) = \sum_{i=1}^{n} w_h(x_0, x_i) y_i \ , \quad \sum_{i=1}^{n} w_h(x_0, x_i) = 1 \ .$$

As illustrated in the following, the bandwidth h determines the extent of locality by specifying how much weight is given to points x_i close to x_0 relative to points further away.

Suppose for simplicity that x_i is a scalar. One might define the weights $\{w_h(x_0, x_i)\}_{i=1}^n$ by considering the $h \in \mathbb{Z}$ nearest neighbors of x_0:

$$w_h(x_0, x_i) = \frac{1}{h} \mathbf{1}\left\{i \in \mathcal{N}_h(x_0)\right\} \ ,$$

where $\mathcal{N}_h(x_0)$ is the set containing the h units i that are closest to x_0 in terms of some distance measure like the L1-norm or absolute value $|x_i - x_0|$. Similarly, one might consider only the points x_i that are in an interval of $\pm h \in \mathbb{R}_{++}$ around x_0:

$$w_h(x_0, x_i) = \frac{\tilde{w}_h(x_0, x_i)}{\sum_{i=1}^n \tilde{w}_h(x_0, x_i)} \ , \quad \tilde{w}_h(x_0, x_i) = \mathbf{1}\left\{\left|\frac{x_i - x_0}{h}\right| \leq 1\right\} \ ,$$

where $\sum_{i=1}^n \tilde{w}_h(x_0, x_i)$ is then the number of such points x_i that are close enough to x_0.

In contrast to these local methods, OLS estimation of $y_i = \beta_0 + \beta_1 x_i + u_i$ yields

$$w_h(x_0, x_i) = \frac{1}{n} + \frac{(x_0 - \bar{x})(x_i - \bar{x})}{\sum_{i=1}^n (x_i - \bar{x})^2} \ ,$$

where $\bar{y}$ and $\bar{x}$ are the sample averages of y_i and x_i, respectively.[4] These weights do not decrease with the distance between x_i and x_0, which illustrates that OLS is a global estimation method.

The drawback of the above two local regression approaches is that they lead to an "edgy" estimated function $\hat{m}$. To obtain a smooth estimate, rather than considering sharp cut-off rules in terms of number of neighbors or maximum permitted distance, we can weigh observations smoothly using a so-called kernel function K. This leads to the general Nadaraya-Watson weights

$$w_h(x_0, x_i) = \frac{\tilde{w}_h(x_0, x_i)}{\sum_{i=1}^n \tilde{w}_h(x_0, x_i)}, \quad \tilde{w}_h(x_0, x_i) = K\left(\frac{x_i - x_0}{h}\right).$$

Essentially, a kernel K is a continuous, positive-valued function with a maximum at zero. More precisely, it should satisfy the following conditions:

1. K is a continuous function
2. K is symmetric around zero
3. $\int_{z\in\mathbb{R}} K(z)dz = 1$, $\int_{z\in\mathbb{R}} |K(z)|dz < \infty$, $\int_{z\in\mathbb{R}} zK(z)dz = 0$
4. either a) $K(z) = 0$ for all $|z| > z_0$ and some z_0 or b) $\lim_{|z|\to\infty} K(z) = 0$
5. $\int_{z\in\mathbb{R}} z^2K(z)dz = c$ with $0 < c < \infty$

The uniform or box-kernel $K(z) = \frac{1}{2}\ \mathbf{1}\{|z| < 1\}$ leads to the estimator discussed above, which considers only points x_i that are in an interval of $\pm h \in \mathbb{R}_{++}$ around x_0 and assigns each weight to all of them. Relative to it, the triangular kernel $K(z) = (1 - |z|)\ \mathbf{1}\{|z| < 1\}$ assigns different weights to these points, according to their distance from x_0. The Epanechnikov kernel $K(z) = \frac{3}{4}(1 - z^2)\ \mathbf{1}\{z \leq 1\}$ also does that, but using a smoother weighting function. With all these Kernels, one can estimate m only at points x_0 which are at most at distance h of at least one point x_i in the sample. This either restricts the space of x at which m can be estimated, or it puts a constraint on how small h and hence the incurred bias (see below) can be. This is somewhat remedies by using the Gaussian kernel $K(z) = \phi(z) = (2\pi)^{-\frac{1}{2}} exp\left\{-\frac{1}{2}z^2\right\}$.[5] However, it is computationally much less efficient than

[4] We get $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$ and $\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$. This results in

$$\hat{y}_0 = \hat{m}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0 = \bar{y} + \hat{\beta}_1(x_0 - \bar{x}) = \sum_{i=1}^n \left\{\frac{1}{n} + \frac{(x_0 - \bar{x})(x_i - \bar{x})}{\sum_{i=1}^n (x_i - \bar{x})^2}\right\} y_i .$$

[5] Note that this implies that the weight given to observation i is proportional to the pdf of a Normal

the above alternatives because at each x_0, all observations i are used to construct $\hat{m}(x_0)$, i.e. $\tilde{w}_h(x_0, x_i) > 0 \; \forall \; x_i$. Regardless of the choice of kernel, the estimation precision is low in regions of x where little data is available, typically the boundaries of x (see below).

Far more important than the choice of the kernel K is the choice of the bandwidth h. Given a continuous K, the resulting Nadaraya-Watson estimator $\hat{m}$ is a smoooth function. However, how smooth it is depends on h. Higher values of h yield smoother estimates, as more observations are taken into account when constructing $\hat{m}(x_0)$ at any given point x_0, which means that $\hat{m}(x_0)$ is less affected by sampling noise. For the same reason, higher values of h lead to a lower variance of $\hat{m}$, but also higher bias (see below). The bias and variance $\hat{m}$ are derived without assuming anything else than Eq. (14.1), and in particular without assuming that m has some parametric functional form. With this in mind, the intuition for the bias is that, with a higher h, we impose smoothness, even though the true function might not be smooth, or at least not at every point x_0. Provided that $h \to 0$ as $n \to \infty$, the bias vanishes asymptotically and we get pointwise consistency of $\hat{m}$.

In contrast to global methods like OLS, under local methods the bias and variance of the estimator $\hat{m}(x_0)$ can vary significantly across x_0, depending on the properties of m at that point x_0. One can show (see e.g. Cameron and Trivedi (2005, ch. 9)) that the bias at x_0 is equal to

$$b(x_0) \equiv \mathbb{E}[\hat{m}(x_0)] - m(x_0) = h^2 \left(m'(x_0) \frac{p'(x_0)}{p(x_0)} + \frac{1}{2} m''(x_0) \right) \int z^2 K(z) dz \; ,$$

and – assuming homoskedasticity – the asymptotic distribution of $\hat{m}(x_0)$ is

$$\sqrt{nh}(\hat{m}(x_0) - m(x_0) - b(x_0)) \xrightarrow{d} N\left(0, V(x_0)\right) \; , \quad V(x_0) = \frac{\mathbb{V}[u_i]}{p(x_0)} \int K(z)^2 dz \; .$$

Thereby, p denotes the pdf of x, and m' and m'' are the first- and second-order derivatives of m, and p' is the first-order derivative of p. The bias is high in regions of the sample space of x where there is little probability mass, and it is high in regions where m has a high slope and curvature.

Based on the above formula, one can approximate the finite sample distribution of $\hat{m}(x_0)$, which in turn enables hypothesis testing and the construction of confidence intervals. Note that this requires estimating the density $p(x)$ – which can be done using kernel density estimation, discussed in the Appendix – as well as the derivatives of m and p. Also, one can derive an optimal bandwidth h that minimizes the expected Mean Squared Error (MSE)

distribution with mean x_0 and standard deviation h.

and therefore balances the bias-variance trade-off. Besides the sample size n, the optimal h depends on the curvature of m and the kernel used. Generally, it is proportional to $n^{-1/5}$. A popular approach to set h is Silverman's rule: $h^* = \hat{\sigma}_x n^{-0.2}$, where $\hat{\sigma}_x$ is the sample standard deviation of x.

Under the optimal bandwidth, $\hat{m}$ converges to m at the rate $n^{2/5}$. This is lower than the $n^{1/2}$ obtained under most global, parametric models, including in particular the linear regression. This means that local regressions need significantly (far) more data to achieve the same precision.[6] However, keep in mind that consistency of parametric models is based on the assumption that the supposed parametric model is correctly specified, whereas local regressions do not make this assumption, rendering them robust to model-misspecification! In sum, to obtain robustness, we pay a price in terms of precision and efficiency.

Rather than relying on analytical derivations of the bias or (asymptotic) variance, one can also select h using CV or Bootstrapping. Thereby, one can downweigh observations i with x_i at the boundary of the support of x_i. The formulas in Eq. (14.2) and Eq. (14.3) then become

$$h^* = \arg\min_h \ \frac{1}{n}\sum_{i=1}^{n}\frac{1}{R}\sum_{r=1}^{R} L(y_i, \hat{m}_h^{-\kappa_r(i)}(x_i))\pi(x_i)$$

and

$$h^* = \arg\min_h \ \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|\mathcal{B}^{-i}|}\sum_{b\in\mathcal{B}^{-i}} L(y_i, \hat{m}_h^b(x_i))\pi(x_i) \ ,$$

respectively, where $\pi(x_i)$ is a weighting function. For example, one might take $\pi(x_i)$ to be one if x_i is in-between the 5th and 95th quantiles of the sample distribution of x_i and zero otherwise. Under local regressions, applying a bootstrap has the added advantage that one can estimate the variability of $\hat{m}(x_0)$ without relying on analytical formulas that require the estimation of p and derivatives of m and p.

Extension to Multivariate Covariates Local regressions are readily extended to accommodate multivariate covariates $x_i \in \mathbb{R}^k$ by applying the kernel function K not on $(x_i - x_0)/h$ but on $d(x_i, x_0)/h$, where d is some distance measure between the vectors x_i and x_0. For example, one could use the L1 or L2 norms

$$d(x_i, x_0) = ||x_i - x_0||_1 = \sum_{j=1}^{k}|x_{ij} - x_{0j}| \quad \text{or} \quad d(x_i, x_0) = ||x_i - x_0||_2 = \sum_{j=1}^{k}(x_{ij} - x_{0j})^2 \ .^7$$

[6] In this regard, 900 observations under a linear regression are equivalent to 5000 observations under a local regression, and 5000 observations under a linear regression are equivalent to 42000 observations under a local regression.

One can also apply a weighting matrix A to form $d(x_i, x_0) = (x_i - x_0)'A(x_i - x_0)$. An example thereof is the Mahalanobis distance, which takes the inverse of the sample covariance matrix of x: $A = \hat{\Sigma}_x^{-1}$. Alternatively, instead of replacing $K((x_i - x_0)/h)$ by $K(d(x_i, x_0)/h)$, one can extend local regressions to higher dimensions by replacing $K((x_i - x_0)/h)$ by $\prod_{j=1}^{k} K((x_{ij} - x_{0j})/h)$.

In higher dimensions, the speed of convergence drops to $n^{2/(k+4)}$, which illustrates that local regressions truly suffer from the curse of dimensionality! Thereby, Silverman's rulle becomes

$$h^* = \left(\frac{4}{k+2}\right)^{\frac{1}{k+4}} \hat{\sigma} n^{-1/(k+4)} , \quad \hat{\sigma}^2 = \frac{1}{k}\sum_{j=1}^{k} \hat{\sigma}_j^2 ,$$

where $\hat{\sigma}_j^2$ is the sample variance of covariate j, x_{ij}.

Semi-Parametric Local Regression Ultimately, non-parametric local regression methods fit an average – and therefore a constant – as the estimate of $m(x_0)$. As discussed above, this typically leads to a high bias at the boundaries of the sample space of x. To reduce the bias at the boundary, one can use a semi-parametric local regression. For example, rather than fitting constants locally, one could fit linear regressions locally:

$$\hat{m}(x_0) = x_0'\hat{\beta}(x_0) , \quad \hat{\beta}(x_0) = \arg\min_{\beta} \sum_{i=1}^{n} K\left(\frac{d(x_i, x_0)}{h}\right)(y_i - x_i'\beta)^2 .$$

In a similar fashion, one can fit other parametric models locally. For example, one might conduct local maximum likelihood estimation:

$$\hat{m}(x_0) = \mathbb{E}[y_i|x_0; \hat{\theta}(x_0)] , \quad \hat{\theta}(x_0) = \arg\max_{\theta} \sum_{i=1}^{n} K\left(\frac{d(x_i, x_0)}{h}\right)\log p(y_i|x_i, \theta) ,$$

where $\mathbb{E}[y_i|x_i; \theta]$ is the expectation of y_i taken under the conditional density (likelihood) $p(y_i|x_i, \theta)$ indexed by the parameter θ.

[7] The Euclidean distance is given by the square root of the L2 norm.

14.3 Regression Trees

A regression tree partitions the sample space of x, $\mathcal{X}$,[8] into strata $\{\mathcal{X}_s\}_{s=1}^S$ and constructs $\hat{m}$ by fitting a constant into each stratum $\mathcal{X}_s$:

$$\hat{m}(x_0) = \sum_{s=1}^{S} c_s^* \, \mathbf{1}\{x_0 \in \mathcal{X}_s\} \ .$$

The strata are also referred to as nodes or leaves of the tree. Choosing the L2 loss function as the minimization criterion leads to c_s^* being the average y_i for observations $x_i \in \mathcal{X}_s$. Formally, letting $\mathcal{n}_s = \{i : x_i \in \mathcal{X}_s\}$ be the set of observations i in stratum $\mathcal{X}_s$, we have

$$c_s^* = \arg\min_{c_s} \ \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{m}(x_i))^2 = \arg\min_{c_s} \ \sum_{i\in\mathcal{n}_s}(y_i - c_s)^2 = \frac{1}{|\mathcal{X}_s|}\sum_{i\in\mathcal{n}_s} y_i \equiv \bar{y}_{\mathcal{X}_s} \ .$$

To simplify notation, the rest of this section is based on the assumption of such an L2 loss. By devising smart algorithms to selet the strata $\{\mathcal{X}_s\}_{s=1}^S$, regression tree-based methods break the curse of dimensionality and lead to great forecasts of y_i even for high-dimensional covariates. However, little is known about their theoretical properties, which is why they are seldom used for inference.

It would be desirable to determine the strata $\{\mathcal{X}_s\}_{s=1}^S$ by minimizing the MSE (subject to some overfitting-constraint). However, it is computationally infeasible to consider every possible partition of $\mathcal{X}$.[9] Instead, one uses the recursive binary splitting algorithm. It iteratively searches for the next split that minimzes the MSE given the splits so far, whereby only splits based on one variable and one value of this variable are considered. Let $\mathcal{R}_1(j,\tau) = \{x_i : x_{ij} < \tau\}$ and $\mathcal{R}_2(j,\tau) = \{x_i : x_{ij} \geq \tau\}$ be two sets that partition $\mathcal{X}$ into two regions, according to whether covariate j, x_{ij}, is smaller or larger than some threshold τ. Also, let $\mathcal{n}_1(j,\tau) = \{i : x_i \in \mathcal{R}_1(j,\tau)\}$ and $\mathcal{n}_2(j,\tau) = \{i : x_i \in \mathcal{R}_2(j,\tau)\}$ be the sets of observations in these two regions. You can think of the algorithm as starting from a situation where there is a single leaf given by the sample space of x, $\mathcal{X}$, and the corresponding prediction $\hat{m}^0(x_0) = \frac{1}{n}\sum_{i=1}^n y_i \equiv \bar{y}$. At the first iteration, it considers predictions of the form

$$\hat{m}^1(x_0; j, \tau) = \sum_{s=1}^{2} \bar{y}_{\mathcal{R}_s(j,\tau)} \, \mathbf{1}\{x_0 \in \ \mathcal{R}_s(j,\tau)\}$$

[8] In the context of machine learning, x_i is often referred to as the feature(s) of observation i and $\mathcal{X}$ as the feature space.

[9] This holds regardless of whether thte MSE in the training- or the validation-sample is used.

and searches for the variable j and threshold τ that minimize the MSE (or lead to the biggest reduction in MSE relative to the one obtained under $\hat{m}^0$, i.e. relative to simply predicting $\bar{y}$ for all observations i):

$$\begin{aligned}(j^*, \tau^*) &= \arg\min_{j,\tau} \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{m}^1(x_0; j, \tau)) \\ &= \arg\min_{j,\tau} \left[\sum_{i \in \mathcal{N}_1(j,\tau)} (y_i - \bar{y}_{\mathcal{R}_1(j,\tau)})^2 + \sum_{i \in \mathcal{N}_2(j,\tau)} (y_i - \bar{y}_{\mathcal{R}_2(j,\tau)})^2\right] .\end{aligned}$$

Graphically, in a two-dimensional space $\mathcal{X}$, this corresponds to cutting the rectangular $\mathcal{X}$ into two pieces – with a line that cuts two of its sides and is parallel to the other two – and computing the average y_i in each of the two pieces. The optimal prediction at iteration one, $\hat{m}^1(\cdot) = \hat{m}^1(\cdot; j^*, \tau^*)$, is based on strata given by the regions corresponding to j^* and τ^*: $\{\mathcal{X}_s^1\}_{s=1}^2 = \{\mathcal{X}_1, \mathcal{X}_2\} = \{\mathcal{R}_1(j^*, \tau^*), \mathcal{R}_2(j^*, \tau^*)\}$.

Subsequent iterations are a bit harder to define mathematically, because one does not start from $\mathcal{X}$ as the strata and $\hat{m}^0(x_0) = \bar{y}$ as the prediction. Instead, iteration 2 starts from $\{\mathcal{X}_s^1\}_{s=1}^2$ and $\hat{m}^1(x_0)$. Graphically, it starts from a rectangle cut into two pieces by a line that cuts two of its sides and is parallel to the other two, and it searches for a further cut, which can either cut the other two sides – leading to 4 pieces – or the same two sides at a different value – leading to 3 pieces. Mathematically, the second iteration again considers regions $\mathcal{R}_1(j, \tau)$ and $\mathcal{R}_2(j, \tau)$ with observations $\mathcal{N}_1(j, \tau)$ and $\mathcal{N}_2(j, \tau)$, this time defining

$$\hat{m}^2(x_0; j, \tau) = \sum_{s=1}^{S(j,\tau)} \bar{y}_{\mathcal{X}_s(j,\tau)} \mathbf{1}\{x_0 \in \mathcal{X}_s(j, \tau)\} ,$$

where

$$\{\mathcal{X}_s(j, \tau)\}_{s=1}^{S(j,\tau)} = \text{unique}\{\mathcal{X}_1^1 \cap \mathcal{R}_1(j, \tau), \mathcal{X}_2^1 \cap \mathcal{R}_1(j, \tau), \mathcal{X}_1^1 \cap \mathcal{R}_2(j, \tau), \mathcal{X}_2^1 \cap \mathcal{R}_2(j, \tau)\}$$

contains all the intersections of the sets $\{\mathcal{X}_s^1\}_{s=1}^2$ with $\mathcal{R}_1(j, \tau)$ and $\mathcal{R}_2(j, \tau)$. Given this $\hat{m}^2(x_0; j, \tau)$, one again chooses j and τ so as to minimize the MSE, i.e. bring about the largest reduction in MSE relative to the one obtained under $\hat{m}^1$, leading to $\hat{m}^2(\cdot) = \hat{m}^2(\cdot; j^*, \tau^*)$ and $\{\mathcal{X}_s^2\}_{s=1}^{S_2} = \{\mathcal{X}_s(j^*, \tau^*)\}_{s=1}^{S(j^*,\tau^*)}$.

This procedure is repeated until the decrease in MSE from one iteration to the next is smaller than some threshold $\underline{r}$ or until the number of leaves reaches some number $\bar{n}_l$ or until the number of observations in the smallest leaves reaches some number $\underline{n_o}$. Algorithm 29 below

defines the full recursive binary splitting algorithm under the first stopping rule.[10]

Algorithm 29 (Fitting Regression Trees: Recursive Binary Splitting)**.**

- *Initialize* $\{\mathcal{X}_s^0\}_{s=1}^{S_0} = \{\mathcal{X}\}$ *and* $MSE^0 = \frac{1}{n}\sum_{i=1}^n (y_i - \bar{y})^2$*, and specify* $\underline{r}$*.*
- *For iterations* $m = 1, 2, ...,$ *given the parition* $\{\mathcal{X}_s^{m-1}\}_{s=1}^{S_{m-1}}$ *from iteration* $m-1$*,*
  1. *Find*

$$(j^*, \tau^*) = \arg\min_{j,\tau} \; MSE(j,\tau) = \arg\min_{j,\tau} \; \frac{1}{n}\sum_{i=1}^n (y_i - \hat{m}(x_0; j, \tau))$$

where

$$\hat{m}(x_0; j, \tau) = \sum_{s=1}^{S(j,\tau)} \bar{y}_{\mathcal{X}_s(j,\tau)} \, \mathbf{1}\{x_0 \in \; \mathcal{X}_s(j,\tau)\}$$

and

$$\{\mathcal{X}_s(j,\tau)\}_{s=1}^{S(j,\tau)} = unique\Big\{\mathcal{X}_1^{m-1} \cap \mathcal{R}_1(j,\tau), ..., \mathcal{X}_{S_{m-1}}^{m-1} \cap \mathcal{R}_1(j,\tau), \\ \mathcal{X}_1^{m-1} \cap \mathcal{R}_2(j,\tau), ..., \mathcal{X}_{S_{m-1}}^{m-1} \cap \mathcal{R}_2(j,\tau)\Big\} \; .$$

  2. *Let* $\hat{m}(\cdot) = \hat{m}(\cdot; j^*, \tau^*)$*,* $\{\mathcal{X}_s^m\}_{s=1}^{S_m} = \{\mathcal{X}_s(j^*, \tau^*)\}_{s=1}^{S(j^*,\tau^*)}$ *and* $MSE^m = MSE(j^*, \tau^*)$*.*
  3. *If* $MSE^{m-1} - MSE^m > \underline{r}$*, continue. Else, break, and take* $\hat{m}_{\underline{r}} = \hat{m}^m$ *with* $\{\mathcal{X}_s^m\}_{s=1}^{S_m}$ *as the estimate of* m*.*

In principle, the stopping rule-parameters $\underline{r}$, $\bar{n_l}$ or $\underline{n_o}$ are important tuning parameters. However, they are typically not selected by CV or bootstrapping because this leads to a model that overfits the data. This is related to the fact that the above algorithm is "greedy", as it considers at each iteration the best possible step, rather than being forward-looking and building a tree that allows for better splits in subsequent iterations. To remedy these issues, one typically selects $\underline{r}$ or $\underline{n_o}$ too low or $\bar{n_l}$ too high on purpose, and then applies a "trimming" or "pruning" algorithm to the full, overfitting tree. This involves "cutting branches" of the original tree so as to find a sub-tree that minimizes MSE subject to some penalty for model complexity. Formally, a tree T_0 is defined by its leaves $\{\mathcal{X}_s^0\}_{s=1}^{S_0}$, which yield the prediction

[10] Under the alternative stopping rules, replace $\text{MSE}^{m-1} - \text{MSE}^m > \underline{r}$ in the last step with $|\{\mathcal{X}_s^m\}_{s=1}^{S_m}| < \bar{n_l}$ or with $\min_{s=1,...,S} \; \mathcal{n}_s^m > \underline{n_o}$ where $\mathcal{n}_s^m = \{i : x_i \in \mathcal{X}_s^\}$.

$\hat{m}^0_{\underline{r}}$. Let $|T_0| = S_0$ be the number of leaves in tree T_0, and let

$$\text{MSE}(T_0) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{m}^0_{\underline{r}}(x_i))^2 \,, \quad \hat{m}^0_{\underline{r}}(x_i) = \sum_{s=1}^{S_0} \bar{y}_{\mathcal{X}^0_s} \, \mathbf{1}\left\{x_i \in \mathcal{X}^0_s\right\}$$

be the MSE obtained under tree T_0. Pruning seeks to find a sub-tree T^* that minimizes

$$C_\alpha(T^*) = \text{MSE}(T^*) + \alpha|T^*|$$

for a given tuning parameter α. While it is hard to define sub-trees and the pruning-objective function in general, we will limit ourselves to the weakest link-pruning algorithm. This procedure iteratively "cuts one branch of the tree" by combining two strata from $\{\mathcal{X}_s\}_{s=1}^S$. Given T_0, define a sub-tree of it, T_1, as the tree that satisfies

1. $|T_1| = |T_0| - 1$ (it has one leaf less than T_0)
2. $\exists \mathcal{X}^1_r = \mathcal{X}^0_u \cup \mathcal{X}^0_v$ for some $\mathcal{X}^0_u, \mathcal{X}^0_v$, while $\mathcal{X}^1_t \in \{\mathcal{X}^0_s\}_{s=1}^{S_0} \; \forall \, t \neq r$ (it combines two leaves of T_0, leaving the rest unchanged)

Let $\mathcal{T}_0$ be the set of such sub-trees possible given T_0.[11] The first iteration of the weakest link-pruning algorithm starts from T_0 and $C_\alpha(T_0)$. It then finds the sub-tree $T^*_1 \in \mathcal{T}_0$ that satisfies

$$T^*_1 = \arg\min_{T_1 \in \mathcal{T}_0} \; C_\alpha(T_1) \; = \arg\min_{T_1 \in \mathcal{T}_0} \; \text{MSE}(T_1) \,.$$

Minimizing $C_\alpha(T_1)$ boils down to minimizing the MSE under T_1 because all sub-trees T_1 share the same number of leaves. As a result, the algorithm looks for the sub-tree T_1 that results in the smallest increase in MSE, i.e. it cuts the branch that contributes the least to reducing MSE. Typically, one takes α large enough so that it is possible to reduce the objective function, i.e. $C_\alpha(T^*_1) < C_\alpha(T_0)$. This procedure is then repeated until no further sub-tree can be found that further reduces C_α, as outlined in Algorithm 30. This results in a final prediction $\hat{m}_{\alpha,\underline{r}}$. A higher α results in smaller trees, and its optimal value can be determined using CV or bootstrapping.

[11] There are $|T_0|(|T_0|-1)/2$ such sub-trees.

Algorithm 30 (Fitting Regression Trees: Weakest Link-Pruning)**.**

- *Using Algorithm 29, grow a tree T_0 with corresponding leaves $\{\mathcal{X}_s^0\}_{s=1}^{S_0}$ and prediction $\hat{m}_{\underline{r}}^0$.*
- *For iterations $m = 1, 2, ...,$ given the tree T_{m-1} from iteration $m-1$,*
  1. *Among the subtrees $\mathcal{T}_{m-1}$ of T_{m-1}, find*

$$T_m^* = \arg\min_{T_m \in \mathcal{T}_{m-1}} C_\alpha(T_m) = \arg\min_{T_m \in \mathcal{T}_{m-1}} MSE(T_m) .$$

  2. *If $C_\alpha(T_m^*) < C_\alpha(T_{m-1})$, continue with $T_m = T_m^*$. Else, break, and take T_{m-1} as the final tree, i.e. take $\hat{m}_{\alpha,\underline{r}} = \hat{m}_{\underline{r}}^{m-1}$ with $\{\mathcal{X}_s^{m-1}\}_{s=1}^{S_{m-1}}$ as the estimate of m.*

Regression trees are among the most intuitive and interpretable statistical learning algorithms. Because they can accommodate continuous as well as binary outcome variables y_i, they also can be used for classification purposes. They are rather robust to the inclusion of irrelevant variables in x_i and can easily accommodate different types of variables in x_i, which is particularly important in big-data environments where data is often "messy".[12] However, an important shortcoming of regression trees is that they are rather unstable, meaning that a slight change in the sample $\{y_i, x_i\}_{i=1}^n$ can leads to very different trees and predictions. Relatedly, very little is known about the theoretical properties of regression trees, though this holds to similar extents for all statistical learning methods.

Bootstrap Averaging (Bagging) & Random Forests To reduce the instability of regression tree-based predictions, bagging estimates m by averaging the predictions of many trees, each fitted on a separate bootstrap sample b. Formally, write the B bootstrap samples as $\left\{\left\{x_i^b\right\}_{i=1}^{n_b}\right\}_{b=1}^B$ and let $\hat{m}_{\alpha,\underline{r}}^b$ be the prediction provided by a regression tree applied so sample $\left\{x_i^b\right\}_{i=1}^{n_b}$. Bagging constructs

$$\hat{m}_{B,\alpha,\underline{r}}(x_0) = \frac{1}{B}\sum_{b=1}^B \hat{m}_{\alpha,\underline{r}}^b(x_0) .$$

Thereby, B can be chosen so as to stabilize the MSE in the validation sample. While bagging makes the prediction more stable, it also reduces interpretability and – depending on B – increases computational time.

To further reduce the variability of the prediction, a random forest grows each of the trees

[12] To enable sensible splitting, categorical variables in x_i should be coded as a set of binary variables for each category.

T_b used to construct $\hat{m}^b_{\alpha,\underline{r}}$ by considering at each iteration in Algorithm 29 only considers splits along a random subset of the k covariates in x_i. The number of covariates considered at each iteration, $\underline{k}$, becomes a further tuning parameter. If $\underline{k} = k$, then the random forest leads to the above prediction obtained under bagging (and is in fact not random). Taking $\underline{k} < k$ introduces some stochasticity in the procedure used to grow the trees $\{T_b\}_{b=1}^B$, thereby reducing the variance of the final prediction, $\hat{m}_{B,\alpha,\underline{r},\underline{k}}(x_0)$. A typical choice is $\underline{k} = ceil(\sqrt{k})$. Typically, smaller values of $\underline{k}$ are useful if there are lots of correlated variables in x_i. In the context of bagging and random forests, it is natural to select α and $\underline{k}$ by minimizing the out-of-bag MSE, avoiding the additional cost of applying CV when bootstrapping is done anyway.

Boosting A boosted tree estimates m by summing up several trees that are grown sequentiallly, where each tree is fit on the residuals left after using all previous trees to predict outcomes y_i. It constructs

$$\hat{m}_{n_T,\alpha,\underline{r}}(x_0) = \sum_{t=1}^{n_T} \hat{e}^t_{\alpha,\underline{r}}(x_0) \ ,$$

where $\hat{e}^1_{\alpha,\underline{r}} = \hat{m}_{1,\alpha,\underline{r}}(x_0)$ is the prediction from a regression tree applied to outcomes y_i, $\hat{e}^2_{\alpha,\underline{r}}$ is the prediction from a regression tree applied to outcomes $y^1_i \equiv y_i - \hat{m}_{1,\alpha,\underline{r}}(x_0) = y_i - \hat{e}^1_{\alpha,\underline{r}}(x_i)$, $\hat{e}^3_{\alpha,\underline{r}}$ is the prediction from a regression tree applied to outcomes $y^2_i \equiv y_i - \hat{m}_{1,\alpha,\underline{r}}(x_0) = y_i - \sum_{t=1}^2 \hat{e}^t_{\alpha,\underline{r}}(x_i)$, and so forth.

Gradient boosting is a variant of this algorithm that can lead to an improved out-of-sample predictive ability by building a tree based on the gradient of the loss function for each observation i rather than on residual outcomes. It is outlined in Algorithm 31, whereby taking $r^t_i = y^{t-1}_i$ yields the regular boosting algorithm.

Algorithm 31 (Fitting Boosted Regression Trees: Gradient Boosting)**.**

- *Using Algorithm 30, construct a predictive function* $\hat{e}^1_{\alpha,\underline{r}}$ *for the outcome variables* $\{y_i\}_{i=1}^n$ *based on the covariates* $\{x_i\}_{i=1}^n$*. Let* $\hat{m}_{1,\alpha,\underline{r}} = \hat{e}^1_{\alpha,\underline{r}}$ *and define the residuals* $y_i^1 \equiv y_i - \hat{m}_{1,\alpha,\underline{r}}(x_i)$*. Denote the*
- *For iterations* $t = 2, ..., n_T$*, given the predictive function* $\hat{m}_{t-1,\alpha,\underline{r}}$ *and residuals* $\{y_i^{t-1}\}_{i=1}^n$ *from iteration* $t-1$*,*
  1. *Compute the loss-gradients*

$$r_i^t = -\left.\frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i}\right|_{\hat{y}_i = \hat{m}^{t-1}_{\alpha,\underline{r}}(x_i)} = -2(y_i - \hat{m}_{t-1,\alpha,\underline{r}}(x_i)) \ .$$

  2. *Use Algorithm 30 to fit a regression tree to predict outcomes* $\{r_i^t\}_{i=1}^n$ *based on covariates* $\{x_i\}_{i=1}^n$*, yielding leaves* $\{\mathcal{X}_s^t\}_{s=1}^{S_t}$*.*
  3. *Based on these leaves, construct*

$$\hat{e}^t_{\alpha,\underline{r}}(x_i) = \sum_{s=1}^{S_t} \mathbf{1}\left\{x_i \in \mathcal{X}_s^t\right\} \bar{y}^{t-1}_{\mathcal{X}_s^t} \ ,$$

  where $\bar{y}^{t-1}_{\mathcal{X}_s^t}$ *is the average residual* y_i^{t-1} *among observations in* $\mathcal{X}_s^t$*.*
  4. *Update the predictive function and residuals:*

$$\hat{m}_{t,\alpha,\underline{r}}(\cdot) = \hat{m}_{t-1,\alpha,\underline{r}}(\cdot) + \hat{e}^t_{\alpha,\underline{r}}(\cdot) = \sum_{u=1}^{t} \hat{e}^u_{\alpha,\underline{r}}(\cdot) \ ,$$
$$y_i^t = y_i^{t-1} - \hat{e}^t_{\alpha,\underline{r}}(x_i) = y_i - \hat{m}_{t,\alpha,\underline{r}}(x_i) \ \forall \ i \ .$$

The number of added trees n_T can be chosen by CV or bootstrapping. The latter option becomes particularly attractive when bagging boosted trees, i.e. averaging the prediction of many boosted trees, each constructed on a separate bootstrap-sample.

A further variant constructs residuals by shrinking the optimal forecasts of residuals in each iteration, i.e. it sets $y_i^1 \equiv y_i - \lambda\hat{e}^1_{\alpha,\underline{r}}(x_i)$ and $y_i^t = y_i^{t-1} - \lambda\hat{e}^t_{\alpha,\underline{r}}(x_i)$ for $t = 2, ..., T$. The shrinkage parameter $\lambda \in (0, 1]$ becomes a further tuning parameter, for which small values are commonly taken, inducing a slow learning of the algorithm. Under stochastic gradient boosting at each iteration t only a certain fraction $\underline{n}$ of observations is used to construct $\hat{e}^t_{\alpha,\underline{r}}$. This decreases the computational time and increases the variability in trees used to fit the residuals, which can lead to improved predictive performance by reducing the chance of

overfitting. All of these variants can be combined with bagging and random forests.

14.4 Neural Networks

A neural network predicts y_i by constructing linear combinations of covariates x_i and sequentially applying simple non-linear functions to them. For illustration purposes, consider first a neural network with a single hidden layer. It constructs

$$\begin{aligned}\hat{m}_H(x_i) &= \beta_0 + z(x_i)'\beta_1 \ , \quad z(x_i) = (z_1(x_i), ..., z_H(x_i)) \ , \\ z_h(x_i) &= \sigma(\alpha_{0,h} + x_i'\alpha_{1,h}) \ ,\end{aligned}$$

where the vector of constructed covariates (or "derived features") $z(x_i) \in \mathbb{R}^H$ makes up the hidden layer. The prediction is a linear combination of these constructed covariates $z(x_i)$ (and a constant), and in turn the elements of $z(x_i)$ are obtained by applying a nonlinear function σ to different linear combinations of the original covariates x_i. If this process is repeated twice, we speak of a neural network with two hidden layers. It constructs

$$\begin{aligned}\hat{m}_{H_1,H_2}(x_i) &= \beta_0 + z^1(x_i)'\beta_1 \ , \quad z^1(x_i) = (z_1^1(x_i), ..., z_{H_1}^1(x_i)) \ , \\ z_h^1(x_i) &= \sigma(\alpha_{0,h}^1 + z^2(x_i)'\alpha_{1,h}^1) \ , \quad z^2(x_i) = (z_1^2(x_i), ..., z_{H_2}^2(x_i)) \ , \\ z_h^2(x_i) &= \sigma(\alpha_{0,h}^2 + x_i'\alpha_{1,h}^2) \ .\end{aligned}$$

This process can be repeated many times, leading to deep neural networks.

The activation function σ is taken to be a simple, locally non-linear function. Typical choices are the sigmoid $\sigma(z) = (1 + e^{-z})^{-1}$ the softplus $\sigma(z) = \log(1 + z)$ or the ReLu (rectified linear unit)-function $\sigma(z) = \max\{0, z\}$. In the case of a single hidden layer, the coefficients $\{\alpha_{0,h}\}_{h=1}^H$ determine where in the sample space of x_i the non-linearity is "activated", whereas the coefficients $\{\alpha_{1,h}\}_{h=1}^H$ determine how "hard" the activation is, i.e. how pronounced the non-linearity is.

The power of neural networks lies in the fact that they iteratively apply the same locally non-linear function to simple linear combinations of (constructed) covariates rather than trying to combine different globally non-linear functions (as other functional approximation methods do, e.g. Chebyshev polynomials). This allows them to approximate well any non-linear function m and generate good forecasts even for high-dimensional x_i. In fact, the Universal Approximation Theorem states that a neural network with at least one hidden layer can approximate any function to any desired degree of accuracy.[13] While neural networks deal

[13]Technically, applies to Borel-measurable functions defined on finite-dimensional spaces, which is always

well with the curse of dimensionality, fitting neural networks is not trivial, which means that it is often a computationally more efficient choice to estimate simpler non-linear functions m using alternative methods.

Neural networks are typically fitted by applying a gradient descent algorithm, choosing the values of coefficients so as to minimize the MSE subject to some penalty for model complexity. As an example, take the neural network with a single hidden layer from above. It features $n_c = H + 1 + H(k+1)$ coefficients: $\rho \equiv (\beta_0, \beta_1', \{\alpha_{0,h}, \alpha_{1,h}'\}_{h=1}^H)$. We choose ρ as

$$\rho^* = \arg\min_\rho \ Q_n(\rho) = \arg\min_\rho \ \frac{1}{n}\sum_{i=1}^n (y_i - \hat{m}_H(x_i;\rho))^2 + \lambda\rho'\rho \ .$$

The partial derivatives are simple to compute:

$$\begin{aligned}
\frac{\partial Q_n}{\partial \beta_0} &= -2\frac{1}{n}\sum_{i=1}^n (y_i - \hat{m}_H(x_i;\rho)) + 2\lambda\beta_0 \ , \\
\frac{\partial Q_n}{\partial \beta_1} &= -2\frac{1}{n}\sum_{i=1}^n (y_i - \hat{m}_H(x_i;\rho))z(x_i) + 2\lambda\beta_1 \ , \\
\frac{\partial Q_n}{\partial \alpha_{0,h}} &= -2\frac{1}{n}\sum_{i=1}^n (y_i - \hat{m}_H(x_i;\rho))\beta_{1,h}\sigma'(\alpha_{0,h} + x_i'\alpha_{1,h}) + 2\lambda\alpha_{0,h} \ , \quad h \in 1:H \ , \\
\frac{\partial Q_n}{\partial \alpha_{1,h}} &= -2\frac{1}{n}\sum_{i=1}^n (y_i - \hat{m}_H(x_i;\rho))\beta_{1,h}\sigma'(\alpha_{0,h} + x_i'\alpha_{1,h})x_i + 2\lambda\alpha_{1,h} \ , \quad h \in 1:H \ .
\end{aligned}$$

The algorithm is ran more efficiently on graphical processing units (GPUs) or other specialized hardware instead of central processsing units (CPUs). A good idea is to initialize all coefficients near (but not exactly at) zero and to standardize all covariates in x_i to have a mean of zero and a variance of one. To avoid local minima, one can ran the algorithm using many different starting values and take the lowest of all obtained minima. Going a step further, one can construct bagged neural networks, perturbing not only the initial values, but also the sample used in each neural network.

Neural networks are a very effective tool for prediction. When applied to binary rather than continuous outcome variables, they can be used for classification purposes, whereby the algorithm can be augmented to predict multiple outcome variables using the same neural network. However, being rather intransparent, neural networks are less useful for analyzing and interpreting the relationship between y_i and x_i and even less so for conducting inference on some aspect of this relationship.

satisfied in econometrics.

Appendix

Kernel Density Estimation

Kernel density estimation applies similar techniques as local regressions do to the estimation of a density rather than a conditional expectation function. The starting point is a set of draws $\{x_i\}_{i=1}^n$, whose pdf $p(x)$ is the object of interest.

One approach to estimate p is to impose a particular, parametric distributional family and estimate its parameters. For example, assuming that p is the Normal distribution, the estimation problem boils down to estimating its mean and variance, whereby their natural estimators are the sample mean and variance, respectively. However, rather than imposing a particular parametric functional form, one might want to estimate p non-parametrically, allowing for flexible functional forms determined by the data.

A natural non-parametric estimator of p is the histogram-estimator

$$\hat{p}(x_0) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2h}\, \mathbf{1}\left\{x_0 - h \leq x_i < x_0 + h\right\} \ ,$$

here with bin-width $2h$. However, this leads to edgy estimate, influenced by sampling noise, whereas we typically think of the underlying density as rather smooth. Instead of giving each observation x_i in a given interval like $x_0 \pm h$ equal weight, one could use a kernel function to weigh them according to their distance to x_0. This leads to the kernel density estimator

$$\hat{p}(x_0) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x_i - x_0}{h}\right) \ . \tag{14.4}$$

Provided that the kernel function is smooth, one obtains a smooth estimate $\hat{p}(x_0)$.

The analogous comments as in Section 14.2 apply; the choice of Kernel is less consequential than the choice of bandwidth h, whereby higher values for h lead to lower variance but increased bias, both of which differ across points x_0. One can show (see e.g. Cameron and Trivedi (2005, ch. 9)) that the bias at x_0 is equal to

$$b(x_0) \equiv \mathbb{E}[\hat{p}(x_0)] - p(x_0) = \frac{1}{2}h^2 p''(x_0) \int z^2 K(z) dz \ ,$$

and the asymptotic distribution of $\hat{p}(x_0)$ is

$$\sqrt{nh}(\hat{p}(x_0) - p(x_0) - b(x_0)) \xrightarrow{d} N\left(0, V(x_0)\right) \ , \quad V(x_0) = p(x_0) \int K(z)^2 dz \ .$$

As with local regression, such an analysis again motivates the computation of optimal bandwidths h under different kernels, sample sizes and properties of p, and it leads to Silverman's plug-in estimate. The convergence rate of $\hat{p}(x_0)$ is again $n^{2/5}$, slower than that of parametric density estimators. In contrast to local regressions, the asymptotic variance is more easily estimated, but the estimation of the bias is once again involved as it depends on the second-order derivative of p.

Extension to Multivariate Densities To estimate a density of a k-dimensional random variable x_i, the expression in Eq. (14.4) becomes

$$\hat{p}(x_0) = \frac{1}{nh^k} \sum_{i=1}^{n} \sum_{j=1}^{k} K\left(\frac{x_{ij} - x_{0j}}{h}\right) .$$

As with local regressions, this reduces the convergence rate to $n^{2/(k+4)}$.

Classification Based on Kernel Density Estimation Kernel density estimates can also be used for classification purposes. For example, you have a sample $\{x_i^s\}_{i=1}^{n_s}$ of characteristics of spam-emails (i.e. $x_i^s \in \mathcal{S}$ or $y_i^s = 1$, y_i being a dummy indicating spam) and a sample $\{x_i^v\}_{i=1}^{n_v}$ of characteristics of valid (no spam) emails (i.e. $x_i^s \in \mathcal{V}$ or $y_i^s = 0$). Given kernel density estimates $\hat{p}_s$ and $\hat{p}_v$, the probability that an email with characteristics x_0 belongs to the spam folder is

$$\hat{\mathbb{P}}[x_0 \in \mathcal{S}] = \frac{\hat{p}_s(x_0)}{\hat{p}_s(x_0) + \hat{p}_v(x_0)} .$$

This is readily extended to classification into C categories and can be augmented with prior probabilities $\{\pi_c\}_{c=1}^{C}$:

$$\hat{\mathbb{P}}[x_0 \in \mathcal{C}_c] = \frac{\pi_c \hat{p}_c(x_0)}{\sum_{c=1}^{C} \pi_c \hat{p}_c(x_0)} .$$