Chapter 4
Nonparametric Econometrics

Simon Fraser University
ECON 483
Summer 2023

Disclaimer

I do not allow this content to be published without my consent.

All rights reserved ©2023 Thomas Vigié

Introduction

- Linear estimators are estimators whose fitted values are obtained by taking a linear combination of the dependent variable observations $\boldsymbol{y_i}$
- In the OLS case, the weights are computed based on the covariances between $\boldsymbol{x_i}$ and $\boldsymbol{y_i}$
- The OLS estimator is a **global estimator**: It considers all the data at once and produces one estimation
- Nonparametric methods are **local**: Free of parametric restrictions about the functional form of the regression function, they can estimate the regression function at a point by considering nearby data
- By nearby data, we mean data close in terms of the covariates
- Why use an observation of 60 years of age to estimate the regression function for someone who is 20 years of age?

Outline

K Nearest Neighbours

K Nearest Neighbors methods

- **K nearest neighbors (KNN)** methods estimate $\boldsymbol{f(x_0)}$ by computing an average of the $\boldsymbol{y_i}$ whose $\boldsymbol{x_i}$ are the closest to the value $\boldsymbol{x_0}$
- Let $\boldsymbol{\mathcal{N}_0}$ be the set of $\boldsymbol{K}$ observations that are the closest to $\boldsymbol{x_0}$. The estimator is defined as

$$\hat{\boldsymbol{f}}(\boldsymbol{x_0}) = \frac{\boldsymbol{1}}{\boldsymbol{K}} \sum_{i \in \mathcal{N}_0} \boldsymbol{y_i}$$

- All the other observations are **not** used to compute $\hat{\boldsymbol{f}}(\boldsymbol{x_0})$

KNN properties

- KNN methods are **consistent**, as long as the number of nearest neighbors $\boldsymbol{K}$ increases as the sample size $\boldsymbol{n}$ increases
- Idea: To keep capturing the main patterns without overfitting, the number of neighbors must increase to lower the bias, **but not too fast** to keep the variance under control
- There is an **asymptotic distribution** for $\hat{\boldsymbol{f}}(\boldsymbol{x_0})$, so we can make inference (hypothesis tests and confidence intervals) about the true value $\boldsymbol{f}(\boldsymbol{x_0})$
- There are many other applications of nearest neighbors methods, but they are not very popular in Economics…

KNN properties (cont'd)

- The choice of K is crucial
  - If $K = 1$: The prediction $\hat{f}(x_0)$ is the observation y that has the closest x to x_0. If x_0 is a point of the data set, its prediction is its corresponding y_0: **Extreme interpolation** (low bias, high variance)
    $\Rightarrow$ **Overfitting**
  - If $K = n$: The prediction is the average point of the whole sample: $\bar{y}$. And **every** prediction is equal to that average: **Extreme smoothing** (high bias, low variance)
    $\Rightarrow$ **Underfitting**
- The optimal K is the one that minimizes a MSE type objective function to find the best bias-variance tradeoff
- If X_i is of dimension q, then the closest observations i are defined as the ones for which x_i is the closest to x_0 in terms of Euclidean distance:

$$\|x_0 - x_i\| \equiv \sqrt{(x_{0,1} - x_{i,1})^2 + \ldots + (x_{0,q} - x_{i,q})^2}$$

K Nearest Neighbors methods with weights

- It is also possible to weigh the observations differently than “in” (and then equal weight) or “out” (and then no weight)
- An observation that is close to the point we try to predict should get a higher weight, and an observation that is far should count less
- Let $\boldsymbol{w_i(x_0)}$ be a weight function such that $\boldsymbol{\sum_{i=1}^n w_i(x_0) = 1}$
- The estimate becomes

$$\boldsymbol{\hat{f}(x_0) = \sum_{i=1}^{n} w_i(x_0) y_i}$$

Selecting the optimal amount of Neighbors

- The number of nearest neighbors K has to mitigate the bias and the variance at the same time
- There exist different objective functions, but we will focus on the **leave-one-out cross validation** criterion function (Stone, 1964), where $\hat{k}$ minimizes

$$CV(k) \equiv \sum_{i=1}^{n} (y_i - \hat{f}_{-i}(x_i))^2$$

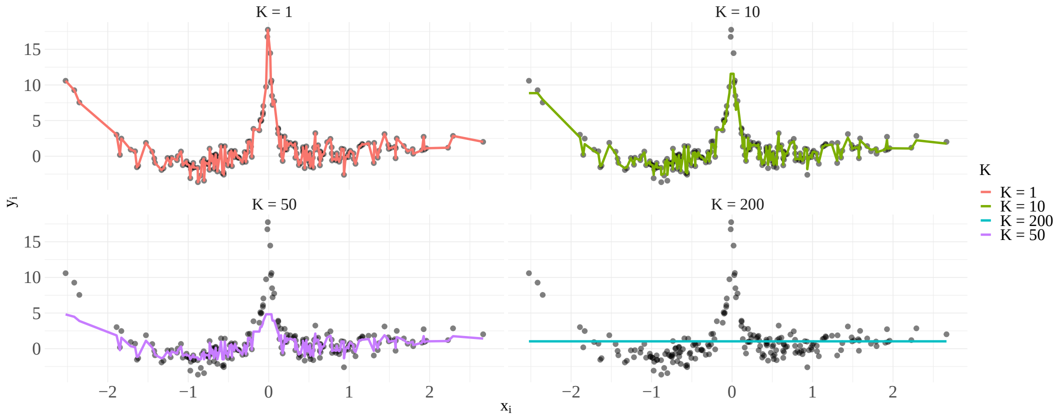
- $\hat{f}_{-i}(x_i)$ is called the **leave-one-out** estimator of $f(x_i)$. It is the estimate of $f(x_i)$ without using observation i in the process $\Rightarrow$ Observation i plays the role of test sample!
- It is equivalent to K-fold cross validation, but there are n folds: Use all the sample but observation 1 to estimate $f(x_1)$, then all the sample but observation 2 to predict $f(x_2)$, ...
- Instead of K MSEs to average over as in the model selection lecture, we now have n MSEs to average over

Illustration in Rstudio: The *FNN* package

- The ***FNN*** package allows to compute nearest neighbors estimates of all kind: Choose the number of neighbors by hand or automatically (by minimizing the leave-one-out CV criterion or other relevant criterion)
- ***knn.reg*** is particularly useful: Specify the dependent variable $\boldsymbol{Y_i}$, specify the training data set (data used to get the estimates), the test data set (the data $\boldsymbol{y_i}$ we try to predict using the corresponding $\boldsymbol{X_i}$), and the algorithm to find the optimal amount of neighbors (you can leave it at its default value)

```r
# K- nearest neighbors estimator
knn_1 <- knn.reg(train = data, test = data, y = y, k = 1)
knn_10 <- knn.reg(train = data, test = data, y = y, k = 10)
knn_50 <- knn.reg(train = data, test = data, y = y, k = 50)
knn_100 <- knn.reg(train = data, test = data, y = y, k = 200)
```

KNN regression in Rstudio



Kernel estimators

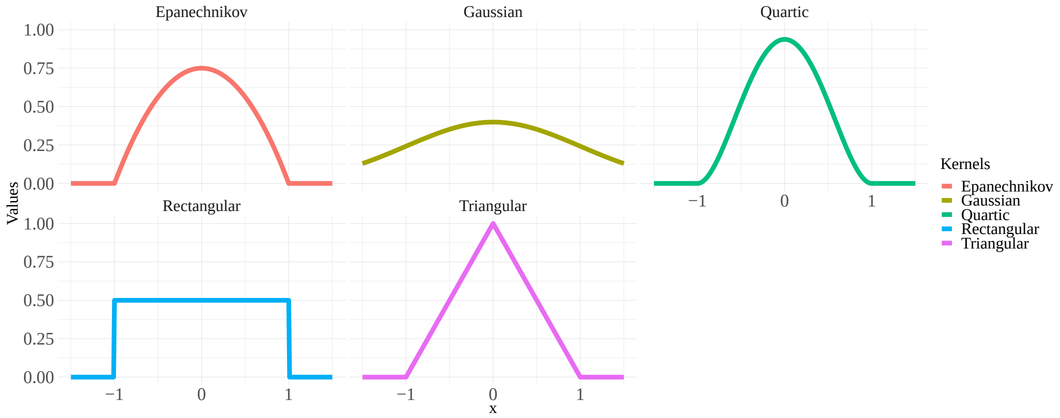
Kernel estimators

- Like the KNN estimators, kernel estimators are **linear** estimators:

$$\hat{f}(x_0) = \sum_{i=1}^{n} w_i(x_0) y_i$$

- But observations get weights based on the distance from the value x_0, not on a specific number of observations around x_0
- Consider a function $K\left(\frac{x_i - x_0}{h}\right)$ where h is called the **bandwidth**. The function $K()$ is called a **Kernel function** and has the following properties:
    - It is **non negative**: $K(u) \geq 0 \; \forall u$ (no weight is negative)
    - It is **symmetric**: $K(u) = K(-u)$ (so observations at the same distance of x_0 but on either side will get the same weight)
- When x_i is close to x_0, a higher weight is given: It is high when $\frac{x_i - x_0}{h}$ is small, and small when $\frac{x_i - x_0}{h}$ is high

Kernel functions



Kernel estimators

- The weights $w_i(x_0)$ are obtained as

$$w_i(x_0) = \frac{K\left(\frac{x_i - x_0}{h}\right)}{\sum_{j=1}^{n} K\left(\frac{x_j - x_0}{h}\right)}$$

  and the estimator is called the **Nadaraya-Watson (1964) estimator** or **local constant estimator**

- We can see that the weights sum to 1:

$$\sum_{i=1}^{n} w_i(x) = \sum_{i=1}^{n} \frac{K\left(\frac{x_i - x}{h}\right)}{\sum_{j=1}^{n} K\left(\frac{x_j - x}{h}\right)} = \frac{\sum_{i=1}^{n} K\left(\frac{x_i - x}{h}\right)}{\sum_{j=1}^{n} K\left(\frac{x_j - x}{h}\right)} = 1$$

Kernel estimators as least squares estimators

- It turns out that kernel estimators can be seen as least squares estimators!

$$\min_{\{a\}} \sum_{i=1}^{n} (y_i - a)^2 K\left(\frac{x_i - x_0}{h}\right)$$

- Take the FOC w.r.t a to get:

$$2\sum_{i=1}^{n} K\left(\frac{x_i - x_0}{h}\right)(y_i - \hat{a}) = 0$$

$$\sum_{i=1}^{n} K\left(\frac{x_i - x_0}{h}\right) y_i = \sum_{i=1}^{n} K\left(\frac{x_i - x_0}{h}\right) \hat{a}$$

$$\hat{a} = \frac{\sum_{i=1}^{n} y_i K\left(\frac{x_i - x_0}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{x_i - x_0}{h}\right)}$$

Kernel estimators vs nearest neighbors estimators

- The straightforward KNN estimator can be seen as the solution to

$$\min_{\{a\}} \sum_{i \in \mathcal{N}_0}^{n} (y_i - a)^2$$

  where $\mathcal{N}_0$ is the set of observations in the neighborhood of x_0

- Nearest neighbors have a **fixed number of observations** around x_0, whereas kernel methods have a **fixed window** around x_0

- If one uses weights for KNN estimator, the formula looks pretty similar to the local constant estimator! The neighbors and bandwidth are different concepts, but they play the same role

- With little data, nearest neighbors might include observations that are far from x_0 while kernel methods might only include one observation or two... In both cases, accuracy will be low

Kernel estimators: Properties

- The problem with accurate predictions using nearby observations is how close the nearby observations are
- A small bandwidth $\boldsymbol{h}$ will imply $\boldsymbol{K}\left(\frac{\boldsymbol{x_i}-\boldsymbol{x}}{\boldsymbol{h}}\right)$ will be small as soon as $\boldsymbol{x_i}$ is a bit far form $\boldsymbol{x}$ $\Rightarrow$ Only very nearby observations have a substantial contribution in estimating $\boldsymbol{f}(\boldsymbol{x})$: The estimation is **more local**, **less global**
- A big bandwidth $\boldsymbol{h}$ will imply $\boldsymbol{K}\left(\frac{\boldsymbol{x_i}-\boldsymbol{x}}{\boldsymbol{h}}\right)$ will be big as soon as $\boldsymbol{x_i}$ is different form $\boldsymbol{x}$ $\Rightarrow$ All the observations will have a similar contribution in estimating $\boldsymbol{f}(\boldsymbol{x})$: The estimation is **less local**, **more global**

Kernel estimators: Properties

- If all the observations $\boldsymbol{x_i}$ are weighed the same (large bandwidth $\boldsymbol{h}$), the weights can be ignored in the minimization problem
- Result: The estimator $\hat{\boldsymbol{f}}(\boldsymbol{x})$ does not pick up the patterns of the true function $\boldsymbol{f}(\boldsymbol{x})$ and is equal to $\bar{\boldsymbol{y}}$: **High bias, low variance**
  $\Rightarrow$ **Underfitting**
- If only the closest observations contribute to $\hat{\boldsymbol{f}}(\boldsymbol{x})$, the estimator picks too much of the pattern around the point of estimation: **Low bias, high variance**
  $\Rightarrow$ **Overfitting**
- Again, a **bias-variance tradeoff**

Kernel estimators: Asymptotic Properties

- It can be shown that

$$\hat{f}(x_0) - f(x_0) = O_p\left(h^2 + \frac{1}{\sqrt{nh}}\right)$$

- For the right hand side to go to zero, we need $h \to 0$ and $nh \to \infty$
- In words: The window around x_0 must decrease as the sample size increases, but the sample size must go to infinity faster than the bandwidth goes to 0

Optimal bandwidth selection

- The bandwidth $\boldsymbol{h}$ is to kernels what $\boldsymbol{K}$ is to nearest neighbors: A **tuning** parameter that has to find the optimal balance between bias and variance
- As for KNN methods, the leave-one-out cross validation can be used:

$$CV(h) \equiv \sum_{i=1}^{n} (y_i - \hat{f}_{-i}(x_i))^2$$

- $\hat{f}_{-i}(x_i)$ is the **leave-one-out** estimator of $f(x_i)$. It is the estimate of $f(x_i)$ without using observation i in the process $\Rightarrow$ Observation i plays the role of test sample!

Beyond one covariate: The curse of dimensionality

- We can equally define a multivariate version of kernel estimators. Say we have $\boldsymbol{p}$ covariates
- Kernel functions are now multivariate kernels. One straightforward candidate is the product kernel:

$$\boldsymbol{\mathcal{K}}\left(\frac{\boldsymbol{x_i - x_0}}{\boldsymbol{h}}\right) \equiv \boldsymbol{K}\left(\frac{\boldsymbol{x_{i,1} - x_{0,1}}}{\boldsymbol{h_1}}\right) \times \ldots \times \boldsymbol{K}\left(\frac{\boldsymbol{x_{i,p} - x_{0,p}}}{\boldsymbol{h_p}}\right)$$

- One bandwidth per covariate, so $\boldsymbol{p}$ bandwidths have to be found (again, leave-one-out cross validation!)

Beyond one covariate: The curse of dimensionality

- It can be shown that

$$\hat{f}(x_0) - f(x_0) = O_p\left(\sum_{j=1}^{p} h_j^2 + \frac{1}{\sqrt{nh_1h_2...h_p}}\right)$$

- For the right hand side to go to zero, we need $h_j \to 0 \; \forall j = 1, ..., p$ and $nh_1h_2...h_p \to \infty$
- Each bandwidth should go to 0, but we also need $nh_1h_2...h_p \to \infty$
- So the sample size needs to increase quickly to keep the second term low (the variance term)
- It is the **curse of dimensionality**: As the number of covariates increases, the sample size needs to increase faster in order to stay accurate
- Result: The rate of convergence to the true regression function is slower than a parametric or semi parametric model

Fighting the curse of dimensionality

- In practice, one deals with more than one covariate, making nearest neighbors and kernel estimators less appealing due to the curse of dimensionality
- Other models have been proposed to deal with it:
- General additive models

$$\boldsymbol{Y_i = f_1(X_{i,1}) + f_2(X_{i,2}) + ... + f_p(X_{i,p}) + u_i}$$

- By separating the functions, the rate of convergence is the one of a univariate nonparametric regression

Fighting the curse of dimensionality

- Other models include a parametric component **and** a nonparametric component: They are **semi parametric**
- **Partially linear models**

$$Y_i = \beta_1 X_{i,1} + \ldots + \beta_{p-1} X_{i,p-1} + f(X_{i,p}) + u_i$$

- We can obtain estimates of the $\beta_j,\ j = 1, \ldots, p-1$ like an OLS estimation while taking the nonparametric component $f(X_{i,p})$ into account using **Robinson (1988)**'s double residuals method (kernel estimation of the nonparametric component, and then OLS using the residuals $y_i - \hat{f}(x_{i,p})$ and $x_{i,j} - \hat{f}(x_{i,p})$)
- **Single index models**

$$Y_i = f(\beta_1 X_{i,1} + \ldots + \beta_p X_{i,p}) + u_i$$

- Only **one** variable now: The linear combination of the $X_{i,j},\ j = 1, \ldots, p$

Beyond local constant: Local linear estimators

- The estimator seen above, by construction, performs poorly when estimating the regression function around points at the boundary of the support of the data
- Intuition: If we want to predict a point that has no observation to its left, we will be using observations to the right only to compute the average, and the prediction will be highly inaccurate: It is the **boundary bias**
- An alternative estimator was proposed. Instead of computing a local average around $\boldsymbol{x_0}$, run a weighted linear regression **centered** around $\boldsymbol{x_0}$:

$$\min_{\{b_0,b_1\}} \sum_{i=1}^{n} (y_i - b_0 - b_1(x_i - x_0))^2 K\left(\frac{x_i - x_0}{h}\right)$$

Local linear estimators

- Note: There is **one** minimization problem per x we want to estimate $f()$ at, instead of one single minimization problem for any prediction like the OLS estimator
- If we want to estimate $f(x_i)\ \forall i = 1, ..., n$ (the observations of the sample), we can use matrix notation to gather these n minimization problems into one nice formula
- The result is still $\hat{f}(x_0) = \sum_{i=1}^{n} w_i(x_0)y_i$, but the weights are a more complicated formula than in the local constant case
- If h is high enough, the weights do not matter and the result is a straight OLS estimation! The **local** estimator becomes **global**
- In practice, the local linear estimator is preferred to the local constant one as both share the same asymptotic properties except for the boundary bias

Local polynomial estimators

- Why stop at a local linear estimator?
- We can go beyond and add polynomial terms to estimate the derivatives of $f()$ at a point x_0:

$$\min_{\{b_0,b_1,\dots,b_q\}} \sum_{i=1}^{n} (y_i - b_0 - b_1(x_i - x_0) - \dots - b_q(x_i - x_0)^q)^2 K\left(\frac{x_i - x_0}{h}\right)$$

  where q is the order of the local polynomial
- We now have two tuning parameters: The bandwidth h and the polynomial order q
- How to get the optimal values for both? Leave-one-out cross validation!

Illustration in **Rstudio**: The *np* package

- The ***np*** contains a lot of nonparametric methods, with different features and tuning parameters selection options (choice of kernel function, bandwidth selection)
- First, use ***npregbw*** to find the optimal bandwidth(s)
- Second, compute the predictions using ***npreg*** and include the bandwidth found in the previous step

Kernel estimators in **Rstudio**: Local constant

```r
# Local constant estimator
bw_lc <- npregbw(y ~ x, regtype = "lc", ckertype = "epanechnikov")
```

```
## Multistart 1 of 1 |Multistart 1 of 1 |Multistart 1 of 1 |Multistart 1 of 1 /Multistart 1 of 1 -Mul
```

```r
model_lc <- npreg(bws = bw_lc)
local_constant <- model_lc$mean
summary(model_lc)
```

```
##
## Regression Data: 200 training points, in 1 variable(s)
##                         x
## Bandwidth(s): 0.04811532
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 1.253306
## R-squared: 0.8400364
##
## Continuous Kernel Type: Second-Order Epanechnikov
## No. Continuous Explanatory Vars.: 1
```

Kernel estimators in **Rstudio**: Local linear

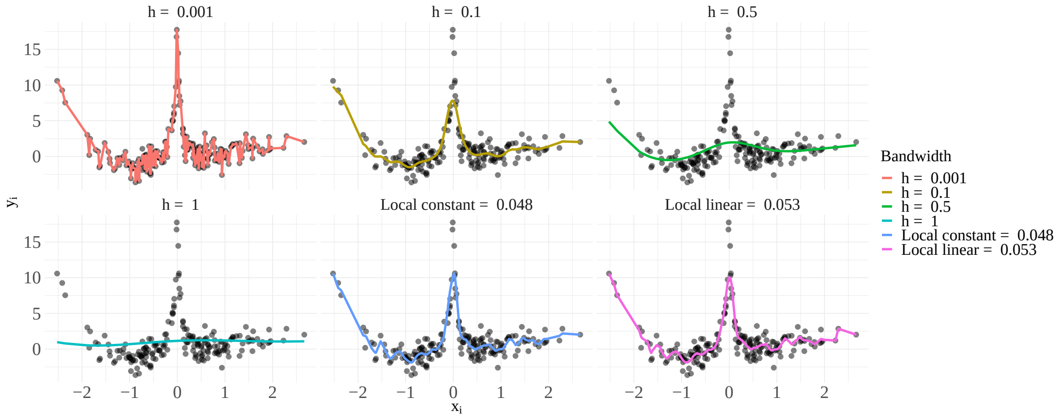
```
# Local linear estimator
bw_ll <- npregbw(y ~ x, regtype = "ll", ckertype = "epanechnikov")
```

```
## Multistart 1 of 1 |Multistart 1 of 1 |Multistart 1 of 1 |Multistart 1 of 1 /Multistart 1 of 1 -Mul
```

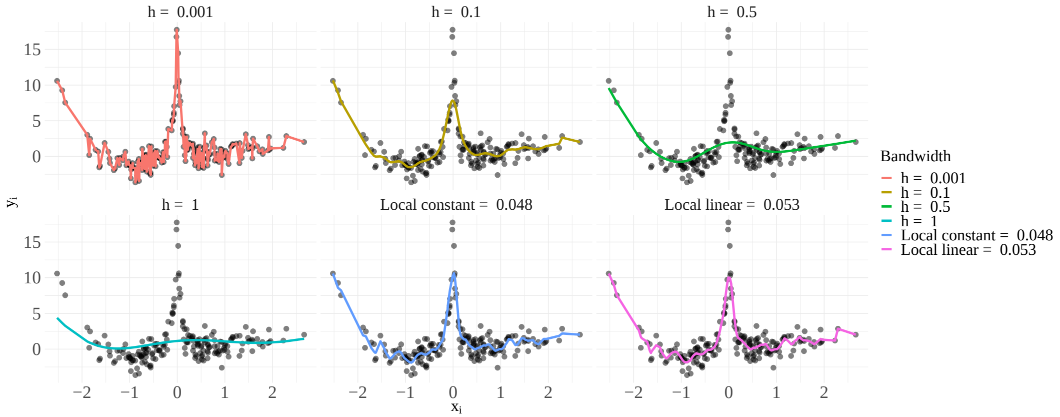
```
model_ll <- npreg(bws = bw_ll)
local_linear <- model_ll$mean
summary(model_ll)
```

```
##
## Regression Data: 200 training points, in 1 variable(s)
##                         x
## Bandwidth(s): 0.05319041
##
## Kernel Regression Estimator: Local-Linear
## Bandwidth Type: Fixed
## Residual standard error: 1.253011
## R-squared: 0.8441363
##
## Continuous Kernel Type: Second-Order Epanechnikov
## No. Continuous Explanatory Vars.: 1
```

Kernel estimators in **Rstudio**: Local constant



Kernel estimators in **Rstudio**: Local linear



Other nonparametric methods

- **Splines regression** consists in estimating piece wise polynomials. Between two knots (say, $\boldsymbol{x = 0}$ and $\boldsymbol{x = 2}$), a polynomial is fitted. Between two other knots (say, $\boldsymbol{x = 2}$ and $\boldsymbol{x = 5}$), another polynomial of possibly different order is fitted
- To keep the function smooth, the method makes sure that at each knot, the polynomials from either side have the same derivative
- **Sieves regression** is a **global** estimation method, consisting in regressing $\boldsymbol{Y_i}$ on sum transformations of $\boldsymbol{X_i}$
- Could be power functions (polynomials), but also sine/cosine functions, as well as orthogonal polynomials
- How many terms to include is the question: For consistency, the number of terms to include must increase at a certain rate with the sample size (similar idea as the bandwidth for kernel methods)

Conclusion

- Nearest neighbors and kernel methods are rich, and many improvements have been developed
- The choice of the kernel function makes a (little) difference: One can show that the Epanechnikov kernel leads to a lower MSE than the others, but the Gaussian kernel is a commonly used one
- These methods can be used to estimate density functions. They deliver a smooth density curve instead of histograms
- Their weakness to a high number of covariates makes them less appealing for big data problems
- But their intuition remains powerful and they are still widely used