Bivariate smoothing and more

Niels Richard Hansen

September 12, 2018
A scatter plot

\[ p \leftarrow \text{qplot}(\text{Year}, \text{Temperature}, \text{data = Nuuk_year}); \quad p \]
Nearest neighbor estimation

Data: \((x_1, y_1), \ldots, (x_n, y_n)\)

The \(k\) nearest neighbor smoother in \(x\) is defined as

\[
\hat{f}(x) = \frac{1}{k} \sum_{j \in N(x)} y_j
\]

where \(N(x)\) is the set of indices for the \(k\) nearest neighbors of \(x\).

This is an estimator of \(f(x) = E(Y \mid X = x)\).
Nearest neighbor estimation

The estimator in $x_i$ is

$$\hat{f}_i = \frac{1}{k} \sum_{j \in N_i} y_j$$

where $N_i = N(x_i)$.

With $S_{ij} = \frac{1}{k} 1(j \in N_i)$ and $S = (S_{ij})$

$$\hat{f} = (\hat{f}_i) = Sy.$$ 

\hat{f} is an estimator of the vector $(f(x_i))$. 
Linear smoothers

Any estimator of $(f(x_i))$ of the form $Sy$ for a smoother matrix $S$ is called a linear smoother.

The $k$ nearest neighbor smoother is a simple example of a linear smoother that works for $x$-values in any metric space.

The representation of a linear smoother as a matrix-vector product,

$$Sy$$

is theoretically useful, but often not the best way to actually compute $\hat{f}$. 
Running mean

When $x_i \in \mathbb{R}$ we can sort data according to $x$-values and then use a symmetric neighbor definition:

$$N_i = \{i - (k - 1)/2, i - (k - 1)/2 + 1, \ldots, i - 1, i, i + 1, \ldots, i + (k - 1)/2\}$$

(for $k$ odd.)

This simplifies computations: we don't need to keep track of metric comparisons, only the order matters.
Bias-variance tradeoff

If data is i.i.d. with $V(Y \mid X) = \sigma^2$:

$$
\text{MSE}(x) = E((f(x) - \hat{f}(x))^2 \mid X = x)
= E((f(x) - E(\hat{f}(x)))^2 \mid X = x)
+ E((\hat{f}(x) - E(\hat{f}(x)))^2 \mid X = x)
= \left[ f(x) - \frac{1}{k} \sum_{l \in N(x)} f(x_l) \right]^2 + \frac{\sigma^2}{k}
$$

Here $f(x) = E(Y \mid X = x)$.

Small $k$ gives large variance and small bias (if $f$ is smooth).

Large $k$ give small variance and potentially large bias (if $f$ is not constant).
Running mean

Implementation using the identity

\[
\hat{f}_{i+1} = \hat{f}_i - y_{i-(k-1)/2} + y_{i+(k+1)/2},
\]

assuming \(y\) in correct order.

```r
## The vector 'y' must be sorted according to the x-values
runMean <- function(y, k) {
  n <- length(y)
  m <- floor((k - 1) / 2)
  k <- 2 * m + 1  
  y <- y / k
  s <- rep(NA, n)
  s[m + 1] <- sum(y[1:k])
  for(i in (m + 1):(n - m - 1))
    s[i + 1] <- s[i] - y[i - m] + y[i + 1 + m]
  s
}
```
Visualization

\[ \text{f\_hat} \leftarrow \text{runMean(Nuuk\_year$Temperature, 11)} \]
\[ p + \text{geom\_line(aes(y = f\_hat, color = "blue"))} \]
Using 'filter'

The R function `filter` computes running means and moving averages.

```r
f_hat_filter <- stats::filter(Nuuk_year$Temperature, rep(1/11, 11))
range(f_hat_filter - f_hat, na.rm = TRUE)
```

```
## [1] -1.332268e-15  4.440892e-16
```

```r
f_hat_filter[c(1:8, 140:147)]
```

```
## [1]     NA     NA     NA     NA     NA -1.8522727
## [7] -1.7151515 -1.5537879 -0.2340909 -0.1886364 -0.1166667     NA
## [13]    NA     NA     NA     NA     NA
```
Benchmarking

<table>
<thead>
<tr>
<th></th>
<th>expr</th>
<th>median</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1 %*% y[1:512]</td>
<td>342.5765</td>
</tr>
<tr>
<td>2</td>
<td>S2 %*% y[1:1024]</td>
<td>1417.2530</td>
</tr>
<tr>
<td>3</td>
<td>S3 %*% y[1:2048]</td>
<td>5543.0610</td>
</tr>
<tr>
<td>4</td>
<td>S4 %*% y[1:4196]</td>
<td>25114.1570</td>
</tr>
<tr>
<td>5</td>
<td>runMean(y[1:512], k = 11)</td>
<td>88.5525</td>
</tr>
<tr>
<td>6</td>
<td>runMean(y[1:1024], k = 11)</td>
<td>165.3515</td>
</tr>
<tr>
<td>7</td>
<td>runMean(y[1:2048], k = 11)</td>
<td>325.8525</td>
</tr>
<tr>
<td>8</td>
<td>runMean(y[1:4196], k = 11)</td>
<td>651.5475</td>
</tr>
<tr>
<td>9</td>
<td>stats::filter(y[1:512], rep(1/11, 11))</td>
<td>107.6195</td>
</tr>
<tr>
<td>10</td>
<td>stats::filter(y[1:1024], rep(1/11, 11))</td>
<td>131.7065</td>
</tr>
<tr>
<td>11</td>
<td>stats::filter(y[1:2048], rep(1/11, 11))</td>
<td>180.9280</td>
</tr>
<tr>
<td>12</td>
<td>stats::filter(y[1:4196], rep(1/11, 11))</td>
<td>311.0550</td>
</tr>
</tbody>
</table>

The Matrix-vector multiplication is $O(n^2)$.

The two other algorithms are $O(n)$. 
LOOCV

The running mean / nearest neighbour smoother is a linear smoother, \( \hat{f} = SY \).

How to predict \( y_i \) if \((x_i, y_i)\) is left out?

A definition for a linear smoother is

\[
\hat{f}_i^{−i} = \sum_{j \neq i} \frac{S_{ij}y_j}{1 - S_{ii}}.
\]

For many smoothing procedures with a natural "out-of-sample" prediction method the identity above holds.

It follows that for leave-one-out cross validation

\[
\text{LOOCV} = \sum_i (y_i - \hat{f}_i^{−i})^2 = \sum_i \left( \frac{y_i - \hat{f}_i}{1 - S_{ii}} \right)^2
\]
LOOCV

```r
loocv <- function(k, y) {
  f_hat <- runMean(y, k)
  mean(((y - f_hat) / (1 - 1/k))^2, na.rm = TRUE)
}

k <- seq(3, 40, 2)
CV <- sapply(k, function(kk) loocv(kk, Nuuk_year$Temperature))
k_opt <- k[which.min(CV)]
qplot(k, CV) + geom_line() + geom_vline(xintercept = k_opt, color = "red")
```
The optimal choice of $k$ is 15.
Smoothing splines

The minimizer of

\[
L(f) = \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \int f''(z)^2 \, dz
\]

is a cubic spline

with knots in the data points \( x_i \), that is, a function

\[
f = \sum_i \beta_i \phi_i
\]

where \( \phi_i \) are basis functions for the \( n \)-dimensional space of such splines.

Cubic splines are piecewise degree 3 polynomials in between knots.
Smoothing splines

In vector notation

$$\hat{f} = \Phi \hat{\beta}$$

with $$\Phi_{ij} = \varphi_j(x_i),$$

and

$$L(f) = (y - f)^T(y - f) + \lambda \|f''\|^2_2$$

$$= (y - \Phi \beta)^T(y - \Phi \beta) + \lambda \beta^T \Omega \beta$$

with

$$\Omega_{ij} = \int \varphi_i''(z) \varphi_j''(z) dz.$$
Smoothing splines

The minimizer is

\[ \hat{\beta} = (\Phi^T\Phi + \lambda \Omega)^{-1} \Phi^T y \]

with resulting smoother

\[ \hat{f} = \Phi((\Phi^T \Phi + \lambda \Omega)^{-1} \Phi^T y). \]

\[ S_\lambda \]
library(splines)

## Note the specification of repeated boundary knots
knots <- c(0, 0, 0, seq(0, 1, 0.2), 1, 1, 1)
xx <- seq(0, 1, 0.005)
B_splines <- splineDesign(knots, xx)
matplot(xx, B_splines, type = "l", lty = 1)
Penalty matrix

I have implemented a function for computing $\Omega$ (see notes).

Omega <- pen_mat(seq(0, 1, 0.1))
image(Matrix(Omega))

Dimensions: 13 x 13
Fitting a smoothing spline

We implement the matrix-algebra directly for computing $S_{\lambda}y$.

```r
inner_knots <- Nuuk_year$Year
Phi <- splineDesign(c(rep(range(inner_knots), 3), inner_knots), inner_knots)
Omega <- pen_mat(inner_knots)
smoother <- function(lambda)
  Phi %*% solve(crossprod(Phi) + lambda * Omega, t(Phi) %*% Nuuk_year$Temperature)
```
Fitting a smoothing spline

```r
p + geom_line(aes(y = smoother(0.1)), color = "blue") +  # Undersmooth
  geom_line(aes(y = smoother(10)), color = "red") +    # Smooth
  geom_line(aes(y = smoother(1000)), color = "purple")  # Oversmooth
```
Generalized cross-validation

With $df = \text{trace}(S) = \sum_{i=1}^{n} S_{ii}$ we replace $S_{ii}$ in LOOCV by $df/n$ to get the generalized cross-validation criterion

$$GCV = \sum_{i=1}^{n} \left( \frac{y_i - \hat{f}_i}{1 - df/n} \right)^2.$$

```r
# Function for gcv

gcv <- function(lambda, y) {
  S <- Phi %*% solve(crossprod(Phi) + lambda * Omega, t(Phi))
  df <- sum(diag(S))  # The trace of the smoother matrix
  sum(((y - S %*% y) / (1 - df / length(y)))^2, na.rm = TRUE)
}

# Grid of lambda values

lambda <- seq(50, 250, 2)
GCV <- sapply(lambda, gcv, y = Nuuk_year$Temperature)
lambda_opt <- lambda[which.min(GCV)]
```

Then we apply this function to a grid of $\lambda$-values and choose the value of $\lambda$ that minimizes GCV.
Generalized cross-validation

\[ qplot(\text{lambda}, \text{GCV}) + \text{geom_vline(xintercept = lambda_opt, color = "red")} \]
Optimal smoothing spline

temp_smooth_opt <- Phi %*% solve(crossprod(Phi) + lambda_opt * Omega, 
t(Phi) %*% Nuuk_year$Temperature) 
p + geom_line(aes(y = temp_smooth_opt), color = "blue")
Using `smooth.spline`

```r
temp_smooth_splines <- smooth.spline(Nuuk_year$Year, Nuuk_year$Temperature, all.knots = TRUE)  ## Don't use fast heuristic
p + geom_line(aes(y = temp_smooth_splines$y), color = "blue")
```
Efficient computations

In practice we use $p < n$ basis functions.

Using the singular value decomposition

$$\Phi = UDV^T$$

it holds that

$$S_\lambda = \tilde{U}(I + \lambda \Gamma)^{-1}\tilde{U}^T$$

where $\tilde{U} = UW$ and

$$D^{-1}V^T\Omega V D^{-1} = \Psi \Gamma \Psi^T.$$
Efficient computations

The interpretation of this representation is as follows.

- First, the coefficients, $\hat{\beta} = \tilde{U}^T y$, are computed for expanding $y$ in the basis given by the columns of $U$.

- Second, the $i$th coefficient is shrunk towards 0,

  $$\hat{\beta}_i(\lambda) = \frac{\hat{\beta}_i}{1 + \lambda y_i}.$$ 

- Third, the smoothed values, $\tilde{U}\hat{\beta}(\lambda)$, are computed as an expansion using the shrunken coefficients.
The Demmler-Reinsch basis (columns of $\tilde{\mathbf{U}}$)
and the eigenvalues $\gamma_i$
The \texttt{ksmooth} function implements the Nadaraya-Watson smoother. There is no automatic choice of bandwidth.

Kernel smoothing for bivariate smoothing is easy to implement along the same lines as kernel smoothing for density estimation. Binning can be useful for handling large data sets.

Local regression smoothing can be implemented using \texttt{lm} and its \texttt{weights} argument, or perhaps more efficiently using \texttt{lm.wfit}. Refits are in principle needed in all points.

The \texttt{locpoly} function in the KernSmooth package does local polynomial regression using binning. It does not do automatic bandwidth selection.

For kernel smoothing it is quite easy to compute the diagonals in the smoothing matrix and thus LOOCV or GCV.
Loess

The nonlinear loess estimator is implemented in the `loess` function. It could be implemented using `lm.wfit`.

The `loess` function does not automatically select the span but has a default of 0.75, which is often too large.

Since loess is nonlinear, the formulas for linear smoothers do not apply. One can instead implement 5- or 10-fold cross validation.

The `loess` function does return an object with a `trace.hat` entry, which might be used as a surrogate for `trace(S)` for GCV, say.

Loess is a robust smoother (linear smoothers are not) and relatively insensitive to outliers.
A loess smoother

```r
p + geom_smooth()
```

```r
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```
Another loess smoother

```
p + geom_smooth(method = "loess", span = 0.5)
```
temp_hist <- hist(Nuuk_year$Temperature)
Histogram objects

class(temp_hist)

## [1] "histogram"

str(temp_hist)

## List of 6
## $ breaks : num [1:9] -5 -4 -3 -2 -1 0 1 2 3
## $ counts : int [1:8] 4 14 23 50 42 13 0 1
## $ density : num [1:8] 0.0272 0.0952 0.1565 0.3401 0.2857 ...
## $ mids : num [1:8] -4.5 -3.5 -2.5 -1.5 -0.5 0.5 1.5 2.5
## $ xname : chr "Nuuk_year$Temperature"
## $ equidist: logi TRUE
## - attr(*, "class")= chr "histogram"
Histogram objects

```r
## $breaks
## [1] -5 -4 -3 -2 -1  0  1  2  3
##
## $counts
## [1]  4 14 23 50 42 13  0  1
##
## $density
## [1] 0.027210884 0.095238095 0.156462585 0.340136054 0.285714286 0.088435374
## [7] 0.000000000 0.006802721
##
## $mids
## [1] -4.5 -3.5 -2.5 -1.5 -0.5  0.5  1.5  2.5
##
## $xname
## [1] "Nuuk_year$Temperature"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```
Getting help

You can find documentation for `plot` using e.g.

```
?plot
```

However, this will be uninformative on how an object of class `histogram` is plotted. Try instead

```
?plot.histogram
```

This will give the documentation for the plot method for objects of class `histogram`. 
Return values

Return values that don't fit into one of the basic data structures can be returned as a list.

tmp <- integrate(sin, 0, 1)
tmp

## 0.4596977 with absolute error < 5.1e-15

str(tmp)

## List of 5
## $ value : num 0.46
## $ abs.error : num 5.1e-15
## $ subdivisions: int 1
## $ message : chr "OK"
## $ call : language integrate(f = sin, lower = 0, upper = 1)
## - attr(*, "class")= chr "integrate"
The return value of \texttt{integrate}

The R function \texttt{integrate} returns a list. It has class

\begin{verbatim}
class(tmp)
\end{verbatim}

\begin{verbatim}
## [1] "integrate"
\end{verbatim}

When you ask R to print this object it doesn't just print out all the values in the list. It gives a suitably formatted result as implemented in the print method for objects of class integrate.
The `print.integrate` function is not exported from the `stats` package. It is in the namespace of the `stats` package, and to access it directly we use `stats:::`.
S3 in two minutes

- S3 classes are standard data structures (typically lists) with class labels.
- It is an informal system. No checks of object content.
- One implements a generic function via `UseMethod`. E.g.

```r
## function (x, y, ...) 
## UseMethod("plot") 
## <bytecode: 0x7fa4ac32ea60> 
## <environment: namespace:graphics>
```

- Then methods for specific classes are implemented as standard R functions with the naming convention `f.classname` for a method for class `classname` of the function `f`.
- Widely used to write methods for the generic functions `print`, `plot` and `summary`.
On microbenchmark

```r
micro_example <- microbenchmark(integrate(sin, 0, 1))
micro_example
```

```
## Unit: microseconds
##                  expr    min     lq     mean median    uq    max neval
##  integrate(sin, 0, 1) 13.251 13.763 14.98458  14.38 14.68 67.431   100
```

```r
class(micro_example)
```

```
## [1] "microbenchmark" "data.frame"
```
On microbenchmark

str(micro_example)

## Classes 'microbenchmark' and 'data.frame': 100 obs. of 2 variables:
## $ expr: Factor w/ 1 level "integrate(sin, 0, 1)"; 1 1 1 1 1 1 1 1 1 1 ...
## $ time: num 67431 21466 15887 14618 14358 ...

head(as.data.frame(micro_example))

## expr time
## 1 integrate(sin, 0, 1) 67431
## 2 integrate(sin, 0, 1) 21466
## 3 integrate(sin, 0, 1) 15887
## 4 integrate(sin, 0, 1) 14618
## 5 integrate(sin, 0, 1) 14358
## 6 integrate(sin, 0, 1) 14676
On microbenchmark

```r
## function (x, unit, order, signif, ...)
## {
##     s <- summary(x, unit = unit)
##     timing_cols <- c("min", "lq", "median", "uq", "max", "mean")
##     if (!missing(signif)) {
##         s[timing_cols] <- lapply(s[timing_cols], base::signif,
##                                  signif)
##     }
##     cat("Unit: ", attr(s, "unit"), "\n", sep = "")
##     if (!missing(order)) {
##         if (order %in% colnames(s)) {
##             s <- s[order(s[[order]]), ]
##         } else {
##             warning("Cannot order results by", order, ".")
##         }
##     }
##     print(s, ..., row.names = FALSE)
## }
## <bytecode: 0x7fa4b3937118>
## <environment: namespace:microbenchmark>
```
On microbenchmark

If package multcomp is installed, the summary method will run a multiple comparison test to produce statistically reliable ranking. This can be time consuming if many expressions are compared.
Exercise

This exercise assumes that you have a working `myHist` function. See my solution for help.

- Modify your `myHist` function so that it returns an object of class `myHistogram`, which is not plotted. Write a print method for objects of this class, which prints just the number of cells.

Hint: It can be useful to know about the function `cat`.

- How can you assign a class label to the returned object so that it is printed using your new print method, but it is still plotted as a histogram when given as argument to `plot`?

- Write a `summary` method that returns a data frame with first column containing the midpoints of the cells and second column containing the counts.

- Write a new `plot` method for objects of class `myHistogram` that uses `ggplot2` for plotting the histogram.
A linear smoother

```r
p + geom_smooth(method = "lm")
```
A polynomial smoother

\[ p + \texttt{geom\_smooth(method = "lm", formula = y \sim \texttt{poly}(x, 5))} \]
Another polynomial smoother

```r
p + geom_smooth(method = "lm", formula = y ~ poly(x, 20))
```
A spline smoother

```r
p + geom_smooth(method = "gam", formula = y ~ s(x))
```
Another spline smoother

```r
p + geom_smooth(method = "gam", formula = y ~ s(x, k=40))
```
Smoothing with ggplot2

The `geom_smooth` function easily adds misc. model fits or scatter plot smoothers to the scatter plot.

The `stat_smooth` function is responsible for delegating the computations. See `?stat_smooth`.

Spline smoothing is performed via the `gam` function in the mgcv package, whereas loess smoothing is via the `loess` function in the stats package.

Any "smoother" can be used that supports a formula interface and has a prediction function adhering to the standards of `predict.lm`. 
An interface for `geom_smooth`.

```r
rMean <- function(..., data, k = 5) {
  ord <- order(data$x)
  structure(list(x = data$x[ord], y = runMean(data$y[ord], k = k)),
               class = "rMean")
}
predict.rMean <- function(object, newdata, ...)
  approx(object$x, object$y, newdata$x)$y  # Linear interpolation
```
A running mean

```r
p + geom_smooth(method = "rMean", se = FALSE, n = 200)
```
Another running mean

\[
p + \text{geom\_smooth(method = "rMean", se = FALSE, n = 200, method.\text{args} = \text{list(k = 13)})}
\]
Boundary

```r
rMean <- function(..., data, k = 5, boundary = NULL) {
  ord <- order(data$x)
  y <- data$y[ord]
  n <- length(y)
  m <- floor((k - 1) / 2)
  if (m > 0 & !is.null(boundary)) {
    if (boundary == "pad")
      y <- c(rep(y[1], m), y, rep(y[n], m))
    if (boundary == "rev")
      y <- c(y[m:1], y, y[n:(n - m + 1)])
  }
  s <- runMean(y, k = k)
  if(!is.null(boundary))
    s <- na.omit(s)
  structure(list(x = data$x[ord], y = s), class = "rMean")
}
```
No boundary
Boundary, padding
Boundary, reversion