Stat 205: Introduction to Nonparametric Statistics

Lecture 09: Tuning Smoothers

Instructor David Donoho; TA: Yu Wang

Smoothers, 1

- Data $Y = (y_i)_{i=1}^n$
- Data $X = (x_i)_{i=1}^n$
- **Black Box**

  $$s = S(Y; X)$$

  here

  - $Y = (Y_i)$ vector of inputs
  - $s = (s_i)$ vector of outputs
  - $s_i = S(Y)_i$ is the i-th output.
- **Smoothing:** informally

  $$|s_i - s_j| \leq C \cdot |x_i - x_j|$$

  (can be true always, or in some 'typical sense')
- **Motivation:**

  $$y_i = \mu(x_i) + z_i, i = 1, \ldots, n$$

  - (z_i) is 'noisy': oscillates rapidly
  - $\mu()$ is ‘smooth': changes gradually

Smoothers, 2

- Many useful smoothers to choose from
- Typical API:
  `output = proc(x,y, tuning)[["fitted.values"]]`
- Examples in R below

```
Linear Fit s = lm(y~x)[["fitted.values"]]
Polynomial Fitting  s = lm(y~x+ I(x^2)+I(x^3))[["fitted.values"]]
Regressogram  s = regressogram(x,y)[["y"]]
(wtd) Local Averaging s = loess(y~x,span=k/n)[["y"]]
Kernel Smoothing  s = ksmooth(x,y,bandwidth=h)[["y"]]
Spline  s = smooth.spline(x, y, lambda=lam,all.knots=FALSE,n.knots=m)[["y"]]
Smoothing Spline s=smooth.spline(x, y, lambda=lam,all.knots=TRUE)[["y"]]
Automatic Smoothing s=smooth.spline(x, y, cv=TRUE)[["y"]]}
```

Regressogram

- Tuning Parameter m number of bins
- Suppose $a = min_i x_i$, $b = max_i x_i$.
- Partition $[a, b]$ into m bins $B_1, \dots, B_m$; each containing k points.
- For a point x, $B(x)$ is the bin containing x.
- Regressogram Estimator

$$\hat{s}_n^{rg}(x) = Ave\{Y_i | X_i \in B(x)\}.$$

k-Nearest-neighbor Procedure

- Tuning Parameter k number of neighbors
- For a point x, $N_k(x)$ is the set of k-nearest neighbors

  $N_k(x) = \{i : d(x_i, x)$ is among the k smallest distances $d(x_j, x$

  (assume no ties, or if ties break randomly)
- k-nn Estimator

$$\hat{s}_n^{knn}(x) = Ave\{Y_i | X_i \in N_k(x)\}.$$

Local Average Procedure

- Tuning Parameter h bandwidth
- For a point x, $B_h(x)$ is the interval $I_h(x) = [x - h/2, x + h/2]$.
- Local Average Estimator

$$\hat{s}_n^{lav}(x) = Ave\{Y_i | X_i \in B_h(x)\}.$$

Kernel Procedure

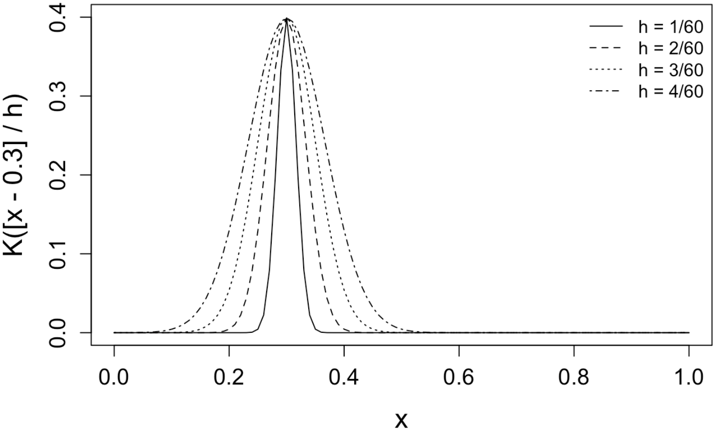
- Tuning Parameter h bandwidth; Kernel $K()$
- Local Weights

$$w_{x,h}(t) = \frac{K(\frac{t-x}{h})}{\sum_i K(\frac{x_i - x}{h})}$$

- Kernel Smooth

$$\hat{s}_h^{ksm}(x) = \sum_i w_{x,h}(x_i) y_i$$

AKA Nadarya-Watson



Bandwidths. Weights for Gaussian kernel regression problem at $x = 0.3$ with different bandwidths.

Credit: Nathaniel Helwig

Local Linear Fit Procedure

- Tuning Parameter h bandwidth
- For a point x, $B_h(x)$ is the interval $I_h(x) = [x - h/2, x + h/2]$.
- Local Mean

$$\mu_h^Y(x) \equiv Ave\{Y_i | X_i \in B_h(x)\} \qquad [= \hat{s}_n^{lav}(x)]$$

$$\mu_h^X(x) \equiv Ave\{X_i | X_i \in B_h(x)\}$$

- Local Slope

$$\beta_h(x) \equiv \frac{Ave\{(Y_i - \mu_h^Y(x))(X_i - \mu_h^X(x)) | X_i \in B_h(x)\}}{Ave\{(X_i - \mu_h^X(x))^2 | X_i \in B_h(x)\}}.$$

- Local Linear Estimator

$$\hat{s}_h^{llf}(x) = \mu_h^Y(x) + \beta_h(x)(x - \mu_h^X(x)).$$

- Note:

$$\hat{s}_h^{llf}(x) = \hat{s}_h^{lav}(x) + \text{correction}$$

Correction vanishes under:

(a) Local symmetry of x_i-data: $x = \mu_h^X(x)$; or
(b) Local Flatness $\beta_h(x) = 0$.

Correction most pronounced at edges of dataset.

Local Polynomial Fit Procedure

- Tuning Parameter h bandwidth; d degree
- For a point x, $B_h(x)$ is the interval $I_h(x) = [x - h/2, x + h/2]$.
- Polynomial

$$P(t; d, \beta) = \sum_{k=0}^{d} \beta_k t^k.$$

- Local Sum of Squares

$$SS_{x,h}((v_i)) = \sum \{v_i^2 | x_i \in B_h(x)\}.$$

- Local Polynomial:

$$\hat{P}_{x;h,d} \equiv \text{argmin}_{P(\cdot;d,\beta)} SS_{x,h}(Y - P(\cdot; d, \beta))$$

- Local Polynomial Fit:

$$\hat{s}_h^{lpf,d}(x) = \hat{P}_{x;h,d}(x).$$

- Special cases

$$\begin{array}{ll} d = 0 & \hat{s}_n^{lpf,0}(x) = \hat{s}_h^{lav}(x) \\ d = 1 & \hat{s}_n^{lpf,1}(x) = \hat{s}_h^{llf}(x) \end{array}$$

Weighted Local Polynomial Fit Procedure

- Tuning Parameter h bandwidth; d degree; $W()$ weight function
- For a point x, $B_h(x)$ is the interval $I_h(x) = [x - h/2, x + h/2]$.
- Polynomial

$$P(t; d, \beta) = \sum_{k=0}^{d} \beta_k t^k.$$

- Local Weighted Sum of Squares

$$WSS_{x,h}((v_i)) = \sum \{v_i^2 W(\frac{x_i - x}{h}) | x_i \in B_h(x)\}.$$

- Weighted Local Polynomial Fit :

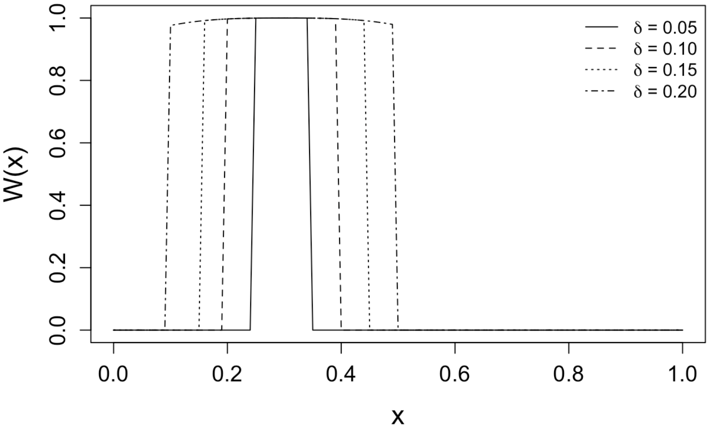
$$\hat{P}_{x;h,d} \equiv \text{argmin}_{P(\cdot;d,\beta)} WSS_{x,h}(Y - P(\cdot; d, \beta))$$

- Weighted Local Polynomial Fit:

$$\hat{s}_h^{wlpf,d}(x) = \hat{P}_{x;h,d}(x).$$

- Special cases:

$$K = W \qquad \hat{s}_n^{wlpf,0}(x) = \hat{s}_h^{ksm}(x)$$



Tricube Function. Weights for local regression problem at $x = 0.3$ with different delta values.

Credit: Nathaniel Helwig.
Note: Please mentally substitute $h \leftrightarrow \delta$.

Penalized Spline Procedure

- Tuning Parameter λ penalty
- Penalized Sum of Squares

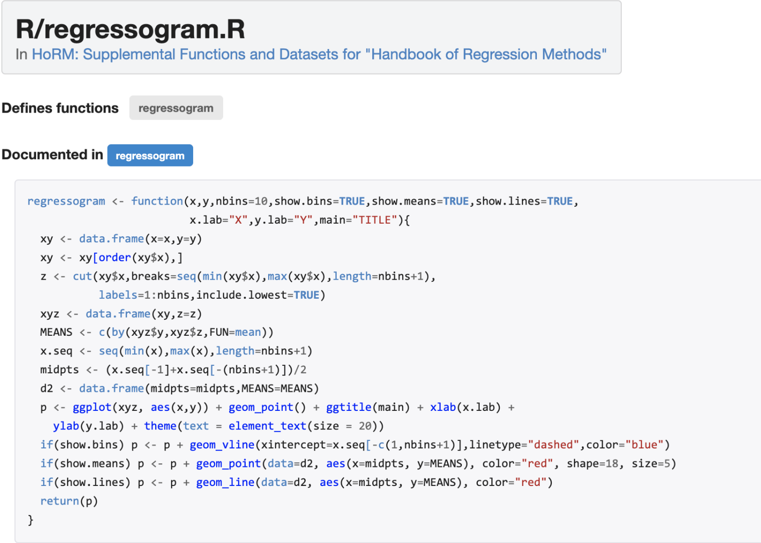
$$PSS_{\lambda}(s; Y) = \sum (y_i - s(x_i))^2 + \lambda \cdot \int_a^b s^2(dt)dt$$

  $a = \min(x_i)$, $b = \max(x_i)$.
- Spline Fit

$$s_{\lambda}^{ss} \equiv \text{argmin}_{s()} PSS_{\lambda}(s; Y)$$

Suggestions in your Journey

- Standard R documentation
  `https://stat.ethz.ch/R-manual/R-devel/library/stats/html/smooth.spline.html`
- Examine Code
  `https://rdrr.io/cran/HoRM/src/R/regressogram.R`
- Online Resources
  `http://users.stat.umn.edu/h̃elwig/notes/smooth-notes.html`



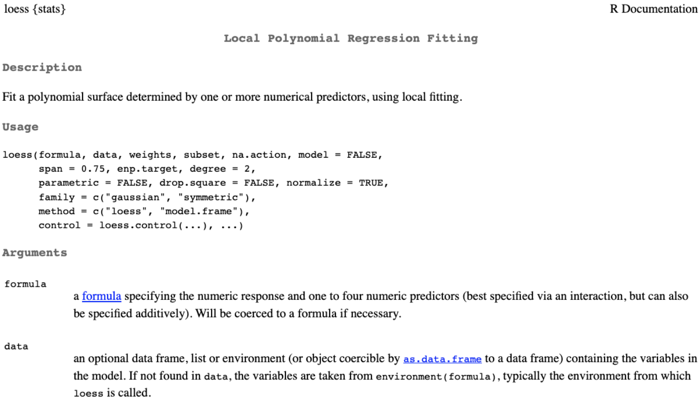
R/regressogram.R

In HoRM: Supplemental Functions and Datasets for "Handbook of Regression Methods"

Defines functions regressogram

Documented in regressogram

```r
regressogram <- function(x,y,nbins=10,show.bins=TRUE,show.means=TRUE,show.lines=TRUE,
                         x.lab="X",y.lab="Y",main="TITLE"){
  xy <- data.frame(x=x,y=y)
  xy <- xy[order(xy$x),]
  z <- cut(xy$x,breaks=seq(min(xy$x),max(xy$x),length=nbins+1),
           labels=1:nbins,include.lowest=TRUE)
  xyz <- data.frame(xy,z=z)
  MEANS <- c(by(xyz$y,xyz$z,FUN=mean))
  x.seq <- seq(min(x),max(x),length=nbins+1)
  midpts <- (x.seq[-1]+x.seq[-(nbins+1)])/2
  d2 <- data.frame(midpts=midpts,MEANS=MEANS)
  p <- ggplot(xyz, aes(x,y)) + geom_point() + ggtitle(main) + xlab(x.lab) +
    ylab(y.lab) + theme(text = element_text(size = 20))
  if(show.bins) p <- p + geom_vline(xintercept=x.seq[-c(1,nbins+1)],linetype="dashed",color="blue")
  if(show.means) p <- p + geom_point(data=d2, aes(x=midpts, y=MEANS), color="red", shape=18, size=5)
  if(show.lines) p <- p + geom_line(data=d2, aes(x=midpts, y=MEANS), color="red")
  return(p)
}
```



loess {stats} R Documentation

Local Polynomial Regression Fitting

Description

Fit a polynomial surface determined by one or more numerical predictors, using local fitting.

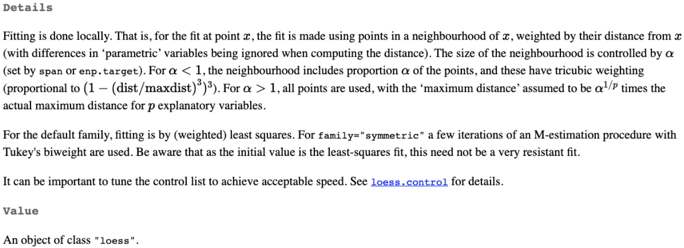
Usage

```
loess(formula, data, weights, subset, na.action, model = FALSE,
      span = 0.75, enp.target, degree = 2,
      parametric = FALSE, drop.square = FALSE, normalize = TRUE,
      family = c("gaussian", "symmetric"),
      method = c("loess", "model.frame"),
      control = loess.control(...), ...)
```

Arguments

`formula`
: a formula specifying the numeric response and one to four numeric predictors (best specified via an interaction, but can also be specified additively). Will be coerced to a formula if necessary.

`data`
: an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `loess` is called.



Details

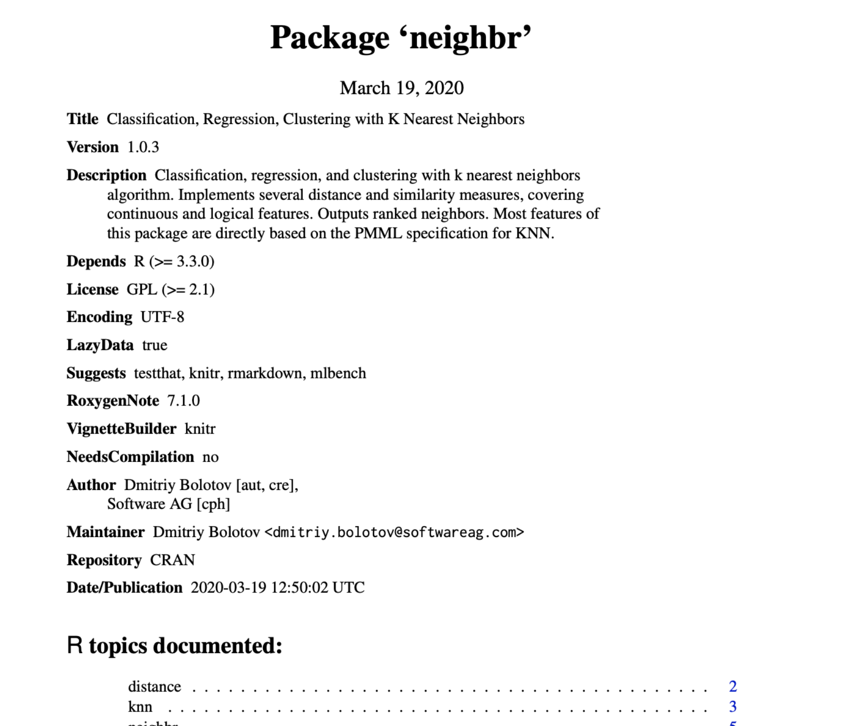
Fitting is done locally. That is, for the fit at point x, the fit is made using points in a neighbourhood of x, weighted by their distance from x (with differences in 'parametric' variables being ignored when computing the distance). The size of the neighbourhood is controlled by α (set by `span` or `enp.target`). For $\alpha < 1$, the neighbourhood includes proportion α of the points, and these have tricubic weighting (proportional to $(1 - (\text{dist}/\text{maxdist})^3)^3$). For $\alpha > 1$, all points are used, with the 'maximum distance' assumed to be $\alpha^{1/p}$ times the actual maximum distance for p explanatory variables.

For the default family, fitting is by (weighted) least squares. For `family="symmetric"` a few iterations of an M-estimation procedure with Tukey's biweight are used. Be aware that as the initial value is the least-squares fit, this need not be a very resistant fit.

It can be important to tune the control list to achieve acceptable speed. See `loess.control` for details.

Value

An object of class `"loess"`.



Package ‘neighbr’

March 19, 2020

Title Classification, Regression, Clustering with K Nearest Neighbors

Version 1.0.3

Description Classification, regression, and clustering with k nearest neighbors algorithm. Implements several distance and similarity measures, covering continuous and logical features. Outputs ranked neighbors. Most features of this package are directly based on the PMML specification for KNN.

Depends R (>= 3.3.0)

License GPL (>= 2.1)

Encoding UTF-8

LazyData true

Suggests testthat, knitr, rmarkdown, mlbench

RoxygenNote 7.1.0

VignetteBuilder knitr

NeedsCompilation no

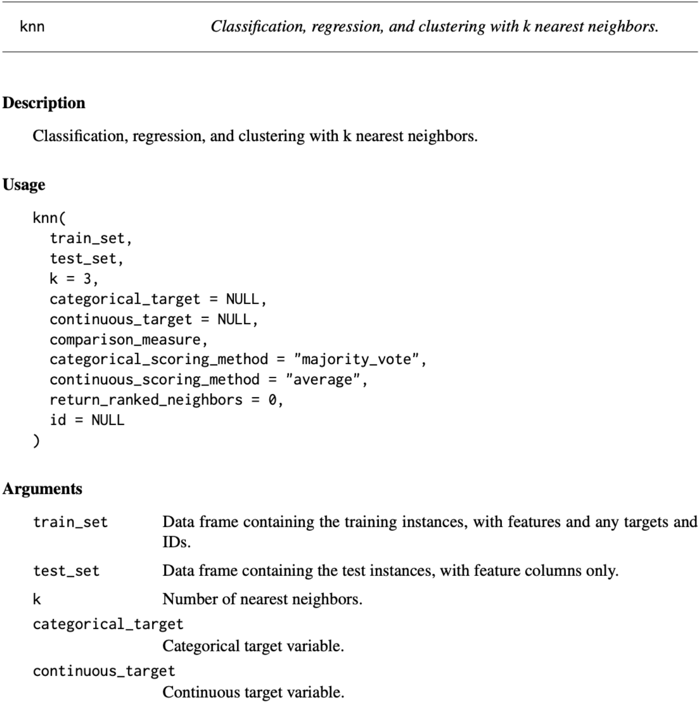
Author Dmitriy Bolotov [aut, cre],
Software AG [cph]

Maintainer Dmitriy Bolotov <dmitriy.bolotov@softwareag.com>

Repository CRAN

Date/Publication 2020-03-19 12:50:02 UTC

R topics documented:



`knn` *Classification, regression, and clustering with k nearest neighbors.*

Description

Classification, regression, and clustering with k nearest neighbors.

Usage

```
knn(
  train_set,
  test_set,
  k = 3,
  categorical_target = NULL,
  continuous_target = NULL,
  comparison_measure,
  categorical_scoring_method = "majority_vote",
  continuous_scoring_method = "average",
  return_ranked_neighbors = 0,
  id = NULL
)
```

Arguments

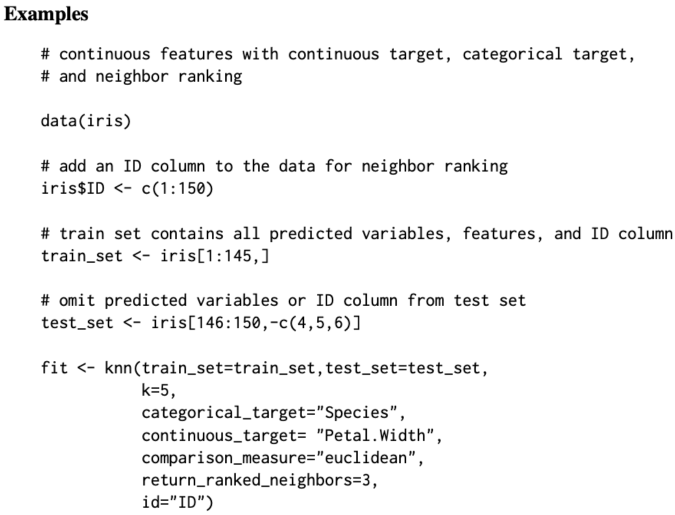
`train_set` Data frame containing the training instances, with features and any targets and IDs.

`test_set` Data frame containing the test instances, with feature columns only.

`k` Number of nearest neighbors.

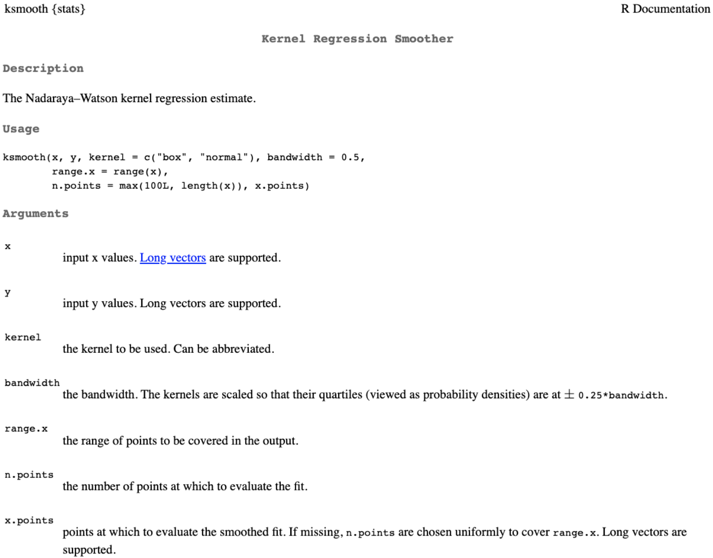
`categorical_target` Categorical target variable.

`continuous_target` Continuous target variable.



Examples

```
# continuous features with continuous target, categorical target,
# and neighbor ranking

data(iris)

# add an ID column to the data for neighbor ranking
iris$ID <- c(1:150)

# train set contains all predicted variables, features, and ID column
train_set <- iris[1:145,]

# omit predicted variables or ID column from test set
test_set <- iris[146:150,-c(4,5,6)]

fit <- knn(train_set=train_set,test_set=test_set,
           k=5,
           categorical_target=”Species”,
           continuous_target= ”Petal.Width”,
           comparison_measure=”euclidean”,
           return_ranked_neighbors=3,
           id=”ID”)
```



ksmooth {stats} R Documentation

Kernel Regression Smoother

Description

The Nadaraya–Watson kernel regression estimate.

Usage

```
ksmooth(x, y, kernel = c("box", "normal"), bandwidth = 0.5,
        range.x = range(x),
        n.points = max(100L, length(x)), x.points)
```

Arguments

`x`
input x values. Long vectors are supported.

`y`
input y values. Long vectors are supported.

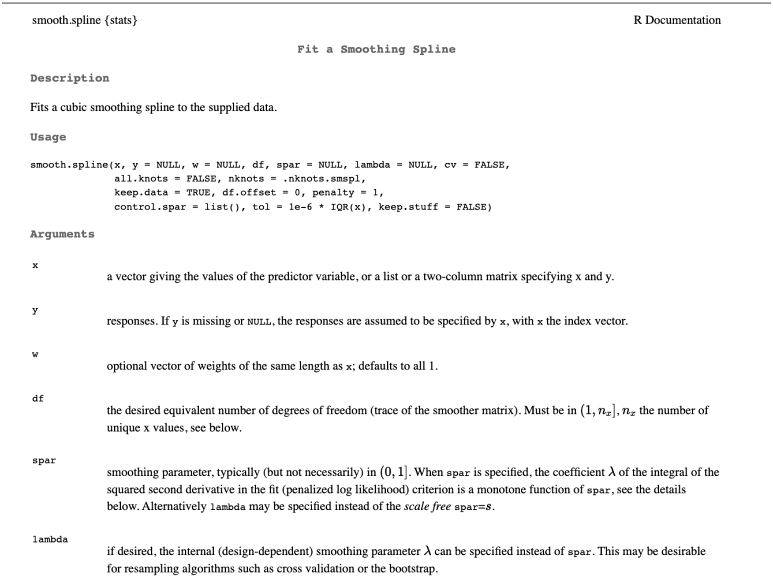
`kernel`
the kernel to be used. Can be abbreviated.

`bandwidth`
the bandwidth. The kernels are scaled so that their quartiles (viewed as probability densities) are at ± `0.25*bandwidth`.

`range.x`
the range of points to be covered in the output.

`n.points`
the number of points at which to evaluate the fit.

`x.points`
points at which to evaluate the smoothed fit. If missing, `n.points` are chosen uniformly to cover `range.x`. Long vectors are supported.



smooth.spline {stats} R Documentation

Fit a Smoothing Spline

Description

Fits a cubic smoothing spline to the supplied data.

Usage

```
smooth.spline(x, y = NULL, w = NULL, df, spar = NULL, lambda = NULL, cv = FALSE,
              all.knots = FALSE, nknots = .nknots.smspl,
              keep.data = TRUE, df.offset = 0, penalty = 1,
              control.spar = list(), tol = 1e-6 * IQR(x), keep.stuff = FALSE)
```

Arguments

`x`
: a vector giving the values of the predictor variable, or a list or a two-column matrix specifying x and y.

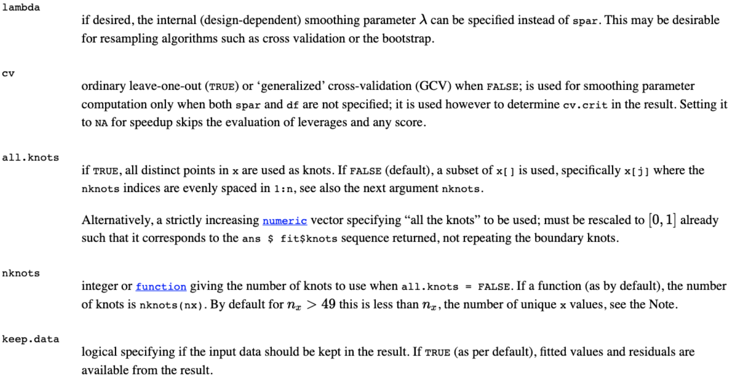
`y`
: responses. If `y` is missing or `NULL`, the responses are assumed to be specified by `x`, with `x` the index vector.

`w`
: optional vector of weights of the same length as `x`; defaults to all 1.

`df`
: the desired equivalent number of degrees of freedom (trace of the smoother matrix). Must be in $(1, n_x]$, n_x the number of unique x values, see below.

`spar`
: smoothing parameter, typically (but not necessarily) in $(0, 1]$. When `spar` is specified, the coefficient λ of the integral of the squared second derivative in the fit (penalized log likelihood) criterion is a monotone function of `spar`, see the details below. Alternatively `lambda` may be specified instead of the *scale free* `spar=`s.

`lambda`
: if desired, the internal (design-dependent) smoothing parameter λ can be specified instead of `spar`. This may be desirable for resampling algorithms such as cross validation or the bootstrap.



`lambda`

if desired, the internal (design-dependent) smoothing parameter λ can be specified instead of `spar`. This may be desirable for resampling algorithms such as cross validation or the bootstrap.

`cv`

ordinary leave-one-out (`TRUE`) or ‘generalized’ cross-validation (GCV) when `FALSE`; is used for smoothing parameter computation only when both `spar` and `df` are not specified; it is used however to determine `cv.crit` in the result. Setting it to `NA` for speedup skips the evaluation of leverages and any score.

`all.knots`

if `TRUE`, all distinct points in `x` are used as knots. If `FALSE` (default), a subset of `x[]` is used, specifically `x[j]` where the `nknots` indices are evenly spaced in `1:n`, see also the next argument `nknots`.

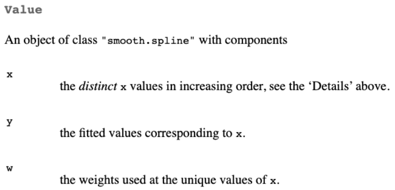
Alternatively, a strictly increasing `numeric` vector specifying “all the knots” to be used; must be rescaled to $[0, 1]$ already such that it corresponds to the `ans $ fit$knots` sequence returned, not repeating the boundary knots.

`nknots`

integer or `function` giving the number of knots to use when `all.knots = FALSE`. If a function (as by default), the number of knots is `nknots(nx)`. By default for $n_x > 49$ this is less than n_x, the number of unique `x` values, see the Note.

`keep.data`

logical specifying if the input data should be kept in the result. If `TRUE` (as per default), fitted values and residuals are available from the result.



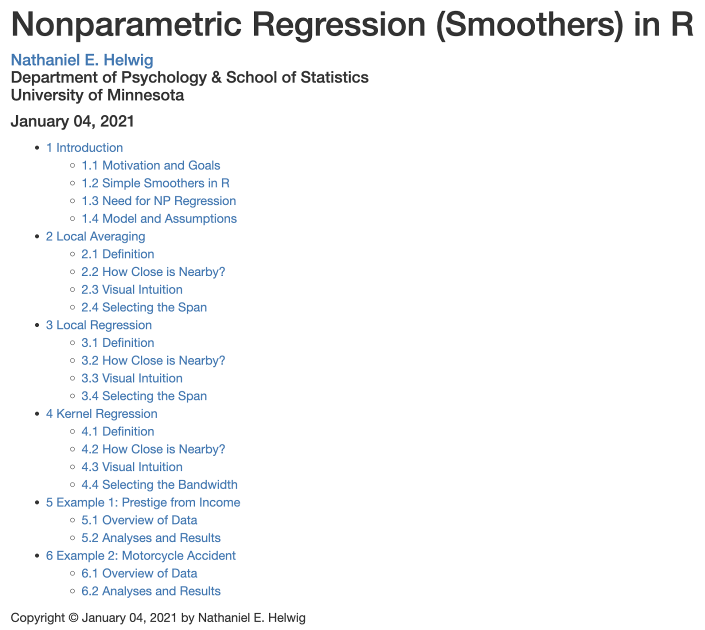
Value

An object of class `"smooth.spline"` with components

`x` the *distinct* `x` values in increasing order, see the 'Details' above.

`y` the fitted values corresponding to `x`.

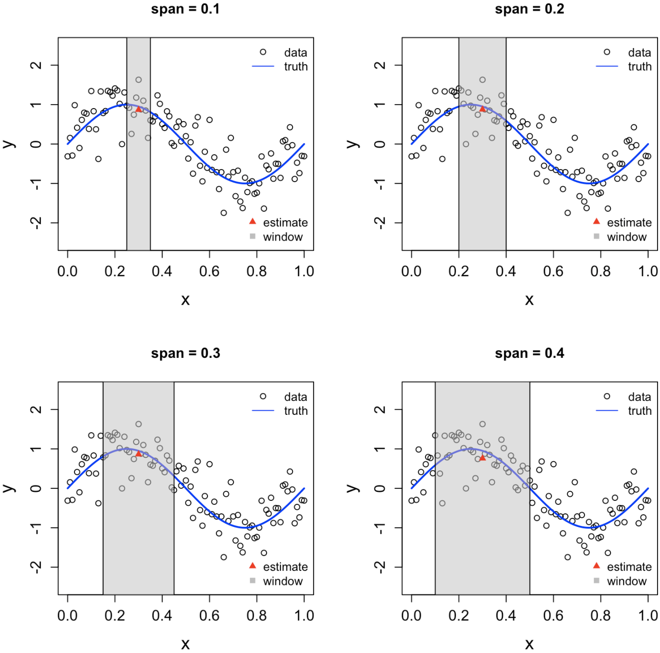
`w` the weights used at the unique values of `x`.



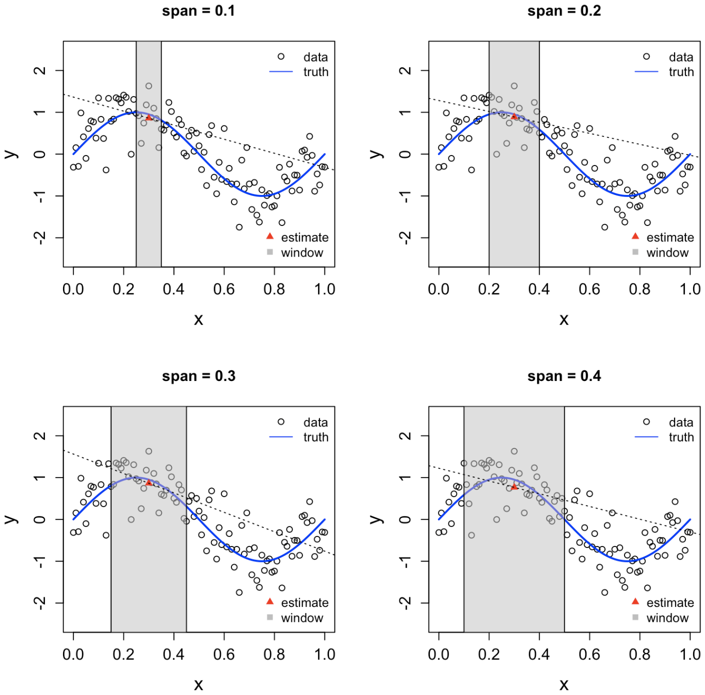
Nonparametric Regression (Smoothers) in R

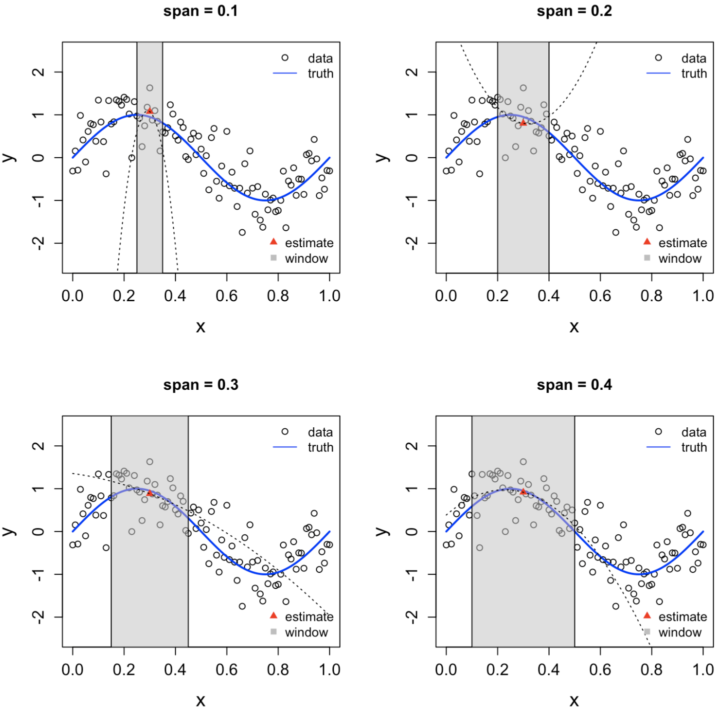
Nathaniel E. Helwig
Department of Psychology & School of Statistics
University of Minnesota

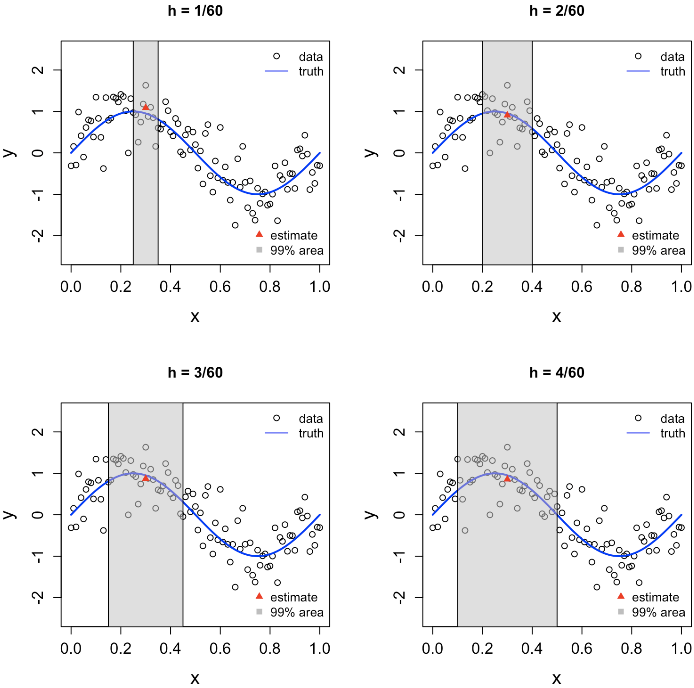
January 04, 2021



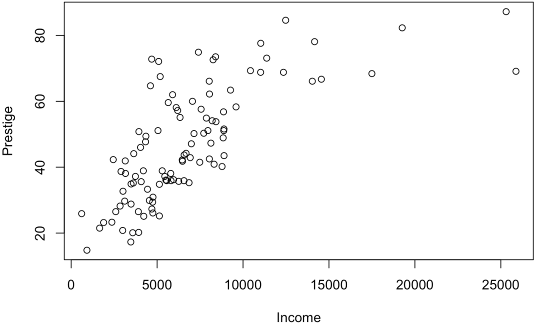
Local averaging estimate of $f(0.3)$ with different span values.


span = 0.1
span = 0.2
span = 0.3
span = 0.4
y
x
2
1
0
-1
-2
0.0
0.2
0.4
0.6
0.8
1.0
○ data
truth
▲ estimate
■ window

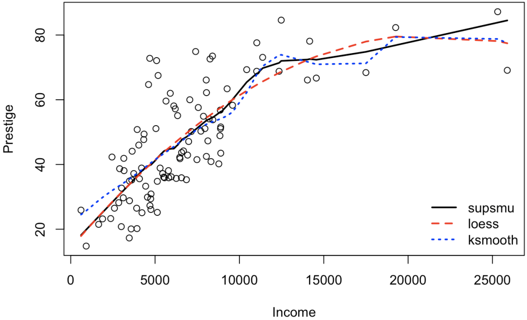

span = 0.1
span = 0.2
span = 0.3
span = 0.4
○ data
truth
▲ estimate
■ window
y
x
-2
-1
0
1
2
0.0
0.2
0.4
0.6
0.8
1.0



Kernel regression estimate of $f(0.3)$ with different bandwidth values



Note: the relationship looks non-linear. For occupations that earn less than $10K, there is a strong (positive) linear relationship between income and prestige. However, for occupations that earn between $10K and $25K, the relationship has a substantially different (attenuated) slope.



From the above plot, it looks like the GCV-tuned LOESS estimate is performing best (i.e., providing the best combination of fit and smoothness), the CV-tuned local average is performing second best, and the GCV-tuned kernel regression estimate is performing worst. In particular, the GCV-tuned kernel regression estimate produces a rather rough/wiggly estimate of the relationship between income and prestige.

The Central Tradeoff: Flexibility vs. Parsimony

Setting	Tuning Parameter Name
Polynomial Fitting	Degree
Local Averaging	# Bins
Kernel Smoothing	Bandwidth
Spline	# Knots
Smoothing Spline	Penalty Coefficient
Orthogonal Series	# Terms
Wavelet Smoothing	Threshold
Neural Net	Width
	Depth
	Connectivity

Any such we will call *smoothing parameter*.

Importance of tuning parameters

- Smoothing parameter [eg Bandwidth, # Bins]
  most important
- Polynomial degree [Linear vs constant vs quadratic]
  next most important
- Kernel $K(\cdot)$, Weights $W(\cdot)$
  less important

So, how do we choose bandwidth?

Choice of Tuning Param.

- Gold standard:
  - Mean-Squared-Error of Estimation

$$MSEE_h = \text{Ave}\{(\mu(x_i) - \hat{\mu}_h(x_i)\}^2$$

  - Possible choice of h:

$$\hat{h}(X, Y) = \text{argmin}_h MSEE_h.$$

  - Unobservable: we don't know $\mu(x_i)$.
- Test Error Proxy: Suppose
  - Train data $\mathcal{D}_{train} = \{(x_i, y_i) : i = 1, \dots\}$ sampled according to:

$$y_i = \mu(x_i) + z_i, \qquad z_i \sim_{iid} N(0, \sigma^2),$$

  - Train estimate $\hat{\mu}_h(\cdot; \mathcal{D}_{train})$ trained on $\mathcal{D}_{train}$.
  - Test data $\mathcal{D}_{test} = \{(x_j', y_j'), j = 1, \dots, m\}$ independent replications of iid samples from same generative mechanism as $\{(x_i, y_i) : i = 1, \dots, n\}$:
  -

$$PMSE_h^{test} = \text{ave}_{(x_j', y_j') \in \mathcal{D}_{test}} \{(y_j' - \hat{\mu}_h(x_j'; \mathcal{D}_{train})\}^2$$

$$E[MSEE_h] \quad = \quad E[PMSE_h^{test}] - \sigma^2$$

looks promising...

Failure of Training MSE

Would-be proxy:

- $PMSE^{train}$, where
  - $\hat{\mu}_h(\cdot; \mathcal{D}_{train})$ trained on *same* data as used for evaluation $\{(x_i, y_i) : i = 1, \dots\}$;

    $$PMSE_h^{train} = \text{ave}_{(x_i,y_i)\in\mathcal{D}_{train}}\{(y_i - \hat{\mu}_h(x_i; \mathcal{D}_{train}))\}^2.$$

  - Compare

    $$PMSE_h^{test} = \text{ave}_{(x_j',y_j')\in\mathcal{D}_{test}}\{(y_j' - \hat{\mu}_h(x_j'; \mathcal{D}_{train}))\}^2$$

- Train MSE suffers from **overfitting**; we will see [next slide] that,

  $$PMSE_h^{train} \to 0, \text{ as } h \to 0.$$

  In contrast Test MSE does not overfit, for the same smoothers as above:

  $$E[PMSE_h^{test}] \to \sigma^2, \text{ as } h \to 0.$$

- In other words, for small h

  $$0 \approx PMSE_h^{train} \ll PMSE_h^{test} \approx \sigma^2.$$

  this is called overfitting, overoptimisim, etc.

Overfitting as $h \to 0, 1$

Let's establish for many smoothers the claim of the previous slide:

$$PMSE_h^{train} \to 0, \text{ as } h \to 0.$$

- Recall the neighborhood definition:

$$B_h(x) = \{x_j : ||x_j - x|| \le h\}.$$

- Consider a smoother defined by local averaging over such neighborhood

$$s_h^{lav}(x) = \text{ave}\{y_j : x_j \in B_h(x)\}$$

- Suppose there are no replicate x's

$$x_i \neq x_j \qquad \forall i \neq j.$$

  more specifically $h_0 = \min_{i \neq j} |x_i - x_j| > 0$.

- As $h \to 0$, eventually x_i is the only observation in the local neighborhood $B_h(x_i)$; i.e.

$$B_h(x_i) = \{x_i\}, \qquad 0 \le h < h_0.$$

- Consequently:

$$s_h^{lav}(x_i) = y_i, \qquad 0 \le h < h_0.$$

  and so:

$$PMSE_h^{train} = n^{-1}\text{ave}|y_i - s_h^{lav}(x_i)|^2 = 0, \qquad 0 \le h < h_0.$$

Overfitting as $h \to 0$, 2

More generally

- Consider any fitting procedure *Fit* that has the property that, when applied to a singleton set, it returns that value:

$$Fit(\{(x_j, y_j)\}) = y_j$$

- Examples include
  - Average
  - Median
  - Fit a least-squares line to the (x, y) data in a neighborhood; except, where there are too few points, fit a constant - by either least squares or least absolute value.
- Define

$$\hat{\mu}_h(x_i; \mathcal{D}_{train}) = Fit(\{(x_j, y_j) : x_j \in B_h(x_i))$$

Then as $h \to 0$,

$$PMSE_h^{train} = \text{ave}_i(y_i - \hat{\mu}_h(x_i; \mathcal{D}_{train}))^2 \to 0.$$

Overfitting as $h \to 0$, 3

- This is called overfitting, because the model is matching the input data perfectly;
- However, we know that one shouldn't do this, as the data are noisy. Our model says:

$$y_i = \mu_i + z_i$$

  where μ_i is the smooth signal we want and z_i is the noise.
- Typically the noise is oscillatory; it's often positive then negative right away.
- An overfit model fits every wiggle in the data; we don't believe the true signal wiggles in that way, the overfit model is 'too wiggly'.

Cross-Validation

- Leave One Out data: $\mathcal{D}_{(j)} = \{(x_i, y_i) : 1 \le i \le n, i \ne j\}$.
- Leave One Out estimate

$$\hat{\mu}_h^{(j)} = \hat{\mu}_h(\mathcal{D}_{(j)})$$

- LOO Cross-validated MSE

$$CVMSE_h = \text{ave}_j(y_j - \hat{\mu}_h^{(j)})^2$$

- Note that, in general, overfitting does not occur:

$$CVMSE_h \not\to 0, \text{ as } h \to 0.$$

- **Theorem.** *Suppose $y_i = \mu_i + z_i$ and that the noise terms z_i are iid with $Ez_i^2 = \sigma^2$, then:*

$$E[CVMSE_h] \ge \sigma^2, \qquad h \ge 0.$$

In short, CVMSE does not overfit, in the sense described on the previous slide.

Success of CVMSE

Wasserman Section 5.3, notation changed:

- Cross-Validated Bandwidth Choice:

$$\hat{h}^{CVC} = \text{argmin}_{h \geq 0} CVMSE_h.$$

- Consider a sequence of datasets $\mathcal{D}_n = \{(x_{i,n}, y_{i,n}) : 1 \leq i \leq n\}$

$$x_{i,n} = i/n; \quad y_i = \mu(x_{i,n}) + z_{i,n}, \qquad i = 1, \ldots n.$$

  where $z_{i,n} \sim_{iid} N(0, \sigma^2)$
- Suppose that $\mu(x)$ has two continuous derivatives on $[0, 1]$.
- Recall the MSEE

$$MSEE_{h,n} = \text{ave}_i(\mu(x_{i,n}) - \hat{\mu}(x_{i,n}))^2.$$

  Note that $MSEE_{h,n}$ is a different quantity than $CVMSE_{h,n}$.
- Define the optimal bandwidth:

$$h_n^* = \text{argmin}_{h \geq 0} MSEE_h.$$

- The optimal MSE

$$MSEE_n^* = MSEE_{h_n^*,n}.$$

- The MSE of the cross-validated choice:

$$MSEE_n^{CVC} = MSEE_{\hat{h}^{CV},n}.$$

- **Theorem.** *Asymptotic Optimality. For $\hat{\mu}$ defined by local averaging, and each fixed $\varepsilon > 0$, eventually almost surely:*

$$MSEE_n^{CVC} < MSEE_n^* \cdot (1 + \varepsilon)$$

- LOO Cross-Validated Bandwidth Choice is effective and in fact optimal (previous slide).
- The previous result is stated under quite specific conditions, but is true much more generally, for example for many other smoothers.
- However, on a problem of size n it generally requires n different runs of the algorithm, one for each point being left out!

$$\hat{\mu}_h^{(j)} = \hat{\mu}_h(\mathcal{D}_{(j)})$$

- When n is large, this becomes expensive, hence LOOCV is said to *scale poorly with* n.
- Fortunately for the important class of *linear smoothers*, the situation is much better: we can compute $CVMSE_h$ using only one run of the algorithm, and some math... as we show next.

Linear Smoothers, 1

- Data $Y = (Y_i)_{i=1}^{n}$
- Smoother

$$s = S(Y)$$

  here
  - $Y = (Y_i)$ vector of inputs
  - $s = (s_i)$ vector of outputs
  - $s_i = S(Y)_i$ is the i-th output.
- **Linear** Smoother:

$$S(Y)_i = \sum_{i=1}^{n} L_{i,j} Y_j$$

  Each output is a linear combination of inputs, with coefficients $L_{i,j}$
- Smoother Matrix:

$$L = (L_{i,j} : 1 \leq i \leq n, 1 \leq j \leq n).$$

- Linear Smoothers are *much* easier to analyze *mathematically*

Linear Smoothers, 2

Many Useful Smoothers are Linear Smoothers.

Setting	Linear?
Polynomial Fitting	Yes
Regressogram	Yes
Local Averaging	Yes
Kernel Smoothing	Yes
Spline	Yes
Smoothing Spline	Yes
Orthogonal Series	Yes
Automatic Smoothing	no
Wavelet Smoothing	no
Neural Net	no

More trendy recent smoothers are **not** linear
More ‘leave it to me’ smoothers are **not** linear

Linear Smoothers, 3

Properties of Smoothers can be derived from properties of L:

- Effective Degrees of Freedom

$$\nu = tr(L) \equiv \sum_i L_{i,i}.$$

- Example: Local Averages: $L_{i,i} = 1/\#B_h(x_i)$

$$\nu = n \cdot \text{Ave} \frac{1}{\#\{B_h(x_i)\}}.$$

- Smoother's *Noise Response*

$$\hat{\mu}(z) = Lz.$$

  Average Variance

$$\bar{\sigma}^2 = E\, Ave_i \hat{\mu}_i^2 = n^{-1} tr(L'L)\sigma^2;$$

- Smoother's *Bias Response*:

$$Bias = \mu - S(\mu) = (I - L)(\mu)$$

  Average Squared Bias

$$Bias^2 = n^{-1}\|(I - L)\mu\|^2$$

Linear Smoothers, 5

Miracle of LOO CV, for local averaging:

- For local averaging $\hat{\mu} \equiv s^{lav}$:

$$\begin{aligned} CVMSE &= \text{ave}_i(y_i - \hat{\mu}(x_i; \mathcal{D}^{(-i)}))^2. \\ &= \text{ave}_i(\frac{y_i - \hat{\mu}(x_i)}{1 - L_{ii}})^2. \end{aligned}$$

  - Exactly true, not an approximation!
  - Involves one run of fitting, not n runs!
- With local averaging: $L_{i,i} = 1/\#B_h(x_i)$
- Equispaced case

$$\begin{aligned} \#B_h(x_i) &= \#\{j : |x_{i,n} - x_{j,n}| \leq h\} \\ &= \#\{j : |i - j| \leq h \cdot n\} \\ &= 2\lfloor h \cdot n \rfloor + 1. \end{aligned}$$

- Define

$$\Xi(h, n) = (\frac{1}{1 - \frac{1}{2\lfloor h \cdot n \rfloor + 1}})^2$$

$$CVMSE = \Xi(h, n) \cdot PMSE_h^{train}$$

- In other words, you just need the train PMSE curve, and then multiply it or 'correct' by an explicit function of h,
- No need for n different LOO runs, only one run!

Linear Smoothers, 6

- Inspired by local averaging story, we often use *Generalized CV*:
$$GCV_h = (1 - \nu_h/n)^{-2} \cdot PMSE_h^{train}$$
involving effective degrees of freedom:
$$\nu_h \equiv tr(L^h) \equiv \sum_i L_{i,i}^h.$$
- Again, you just need to know the train PMSE curve, and multiply it by an explicit function of h, involving ν_h, which you can calculate at the same time as you calculate $\hat{\mu}_h$.
- GCV Bandwidth Choice:
$$\hat{h}^{GCVC} = \text{argmin}_{h \geq 0} GCV_h.$$
- This principle is optimal quite broadly as well.