Nonparametric Smoothing

BTRY/STSCI 4520

- Smoothing with a Nadaraya-Watson est
- Choosing Bandwidths
- Non-parametric density estimation
- Local linear regression
- Basis expansion methods

Curve Fitting

In the most general set-up

- data are (t_i, y_i)
- [illegible] i

- don't want to make assumptions about g except that it is "smooth".

Try and produce methods that fit locally.

Local Fitting

Idea: use only points near the t of interest.

Binning:

-
-

Prod

Use the average of a moving window centered at

$$\hat{g}(t) = \text{average of } y_i \text{ wi}$$

Or weight by distance from t.

Nadaraya-Watson Estimator

$$g(t,h) = \frac{\sum_{i=1}^{n} y_i K\left(\frac{t_i - t}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{t_i - t}{h}\right)}$$

-
- Bandwidth h spreads weight closer or fu

```
NW = function(t,TT,Y,bw){
  return( weighted.mean(Y, dnorm(TT
}
```

Larger h implies that points further away have influence on curve.

Bias-Variance Trade-off

Need to think about choosing h; what do we want to achieve?

Possibly we would like to minimize expected squared error:

$$E\left(\hat{g}(t,h) - g(t)\right)^2$$

By ad

$$E\left[\hat{g}\right. \quad \left.(t,h)\right]^2$$

$$= \text{bias}^2 + \text{variance}$$

Intuitively: larger h leads to larger bias (use po but smaller variance (use more points). We'll do this formally in density estimation.

This is generally referred to as a trade-off; want to balance these two.

Example: Vancouver Precipitation

- Small h: relies on points close to t: low bias, but high variance.
- Large h: spreads weight across many observations: smaller variance, higher bias.
- Finding an ideal compromise is difficult.

Choosing A Bandwidth

Rather than minimizing at just one value of t, we'll average across the interval:

This is

interval.

We can also think of this as the error (minus residual va

predicting a new data point $(t_{new}, y_{new}$

We will approximate this via cross-validation.

Cross Validation

Cross-validation = estimate on some data, evaluate on others.

For each data point (t_i, y_i)

- Remove (t_i, y_i) from the data.

The cross-validated score is

$$\mathrm{OCV}(h) = \sum_{i=1}^{n} (y -$$

Treats each i as a “new” observation. Choose h to minimize $\mathrm{OCV}(h)$.

Cross Validation

Leave each point out in turn and try to predict it

```
CV = function(bw,TT,Y)
{
  err = rep(0,length(Y))  # Prediction Errors
  fo

  }
  return( sum(err^2) )
}
```

We'll search over values, often best by powers of 10

```
hs = 1:20
for(j in 1:length(hs)){
  CVs[j] = CV(hs[j],TT,Y)
}
```

Calculating Cross Validation Efficiently

- Leaving one point out at a time is computationally expensive.
- But, $\hat{g}(t)$ is linear in the y_i; we can write out in vectors

$$\hat{\mathbf{y}} = S(h)\mathbf{y}$$

[illegible] lues

$$[S(h)]_{ij} = \frac{[illegible]}{\sum_{k=[illegible]}^{n} [illegible]}$$

- Some calculation (not shown) gives us that

$$\mathrm{OCV}(h) = \sum_{i=1}^{n} \frac{(y_i - \hat{g}(t_i, h))^2}{(1 - [S(h)]_{ii})^2}$$

This identity means that we can calculate OCV(h) as efficiently as $\hat{\mathbf{y}}$.

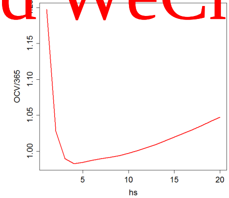
In Code

```
hs = 1:20      # Bandwidths
OCV = 0*hs     # OCV scores

# Matrix of time-points:
X = matrix(TT,365,365)

for(i i
  Kva
  Sh = di

  hat                                          # Predictions
  OCV[i] = sum( (Y-hatY)^2/(1-diag(Sh))^2 )    # OCV formula
}
```



Generalized Cross Validation

OCV tends to under-smooth and has some mathematically undesirable properties.

Instead, *Generalized Cross Validation* is often used

$$\frac{{}^{n}{}_{1}(y_i \quad \hat{g}(t_i, h))^2}{}$$

`GCV [`

within loop above.

Undersmoothing

GCV and OCV generally produce curves that are rougher than is visually appealing.

More associated with human visual perception than statistics

One suggested change is a little extra penalty in GCV:

`GCV2`

Extensions 1: Kernel Density Estimation (Rizzo Ch 10)

If $X_1, \ldots, X_n \sim f(\cdot)$ but we don't know f.

Estimate for f is

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{X_i - x}{h}\right)$$

place

Using 20 points at random for ease of illustration.

Alternative Kernels

Examples here use the normal density; but any symmetric bump that integrates to 1 will do.

Gaussian Density

$\max(1 - x^2, 0)$

Epanechnikov kernel often favored for theoretical reasons.

Bandwidth h

Bandwidth controls the “wigglyness” of the estimate.

$h = 0.1$ $h = 0.6$

Intuitively: smaller h makes estimate more sensitive to data $\Rightarrow$ higher variance.

It works better with all the data

Asymptotics Sketch (details unimportant)

We will look at the *expected* error between $\hat{f}(x)$ and $f(x)$.

As in smoothing, we divide expected error into

$$E(\hat{f}(x) - f(x))^2 = E(\hat{f}(x) - E\hat{f}(x))^2 + (E\hat{f}(x) - f(x))^2$$

whic

It is nat

We will use change of variables:

$$E_{X_i} \frac{1}{h} K\left(\frac{X_i - x}{h}\right) = \int \frac{1}{h} K\left(\frac{y - x}{h}\right)$$

$$= \int \frac{1}{h} hK(u) f(x + hu) du = \int K(u) f(x + hu) du$$

by setting $u = (y - x)/h$ or $y = x + hu$.

Bias

We observe that because each $K((X_i - x)/h)$ are i.i.d:

$$E\hat{f}(x) = \int \frac{1}{h} K\left(\frac{y-x}{h}\right) f(y)\,dy$$

bias “s

We take two terms of a Taylor expansion of

$$E\hat{f}(x) = \int K(u)\left[f(x) + huf'(x) + \frac{1}{2}\right.$$

$$= f(x) + hf'(x)\int uK(u)du + \frac{1}{2}h^2 \int K(u)u^2 f''(x + hu^*)du$$

$$= f(x) + o(h^2)$$

so bias decreases at rate h^2. Need to trade this off with variance.

Variance

Variance of $\hat{f}(x) = \frac{1}{n}\sum K((X_i - x)/h)/h$ is

$$\mathrm{Var}(\hat{f}(x)) = \frac{1}{n}\mathrm{Var}\,\frac{1}{h}K\left(\frac{X-x}{h}\right)$$

[illegible]

$$= \frac{1}{n} \quad \frac{1}{h} \quad \frac{1}{h}K \quad \frac{\quad}{h} \quad f \quad [illegible] \quad hu)du\Big)^2\Big)$$

$$= \frac{1}{nh}\int K(u)^2 f(x+hu)du - [illegible]\, u\Big)^2$$

$$= O\left(\frac{1}{nh}\right)$$

increases at rate $1/h$ for a fixed n.

Trade Off

We can do calculus with order notation by looking at

$$E(\hat{f}(x) - f(x))^2 = E(\hat{f}(x) - E\hat{f}(x))^2 + (E\hat{f}(x) - f(x))^2$$

$$= \frac{A}{nh} + (f(x) + Bh^2 - f(x))^2$$

whic

$$h = \left(\frac{A}{4B^2 n}\right)^{1/5} =$$

or plugging this back into MSE

$$E(\hat{f}(x) - f(x))^2 = O(n^{-4/5}).$$

Notice that this is slower than the $O(n^{-1})$ rate we usually get for squared error of parameter estimates.

Extensions 2: Local Linear Regression

Re-thinking Nadaraya-Watson; the locally-weighted mean solves

$$\hat{m}(t) = \operatorname*{argmin}_{m} \sum_{i=1}^{n} (y_i - m)^2 K\left(\frac{t_i - t}{h}\right)$$

But w

Local

$$(\hat{\beta}_0(t), \hat{\beta}_1(t)) = \operatorname*{argmin}_{\beta_0, \beta_1} \sum_{i=1}^{n} (y_i - \beta_0 \quad \overline{\qquad})$$

has better statistical properties, especially at edge of data and if you want to take derivatives.

Predict $\hat{\beta}_0(t)$ but we can also get a slope from $\hat{\beta}_1(t)$
Local weighting can be applied to any model in this way.

R Packages

(see Lab for examples, but) a number of packages provide local polynomial fitting

- `locpol`
    -
    -
    -
-
    - only Gaussian kernels
    - also allows degree 0
    - density estimation etc
- `locfit` (not as clearly documented, but does multivariate local polynomials)

Generally, require evaluation points to be specified beforehand.

Basis Expansions

Alternative, to local methods, we can use regression approaches.

From multiple linear regression:

$$y_i = \beta_0 + x_{1i}\beta_1 + x_{2i}\beta_2 + \cdots + \epsilon_i$$

Or if th

Which we will write as

$$y_i = \sum_{j=1}^{K} c_j \phi_j(t_i) + \epsilon_i$$

or more compactly

$$g(t) = \mathbf{c}^T \Phi(t)$$

Splines

Using polynomials is generally a bad idea. Many other basis systems available. We'll illustrate with splines here.

- Splines are polynomial segments joined end-to-end
-
- s
- The order m (order = degree+1) of the polynomial segments and
- the location of the knots define the system.
- **Bsplines** are a particularly useful means constraints.

The package `splines` will be used for this.

Splines

Vancouver precipitation with knots at months.

Splines of order 1

Same as dividing data into bins and and use mean in each bin.

Splines

Vancouver precipitation with knots at months.

Splines of order 2

Splines

Vancouver precipitation with knots at months.

Splines of order 3

Splines

Vancouver precipitation with knots at months.

Splines of order 4

Splines

Vancouver precipitation with knots at months.

Splines of order 5

Splines

Vancouver precipitation with knots at months.

Splines of order 6

Least-Squares

We have observations

$$y_i = g(t_i) + \epsilon$$

and w

$$j=1$$

Minimize the sum of squared errors:

$$SSE = \sum_{i=1}^{n} (y_i - g(t_i))^2 = \sum_{i=1} (y_i - \mathbf{c}^T \Phi(t_i))^2$$

This is just linear regression!

The splines Package

To define a B-spline you need

1. A vector of knots
2. The order of the spline

On knots

- [illegible]
- [illegible] edges.
- Can spread these out like regular knots, or pile points.

`splineDesign` gives values of basis function

```
library(splines)
knots = c(rep(0,3),months,rep(365,3))   # Order 4 splines -
bvals = splineDesign(x=TT,knots=knots,ord=4)
```

Linear Regression on Basis Functions

- If the N by K matrix $\mathbf{\Phi}$ contains the values $\phi_k(t_j)$, and $\mathbf{y}$ is the vector $(y_1, \ldots, y_N)$, we can write

- east

$$\hat{\mathbf{c}} = \left(\mathbf{\Phi}^T \mathbf{\Phi}\right)$$

- Then we have the estimate

$$\hat{g}(t) = \Phi(t)\hat{\mathbf{c}} = \Phi(t)\left(\mathbf{\Phi}^T \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^T \mathbf{y}$$

Penalization

- Using splines, or other basis expansion, bias/variance trade-off is in terms of number and order of basis functions.

- This is usually difficult to work with, and estimates have high variability.
-

$$PENSSE(\lambda) = \underset{i=1}{\quad} (y_i - g(t_i))^2 + \lambda \quad g''(t) \quad dt$$

  - Fit to data + penalty for roughness.
  - $\int g''(t)^2 dt = 0$ if $g(t) = a + bt$, but large if g is "wiggly"
  - λ governs trade-off between the two.
- Use more knots that needed (one each data point if possible) and then control variance with λ.

Penalty Matrices

Observe that

$$\lambda \int \left(g''(t)\right)^2 dt = \lambda \int \left(\sum c_i \phi_i''(t)\right)^2 dt$$

$$= \quad c_i c_j \phi_i''(t) \phi_j''(t) dt$$

for P

We will approximate

$$\int \phi_i''(t) \phi_j''(t) \approx \delta \sum_k$$

(see numerical integration next) can write this as

$$P = \delta \mathbf{\Phi}''^T \mathbf{\Phi}''$$

`splineDesign` allows you to specify `derivs` for derivatives.

OCV and GCV

- Write

$$PENSSE(\lambda) = \quad (y_i \quad \Phi(t_i)\mathbf{c})^2 + \lambda \mathbf{c}^T P \mathbf{c}$$

-

$$\hat{\mathbf{c}} = \quad \mathbf{\Phi}^T \mathbf{\Phi} + \lambda \qquad {}^T$$

or

$$\hat{\mathbf{y}} = \mathbf{\Phi} \left(\mathbf{\Phi}^T \mathbf{\Phi} + \lambda P \right)$$

- Same formulae for OCV and GCV for λ.

Fitting Vancouver

```
knots = c( rep(0,3),0:365,rep(365,3) ) # Knot every day
bvals = splineDesign(knots=knots,x=TT)  # Order 4 is default

pvals = splineDesign(knots=knots,x=seq(0,365,by=0.1),derivs=2)
Pen = 0.1*t(pvals)%*%pvals

# Now loo
lambd

OCV = 0*l
GCV = 0*l

Yhats = matrix(0,365,length(lambda))

for(i in 1:length(lambda)){
  Slam = bvals%*%solve( t(bvals)%*%bvals+lambda

  Yhat = Slam%*%Y
  OCV[i] = mean( (Y-Yhat)^2/(1-diag(Slam))^2 )
  GCV[i] = sum( (Y-Yhat)^2)/sum( (1-diag(Slam))^2 )

  Yhats[,i] = Yhat
}
```

Fitting Vancouver

GCV/OCV Curves

Fits

A Comparison

Broader Uses

- Splines (and other bases) allow non-parametric terms to be included in any model.

- Eg. consider the partially linear model with a linear effect for x_1 and non-parametric effect for x_2:

- ^{2} ^{2} s.

- Works in generalized linear models, non-li other places.

- Many basis systems other than B-splines. M Fourier bases

$$1, \cos(\omega t), \sin(\omega t), \cos(2\omega t), \sin(2\omega t), \cos(3\omega t), \ldots$$

and wavelets.

Summary

Two ways of adding non-parametric effects

1. Kernel methods
    - Weight points to account for those that are closest to t.
2.
    - Represent non-parametric effect via a s
    - Add penalization to obtain smoothnes
    - Computationally efficient, can target t
    - Harder to demonstrate mathematica

In both cases, smoothing parameters (h, λ) must be chosen by cross validation or other methods.