Notes

- No Office Hour today, Tuesday Mar 2
- Practise test questions posted
- HW 2 due date Tuesday Mar 9
- HW 2 questions and HW 1 answers Thursday, Mar 4
- Statistical Society of Canada case studies

  `http://www.ssc.ca/documents/case_studies/2010/`

Technology

The data deluge

Businesses, governments and society are only starting to tap its vast potential

Feb 25th 2010 | From *The Economist* print edition

§7.3 Estimating expected prediction error

-

$$\begin{aligned} \mathrm{Err}_{\mathcal{T}} &= E\{L(Y, \hat{f}(X)) \mid \mathcal{T}\} \\ \mathrm{Err} &= E_{\mathcal{T}}[E\{L(Y, \hat{f}(X)) \mid \mathcal{T}\}] \end{aligned}$$

- know that $\overline{err} = \frac{1}{N} \sum L\{y_i, \hat{f}(x_i)\}$ is too small
- can show that on average, $\overline{err}$ should be inflated by

$$2\frac{d\sigma_\epsilon^2}{N}$$

- where d is the number of inputs

... can show (§7.4)

- Define

$$\mathrm{Err}_{\mathrm{in}} = \frac{1}{N}\sum_{i=1}^{N} E\{L(Y_i^0, \hat{f}(x_i)) \mid \mathcal{T}\}$$

- new Y_i^0 at each of the training inputs $x_1, \ldots, x_N$
- and define

$$\omega = E_{\mathbf{y}}(\mathrm{Err}_{\mathrm{in}} - \overline{err})$$

- expectation over $\mathbf{y}$ instead of $\mathcal{T}$ (x's fixed)
-

$$\omega = \frac{2}{N}\sum_{i=1}^{N} \mathrm{cov}(\hat{y}_i, y_i)$$

- leading to (7.22)
-

$$\begin{aligned} E_{\mathbf{y}}(\mathrm{Err}_{\mathrm{in}}) &= E_{\mathbf{y}}(\overline{err}) + \frac{2}{N}\sum_{i=1}^{N} \mathrm{cov}(\hat{y}_i, y_i) \\ &= E_{\mathbf{y}}(\overline{err}) + \frac{2}{N} d\sigma_\epsilon^2 \end{aligned}$$

- for linear fits with d basis functions

... can show

- $$\begin{aligned} E_{\mathbf{y}}(\mathrm{Err}_{\mathrm{in}}) &= E_{\mathbf{y}}(\overline{err}) + \frac{2}{N}\sum_{i=1}^{N}\mathrm{cov}(\hat{y}_i, y_i) \\ &= E_{\mathbf{y}}(\overline{err}) + \frac{2}{N}d\sigma_\epsilon^2 \end{aligned}$$
- (§7.5) $\overline{err}$ estimates $E_{\mathbf{y}}(\overline{err})$ $\quad\quad$ $\frac{2}{N}d\hat{\sigma}_\epsilon^2$ estimates 2nd term
- with log-likelihood loss function, the result is
- as $N \to \infty$,

$$-2E\{\log \Pr(Y;\hat{\theta})\} \simeq -\frac{2}{N}\sum_{i=1}^{N}\log \Pr(y_i;\hat{\theta}) + 2\frac{d}{N}$$

- which motivates

$$AIC = -\frac{2}{N}\ell(\hat{\theta};\mathbf{y}) + \frac{2d}{N}$$

- e.g. in (7.30) $d = d(\alpha)$ depends on a smoothing parameter
- (§7.6) with smoothing splines replace d by $\mathrm{trace}\mathbf{S}_\lambda$
- skip §7.7, 8, 9

Direct estimation of EPE §7.10

- another way to estimate prediction error
- $$Err = E_{\mathcal{T}} E\{L(Y, \hat{f}(X)) \mid \mathcal{T}\}$$
- $$CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(x_i))\}$$
- leave-one-out (N-fold):
- $$\frac{1}{N} \sum L(y_i, \hat{f}^{-i}(x_i)) = (e.g.) \frac{1}{N} \sum \{y_i - \hat{f}^{-i}(x_i)\}^2$$
- $$y_i - \hat{f}^{-i}(x_i) = \frac{y_i - \hat{f}(x_i)}{1 - S_{ii}}$$
- for any linear smoother $\hat{f} = Sy$
- $$CV = \frac{1}{N} \sum \{\frac{y_i - \hat{f}(x_i)}{1 - S_{ii}}\}^2$$
- $$GCV = \frac{1}{N} \sum \{\frac{y_i - \hat{f}(x_i)}{1 - \mathrm{tr}(\mathrm{S})/N}\}^2$$

comments

- §7.10.2,3: CV must be carried out before any model simplification that uses the y's
- both CV and K-fold CV seem to estimate expected prediction error but not prediction error
- CV can be quite variable (especially leave one out)
- p.231: using AIC, CV or GCV to choose smoothing parameter seems to overfit the data (?)
- Example: biasvariance.R

Projection pursuit regression (§11.2)

- response Y, inputs $X = (X_1, \ldots, X_p)$
- model $f(X) = E(Y \mid X)$ or $f(X) = pr(Y = 1 \mid X)$ or $f_k(X) = pr(Y = k \mid X)$
- PPR model $f(X) = \sum_{m=1}^{M} g_m(\omega_m^T X) = \sum g_m(V_m)$, say
- g_m are 'smooth' functions, as in generalized additive models
- $V_m = \omega_m^T X$ are derived variables: the projection of X onto $\omega_m = (\omega_{m1}, \ldots, \omega_{mp})$, with $||\omega_m|| = 1$
- Figure 11.1
- as g_m are nonlinear (in general), we are forming nonlinear functions of linear combinations
- as $M \to \infty$, $\sum g_m(\omega_m^T X)$ can get arbitrarily close to any continuous function on R^p
- if $M = 1$ a generalization of linear regression

PPR fitting

- training data $(x_i, y_i), \quad i = 1, \ldots, N$

$$\min_{\{g_m, \omega_m\}} \sum_{i=1}^{N} \{y_i - \sum_{m=1}^{M} g_m(\omega_m^T x_i)\}^2$$

- $M = 1$: fix ω, form $v_i = \omega^T x_i, i = 1, \ldots, N$
- solve for g using a regression smoother – kernel, spline, `loess`, etc.
- given g, estimate ω by weighted least squares of a derived variable z_i on x_i with weights $g_0^2(\omega_0^T x_i)$ and no constant term
- uses a simple linear approximation to $g(\cdot)$ (see note)
- if $M > 1$ add in each derived input one at a time

PPR fitting

- training data (x_i, y_i), $i = 1, \ldots, N$

$$\min_{\{g_m, \omega_m\}} \sum_{i=1}^{N} \{y_i - \sum_{m=1}^{M} g_m(\omega_m^T x_i)\}^2$$

- $M = 1$: fix ω, form $v_i = \omega^T x_i$, $i = 1, \ldots, N$
- solve for g using a regression smoother – kernel, spline, `loess`, etc.
- given g, estimate ω by weighted least squares of a derived variable z_i on x_i with weights $g_0'^2(\omega_0^T x_i)$ and no constant term
- uses a simple linear approximation to $g(\cdot)$ (see note)
- if $M > 1$ add in each derived input one at a time

$$g(\omega^T x_I) \simeq g(\omega_0^T x_i) + g'(\omega_0^T x_i)(\omega - \omega_0)^T x_i$$

$$\begin{aligned}
\{y_i - g(\omega^T x_i)\}^2 &= \{y_i - g_0 - g_0'(\omega - \omega_0)^T x_i\}^2 \\
&= (g_0')^2 \{\frac{y_i}{g_0'} - \frac{g_0}{g_0'} - (\omega - \omega_0)^T x_i\}^2 \\
&= (g_0')^2 \left\{\omega_0^T x_i + \left(\frac{y_i - g_0}{g_0'}\right) - \omega^T x_i\right\}^2
\end{aligned}$$

weight derived response (target)

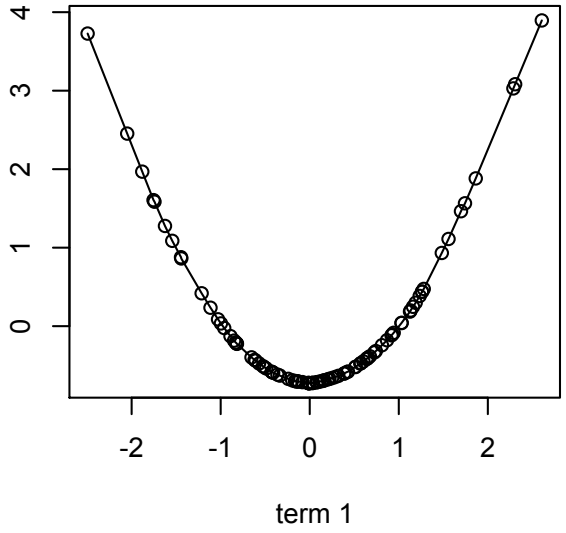
PPR implementation

- a smoothing method that provides derivatives is convenient
- M is usually estimated as part of the fitting
- provided in `MASS` library as `ppr`: fits M_{max} terms and drops least effective term and refits, continues down to M terms: both M and M_{max} provided by the user
- `ppr` also accommodates more than a single response Y; see help file
- difficult to interpret results of model fit, but may give good predictions on test data
- PPR is more general than GAM, because it can accommodate interactions between features: eg. $X_1 X_2 = \{(X_1 + X_2)^2 - (X_1 - X_2)^2\}/4$
- the idea of 'important' or 'interesting' projections can be used in other contexts to reduce the number of features, in classification and in unsupervised learning, for example

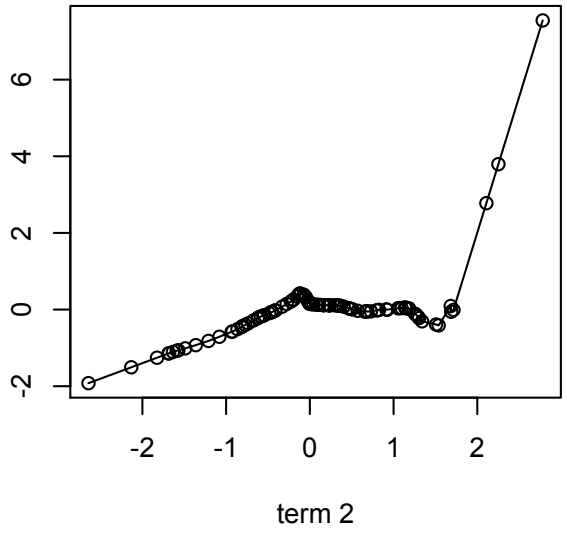
Example

```
> sigmoid = function(x){1/(1+exp(-x))}
> x1 = rnorm(100)
> x2 = rnorm (100)
> z = rnorm(100)
> y = sigmoid(3*x1+3*x2)+ (3*x1-3*x2)^2 + 0.3*z
> simtest = data.frame(cbind(x1,x2,y))
> pairs(simtest)
> sim.ppr = ppr(y ~ x1 + x2, data=simtest, nterms = 2, max.terms=5)
> summary(sim.ppr)
Call:
ppr(formula = y ~ x1 + x2, data = simtest, nterms = 2, max.terms = 5)

Goodness of fit:
 2 terms  3 terms  4 terms  5 terms
48.23182 40.63610 27.36090 27.82126

Projection direction vectors:
   term 1     term 2
x1 -0.7133944 -0.8127244
x2  0.7007627  0.5826483

Coefficients of ridge terms:
    term 1     term 2
24.5680314  0.9625025
> plot(sim.ppr)
> plot(update(sim.ppr,sm.method="gcv",nterms=2))
## adapted from code in Venables and Ripley, Sec.8.9 and help files
```

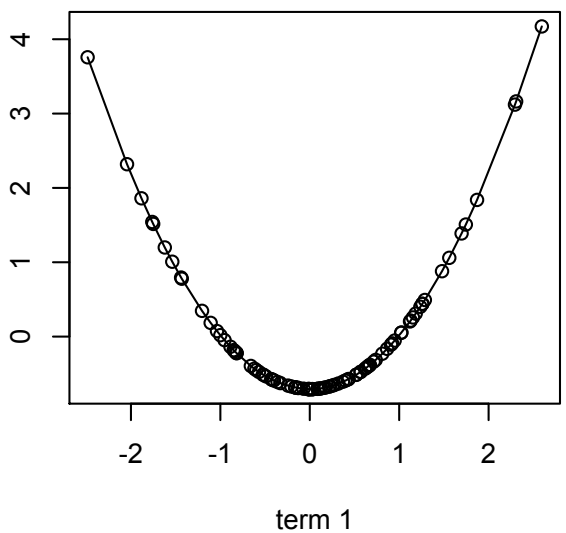
ppr using simulated data (exercise 11.5)



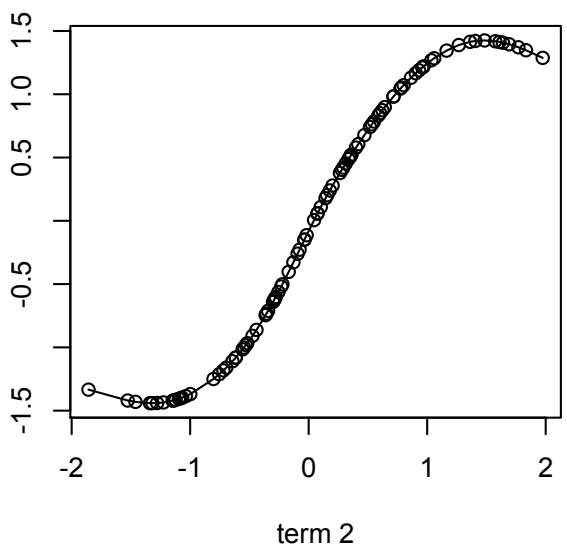
ppr using simulated data (exercise 11.5)



update(..., sm.method="gcv")

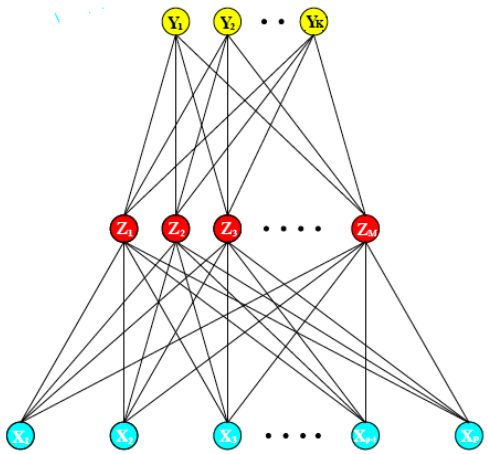


update(..., sm.method="gcv")



Neural networks (§11.3)

- inputs (features) $X_1, \ldots, X_p$
- derived inputs $Z_1, \ldots, Z_M$ (hidden layer)
- output (response) $Y_1, \ldots, Y_K$
  - usual regression has $K = 1$
  - classification has $(Y_1, \ldots, Y_K) = (0, \ldots, 1, 0, \ldots)$
  - also can accommodate multivariate regression with several outputs
- Figure 11.2



... neural networks

- derived inputs $Z_m = \sigma(\alpha_{0m} + \alpha_m^T X)$ for some choice $\sigma(\cdot)$
- called an activation function
- often chosen to be logistic $1/(1 + e^{-v})$ (sigmoid)
- target $T_k = \beta_{0k} + \beta_k^T Z$
- output $Y_k = f_k(X) = g_k(T_1, \ldots, T_K)$ for some choice $g_k(\cdot)$
- in regression g_k would usually be the identity function
- in K-class classification usually use $g_k(T) = \dfrac{e^{T_k}}{\sum_{\ell=1}^{K} e^{T_\ell}}$
- in 2-class classification $g_1(T) = 1(T > 0)$
  hard thresholding

- $Y_k = g_k(\beta_{0k} + \sum_{m=1}^{M} \beta_{km} Z_m)$
- $Y_k = g_k(\beta_{0k} + \sum_{m=1}^{M} \beta_{km} \sigma(\alpha_{0m} + \sum_{\ell=1}^{p} \alpha_{\ell m} X_\ell))$

... neural networks

- connection to PPR: $f(X) = \sum_{m=1}^{M} \gamma_m g_m(\omega_m^T X) = \sum \gamma_m V_m$
- $V_m \rightarrow Z_m = \sigma(\alpha_{0m} + \alpha_m^T X)$
- $g_m \rightarrow \sum_{m=1}^{M} \beta_{km} Z_m$
- i.e. $g_m(V_m)$ (arbitrary but smooth) replaced by
- $\beta_m \sigma(\alpha_{0m} + \alpha_m^T X)$ (linear logistic)
- smooth functions are in principle more flexible, but can use a large number of derived Z's
- the intercept terms α_{0m} and β_{0k} could be absorbed into the general expression by including an input of 1, and a hidden layer input of 1
- these are called 'bias units'
- Eq. (11.7), Figure 11.3

NN fitting (§11.4)

- need to estimate $(\alpha_{0m}, \alpha_m), m = 1, \ldots, M$ $M(p+1)$ and $(\beta_{0k}, \beta_k), k = 1, \ldots K$ $K(M+1)$
- loss function $R(\theta); \theta = (\alpha_{0m}, \alpha_m, \beta_{0k}, \beta_k)$ to be minimized; regularization needed to avoid overfitting
- loss function would be least squares in regression setting, e.g.

$$\sum_{i=1}^{N} \sum_{k=1}^{K} \{y_{ik} - f_k(x_i)\}^2$$

- for classification could use cross-entropy

$$\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i)$$

- the parameters α and β called (confusingly) weights, and regularization is called weight decay

Back propogation

- data $(y_{ik}, x_i), i = 1, \ldots, N, k = 1, \ldots, K$
- let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$ and $z_i = (z_{1i}, \ldots, z_{mi})$
- $R(\theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} \{y_{ik} - f_k(x_i)\}^2 = \sum R_i(\theta)$, say
- $f_k(x_i) = g_k(\beta_k^T z_i)$

$$\begin{aligned}
\frac{\partial R_i}{\partial \beta_{km}} &= -2\{y_{ik} - f_k(x_i)\} g_k'(\beta_k^T z_i) z_{mi} \\
\frac{\partial R_i}{\partial \alpha_{m\ell}} &= -2 \sum_{k=1}^{K} \{y_{ik} - f_k(x_i)\} g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i) x_{i\ell}
\end{aligned}$$

at each iteration use $\partial R / \partial \theta$ to guide choice to next point

$$\begin{aligned}
\beta_{km}^{(r+1)} &= \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}} \\
\alpha_{m\ell}^{(r+1)} &= \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}} \qquad (11.13) \\
\delta_{ki} &= -2\{y_{ik} - f_k(x_i)\} g_k'(\beta_k^T z_i) \\
s_{mi} &= -2 \sum_{k=1}^{K} \{y_{ik} - f_k(x_i)\} g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i) \\
s_{mi} &= \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki} \qquad (11.15)
\end{aligned}$$

use current estimates to get $\hat{f}_k(x_i)$
compute δ_{ki} and hence s_{mi} from (11.15)
put these into (11.13)

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}} \quad (11.13)$$

$$\delta_{ki} = -2\{y_{ik} - f_k(x_i)\} g_k'(\beta_k^T z_i)$$

$$s_{mi} = -2 \sum_{k=1}^{K} \{y_{ik} - f_k(x_i)\} g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i)$$

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki} \quad (11.15)$$

use current estimates to get $\hat{f}_k(x_i)$
compute δ_{ki} and hence s_{mi} from (11.15)
put these into (11.13)

- the coefficients $(\alpha_{m\ell}, \beta_{km})$ are usually called weights
- the algorithm is called back propogation or the δ-rule
- can be computed in time linear in the number of hidden units
- can be processed one instance (case) at a time
- any continuous function can be represented this way (with enough Z's)

Training NNs (§11.5)

- with small $\alpha_{m\ell}$, $\sigma(v) \simeq v$; large linear regression
- if algorithm stops early, $\alpha_{m\ell}$ still small; fit 'nearly' linear or shrunk towards a linear fit
- use penalty as in ridge regression to avoid overfitting
- $\min_\theta\{R(\theta) + \lambda J(\theta)\}$
- $J(\theta) = \sum \beta_{km}^2 + \sum \alpha_{m\ell}^2$
- as in ridge regression need to scale inputs to mean 0, var 1 (at least approx.)
- λ called weight decay parameter; seems to be more crucial than the number of hidden units
- `nnet` in `MASS` library; rec'd $\lambda \in (10^{-4}, 10^{-2})$ for LS; $\lambda \in (.01, .1)$ for entropy
- regression examples: §11.6, simulated Figures 11.6, 7, 8
- classification example: Figure 11.4 and Section 11.7