Section 1
Linear Models

Stat341
T. Hastie

Handout #1
Apr 2, 1997.

The linear model has been the mainstay of statistics. Despite the great inroads made by modern nonparametric regression techniques, linear models remain important, and so we need to understand them well.

- theory of least squares
- computational aspects
- distributional aspects
- linear models in Splus
- formulas for expressing models
- contrasts

Theory of Least Squares

N measurements $x_i \in \mathbf{R}^p$, $\;y_i \in \mathbf{R}$, $\;i = 1, \ldots, N$, $N > p$.
Linear Model:

$$y_i = \beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j + \varepsilon_i \tag{1}$$

with ε_i $i.i.d.$, $\;\mathbf{E}(\varepsilon_i) = 0$, $\;\mathbf{Var}(\varepsilon_i) = \sigma^2$. We either *assume* the linear model is correct, or more realistically think of it as a linear approximation to the regression model

$$\mathbf{E}(y_i|x_i) = f(x_i)$$

Either way, the most popular way of fitting the model is *least squares*: pick $\beta_0, \; \beta_j, \; j = 1, \ldots, p$, to minimize

$$\mathbf{RSS}(\beta_0, \beta_1, \ldots, \beta_p) = \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 \tag{2}$$

Vector notation

- Absorb β_0 into β, and augment the vector x_i with a 1 (and let the new dimension be p for simplicity).
- Write

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}_{(N\times 1)} \qquad X = \begin{bmatrix} x_1^T \\ \vdots \\ x_N^T \end{bmatrix}_{(N\times p)}$$

Then (2) can be written

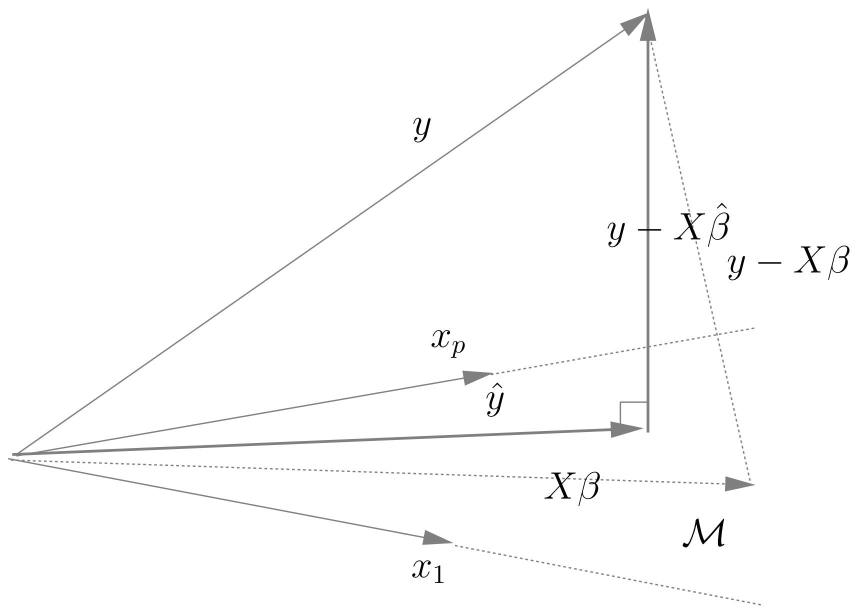
$$\mathbf{RSS}(\beta) = \|y - X\beta\|^2 = (y - X\beta)^T(y - X\beta) \qquad (3)$$

$$\begin{gathered} \partial\,\mathbf{RSS}\,/\partial\beta = -2X^T(y - X\beta) = 0 \\ \Downarrow \\ \hat{\beta} = (X^T X)^{-1} X^T y \end{gathered}$$

if $X^T X$ is invertible. This is the *text book* solution to the least squares problem.

Geometry of Least Squares

The geometrical solution is more revealing.



$\hat{y} = X\hat{\beta}$ is the *orthogonal projection* of y onto the subspace $\mathcal{M} \subset \mathbf{R}^n$ spanned by the columns of X. This is true even if X is not of full column rank.
Proof: Pythagoras.

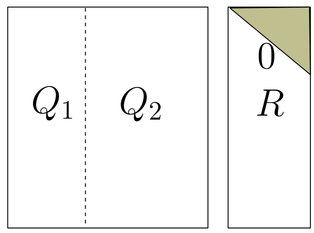
$$y - \hat{y} \perp \mathcal{M}$$
$$\Updownarrow$$
$$(y - X\hat{\beta}) \perp x_j \ \forall j \qquad (x_j \text{ is a column of } X \text{ here})$$
$$\Updownarrow$$
$$X^T(y - X\hat{\beta}) = 0$$

Computational Aspects

Q-R decomposition of X:

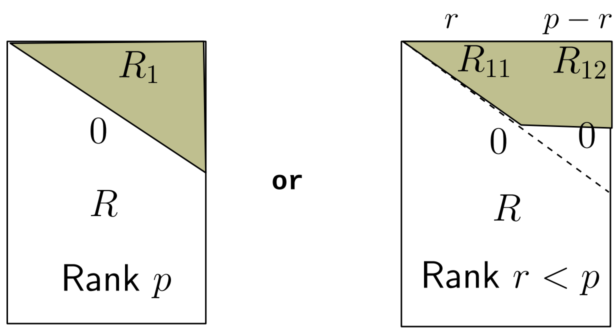
$$X_{N\times p} = Q_{N\times N} R_{N\times p}$$

$$=$$



where Q has orthonormal columns: $Q^T Q = I$ (and rows?)

R is upper triangular, and may not have full rank:



- For the full rank case,

$$\begin{aligned}
\|y - X\beta\|^2 &= \left\|Q^T y - R\beta\right\|^2 \\
&= \left\|Q_1^T y - R_1\beta\right\|^2 + \left\|Q_2^T y\right\|^2 \\
&\Rightarrow \hat{\beta} = R_1^{-1} Q_1^T y \\
\mathbf{RSS}(\hat{\beta}) &= \left\|Q_2^T y\right\|^2
\end{aligned}$$

- Effects: $e = Q^T y$ - Coordinates of y on columns of Q.
- $\hat{y} = Q_1 Q_1^T y = Hy = X(X^T X)^{-1} X^T y$ — H is known as the *hat* matrix (because it puts the hat on y).
- Non full rank case - $\mathbf{Rank}(X) = r < p$. We need to solve $Q_1^T y = R_{11}\beta_1 + R_{12}\beta_2$, where Q_1 has r columns. There are infinite solutions (more linear parameters than equations). We can set $\beta_2 = 0$, and solve for β_1, but this solution is arbitrary.
- $\hat{y} = Q_1 Q_1^T y$ is still well defined, and unique.
- Least squares computations using the QR decomposition is standard practice, and is what is used in Splus. The computations are efficient, and numerically stable. Inverting $X^T X$ directly is *seldom* reccomended.

Distributional Aspects

- $\mathbf{Cov}\,\hat{\beta} = (X^TX)^{-1}\sigma^2 = (R^TR)^{-1}\sigma^2$

- If $\varepsilon \sim N(0, \sigma^2 I)$ and the linear model is correct, then $\hat{\beta} \sim N(\beta, (X^TX)^{-1}\sigma^2)$, and this leads to the t-tests for individual parameters that often get printed out by LS software.

- $e = Q^Ty \sim N(R\beta, \sigma^2 I)$, *i.e.*

$$\begin{pmatrix} e_1 \\ e_2 \end{pmatrix} \sim N\left(\begin{pmatrix} R_1\beta \\ 0 \end{pmatrix}, \sigma^2 I)\right)$$

  and hence $\|e_2\|^2 = \left\|Q_2^Ty\right\|^2 = \mathbf{RSS}(\hat{\beta}) \sim \sigma^2\chi^2_{N-p}$

- Under $H_0: \ \beta = 0$, $\|e_1\|^2 \sim \sigma^2\chi^2_p$, and e_1 is independent of e_2 (why?), hence

$$\frac{\|e_1\|^2}{p} / \frac{\|e_2\|^2}{N-p} \sim F_{p,N-p}$$

  Note that $\|e_1\|^2 = \|\hat{y}\|^2$.

A Language for expressing linear models

Venables and Ripley, page 153+, Chambers and Hastie, 18-44.

```
Hwt  ~ Bwt + Sex
```

"Heart Weight is modelled as Body Weight plus Sex"
This implies some numerical setup, namely

$$X = \begin{bmatrix} 1 & \mathtt{Bwt}_1 & \mathtt{Sex}_1 \\ 1 & \mathtt{Bwt}_2 & \mathtt{Sex}_2 \\ \vdots & \vdots & \vdots \\ 1 & \mathtt{Bwt}_N & \mathtt{Sex}_N \end{bmatrix} \qquad y = \begin{bmatrix} \mathtt{Hwt}_1 \\ \mathtt{Hwt}_2 \\ \vdots \\ \mathtt{Hwt}_N \end{bmatrix}$$

Sex is a factor (Male and Female) — What is coded is a *contrast* — in this case -1 for `Sex = F`, 1 for `Sex = M`.

Question: Why not use a two column matrix instead?

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 1 & 0 \end{bmatrix}$$

Note that the columns sum to 1 — we have introduced a degeneracy or aliasing (more later.)

Formulas in General

`y ~ a + b + c + ...`

where `a, b, c, ...` can be

- numeric vectors — these get included into X as is.
- numeric matrices — again these get included as is.
- k-level factors — these typically get converted to $k-1$ column contrast matrices, and then inserted into X.
- *any expression that evaluates to one of the above*

For example:

- `log(y) ~sin(x) + cut(z, 3)`:
  here we first apply the $\log$ and $\sin$ functions to `y` and `x` resp.; `cut(z,3)` creates a 3-level factor by cutting `z` in two places (roughly the tertiles), which in turn get coded as contrasts and included in X.
- `1/y ~poly(x, 4) + I(z>0)`:
  `poly(x, 4)` produces a matrix of orthogonal polynomials in `x` — four columns in all, since the constant is omitted. `I(z>0)` is a dummy variable created from the logical variable `z>0`.

Contrasts

Consider the one-way layout: $m_i = \mu_0 + \mu_i, \; i = 1, \ldots, k$

$$X = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

X is not of full rank, so

$$\begin{aligned} \hat{y}_i = y_i &= \mu_0' + \mu_i' \\ &= \mu_0'' + \mu_i'' \end{aligned}$$

and hence there is no way to extract the individual parameters uniquely. But $\hat{y}_i - \hat{y}_j = \mu_i' - \mu_j' = \mu_i'' - \mu_j''$ is unique. The latter is called an *estimable contrast.* Similarly $\mu_i - \bar{\mu}$ is estimable.

The *Gauss-Markov* theorem tells us what contrasts are estimable - namely $A\mu$ where A is a linear combination of the rows of X.

It makes sense with one row per mean. These are all we have, so we cannot extract *more* parameters than there are different means.

Contrast Matrix

$$\begin{bmatrix} m_1 \\ \vdots \\ m_k \end{bmatrix} = \begin{bmatrix} 1 & \\ 1 & C \\ \vdots & {\scriptstyle (k \times k-1)} \\ 1 & \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{k-1} \end{bmatrix}$$

with $C^T\mathbf{1} = 0$. Then code u_i via $C_{p\times p-1}$ rather than $I_{p\times p}$. Note that if $u = C\beta$, then $\mathbf{1}^T u = \mathbf{1}^T C\beta = 0$, and $\therefore \sum_i u_i = 0$.

Example: Helmert contrasts (`contr.helmert` in Splus):

$$C = \begin{pmatrix} -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ 0 & 2 & -1 & -1 \\ 0 & 0 & 3 & -1 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

Example: Traditional mean-zero contrasts (`contr.sum` in Splus):

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 \end{pmatrix}$$

Read page 155 of Venables & Ripley and page 32 of Chambers & Hastie.

More formulas

- interactions — `y ~a:b` and `y ~a*b`
  these imply parameters of the form β_{ij} for each crossing of level i of factor `a` with level j of factor `b`. What about redundancies caused by intercept? and main effects? How do two way contrasts get coded?

- `~a*b` or equivalently `~1 + a + b + a:b`
  This creates an intercept term, main effects for `a` and `b`, and interactions. Suppose we use C_a to code the 3 levels of `a` (using the *sum* contrasts), and C_b to code the 4 levels of `b` (using the helmert contrasts):

$$C_a = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & -1 \end{pmatrix} \quad C_b = \begin{pmatrix} -1 & -1 & -1 \\ 1 & -1 & -1 \\ 0 & 2 & -1 \\ 0 & 0 & 3 \end{pmatrix}$$

  Then the model matrix corresponding to the run sequence $(a_1, b_1), (a_1, b_2), \ldots, (a_2, b_1), \ldots, (a_3, b_4)$ and the formula above would consist of particular tensor products of $\mathbf{1}$, C_a and C_b, best illustrated by the example:

```
> a <- factor(rep(1:3,c(4,4,4))
> a <- C(a, "contr.sum") #
> b <- factor(rep(1:4,3))
> b <- C(a, "contr.helmert")
> model.matrix( ~ a*b)

   (Intercept) a1 a2 b1 b2 b3 a1b1 a2b1 a1b2 a2b2 a1b3 a2b3
 1           1  1  0 -1 -1 -1   -1    0   -1    0   -1    0
 2           1  1  0  1 -1 -1    1    0   -1    0   -1    0
 3           1  1  0  0  2 -1    0    0    2    0   -1    0
 4           1  1  0  0  0  3    0    0    0    0    3    0
 5           1  0  1 -1 -1 -1    0   -1    0   -1    0   -1
 6           1  0  1  1 -1 -1    0    1    0   -1    0   -1
 7           1  0  1  0  2 -1    0    0    0    2    0   -1
 8           1  0  1  0  0  3    0    0    0    0    0    3
 9           1 -1 -1 -1 -1 -1    1    1    1    1    1    1
10           1 -1 -1  1 -1 -1   -1   -1    1    1    1    1
11           1 -1 -1  0  2 -1    0    0   -2   -2    1    1
12           1 -1 -1  0  0  3    0    0    0    0   -3   -3
```

- ~`a:b` ignores the main effects, having just an intercept and interactions.
- ~`a:b -1` — no intercept, pure interactions.

```
> model.matrix(~ a:b -1)

   a1b1 a2b1 a3b1 a1b2 a2b2 a3b2 a1b3 a2b3 a3b3 a1b4 a2b4 a3b4
 1    1    0    0    0    0    0    0    0    0    0    0    0
 2    0    0    0    1    0    0    0    0    0    0    0    0
 3    0    0    0    0    0    0    1    0    0    0    0    0
 4    0    0    0    0    0    0    0    0    0    1    0    0
 5    0    1    0    0    0    0    0    0    0    0    0    0
```

```
 6    0    0    0    0    1    0    0    0    0    0    0    0
 7    0    0    0    0    0    0    0    1    0    0    0    0
 8    0    0    0    0    0    0    0    0    0    0    1    0
 9    0    0    1    0    0    0    0    0    0    0    0    0
10    0    0    0    0    0    1    0    0    0    0    0    0
11    0    0    0    0    0    0    0    0    1    0    0    0
12    0    0    0    0    0    0    0    0    0    0    0    1
```

- `~a + a:b` or `~a/b` or `~1 + a+ b %in% a`
  This specifies nesting, e.g. State, and County within State.

```
> model.matrix(~ a/b)

  (Intercept) a1 a2 a1b1 a2b1 a3b1 a1b2 a2b2 a3b2 a1b3 a2b3 a3b3
 1          1  1  0   -1    0    0   -1    0    0   -1    0    0
 2          1  1  0    1    0    0   -1    0    0   -1    0    0
 3          1  1  0    0    0    0    2    0    0   -1    0    0
 4          1  1  0    0    0    0    0    0    0    3    0    0
 5          1  0  1    0   -1    0    0   -1    0    0   -1    0
 6          1  0  1    0    1    0    0   -1    0    0   -1    0
 7          1  0  1    0    0    0    0    2    0    0   -1    0
 8          1  0  1    0    0    0    0    0    0    0    3    0
 9          1 -1 -1    0    0   -1    0    0   -1    0    0   -1
10          1 -1 -1    0    0    1    0    0   -1    0    0   -1
11          1 -1 -1    0    0    0    0    0    2    0    0   -1
12          1 -1 -1    0    0    0    0    0    0    0    0    3
```

- `~a*b*c` of `(a+b)*c` — more complicated interaction models.

Lots of flexibility — see 2.3 and 2.4 of C&H. Scripts 6.1, 6.2, 6.3.

Linear models in Splus

```
fm <- lm( y ~ x + a * b, data = mydata )
```

where `mydata` is a dataframe that includes the variables `x`, `y`, and `b`. `fm` is an Splus object of *class* `"lm"`.

The modelling language in Splus is *object-oriented* — generic functions recognize the class of an object, and invoke class-specific methods.
Examples of generic functions with methods for `lm` objects are

- `fitted()`: extract fitted values.
- `residuals()`: extract residuals.
- `coefficients()` or `coef()`: extract coefficients.
- `model.matrix()`: extract the model matrix that was built from the formula, and used to fit the model.
- `summary()`: produce a summary of the properties of the fitted model.
- `print()`: a more succinct summary, also by simply typing the name of the object.
- `plot()`: produce a plot of the object.

`lm()` has a number of additional arguments, such as `weights=`, `subset=`, and more; see the (online) documentation, and experiment.