Statistical Models in Julia

Douglas Bates

January 16, 2013

Leveraging types and multiple dispatch in statistics

- The *R* community is accustomed to using generic functions, methods and composite types, especially for representing statistical models. In fact, generics and methods were introduced in *R*'s predecessor, *S*, with the "white book" ("Statistical Models in S", 1988)
- The concept was sound; the implementation rather casual. 'S3' classes are simply tags on a general type of structure and the generics allowed for single dispatch only.
- The 'S4' system of classes, generics and methods are closer in design to Julia types and multiple dispatch but have not gained much of a following nor are they well supported.
- The best known *R* packages that use S4 are the 'Matrix' package and the packages from Bioconductor.

Statistical models, probability distributions & parameters

- Statistical models express the distribution of the response vector as depending on the values of parameters and of covariates.
- Given the observed data we estimate values of the parameters by optimizing an objective function (e.g. log-likelihood, posterior density) with respect to the parameters.
- For example, in a linear regression model, the vector of responses, $\mathbf{y}$, is expressed as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$$

  where the $n \times p$ model matrix, $\mathbf{X}$, is derived from the values of covariates.
- The maximum likelihood estimates of $\boldsymbol{\beta}$ are the least squares estimates

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2$$

A fitted linear regression model in R and extractors

```
> fm = lm(optden ~ carb, Formaldehyde)
> class(fm)
[1] "lm"
> coef(fm)
(Intercept)        carb
0.005085714 0.876285714
> residuals(fm)
          1            2            3            4
-0.006714286  0.001028571  0.002771429  0.007142857  0.007
> fitted(fm)
         1          2          3          4          5
0.09271429 0.26797143 0.44322857 0.53085714 0.61848571 0.7
> deviance(fm)  # residual sum of squares
[1] 0.0002992
```

```
> summary(fm)

Call:
lm(formula = optden ~ carb, data = Formaldehyde)

Residuals:
        1         2         3         4         5         6
-0.006714  0.001029  0.002771  0.007143  0.007514 -0.011743

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.005086   0.007834   0.649    0.552
carb        0.876286   0.013535  64.744 3.41e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘

Residual standard error: 0.008649 on 4 degrees of freedom
Multiple R-squared: 0.999,  Adjusted R-squared: 0.9988
F-statistic:  4192 on 1 and 4 DF,  p-value: 3.409e-07
```

Methods with ‘lm’ as the class of the first argument

```
> methods(class = "lm")
 [1] add1.lm*           alias.lm*             anova.lm
 [5] confint.lm*        cooks.distance.lm*   deviance.lm*
 [9] dfbetas.lm*        drop1.lm*             dummy.coef.lm*
[13] extractAIC.lm*     family.lm*            formula.lm*
[17] influence.lm*      kappa.lm              labels.lm*
[21] model.frame.lm     model.matrix.lm       nobs.lm*
[25] predict.lm         print.lm              proj.lm*
[29] residuals.lm       rstandard.lm          rstudent.lm
[33] summary.lm         variable.names.lm*   vcov.lm*
```

Categorical covariates in linear models

```
> str(InsectSprays)
'data.frame':   72 obs. of  2 variables:
 $ count: num  10 7 20 14 14 12 10 23 17 20 ...
 $ spray: Factor w/ 6 levels "A","B","C","D",..: 1 1 1 1 1
> anova(fm2 <- lm(count ~ spray, InsectSprays))
Analysis of Variance Table

Response: count
          Df Sum Sq Mean Sq F value    Pr(>F)
spray      5 2668.8  533.77  34.702 < 2.2e-16
Residuals 66 1015.2   15.38
```

Generalized linear models

- The Gaussian (or "normal") distribution has many special properties. When generalizing to other distributions it helps to re-write the model as

$$\mathcal{Y} \sim \mathcal{N}(\mathbf{X}\boldsymbol{\beta}, \sigma^2\mathbf{I})$$

- A Poisson regression model for count data (as in the last example)

$$\mathbf{y} \sim \mathcal{P}(\exp(\mathbf{X}\boldsymbol{\beta}))$$

  where the exp of the vector is element-wise, $\mu_i = \exp(\eta_i)$
- For historical reasons, the mapping from $\boldsymbol{\mu}$ to $\boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta}$ is called the 'link function' so this model has a log-link.
- 'logistic regression' for a binary response uses a logistic or log-odds link, $\eta_i = \log(\mu_i/(1-\mu_i))$

Fitting GLM's

- Nelder and Wedderburn realized that models defined by a distribution, a linear predictor and a link function could be fit by iteratively re-weight least squares (IRLS).
- As the name implies, IRLS involves repeatedly solving a weighted least squares problem, updating the weights and the residual and iterating.
- This is similar to iterative methods for nonlinear least squares. In fact, the underlying operations of nonlinear regression and IRLS can be the same.

Linear mixed-effects models (LMM's)

- In a mixed-effects model some of the coefficients in the linear predictor apply to particular 'subjects' sampled from a population and we are interested in the distribution of these 'effects'.
- The model is defined by the conditional distribution

$$(\mathcal{Y}|\mathcal{B} = \mathbf{b}) \sim \mathcal{N}(\mathbf{X}\beta + \mathbf{Z}\mathbf{b}, \sigma^2\mathbf{I})$$

  and the unconditional distribution, $\mathcal{B} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$
- The dimension of $\mathbf{b}$ can be very large (in the millions is not uncommon), but $\mathbf{Z}$ is very sparse, as is $\boldsymbol{\Sigma}$.
- Because $\boldsymbol{\Sigma}$ is positive (semi-)definite we can express it as $\boldsymbol{\Sigma} = \sigma^2\boldsymbol{\Lambda}\boldsymbol{\Lambda}^T$. $\boldsymbol{\Lambda}$ is a function of $\boldsymbol{\theta}$ a parameter vector whose dimension is small.

Fitting LMM's

- By solving a penalized least squares (PLS) problem

$$\begin{bmatrix} \boldsymbol{\Lambda}^T\mathbf{Z}^T\mathbf{Z}\boldsymbol{\Lambda}+\mathbf{I} & \boldsymbol{\Lambda}^T\mathbf{Z}^T\mathbf{X} \\ \mathbf{X}^T\mathbf{Z}\boldsymbol{\Lambda} & \mathbf{X}^T\mathbf{X} \end{bmatrix} \begin{bmatrix} \widehat{\beta} \\ \tilde{\mathbf{b}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Lambda}^T\mathbf{Z}^T\mathbf{y} & \mathbf{X}^T\mathbf{y} \end{bmatrix}$$

  we can evaluate a 'profiled log-likelihood' that can be optimized with respect to $\boldsymbol{\theta}$.
- This can be extended to generalized linear mixed-effects models (GLMMM's) requiring penalized iteratively reweigthed least squares (PIRLS) or nonlinear mixed-effects models (NLMMs) requiring penalized nonlinear least squares (PNLS), etc.
- In the goriest forms these kinds of optimizations are nested within yet another optimization problem taking into account time-series dependencies or spatial dependencies.

lm and glm in Julia

Add the **DataFrames**, **Distributions**, **GLM** and **RDataSets** packages if you don't already have them

```
julia> form = data("datasets", "Formaldehyde")
6x3 DataFrame:
          carb optden
[1,]     1  0.1  0.086
[2,]     2  0.3  0.269
[3,]     3  0.5  0.446
[4,]     4  0.6  0.538
[5,]     5  0.7  0.626
[6,]     6  0.9  0.782
```

```
julia> fm = lm(:(optden ~ carb), form)
Formula: optden ~ carb

Coefficients:
         Term  Estimate Std. Error  t value  Pr(>|t|)
  (Intercept)   0.00509    0.00783    0.649     0.552
         carb   0.87629    0.01353   64.744     0.000 ***

Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘

R-squared: 0.0000

julia> deviance(fm)
0.0002992000000000012
```

This morning *dump* of a model frame is broken so we will look at components

```
julia> typeof(fm)
LmMod
julia> typeof(fm).names
(fr,mm,rr,pp)
julia> dump(fm.rr)  # the response component
LmResp
  mu: Array(Float64,(6,)) [0.0927143, 0.267971, 0.443229, 0
  offset: Array(Float64,(6,)) [0.0, 0.0, 0.0, 0.0, 0.0, 0.0
  wts: Array(Float64,(6,)) [1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
  y: Array(Float64,(6,)) [0.086, 0.269, 0.446, 0.538, 0.626
```

```
julia> dump(fm.pp)  # the predictor component
DensePredQR
  X: Array(Float64,(6,2)) 6x2 Float64 Array:
 1.0  0.1
 1.0  0.3
 1.0  0.5
 1.0  0.6
 1.0  0.7
 1.0  0.9
  beta0: Array(Float64,(2,)) [0.00508571, 0.876286]
  delbeta: Array(Float64,(2,)) [0.0, 0.0]
  qr: QRDense{Float64}
    hh: Array(Float64,(6,2)) 6x2 Float64 Array:
 -2.44949   -1.26557
  0.289898   0.63901
  0.289898  -0.141688
  0.289898  -0.277763
  0.289898  -0.413839
  0.289898  -0.68599
    tau: Array(Float64,(2,)) [1.40825, 1.15004]
```

Multiple linear regression

```
julia> LifeCycleSavings = data("datasets", "LifeCycleSavin
...
julia> fm2 = lm(:(sr ~ pop15 + pop75 + dpi + ddpi), LifeCy
Formula: sr ~ :(+(pop15,pop75,dpi,ddpi))

Coefficients:
        Term  Estimate Std. Error t value Pr(>|t|)
  (Intercept)  28.56609    7.35452   3.884    0.000 ***
       pop15  -0.46119    0.14464  -3.189    0.003 **
       pop75  -1.69150    1.08360  -1.561    0.126
         dpi  -0.00034    0.00093  -0.362    0.719
        ddpi   0.40969    0.19620   2.088    0.042 *
```

Generalized Linear models

```
julia> dobson = DataFrame({[18.,17,15,20,10,20,25,13,12],
                            gl(3,1,9), gl(3,3)},
                           ["counts","outcome","treatment"])
9x3 DataFrame:
        counts outcome treatment
[1,]      18.0       1         1
[2,]      17.0       2         1
[3,]      15.0       3         1
[4,]      20.0       1         2
[5,]      10.0       2         2
[6,]      20.0       3         2
[7,]      25.0       1         3
[8,]      13.0       2         3
[9,]      12.0       3         3
```

```
julia> fm3 = glm(:(counts ~ outcome + treatment), dobson,
1: 46.81189638788046, Inf
2: 46.76132443472076, 0.0010814910349747592
3: 46.761318401957794, 1.2901182375636843e-7

Formula: counts ~ :(+(outcome,treatment))

Coefficients:
         Term  Estimate Std. Error  t value  Pr(>|t|)
  (Intercept)   3.04452    0.17089   17.815     0.000 ***
    outcome:2  -0.45426    0.20215   -2.247     0.025 *
    outcome:3  -0.29299    0.19273   -1.520     0.128
  treatment:2   0.00000    0.19998    0.000     1.000
  treatment:3   0.00000    0.19999    0.000     1.000
```

An LmMod and a GlmMod differ mostly in the response object

```
julia> typeof(fm3)
GlmMod
julia> dump(fm3.rr)
GlmResp
  d: Poisson
    lambda: Float64 1.0
  l: LogLink
  eta: Array(Float64,(9,)) [3.04452, 2.59027, 2.75154, 3.04
  mu: Array(Float64,(9,)) [21.0, 13.3333, 15.6667, 21.0, 13
  offset: Array(Float64,(9,)) [0.0, 0.0, 0.0, 0.0, 0.0, 0.0
  wts: Array(Float64,(9,)) [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1
  y: Array(Float64,(9,)) [18.0, 17.0, 15.0, 20.0, 10.0, 20.
```

- Response types: LmResp, GlmResp (to be added NlsResp)
- Predictor types: LinPred, DensePred, DensePredChol, DensePredQR
- to be added: DensePredCholPiv, DensePredQRPiv, DDensePred, DDensePredChol, SparsePred, SparsePredChol, SparsePredQR, MixedPred, MixedPredChol, MixedPredDiag, ...
- Distribution types: Bernoulli, Beta, Binomial, Categorical, Cauchy, Chisq, Dirichlet, Exponential, FDist, Gamma, Geometric, HyperGeometric, Logistic, logNormal, Multinomial, NegativeBinomial, NoncentralBeta, NoncentralChisq, NoncentralF, NoncentralT, Normal, Poisson, TDist, Uniform, Weibull
- Link types: CauchitLink, CloglogLink, IdentityLink, InverseLink, LogitLink, LogLink, ProbitLink

What does Julia offer that R doesn't

- One language providing flexibility and efficiency