Introduction to the Stat-JR software package

Professor William Browne
Video 1 What is StatJR

- A statistical software package written in Python and first released in 2013.
- Named after our former colleague Jon Rasbash and pronounced “Stature”.
- Stat-JR is meant to appeal to novice users, expert users and other algorithm developers.
- It has its own MCMC estimation engine built into the software but also allows interoperability with other software packages (this talk).
- Has several interfaces including an electronic book interface including “statistical analysis assistant” features (talk 2).
- Can also be used to create “bespoke” training materials in combination with the SPSS software package (talk 3).
StatJR component based approach

Below is an early diagram of how we envisioned the system. Here you will see boxes representing components some of which are built into the STAT-JR system. The system is written in Python with a VB.net algebra processing system. A team of coders have worked together on the system.
Templates

Backbone of Stat-JR.
Consist of a set of code sections for advanced users to write. A bit like R packages.

For a model template it consists of at least:

- an *inputs* method which specifies inputs and types
- A *model* method that creates (BUGS like) model code for the algebra system
- An (optional) *latex* method can be used for outputting LaTeX code for the model.

Other optional functions required for more complex templates
Regression 1 Example

from EStat.Templating import *

class Regression1(Template):
    'A model template for fitting 1 level Normal multiple regression model in eStat only.'
    tags = ['Model', '1-Level', 'eStat', 'Normal']
    engines = ['eStat']
    inputs = '''
y = DataVector('Response: ')
x = DataMatrix('Explanatory variables: ', allow_cat=True, help='predictor variables')
beta = ParamVector(parents=[x], as_scalar=True)
tau = ParamScalar()
sigma = ParamScalar(modelled=False)
sigma2 = ParamScalar(modelled=False)
deviance = ParamScalar(modelled=False)
'''

model = '''
model{
    for (i in 1:length(${y})) {
        ${y}[i] ~ dnorm(mu[i], tau)
        mu[i] <- ${mmult(x, 'beta', 'i')}
    }
}

    # Priors
    % for i in range(0, x.ncols()):
    beta[i] ~ dflat()
    % endfor
    tau ~ dgamma(0.001000, 0.001000)
    sigma2 <- 1 / tau
    sigma <- 1 / sqrt(tau)
}
'''

latex = r'''
\begin{aligned}
\mbox{$${y}_i$$} & \sim \mbox{N}(\mu_i, \sigma^2) \\
\mu_i & = \mbox{mmulttex}(x, r'\beta', 'i') \\
%for i in range(0, len(x)):
\beta_{i} & \propto 1 \\
%endfor
\tau & \sim \mbox{Gamma} (0.001,0.001) \\
\sigma^2 & = 1 / \tau
\end{aligned}
'''

An example of STAT-JR – setting up a model
An example of STAT-JR – setting up a model

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response</td>
<td>normexam remove</td>
</tr>
<tr>
<td>Explanatory variables</td>
<td>cons, standlrit remove</td>
</tr>
<tr>
<td>Number of chains</td>
<td>3 remove</td>
</tr>
<tr>
<td>Random Seed</td>
<td>1 remove</td>
</tr>
<tr>
<td>Length of burnin</td>
<td>500 remove</td>
</tr>
<tr>
<td>Number of iterations</td>
<td>2000 remove</td>
</tr>
<tr>
<td>Thinning</td>
<td>1 remove</td>
</tr>
<tr>
<td>Use default algorithm settings</td>
<td>Yes remove</td>
</tr>
<tr>
<td>Generate prediction dataset</td>
<td>Yes remove</td>
</tr>
<tr>
<td>Use default starting values</td>
<td>Yes remove</td>
</tr>
<tr>
<td>Name of output results</td>
<td>out</td>
</tr>
</tbody>
</table>
Equations for model

\[ \text{normexam}_i \sim N(\mu_i, \sigma^2) \]

\[ \mu_i = \beta_0 \text{cons}_i + \beta_1 \text{standlrt}_i \]

\[ \beta_0 \propto 1 \]

\[ \beta_1 \propto 1 \]

\[ \tau \sim \Gamma(0.001, 0.001) \]

\[ \sigma^2 = 1/\tau \]

- All objects created available from one pull down and can be popped out to separate tabs in browser.
Equations for model

\[ \text{normexam}_i \sim N(\mu_i, \sigma^2) \]
\[ \mu_i = \beta_0 \text{cons}_i + \beta_1 \text{standlrt}_i \]
\[ \beta_0 \propto 1 \]
\[ \beta_1 \propto 1 \]
\[ \tau \sim \Gamma(0.001, 0.001) \]
\[ \sigma^2 = 1/\tau \]

- Note: Equations use MATHJAX and so underlying LaTeX can be copied and paste. The model code is based around the WinBUGS language with some variation.
Model code

- All objects created available from one pull down and can be popped out to separate tabs in browser.
Model code in detail

```r
model{
  for (i in 1:length(normexam)) {
    normexam[i] ~ dnorm(mu[i], tau)
    mu[i] <- cons[i] * beta0 + standlrt[i] * beta1
  }

  # Priors
  beta0 ~ dflat()
  beta1 ~ dflat()
  tau ~ dgamma(0.001000, 0.001000)
  sigma2 <- 1 / tau
  sigma <- 1/sqrt(tau)
}
```

For this template the code is, aside from the length function, standard WinBUGS model code.
Algebra system steps

Log posterior: \( \tau \left( \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_1(\text{normexam}_i - \beta_{-1}\text{standlrt}_i) \right) \beta_0 - \frac{\tau \left( \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_1^2 \right) \beta_0^2}{2} \)

Distribution: \( \text{dnorm} \)

Match A: \( \tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_1(\text{normexam}_i - \beta_{-1}\text{standlrt}_i) \)

Match B: \( -\left( \frac{\tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_1^2}{2} \right) \)

Sampling parameter: \( \mu = \tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_1(\text{normexam}_i - \beta_{-1}\text{standlrt}_i) \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_1^2 \)

Sampling parameter: \( \tau = \tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_1^2 \)

Sampling distribution: \( \beta_0 \sim \text{dnorm} \left( \frac{\tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_1(\text{normexam}_i - \beta_{-1}\text{standlrt}_i)}{\tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_1^2} \right), \tau \sum_{i=1}^{\text{length}(\text{normexam})} \text{cons}_1^2 \)
Algebra system steps
Algebra system steps

Use Gibbs sampling from conditional posterior for beta1:

$\beta_1 \sim N\left(\frac{\sum_{i=1}^{\text{length(normexam)}} \text{standlrt}_i \times (\text{normexam}_i - \beta_0 \times \text{cons}_i)}{\sum_{i=1}^{\text{length(normexam)}} \text{standlrt}_i^2}, \tau \times \left(\sum_{i=1}^{\text{length(normexam)}} \text{standlrt}_i^2\right)\right)$

$\beta_1 \sim N(0.000249799765395 \times (2382.12631198 + \beta_0 \times (-7.34783096611)), 4003.20632175 \times \tau)$

- Here the first line is what is returned by the algebra system – which works solely on the model code.
- The second line is what can be calculated when values are added for constants and data etc.
- System then constructs C code and fits model
The package can output C++ code that can then be taken away by software developers and modified.
Output of generated C++ code

// Update beta1
{
beta1 = dnorm((0.000249799765395*(2382.12631198+(beta0*(-7.34783096611)))),(4003.20632175*tau));
}

// Update beta0
{
beta0 = dnorm((((-0.462375992909)+((-7.34783096611)*beta1))*0.000246366100025),(tau*4059.0));
}

- Note now that the code includes the actual data in place of constants and so looks less like the familiar algebraic expressions
Output from the E-STAT engine

- Estimates and the DIC diagnostic can be viewed for the model fitted.
Output from the E-STAT engine

- E-STAT offers multiple chains so that we can use multiple chain diagnostics to aid convergence checking.
- Graphics are in svg format so scale nicely.
Interoperability with WinBUGS (Regression 2)

- This template offers the choice of many software packages for fitting a regression model.
- STAT-JR checks what is installed on the machine and only offers packages that are installed. Here we choose WinBUGS.
- Interoperability in the user interface is obtained via a few extra inputs. In fact in the template code user written functions are required for all packages apart from WinBUGS, OpenBUGS and JAGS. The transfer of data between packages is however generic.
Interoperability with WinBUGS (Regression 2)

- Here we can view the files required to run WinBUGS in the pane (script file shown but model, inits and data also available)
- The model can be run by press of a button.
Interoperability with R

- R can be chosen as another alternative. In fact here we have 2 choices – glm or MCMCglmm.
- You will see in the pane the script file ready for input to R. There will also be the data file that R requires.
Interoperability with R

- If written in to the code in the template – graphics from other software can be extracted.
- Here for example is a residual plot associated with the R fit of the model.
There are also templates for plotting. For example here is a plot using the XYplot template. Shown is the plot whilst the Python command script is also available. For more details on StatJR go to http://www.bristol.ac.uk/cmm/software/statjr/
For more information visit www.ncrm.ac.uk