Software for Generalized Bayesian Inference for Samples from Exponential Families

An object-oriented R implementation of generalized iLUCK-models

Gero Walter

Department of Statistics
Ludwig-Maximilians-University Munich

February 17th, 2009

Generalized Bayesian Inference – General Idea

Bayesian Inference on some parameter θ:

prior knowledge on θ	+	data x	→	updated knowledge on θ
prior distribution $p(\theta)$	+	likelihood $f(x \mid \theta)$	→	posterior distribution $p(\theta \mid x)$
set of priors	+	likelihood	→	**set of** posteriors

Tractability: use **conjugate** priors $\iff$
choose $p(\theta)$ such that $p(\theta \mid x)$ is from same parametric class
→ update only parameters!

LUCK-models: Single Conjugate Prior

$X \overset{iid}{\sim}$ *linear, canonical exponential familiy*, i.e.

$$p(x \mid \theta) \propto \exp\left\{\langle\psi, \tau(x)\rangle - n\mathbf{b}(\psi)\right\} \qquad \left[\psi \text{ transformation of } \theta\right]$$

→ conjugate prior:

$$p(\theta) \propto \exp\left\{n^{(0)}\left[\langle\psi, y^{(0)}\rangle - \mathbf{b}(\psi)\right]\right\}$$

→ (conjugate) posterior:

$$p(\theta \mid x) \propto \exp\left\{n^{(1)}\left[\langle\psi, y^{(1)}\rangle - \mathbf{b}(\psi)\right]\right\}$$

$$y^{(1)} = \frac{n^{(0)}y^{(0)} + \tau(x)}{n^{(0)} + n} \qquad \text{and} \qquad n^{(1)} = n^{(0)} + n\,.$$

LUCK-models: Interpretation of $y^{(0)}$ and $n^{(0)}$

$y^{(0)}$: "main prior parameter"

- for samples from a $\mathrm{N}(\mu, 1)$, $p(\mu)$ is a $\mathrm{N}(y^{(0)}, \frac{1}{n^{(0)}})$
- for samples from a $\mathrm{M}(\boldsymbol{\theta})$, $p(\boldsymbol{\theta})$ is a $\mathrm{Dir}(n^{(0)}, y^{(0)})$
  ($y_j^{(0)} = t_j \widehat{=}$ prior probability for class j, $n^{(0)} = s$)

$n^{(0)}$: "prior strength" or "pseudocounts"

with $\tilde{\tau}(x) =: \frac{1}{n}\tau(x)$: $\quad [\ \tau(x) = \sum_{i=1}^{n} \tau(x_i)\ ]$

$$y^{(1)} = \frac{n^{(0)}}{n^{(0)} + n} \cdot y^{(0)} + \frac{n}{n^{(0)} + n} \cdot \tilde{\tau}(x) \,.$$

sets of LUCK-models – iLUCK-model

iLUCK-model: vary $y^{(0)}$ in $\mathcal{Y}^{(0)}$ $\quad$ [$\mathcal{Y}^{(0)}$ convex] $\Longleftrightarrow$
allow for ambiguity on the main prior parameter

- prior credal set contains *all finite convex mixtures* of $p(\theta)$s with $y^{(0)} \in \mathcal{Y}^{(0)}$
- posterior credal set easy to calculate:
  all finite convex mixtures of $p(\theta \mid x)$s with

$$y^{(1)} \in \mathcal{Y}^{(1)} = \frac{n^{(0)}}{n^{(0)} + n} \cdot \mathcal{Y}^{(0)} + \frac{n}{n^{(0)} + n} \cdot \tilde{\tau}(x)$$

$\boxed{\text{⚠ unfavourable behavior in case of prior–data conflict! ⚠}}$

sets of LUCK-models – Generalized iLUCK-model

generalized iLUCK-model:

vary $y^{(0)}$ in $\mathcal{Y}^{(0)}$ **and** $n^{(0)}$ in $\mathcal{N}^{(0)}$ $\iff$
weigh prior information $\mathcal{Y}^{(0)}$ and sample information $\tilde{\tau}(x)$ in

$$y^{(1)} \in \mathcal{Y}^{(1)} = \frac{n^{(0)}}{n^{(0)} + n} \cdot \mathcal{Y}^{(0)} + \frac{n}{n^{(0)} + n} \cdot \tilde{\tau}(x)$$

more flexible!

- prior credal set contains *all finite convex mixtures* of $p(\theta)$s with $y^{(0)} \in \mathcal{Y}^{(0)}$ **and** $n^{(0)} \in \mathcal{N}^{(0)}$
- posterior credal set still quite easy to calculate: *all finite convex mixtures* of $p(\theta \mid x)$s with

$$\left\{ \left(n^{(1)}, y^{(1)}\right) \middle| \, n^{(1)} = n^{(0)} + n,\ y^{(1)} = \frac{n^{(0)} y^{(0)} + \tau(x)}{n^{(0)} + n},\ n^{(0)} \in \mathcal{N}^{(0)},\ y^{(0)} \in \mathcal{Y}^{(0)} \right\}$$

Demonstration

The R project for Statistical Computing

- not just a (statistical) software package,
  rather a full-grown programming language
- open source implementation of the (award-winning) S language
- extremely widespread in universitary research
  (reference implementation of new methods are often in R)
- extensions providing additional functionality can be made readily available as "packages"
- can be linked with LaTeX (included package Sweave)
- can be used as imperative or as object-oriented language

Imperative vs. Object-oriented Programming

imperative: do this, then that

➥ functions (on arguments)

object-oriented: create 'objects', do things with them

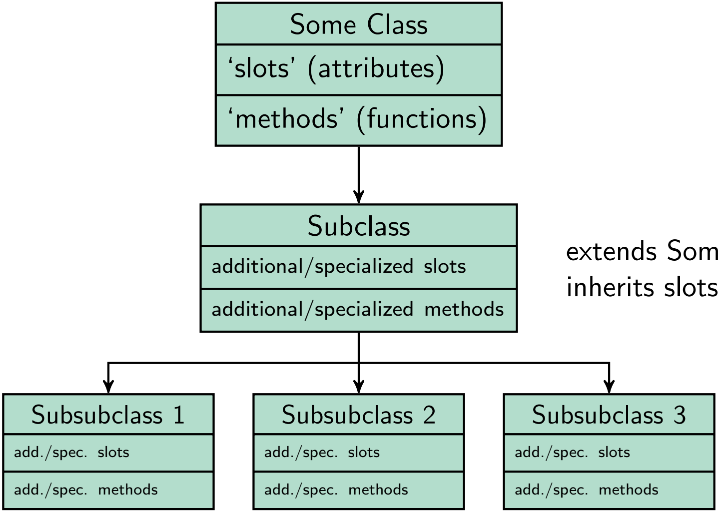
➥ blueprints for objects called 'classes'

objects created according to a blueprint are called an 'instance'

example:
banking company administrating their customers' accounts

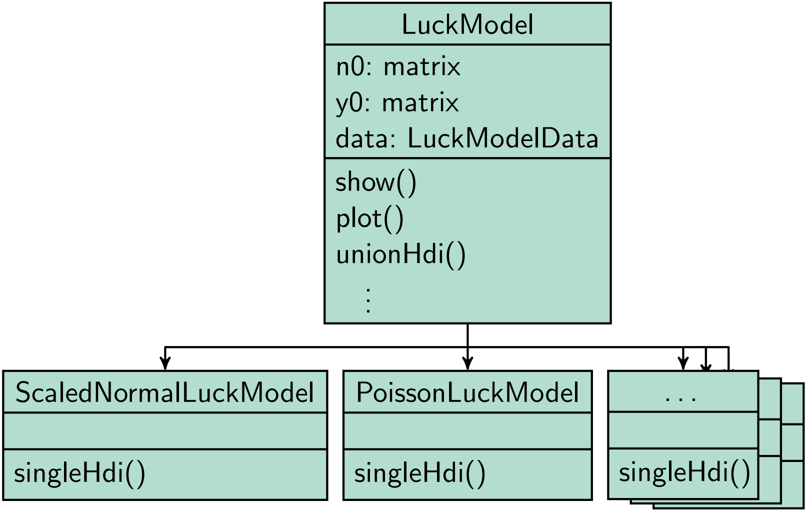
class:	`BankAccount`
instances:	bank account for customer A
	bank account for customer B
	⋮

Object-oriented Programming: Class hierarchies



extends Some Class
inherits slots & methods

Implementation – Class Structure



Implementation – Class Structure