Bayesian SAE using Complex Survey Data

Lecture 3B: Hierarchical Bayes Modeling in R

Richard Li

Department of Statistics
University of Washington

Review

In lecture 2, we showed Bayesian estimation of the following two models:

- Let y_i and n_i denote the number of individuals having type II diabetes and the number of samples in the i-th area.
- Let w_{ij} denote the weight of the j-th individual in the i-th area.
- Our model assumes for $i = 1, ..., n$

$$y_i | p_i \sim \text{Binomial}(n_i, p_i), \quad p_i \sim \text{Beta}(a, b)$$

and

$$w_{ij} | \mu \sim \text{Normal}(\mu_i, \sigma^2), \quad \mu_i \sim \text{Normal}(\mu_0, \sigma_0^2)$$

- We assumed we know σ^2 and fix a, b, μ_0, and σ_0^2.
- These simple models assume no between-area smoothing.

Hierarchical models

In this lecture, we show analysis of hierarchical models that allows smoothing over areas. We will implement three models:

- beta-binomial model
  - full Bayes (supplement)
- normal-normal model
  - INLA
  - full Bayes (supplement)
- binomial-normal model
  - INLA

Outline

Beta binomial model

Normal hierarchical model

Binomial-normal model and INLA

Additional topics: Beta-binomial model via Metropolis Hastings algorithm

Additional topics: Normal hierarchical model using Gibbs sampler

Beta binomial model

Beta-binomial model

Recall the model in Lecture 2

$$y_i|p_i \sim \text{Binomial}(n_i, p_i) \quad \theta_i \sim \text{Beta}(\alpha, \beta)$$

Previously we implemented this model assuming we know α and β. Now we estimate α and β with empirical Bayes approach.

Beta binomial model: full Bayes

For a fully Bayesian treatment, we use the prior from Gelman et al. (Section 5.3)

$$p(\alpha, \beta) \propto (\alpha + \beta)^{-5/2}$$

This leads to the joint distribution of y_i, p_i, a, b to be

$$p(\boldsymbol{y}, \boldsymbol{p}, \alpha, \beta) = \propto (\alpha+\beta)^{-5/2} \prod_i \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p_i^{\alpha-1}(1-p_i)^{\beta-1} \binom{n_i}{y_i} p_i^{y_i}(1-p_i)^{n_i-y_i}$$

Unfortunately, the conditional distributions of α and β are not in closed form and needs to be sampled with Metropolis-Hasting algorithm. See the last section for an example.

Normal hierarchical model

Normal hierarchical model

Let's consider the following hierarchical model (Notice we change the notation as model gets more complicated)

$$\begin{aligned} y_{ij}|\theta_i, \sigma^2 &\sim \text{Normal}\left(\theta_i, \sigma^2\right) \\ \theta_i|\mu, \tau^2 &\sim \text{Normal}\left(\mu, \tau^2\right) \end{aligned}$$

And we use the following 'default' priors on the unknown parameters

$$\begin{aligned} \mu &\sim \text{Normal}\left(\mu_0, \gamma_0^2\right) \\ \sigma^2 &\sim \text{InvGamma}\left(\nu_0/2, \nu_0\sigma_0^2/2\right) \\ \tau^2 &\sim \text{InvGamma}\left(\eta_0/2, \eta_0\tau_0^2/2\right) \end{aligned}$$

The hyperparameters we need to specify are

- μ_0 and τ_0^2 are the prior guess at the μ and the certainty of this guess.
- σ_0 and ν_0 are the prior guess at the within-area variance σ^2 and the certainty of this guess.
- τ_0 and η_0 are the prior guess at the across-area variance τ and the certainty of this guess.

Integrated nested Laplace approximation (INLA)

INLA is a computationally convenient alternative to MCMC that uses Laplace approximation and numerical integration very efficiently (no sampling is needed). We will be mostly using INLA to estimate hierarchical models from now on.
The homepage of INLA software is:

`http://www.r-inla.org/home`

There are many useful materials of INLA on its homepage. For beginners, there are several books available:

`http://www.r-inla.org/books`

To install,

```
install.packages("INLA", repos = c(getOption("repos"),
    INLA = "https://inla.r-inla-download.org/R/stable"),
    dep = TRUE)
```

Hierarchical normal : MLE

Load data and sample $2,000$ observations

```
load(url("http://faculty.washington.edu/jonno/PAA-SAE/simKing.rda"))
```

Or from the downloaded local file:

```
load("../data/simKing.rda")
set.seed(1)
samp <- pop[sample(1:dim(pop)[1], 2000), ]
regionnames <- samp[match(c(1:48), samp[,"area"]), "areaname"]
samp <- data.frame(index=samp$area,  value=samp$weight)
```

Recall the calculation of MLE for each individual area without smoothing

```
theta.true <- aggregate(weight ~ area, pop, mean)[,
    2]
theta.naive <- aggregate(value ~ index, samp, mean)[,
    2]
sd.naive <- sqrt(aggregate(value ~ index, samp, var)[,
    2]/aggregate(value ~ index, samp, length)[, 2])
```

Fitting hierarchical normal model

We put weekly informative priors, invGamma$(1, 0.05)$ on σ^2 and invGamma$(1, 0.01)$ on τ^2, and a very flat default prior on μ.

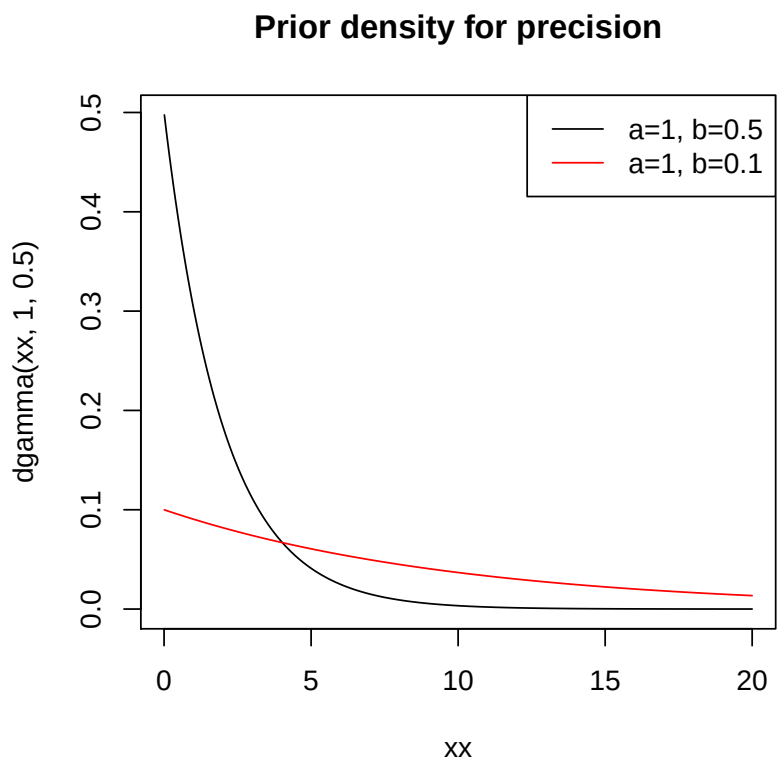
```
library(INLA)
newx <- data.frame(value=NA, index = 1:48)
hyperprior1 <- list(theta=list(prior='loggamma',
        param=c(1, 0.5)))
hyperprior2 <- list(theta=list(prior='loggamma',
        param=c(1, 0.1)))
formula <- value ~ 1+f(index, model="iid",
        hyper = hyperprior2)
fit <- inla(formula,
        data=rbind(newx, samp), family = "gaussian",
        control.family = list(hyper = hyperprior1),
        control.predictor = list(compute = TRUE))
```

The `compute=TRUE` within `control.predictor` asks inla to compute the fitted value for each observations. We included new data with NA weights at the beginning of the data frame, so that we can obtain predicted values for each region. *This is not very efficient; faster alternatively exist.*

Fitting hierarchical normal model

Prior density illustration

```
xx <- seq(0.01, 20, len = 1000)
plot(xx, dgamma(xx, 1, 0.5), type = "l",
        main="Prior density for precision")
lines(xx, dgamma(xx, 1, 0.1), col = 2)
legend("topright", c("a=1, b=0.5", "a=1, b=0.1"),
        col=c(1,2), lty=c(1,1))
```



Fitting hierarchical normal model: model summary

```
summary(fit)
```

```
##
## Call:
## c("inla(formula = formula, family = \"gaussian\", data = rbind(newx, ",  "    samp), co
##
## Time used:
##  Pre-processing    Running inla Post-processing           Total
##          1.2932         13.7509          0.9232         15.9673
##
## Fixed effects:
##                 mean     sd 0.025quant 0.5quant 0.975quant     mode kld
## (Intercept) 179.9324 0.8182   178.3143  179.934   181.5398 179.9372   0
##
## Random effects:
## Name   Model
##  index   IID model
##
## Model hyperparameters:
##                                             mean     sd 0.025quant 0.5quant 0.975quant
## Precision for the Gaussian observations  0.0024 0.0001     0.0022   0.0024     0.0025
## Precision for index                     16.6845 9.7291     5.4780  14.2568    42.0439
##                                             mode
## Precision for the Gaussian observations  0.0024
## Precision for index                     10.7436
##
## Expected number of effective parameters(std dev): 31.36(3.276)
## Number of equivalent replicates : 63.78
##
```

Understand the INLA object

A few useful fields in an INLA fitted object

- `marginals.fixed`, `marginals.random`, `marginals.hyperpar`: the densities of fixed effect, random effect, hyperparameters.
- `summary.linear.predictor` fitted values corresponding to each row of the input data.
- `summary.lincomb` we did not use here, but it can be useful to calculate linear combination of the parameters when the model gets more complicated.

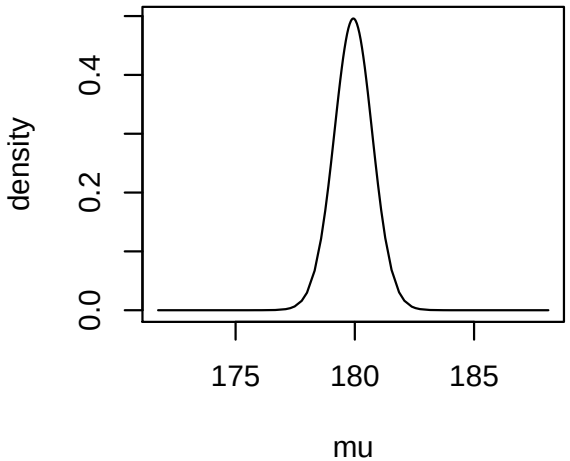
Some particular useful functions in working with INLA objects are `inla.rmarginal, inla.tmarginal`, the former draws random variables from an estimated density, and the latter calculates the density on transformed variables. We will see how to use them later. `inla.hyperpar()` command improves the estimation of the hyperparameters using grid integration strategy

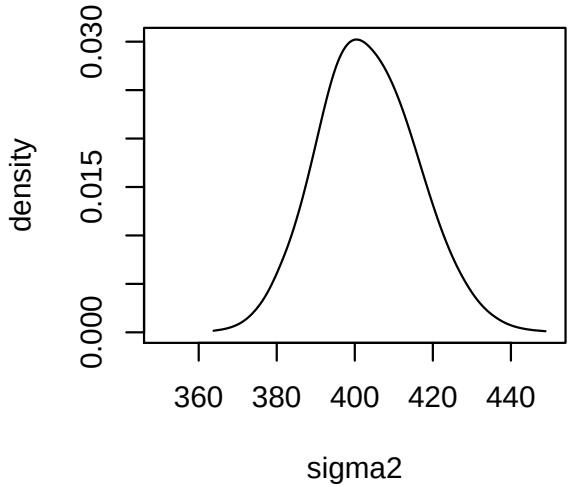
```
hyper <- inla.hyperpar(fit)
```

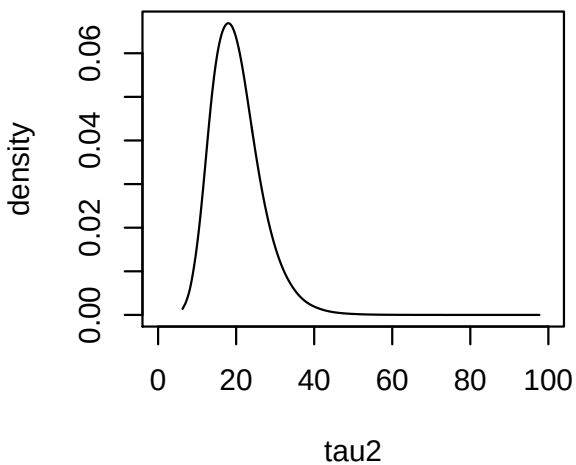
Posterior summary: hyperparameters

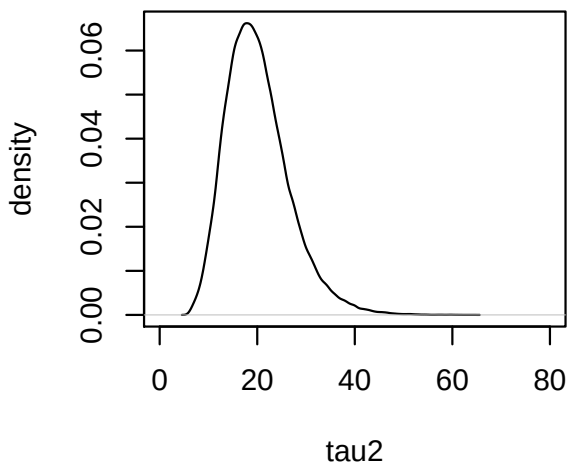
```
par(mfrow = c(2, 2))
plot(fit$marginals.fixed[[1]], type = "l", xlab = "mu",
    ylab = "density")
plot(inla.tmarginal(function(x) 1/x, hyper$marginals.hyperpar[[1]]),
    type = "l", xlim = c(350, 450), xlab = "sigma2",
    ylab = "density")
plot(inla.tmarginal(function(x) 1/x, hyper$marginals.hyperpar[[2]]),
    type = "l", xlim = c(0, 100), xlab = "tau2", ylab = "density")
## alternative estimation of transformed parameter
plot(density(inla.rmarginal(1e+05, inla.tmarginal(function(x) 1/x,
    hyper$marginals.hyperpar[[2]]))), type = "l", xlim = c(0,
    80), xlab = "tau2", ylab = "density", main = "")
```

Posterior summary: hyperparameters



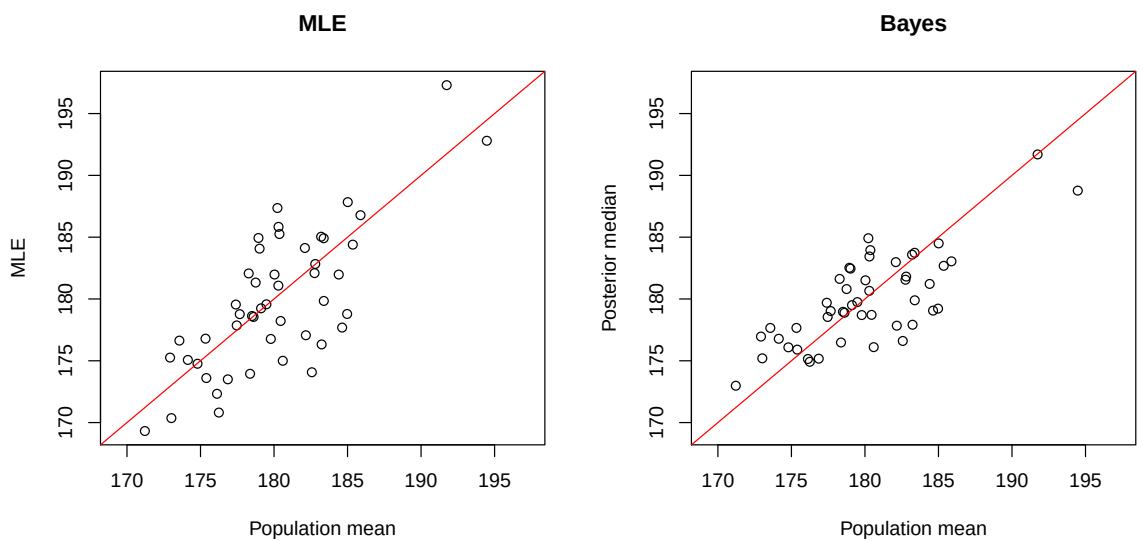






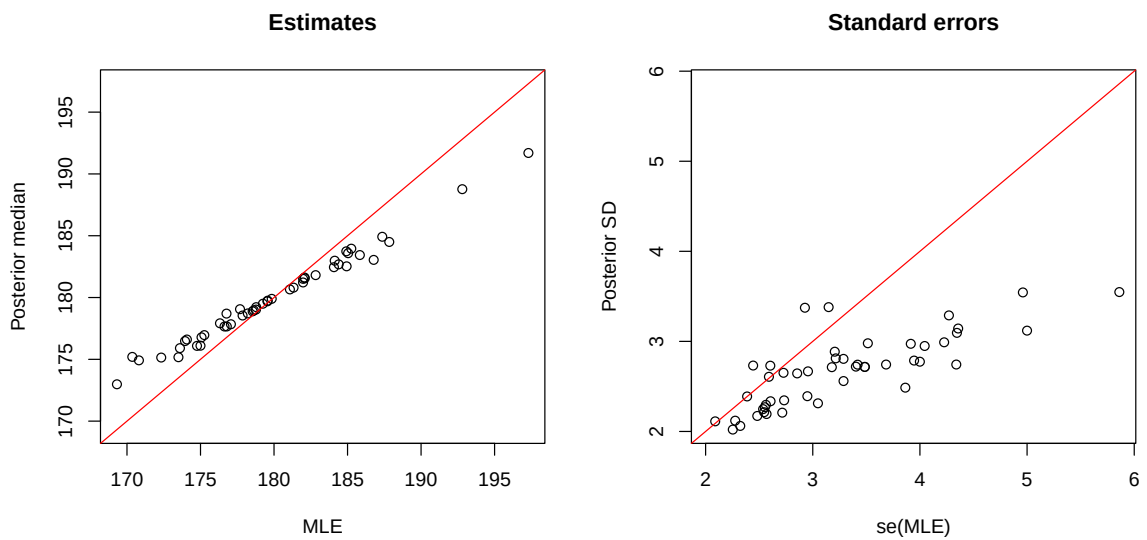
Compare INLA and MLE: estimates

```
theta.median <- fit$summary.linear.predictor[1:48,
    "0.5quant"]
lim <- range(c(theta.true, theta.naive, theta.median))
par(mfrow = c(1, 2))
plot(theta.true, theta.naive, xlim = lim, ylim = lim,
    xlab = "Population mean", ylab = "MLE", main = "MLE")
abline(c(0, 1), col = "red")
plot(theta.true, theta.median, xlim = lim, ylim = lim,
    xlab = "Population mean", ylab = "Posterior median",
    main = "Bayes")
abline(c(0, 1), col = "red")
```



Compare INLA and MLE: uncertainty

```
theta.sd <- fit$summary.linear.predictor[1:48, "sd"]
par(mfrow = c(1, 2))
lim1 <- range(c(theta.naive, theta.median))
plot(theta.naive, theta.median, xlim = lim1, ylim = lim1,
    xlab = "MLE", ylab = "Posterior median", main = "Estimates")
abline(c(0, 1), col = "red")
lim2 <- range(c(theta.sd, sd.naive))
plot(sd.naive, theta.sd, xlim = lim2, ylim = lim2,
    xlab = "se(MLE)", ylab = "Posterior SD", main = "Standard errors")
abline(c(0, 1), col = "red")
```



Read map files into R

Download the shapefiles from `http://faculty.washington.edu/jonno/PAA-SAE/HRA_ShapFiles/`
`maptools` package provide the easiest way to read in shapefile. However, it does not load in the spatial referencing information.

```
# install.packages('maptools')
library(maptools)
f <- "../data/HRA_ShapeFiles/HRA_2010Block_Clip.shp"
kingshape <- readShapePoly(f)
```

`rgdal` package provides more powerful ways to read in geographical data. But, additional steps to install GDAL is required.

```
# install.packages('rgdal')
library(rgdal)
kingshape <- readOGR("../data/HRA_ShapeFiles", layer = "HRA_2010Block_Clip")

## OGR data source with driver: ESRI Shapefile
## Source: "../data/HRA_ShapeFiles", layer: "HRA_2010Block_Clip"
## with 48 features
## It has 9 fields
```

Compare INLA and MLE on the map

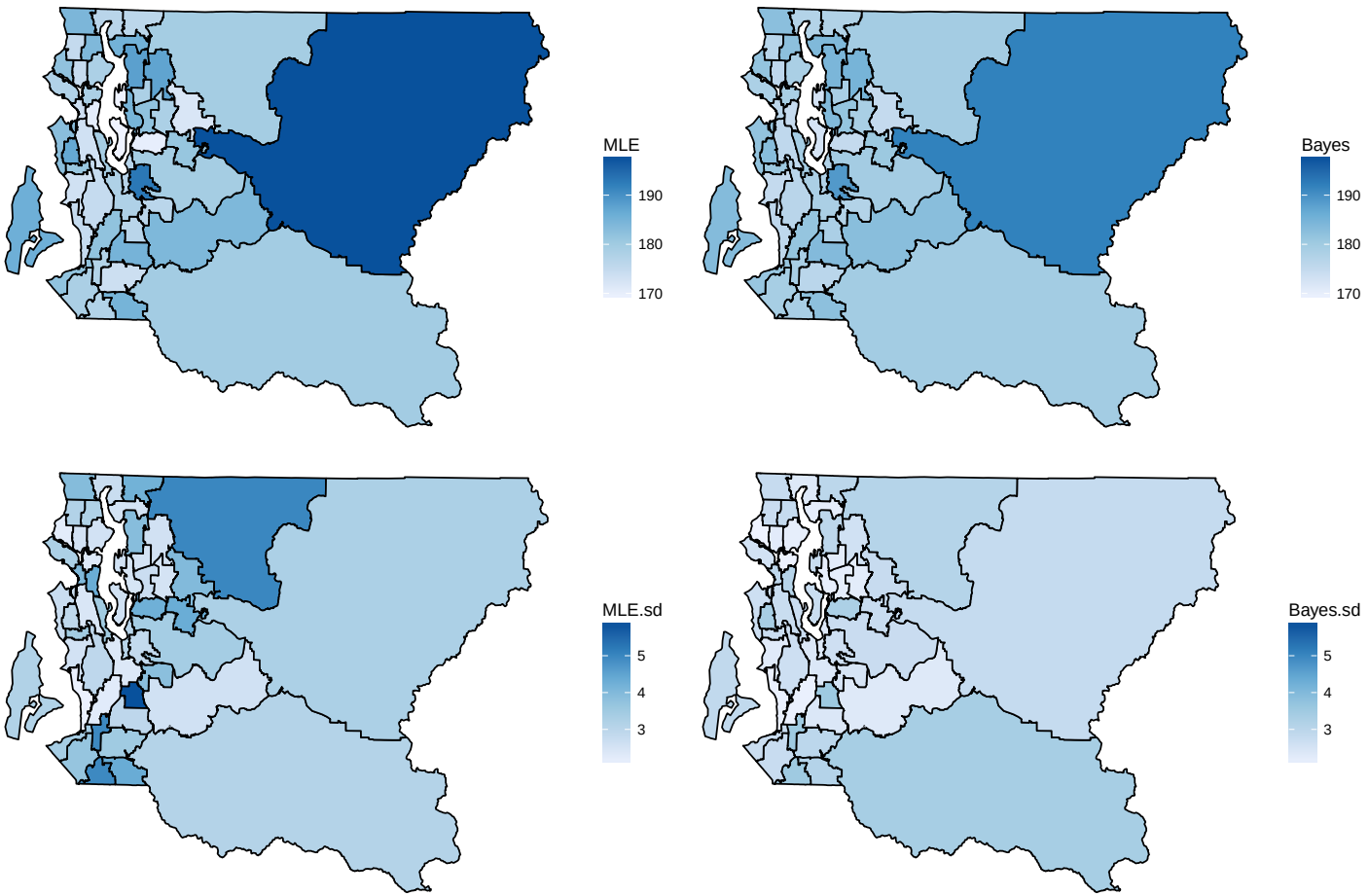
```
compare <- data.frame(MLE=theta.naive,
        Bayes=theta.median, MLE.sd=sd.naive,
        Bayes.sd=theta.sd, Region=regionnames)
library(ggplot2)
geo <- fortify(kingshape, region = "HRA2010v2_")
geo1 <- merge(geo, compare, by = "id", by.y = "Region")
g1 <- ggplot(geo1)
g1 <- g1 + geom_polygon(aes(x=long, y=lat, group = group,
        fill=MLE), color="black")
g1 <- g1 + scale_fill_distiller(direction=1, limits=lim1)
g1 <- g1 + theme_void()
```

Compare INLA and MLE on the map

```
# Bayes estimates
g2 <- ggplot(geo1)
g2 <- g2 + geom_polygon(aes(x = long, y = lat, group = group,
    fill = Bayes), color = "black")
g2 <- g2 + theme_void() + scale_fill_distiller(direction = 1,
    limits = lim1)
# MLE SE
g3 <- ggplot(geo1)
g3 <- g3 + geom_polygon(aes(x = long, y = lat, group = group,
    fill = MLE.sd), color = "black")
g3 <- g3 + theme_void() + scale_fill_distiller(direction = 1,
    limits = lim2)
# Bayes SD
g4 <- ggplot(geo1)
g4 <- g4 + geom_polygon(aes(x = long, y = lat, group = group,
    fill = Bayes.sd), color = "black")
g4 <- g4 + theme_void() + scale_fill_distiller(direction = 1,
    limits = lim2)
```

Compare INLA and MLE on the map

```
library(gridExtra)
grid.arrange(grobs = list(g1, g2, g3, g4), ncol = 2)
```



Binomial-normal model and INLA

Binomial-normal model

A more common framework to model binomial data is through *generalized linear mixed model*. We assume

$$\begin{aligned} \log\left(\frac{p_i}{1-p_i}\right) &= \mu + \epsilon_i \\ \mu|\sigma_\mu &\sim \text{Normal}(0, \sigma_\mu^2) \\ \epsilon_i|\sigma_\epsilon^2 &\sim \text{Normal}(0, \sigma_\epsilon^2), \quad i = 1, ..., n \\ \sigma_\epsilon^2 &\sim \text{InvGamma}(a, b) \end{aligned}$$

where hyperparameters (σ_μ^2, a, b) are fixed.
When area-level covariates are available, this can be extended to $\text{logit}(p_i) = \mu + x_i^T \beta + \epsilon_i$.

Binomial-normal model: Data

We again take 2,000 samples from the simulated dataset, and calculate the number of diabetes in each area, along with the sample size in each area.

```
set.seed(1)
samp <- pop[sample(1:dim(pop)[1], 2000), ]
samp <- data.frame(index = samp$area, value = samp$diabetes)
samp.aggre <- aggregate(value ~ index, samp, sum)
samp.aggre$n <- aggregate(value ~ index, samp, length)[,
    2]
prev <- data.frame(truth = aggregate(diabetes ~ area,
    pop, mean)[, 2], mle = aggregate(value ~ index,
    samp, mean)[, 2], size = aggregate(value ~ index,
    samp, length)[, 2])
prev$mle.sd <- sqrt(prev$mle * (1 - prev$mle)/prev$size)
prev$mle.sd[prev$mle.sd == 0] <- NA
```

Binomial-normal model: Data

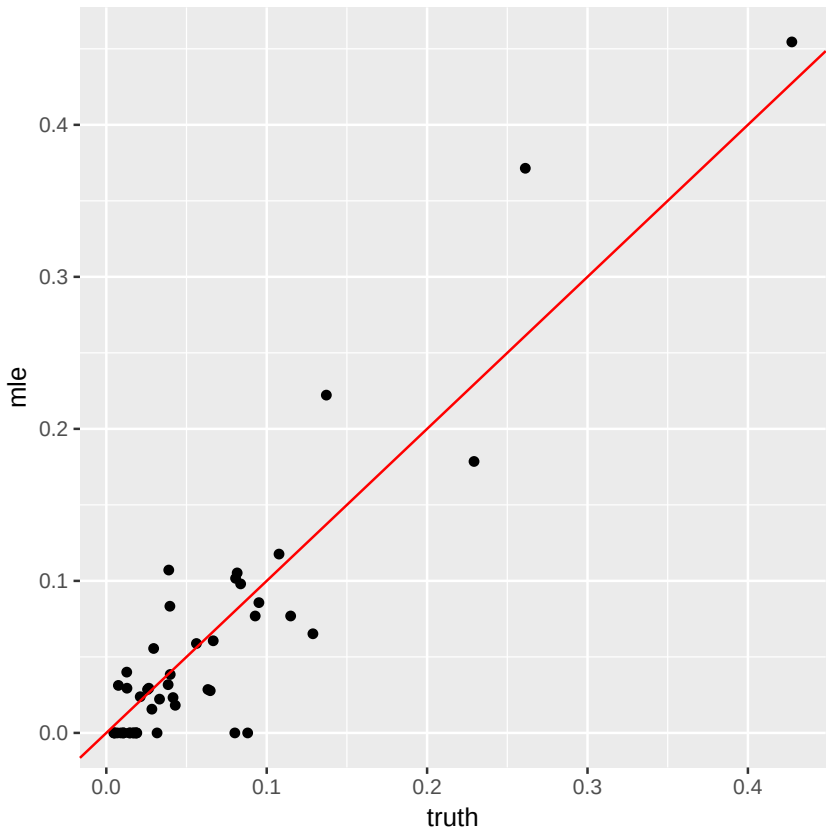
We also calculate the logit of the maximum likelihood estimators, and their asymptotic standard errors (using delta method, more on this later):

$$se(log(\frac{\hat{p}}{1-\hat{p}})) = \frac{\widehat{se}(\hat{p})}{\hat{p}(1-\hat{p})}$$

```
prev$mle.logit <- log(prev$mle/(1 - prev$mle))
prev$mle.logit[prev$mle.logit == -Inf] <- NA
prev$mle.logit.sd <- prev$mle.sd/(prev$mle * (1 - prev$mle))
prev$mle.logit.sd[is.nan(prev$mle.logit.sd)] <- NA
```

Binomial-normal model: MLE

```
library(ggplot2)
g <- ggplot(prev, aes(x = truth, y = mle))
g <- g + geom_point() + geom_abline(color = "red")
g
```



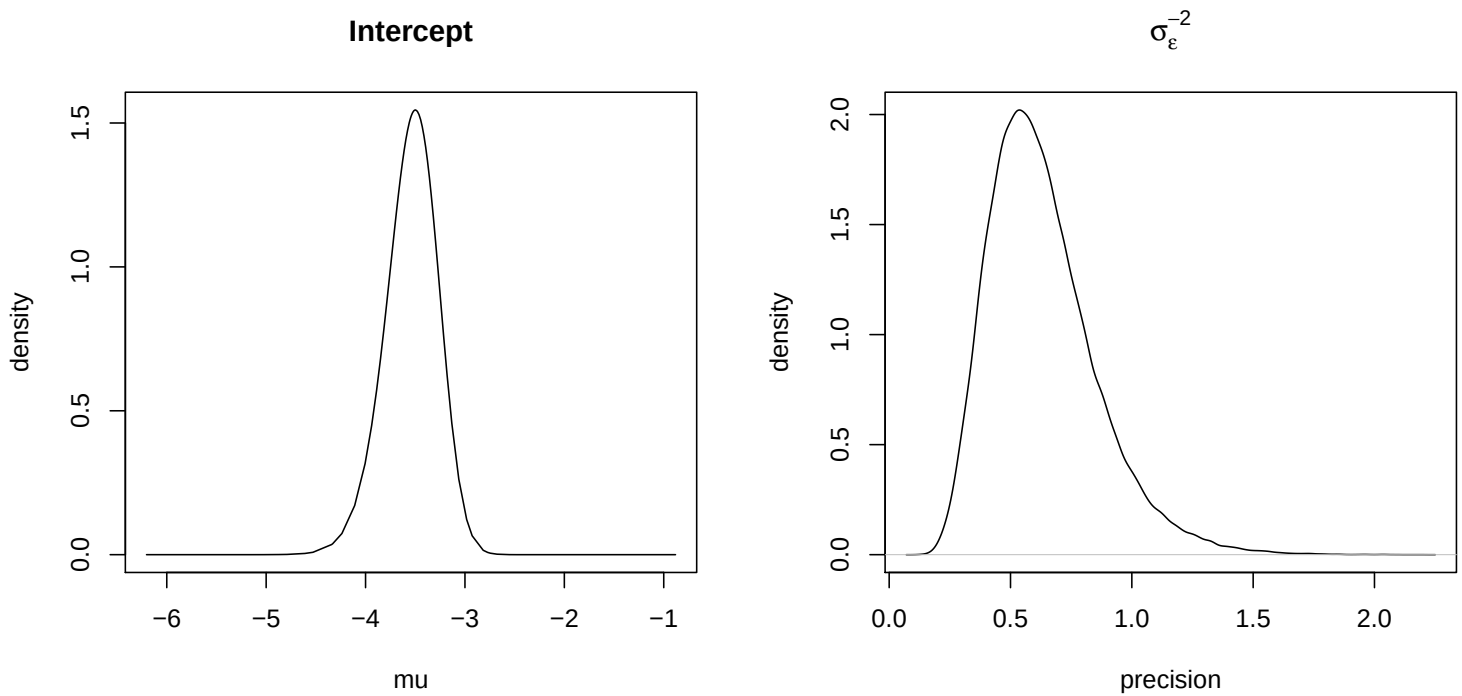
Binomial-normal model: INLA

Similar to before, we use INLA to estimate the model with default prior choices.

```
formula = value ~ 1 +
        f(index, model='iid',
                hyper=list(theta=list(prior='loggamma',
                                      param=c(0.5, 0.01))))
fit3 <- inla(formula,
        family="binomial",
        data=samp.aggre, Ntrials=n,
        control.predictor = list(compute = TRUE))
```

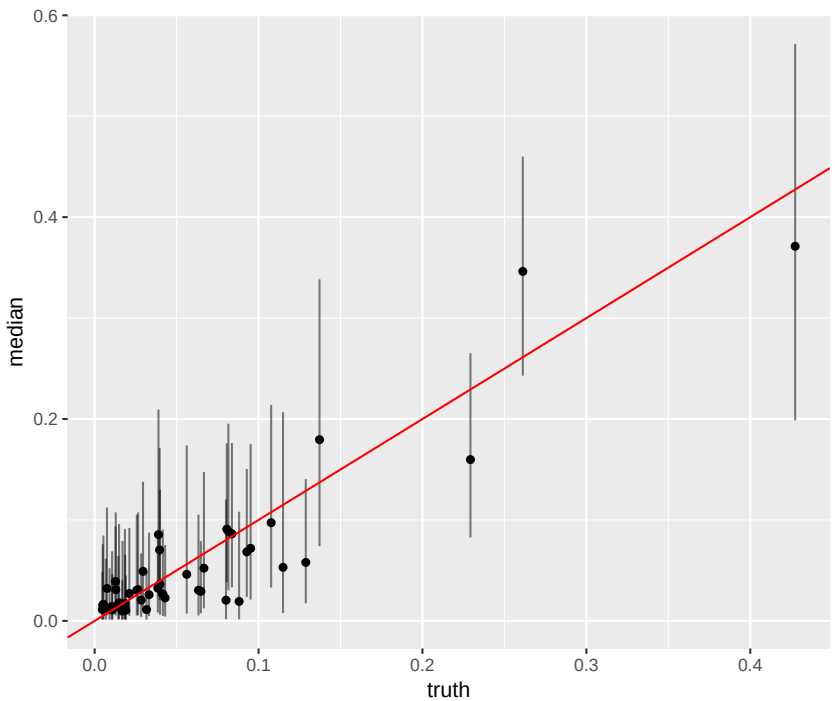
Binomial-normal model: INLA

```
par(mfrow = c(1, 2))
plot(fit3$marginals.fixed[[1]], type = "l", xlab = "mu",
    ylab = "density", main = "Intercept")
plot(density(inla.rmarginal(1e+05, fit3$marginals.hyperpar[[1]])),
    type = "l", xlab = "precision", ylab = "density",
    main = expression(sigma[epsilon]^-2))
```



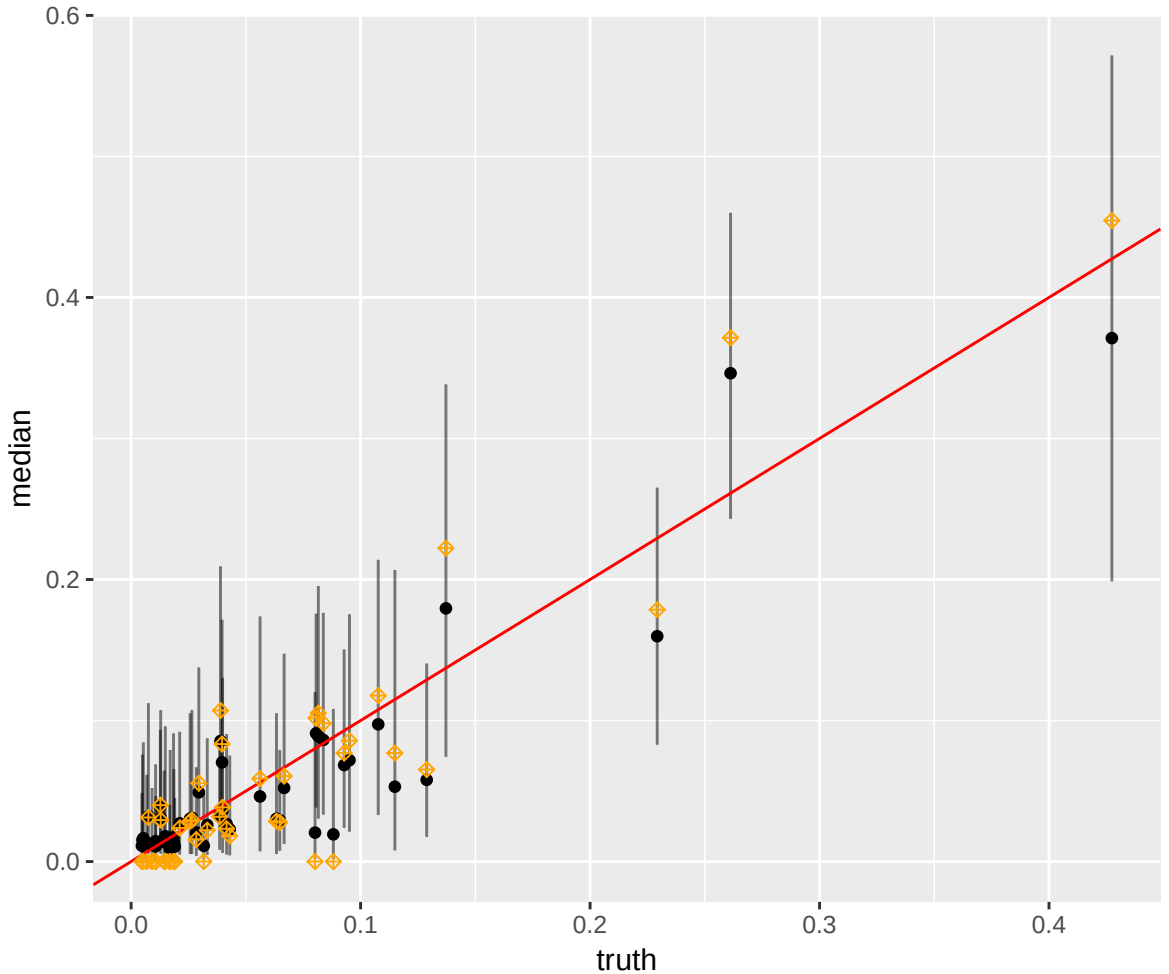
Binomial-normal model: Comapare

```
prev <- cbind(prev, fit3$summary.fitted.values)
colnames(prev)[9:11] <- c("lower", "median", "upper")
g <- ggplot(prev, aes(x = truth, y = median, ymin = lower,
    ymax = upper))
g <- g + geom_point() + geom_errorbar(alpha = 0.5) +
    geom_abline(color = "red")
g
```



Binomial-normal model: Comapare

```
g + geom_point(aes(x = truth, y = mle), color = "orange",
    shape = 9)
```



Binomial-normal model: Maps

Now we merge all the estimates into one data frame

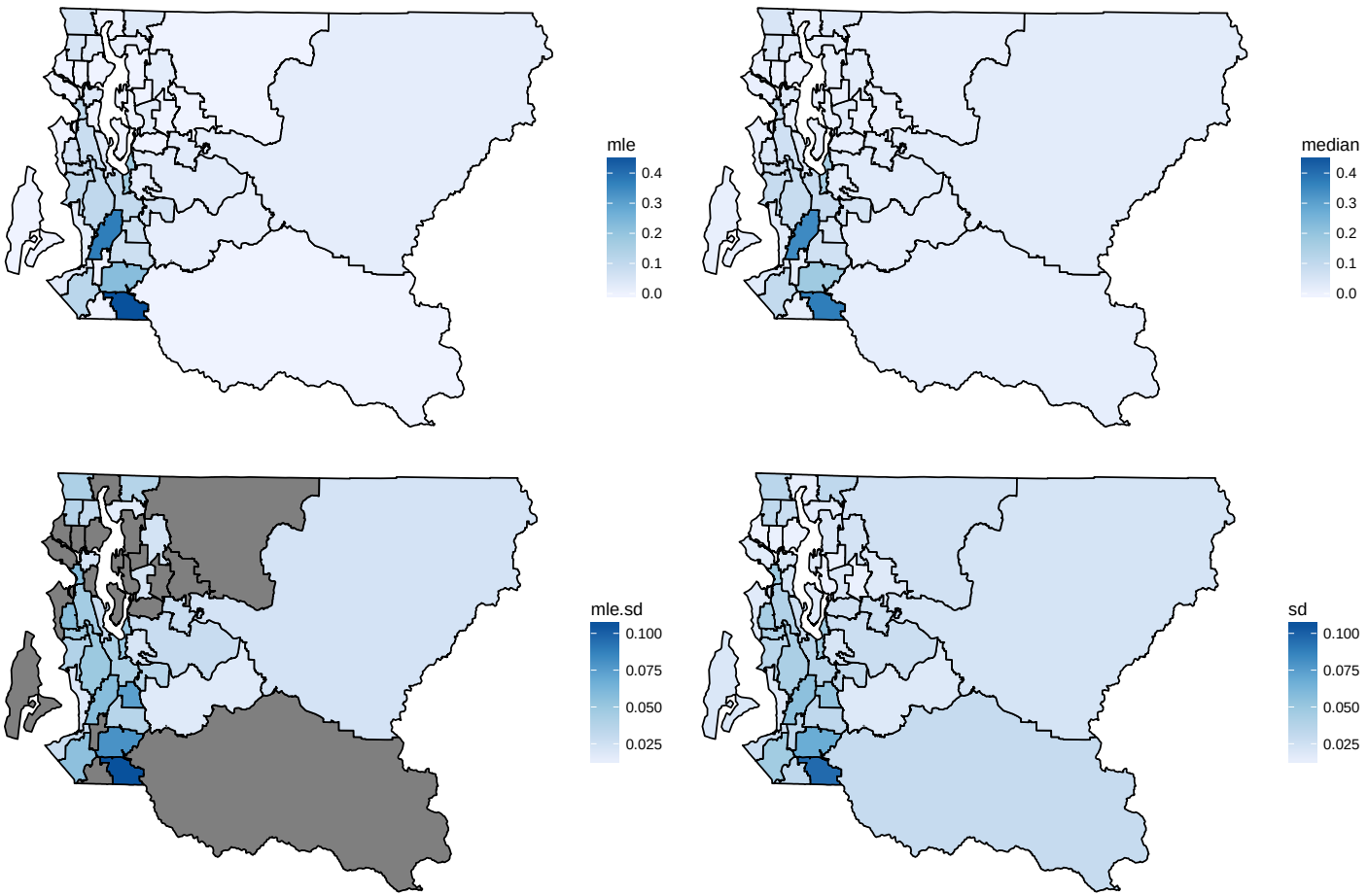
```
prev$logit.median <- fit3$summary.linear.predictor[,
    "0.5quant"]
prev$logit.sd <- fit3$summary.linear.predictor[, "sd"]
prev$Region <- regionnames
geo1 <- merge(geo, prev, by = "id", by.y = "Region")
lim1 <- range(c(prev$mle, prev$median))
lim2 <- range(c(prev$mle.sd, prev$sd), na.rm = T)
lim3 <- range(c(prev$mle.logit, prev$logit.median),
    na.rm = T)
lim4 <- range(c(prev$mle.logit.sd, prev$logit.sd),
    na.rm = T)
```

Binomial-normal model: Maps

```
g1 <- g2 <- g3 <- g4 <- ggplot(geo1) + theme_void()
# MLE
g1 <- g1 + geom_polygon(aes(x = long, y = lat, group = group,
    fill = mle), color = "black") + scale_fill_distiller(direction = 1,
    limits = lim1)
# Bayes estimates
g2 <- g2 + geom_polygon(aes(x = long, y = lat, group = group,
    fill = median), color = "black") + scale_fill_distiller(direction = 1,
    limits = lim1)
# MLE SE
g3 <- g3 + geom_polygon(aes(x = long, y = lat, group = group,
    fill = mle.sd), color = "black") + scale_fill_distiller(direction = 1,
    limits = lim2)
# Bayes SD
g4 <- g4 + geom_polygon(aes(x = long, y = lat, group = group,
    fill = sd), color = "black") + scale_fill_distiller(direction = 1,
    limits = lim2)
```

Binomial-normal model: Maps

```
grid.arrange(grobs = list(g1, g2, g3, g4), ncol = 2)
```

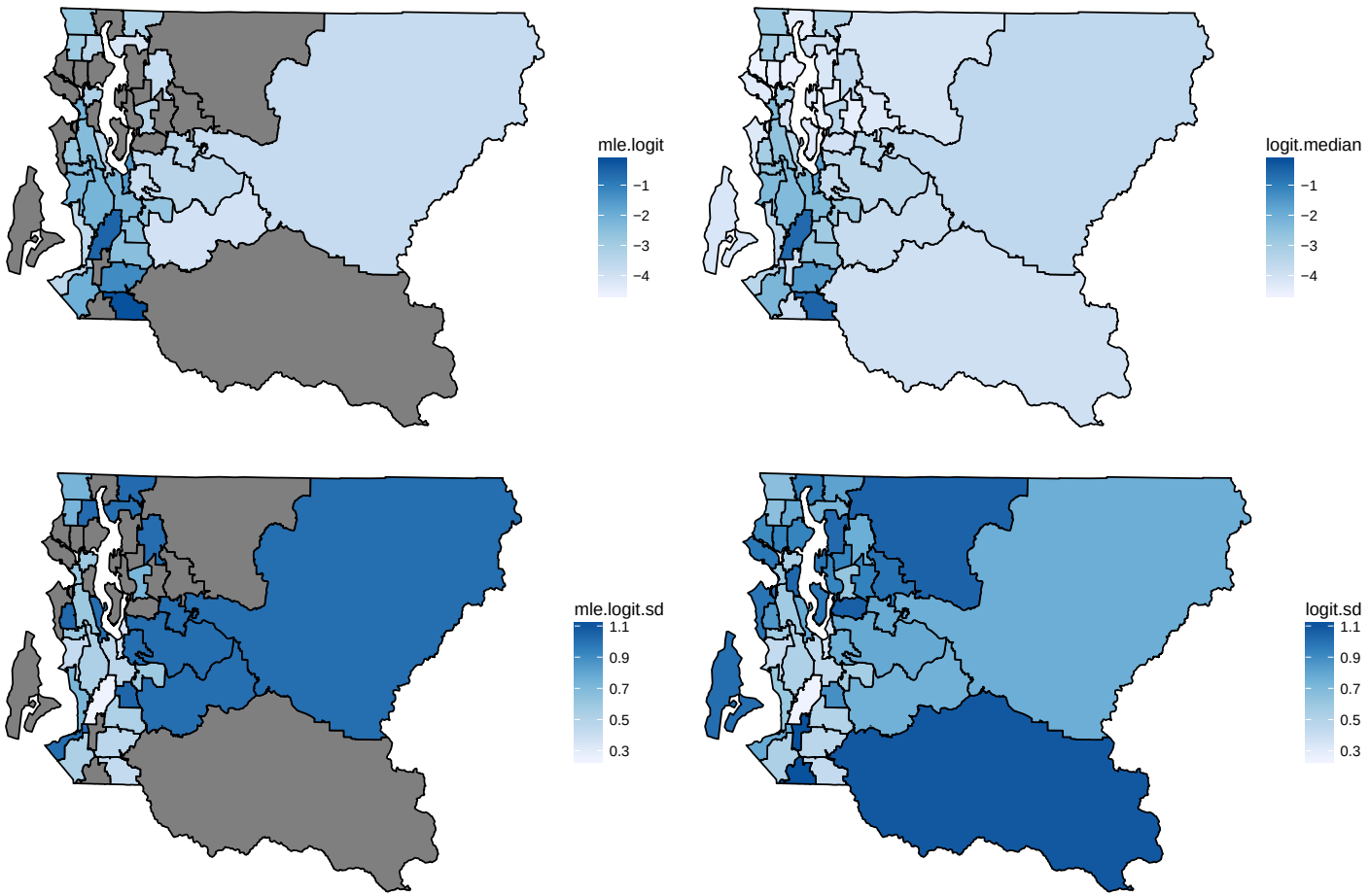


Binomial-normal model: logit scale

```
g1 <- g2 <- g3 <- g4 <- ggplot(geo1) + theme_void()
# MLE logit
g1 <- g1 + geom_polygon(aes(x = long, y = lat, group = group,
    fill = mle.logit), color = "black") + scale_fill_distiller(direction = 1,
    limits = lim3)
# Bayes logit
g2 <- g2 + geom_polygon(aes(x = long, y = lat, group = group,
    fill = logit.median), color = "black") + scale_fill_distiller(direction = 1,
    limits = lim3)
# MLE logit asymptotic SE
g3 <- g3 + geom_polygon(aes(x = long, y = lat, group = group,
    fill = mle.logit.sd), color = "black") + scale_fill_distiller(direction = 1,
    limits = lim4)
# Bayes logit SD
g4 <- g4 + geom_polygon(aes(x = long, y = lat, group = group,
    fill = logit.sd), color = "black") + scale_fill_distiller(direction = 1,
    limits = lim4)
```

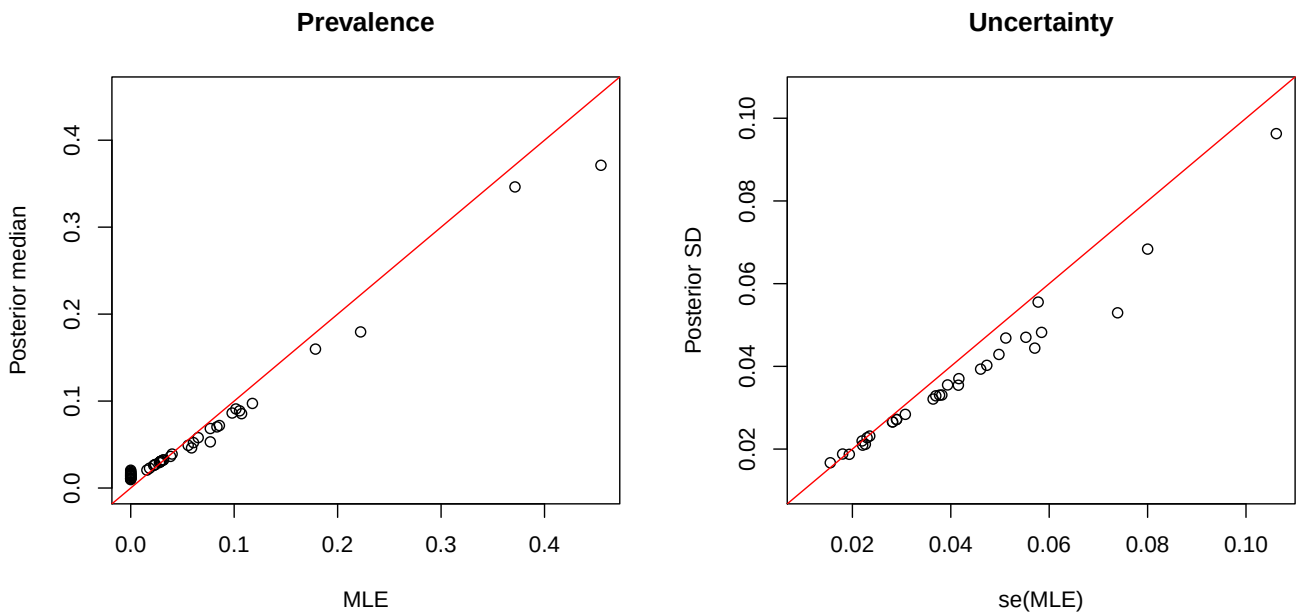
Binomial-normal model: logit scale

```
grid.arrange(grobs = list(g1, g2, g3, g4), ncol = 2)
```



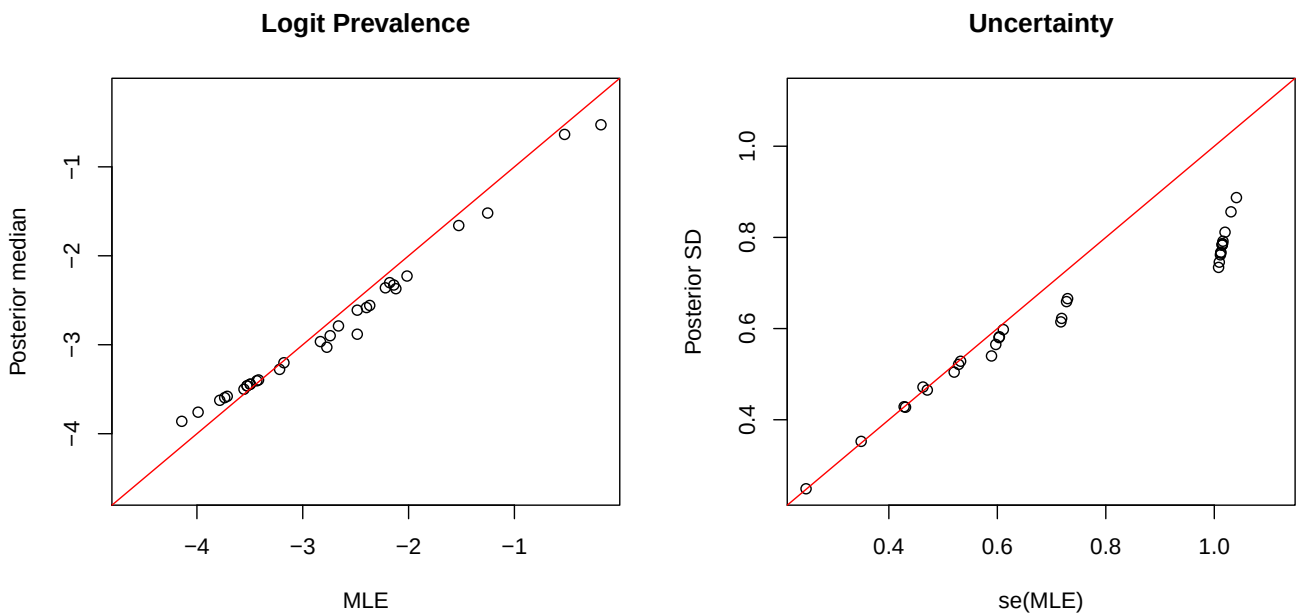
Summary: probability scale

```
par(mfrow = c(1, 2))
plot(prev$mle, prev$median, xlab = "MLE", ylab = "Posterior median",
    main = "Prevalence", xlim = lim1, ylim = lim1)
abline(c(0, 1), col = "red")
plot(prev$mle.sd, prev$sd, xlab = "se(MLE)", ylab = "Posterior SD",
    main = "Uncertainty", xlim = lim2, ylim = lim2)
abline(c(0, 1), col = "red")
```



Summary: logit scale

```
par(mfrow = c(1, 2))
plot(prev$mle.logit, prev$logit.median, xlab = "MLE",
    ylab = "Posterior median", main = "Logit Prevalence",
    xlim = lim3, ylim = lim3)
abline(c(0, 1), col = "red")
plot(prev$mle.logit.sd, prev$logit.sd, xlab = "se(MLE)",
    ylab = "Posterior SD", main = "Uncertainty", xlim = lim4,
    ylim = lim4)
abline(c(0, 1), col = "red")
```



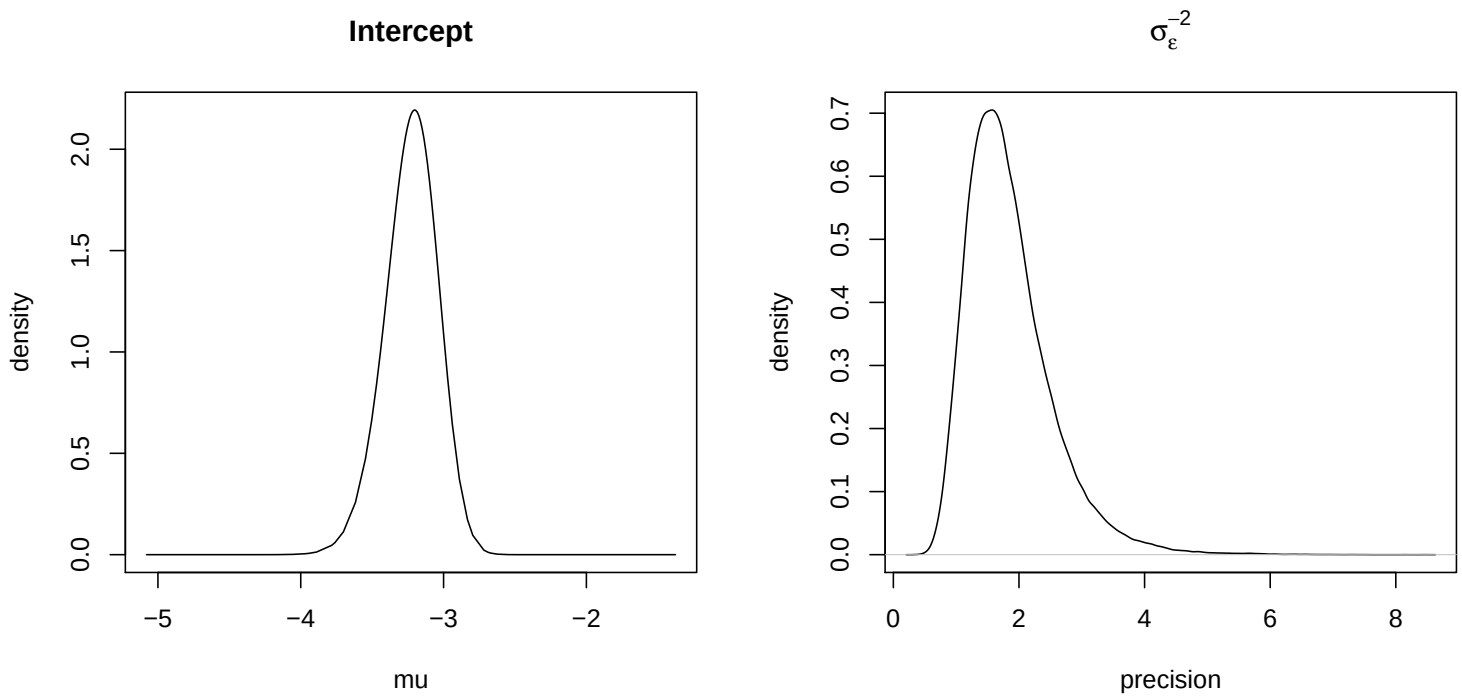
Binomial-normal model: Stronger smoothing

If we want stronger smoothing effect, we may change the hyperpriors on σ_ϵ to center more closely on 0. For a more extreme illustration,

```
formula = value ~ 1 +
        f(index, model='iid',
                hyper=list(theta=list(prior='loggamma',
                                      param=c(10, 0.1))))
fit4 <- inla(formula,
        family="binomial",
        data=samp.aggre, Ntrials=n,
        control.predictor = list(compute = TRUE))
```

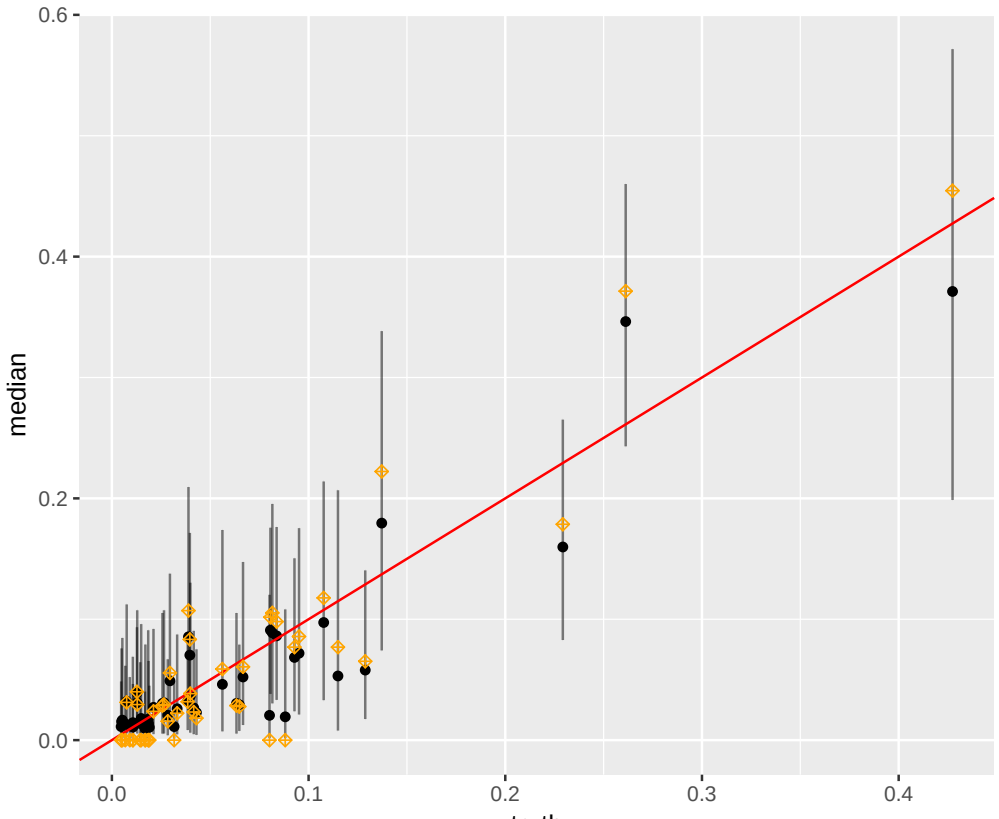
Binomial-normal model: Stronger smoothing

```
par(mfrow = c(1, 2))
plot(fit4$marginals.fixed[[1]], type = "l", xlab = "mu",
    ylab = "density", main = "Intercept")
plot(density(inla.rmarginal(1e+05, fit4$marginals.hyperpar[[1]])),
    type = "l", xlab = "precision", ylab = "density",
    main = expression(sigma[epsilon]^-2))
```



Binomial-normal model: Stronger smoothing

```
prev[3:8] <- fit4$summary.fitted.values
g <- ggplot(prev, aes(x = truth, y = median, ymin = lower,
    ymax = upper))
g <- g + geom_point() + geom_errorbar(alpha = 0.5) +
    geom_abline(color = "red")
g + geom_point(aes(x = truth, y = mle), color = "orange",
    shape = 9)
```



Additional topics: Beta-binomial model via Metropolis Hastings algorithm

Beta-binomial model using Metropolis Hastings algorithm

- Previously we have implemented Gibbs sampler to draw posterior samples from the conditional distributions of the parameters.
- Gibbs sampling requires the conditional distributions of each parameter $\theta_i|\boldsymbol{\theta}_{-i}$ to have a closed form and can be sampled directly. However, in many models, some conditional distributions are only known up to an unknown constant.
- That is, we may be able to write $\pi(\theta) = \frac{1}{Z}\tilde{\pi}(\theta)$, but does not know the normalizing constant Z.
- Here we implement a Metropolis-Hastings algorithm to sample from the Beta-binomial model discussed before.

Metropolis Hastings algorithm

In a nutshell, M-H algorithm samples from the distribution $\pi(\theta) \propto \tilde{\pi}(\theta)$ in the following fashion (with sloppy notation):

1. Start with some initial value θ_0 and a proposal distribution $q(y|x)$. For $t = 1, ..., T$, repeat steps 2 and 3.
2. Generate $\theta^* \sim q(y|\theta^{(t-1)})$.
3. Take $\theta^{(t)} = \theta^*$ with probability p, and keep $\theta^{(t-1)}$ with probability $1 - p$, where

$$p = min(\frac{\tilde{\pi}(\theta^*)q(\theta^{(t-1)}|\theta^*)}{\tilde{\pi}(\theta^{(t-1)})q(\theta^*|\theta^{(t-1)})}, 1)$$

4. The sampled $\theta^{(t)}, t = 1, ..., T$ forms draws from $\pi(\theta)$.

For a more detailed self-contained introduction, see Robert (2016): `https://arxiv.org/pdf/1504.01896.pdf`

Beta-binomial model

The following algorithm is adapted from the example in `http://vnminin.github.io/SISMID_MCMC_I/`. We use the prior $p(\alpha, \beta) \propto (\alpha + \beta)^{-2.5}$. To propose new values for both α and β, we let $\alpha^* = \alpha e^{\lambda(u-0.5)}$, and $\beta^* = \beta e^{\lambda(u-0.5)}$, where $u \sim \text{Uniform}[0, 1]$. With a change of variable, it can be shown that the proposal density is

$$p(y|x) = \frac{1}{\lambda y}$$

λ is a tuning parameter that controls the proposal.

Beta-binomial model: simulate data

We first simulate $y_i, i = 1, ..., 100$, observations from Binomial(n_i, p_i), where $p_i \sim$ Beta$(1, 15)$.

```
# simulate data
set.seed(1)
size = 100
n <- sample(1:50, size, replace = TRUE)
a <- 1
b <- 15
p = rbeta(size, a, b)
y = rbinom(size, n, p)
data <- data.frame(y = y, n = n)
```

Beta-binomial model: functions

These functions calculates the log prior density, proposes new parameters (α or β), and draws new samples of p.

```
# log prior evaluation function
log.prior <- function(alpha, beta) {
    -2.5 * log(alpha + beta)
}
# sample proposal value
getstar = function(current, lambda) {
    return(current * exp(lambda * (runif(1) - 0.5)))
}
# conditional sample of p
draw.p <- function(alpha, beta) {
    return(rbeta(size, alpha + y, beta + n - y))
}
```

Beta-binomial model: MH draw

This function performs the M-H step sampling of α.

```
draw.alpha <- function(alpha, beta, p, lambda) {
    alpha.star <- getstar(alpha, lambda)
    num <- size * (lgamma(alpha.star + beta) - lgamma(alpha.star)) +
        alpha.star * sum(log(p))
    num <- num + log.prior(alpha.star, beta)
    num <- num + log(alpha.star)
    den <- size * (lgamma(alpha + beta) - lgamma(alpha)) +
        alpha * sum(log(p))
    den <- den + log.prior(alpha, beta)
    den <- den + log(alpha)

    acc <- ifelse((log(runif(1)) <= num - den) && (alpha.star >
        0), 1, 0)
    return(c(acc, ifelse(acc, alpha.star, alpha)))
}
```

Beta-binomial model: MH draw

This function performs the M-H step sampling of β.

```
draw.beta <- function(alpha, beta, p, lambda) {
    beta.star <- getstar(beta, lambda)
    num <- size * (lgamma(alpha + beta.star) - lgamma(beta.star)) +
        beta.star * sum(log(1 - p))
    num <- num + log.prior(alpha, beta.star)
    num <- num + log(beta.star)
    den <- size * (lgamma(alpha + beta) - lgamma(beta)) +
        beta * sum(log(1 - p))
    den <- den + log.prior(alpha, beta)
    den <- den + log(beta)
    acc <- ifelse((log(runif(1)) <= num - den) && (beta.star >
        0), 1, 0)
    return(c(acc, ifelse(acc, beta.star, beta)))
}
```

Beta-binomial model: Initialization

```
Nitr <- 10000
a.draw <- b.draw <- matrix(NA, Nitr, 2)
ps <- matrix(NA, nrow = Nitr, ncol = size)

# Metropolis tuning parameters
lambda.alpha <- 0.7
lambda.beta <- 0.7

# Initial values for the chain
a.draw[1, 2] <- 1
b.draw[1, 2] <- 1
ps[1, ] <- draw.p(a.draw[1, 2], b.draw[1, 2])
```

Beta-binomial model: Sampling

```
# MCMC simulation
for (m in 2:Nitr) {
    a.draw[m, ] <- draw.alpha(a.draw[m - 1, 2], b.draw[m -
        1, 2], ps[m - 1, ], lambda.alpha)
    b.draw[m, ] <- draw.beta(a.draw[m, 2], b.draw[m -
        1, 2], ps[m - 1, ], lambda.beta)
    ps[m, ] <- draw.p(a.draw[m, 2], b.draw[m, 2])
}
# thinning
draws <- seq(1001, Nitr, by = 5)
a.draw <- a.draw[draws, ]
b.draw <- b.draw[draws, ]
ps <- ps[draws, ]
```

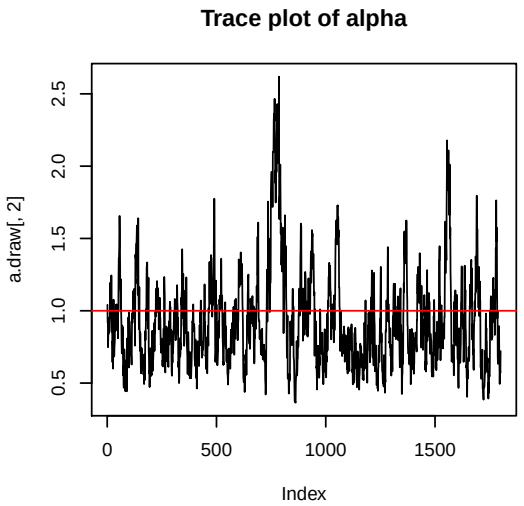
Beta-binomial model: Compare results

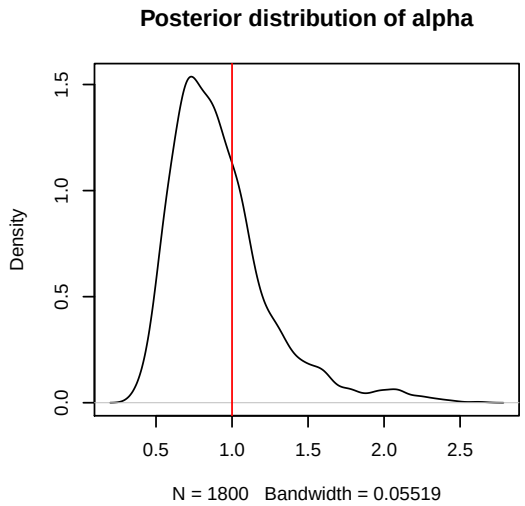
```
par(mfcol = c(2, 3))
plot(a.draw[, 2], type = "l", main = "Trace plot of alpha")
abline(h = a, col = "red")
plot(b.draw[, 2], type = "l", main = "Trace plot of beta")
abline(h = b, col = "red")
plot(density(a.draw[, 2]), main = "Posterior distribution of alpha")
abline(v = a, col = "red")
plot(density(b.draw[, 2]), main = "Posterior distribution of beta")
abline(v = b, col = "red")
plot(p, y/n, xlim = c(0, 0.4), ylim = c(0, 0.4), xlab = "Truth",
    ylab = "MLE", main = "Prevalence: MLE")
abline(c(0, 1))
plot(p, apply(ps, 2, median), xlim = c(0, 0.4), ylim = c(0,
    0.4), xlab = "Truth", ylab = "Posterior median",
    main = "Prevalence: Bayes")
abline(c(0, 1))
print(round(c(alpha.rate = mean(a.draw[, 1]), beta.rate = mean(b.draw[,
    1])), 2))
```

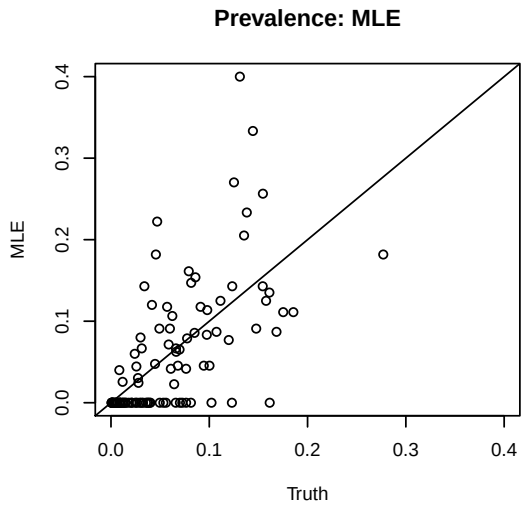
Beta-binomial model: Compare results

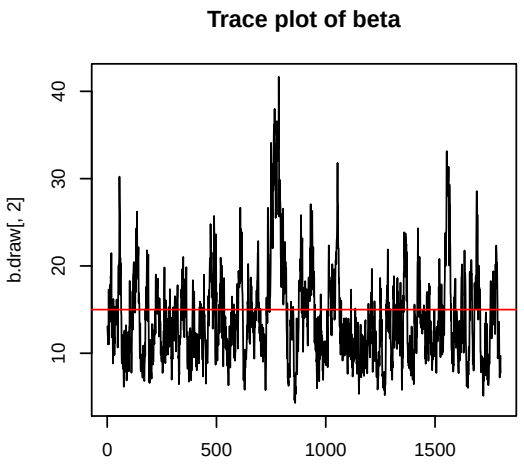
Acceptance rate and posterior summary:

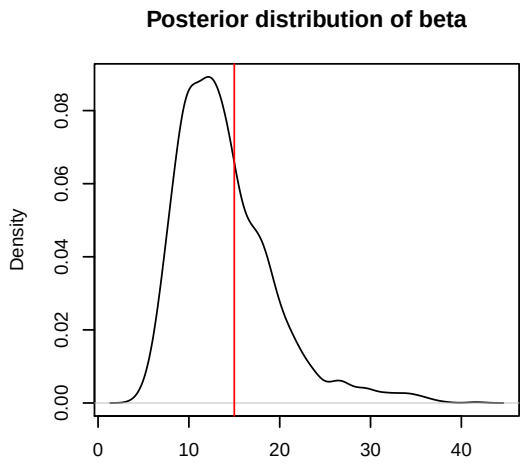
```
## alpha.rate  beta.rate
##        0.37       0.46
```

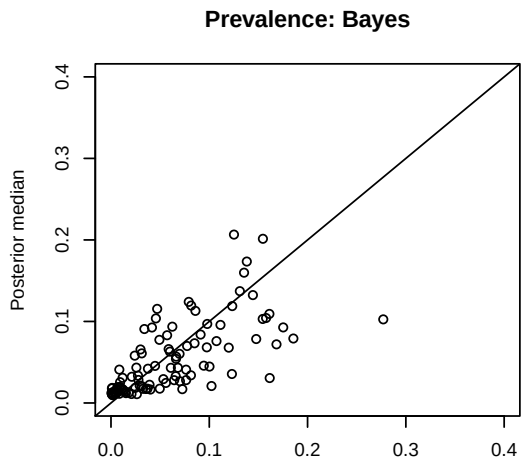












Additional topics: Normal hierarchical model using Gibbs sampler

Normal hierarchical model

Let's consider the following hierarchical model (Notice we change the notation as model gets more complicated)

$$\begin{aligned} y_{ij}|\theta_i, \sigma^2 &\sim \text{Normal}\left(\theta_i, \sigma^2\right) \\ \theta_i|\mu, \tau^2 &\sim \text{Normal}\left(\mu, \tau^2\right) \end{aligned}$$

And we use the following 'default' priors on the unknown parameters

$$\begin{aligned} \mu &\sim \text{Normal}\left(\mu_0, \gamma_0^2\right) \\ \sigma^2 &\sim \text{InvGamma}\left(\nu_0/2, \nu_0\sigma_0^2/2\right) \\ \tau^2 &\sim \text{InvGamma}\left(\eta_0/2, \eta_0\tau_0^2/2\right) \end{aligned}$$

The hyperparameters we need to specify are

- μ_0 and τ_0^2 are the prior guess at the μ and the certainty of this guess.
- σ_0 and ν_0 are the prior guess at the within-area variance σ^2 and the certainty of this guess.
- τ_0 and η_0 are the prior guess at the across-area variance τ and the certainty of this guess.

Conditional distribution

As we have seen in simple normal model, the conditional distribution of θ given data and other unknown parameters are

$$\theta_i|\mu, \sigma^2, \boldsymbol{y} \sim \text{Normal}(w\bar{y}_i + (1-w)\mu, w\frac{\sigma^2}{n_i})$$

where

$$w_i = \frac{\tau^2}{\tau^2 + \sigma^2/n_i}$$

```
samp.theta <- function(ybar, mu, sigma.sq, tau.sq,
    ni) {
    w <- tau.sq/(tau.sq + sigma.sq/ni)
    m <- w * ybar + (1 - w) * mu
    v <- w * sigma.sq/ni
    return(rnorm(length(m), m, sqrt(v)))
}
```

Conditional distribution

We can show that the conditional distribution of μ given data and other unknown parameters are

$$\mu|\tau, \boldsymbol{\theta} \sim \text{Normal}(w\bar{\theta} + (1-w)\mu_0, w\frac{\tau^2}{n})$$

where

$$w = \frac{\gamma_0^2}{\gamma_0^2 + \tau^2/n}$$

```
samp.mu <- function(theta, tau.sq, mu0, gamma0.sq,
    n) {
    w <- gamma0.sq/(gamma0.sq + tau.sq/n)
    mu.n <- w * mean(theta) + (1 - w) * mu0
    var.n <- w * tau.sq/n
    return(rnorm(1, mu.n, sqrt(var.n)))
}
```

Conditional distribution

We can show that the conditional distribution of τ^2 given data and other unknown parameters are

$$\tau^2|\mu, \boldsymbol{\theta} \sim \text{InvGamma}(a_n, b_n)$$

where

$$a_n = (n + \eta_0)/2, \quad b_n = \eta_0\tau_0^2 + \sum_i (\theta_i - \mu)^2$$

```
samp.tau.sq <- function(theta, mu, eta0, tau0.sq, mu0,
    n) {
    a <- (n + eta0)/2
    b <- eta0 * tau0.sq/2 + sum((theta - mu)^2)/2
    return(1/rgamma(1, a, b))
}
```

Conditional distribution

We can show that the conditional distribution of σ^2 given data and other unknown parameters are

$$\sigma^2|\mu, \boldsymbol{\theta}, \boldsymbol{y} \sim \text{InvGamma}(a_n, b_n)$$

where

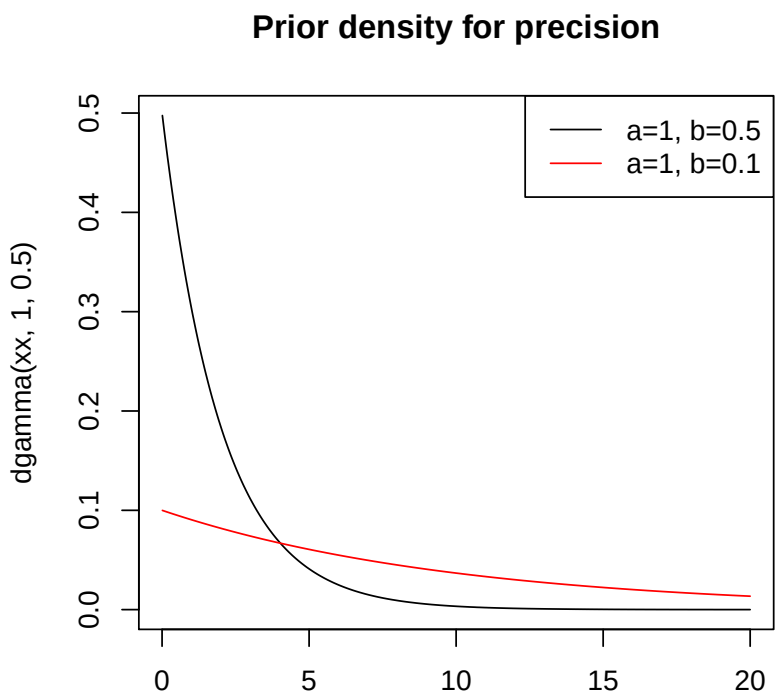
$$a_n = (\nu_0 + \sum_i n_i)/2, \quad b_n = \nu_0\sigma_0^2/2 + \sum_i \sum_j (y_{ij} - \mu_i)^2/2$$

```
samp.sigma.sq <- function(theta, S, v0, sigma0.sq,
    n, N) {
    a <- (n + N)/2
    b <- (v0 * sigma0.sq + S)/2
    return(1/rgamma(1, a, b))
}
```

Full Bayes estimation: Gibbs sampling

With suitable starting values, we can iteratively sample from the conditional distributions of the unknown parameters $(\boldsymbol{\theta}, \mu, \sigma^2, \tau^2)$.
We put weekly informative priors, invGamma$(1, 0.05)$ on σ^2 and invGamma$(1, 0.01)$ on τ^2, and a very flat Normal$(0, 10000)$ prior on μ.

```
xx <- seq(0.01, 20, len = 1000)
plot(xx, dgamma(xx, 1, 0.5), type = "l", main = "Prior density for precision")
lines(xx, dgamma(xx, 1, 0.1), col = 2)
legend("topright", c("a=1, b=0.5", "a=1, b=0.1"), col = c(1,
    2), lty = c(1, 1))
```



Gibbs sampling details

```
# data is a three column data.frame [index=i, value=y_ij]
sampleHN <- function(data, n, Nsim, burnin, eta0, tau0.sq, v0, sigma0.sq,
    mu0, gamma0.sq) {
    N <- dim(data)[1]
    ni <- as.numeric(table(c(1:n, data[, 1])) - 1)
    # saved values
    thetas <- matrix(NA, Nsim, n)
    mus <- sigma.sqs <- tau.sqs <- rep(NA, Nsim)
    # initiate values
    theta <- ybar <- aggregate(value ~ index, data, mean)[, 2]
    mu <- mean(theta)
    sigma.sq <- mean((data[, 2] - theta[data[, 1]])^2)
    tau.sq <- mean((theta - mu)^2)
    for (i in 1:Nsim) {
        thetas[i, ] <- theta <- samp.theta(ybar, mu, sigma.sq, tau.sq,
            ni)
        S <- sum((data[, 2] - theta[data[, 1]])^2)
        sigma.sqs[i] <- sigma.sq <- samp.sigma.sq(theta, S, v0, sigma0.sq,
            n, N)
        mus[i] <- mu <- samp.mu(theta, tau.sq, mu0, gamma0.sq, n)
        tau.sqs[i] <- tau.sq <- samp.tau.sq(theta, mu, eta0, tau0.sq, mu0,
            n)
    }
    thetas <- thetas[(burnin + 1):Nsim, ]
    mus <- mus[(burnin + 1):Nsim]
    sigma.sqs <- sigma.sqs[(burnin + 1):Nsim]
    tau.sqs <- tau.sqs[(burnin + 1):Nsim]
    return(list(theta = thetas, mu = mus, sigma.sq = sigma.sqs, tau.sq = tau.sqs))
}
```

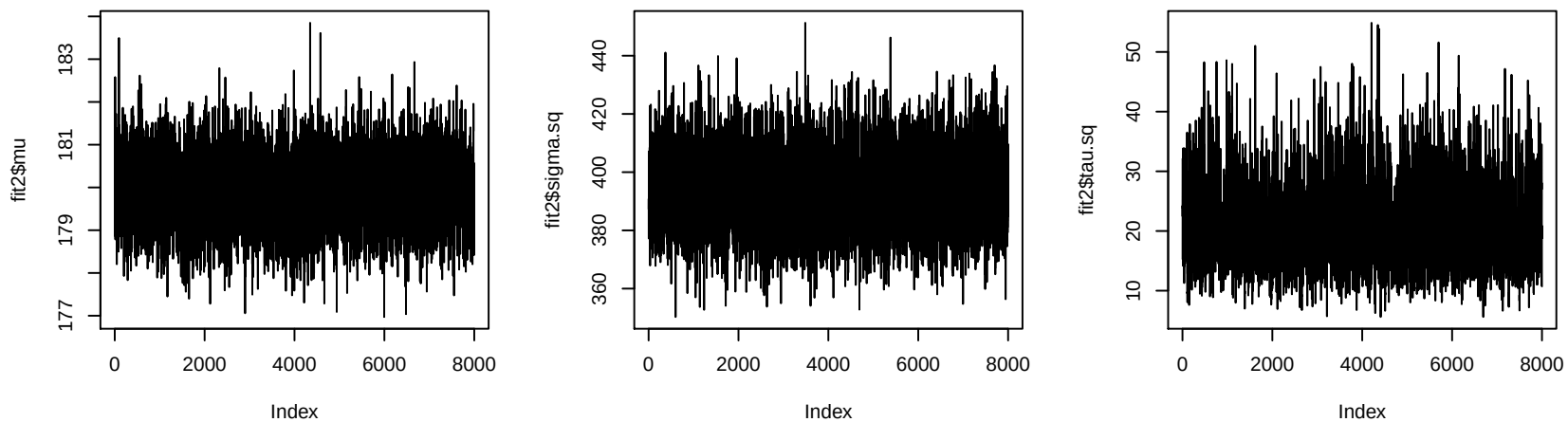
Estimating hierarchical normal model with simulated data

We sample 1,000 observations from the simulated dataset, draw 10,000 samples and discard the first 2,000 samples (burn-in).

```
load("../data/simKing.rda")
set.seed(1)
samp <- pop[sample(1:dim(pop)[1], 2000), ]
samp <- data.frame(index = samp$area, value = samp$weight)
fit2 <- sampleHN(data = samp, n = 48, Nsim = 10000,
    burnin = 2000, eta0 = 2, tau0.sq = 0.1, v0 = 2,
    sigma0.sq = 0.5, mu0 = 0, gamma0.sq = 10000)
```

Posterior summary: trace plot

```
par(mfrow = c(1, 3))
plot(fit2$mu, type = "l")
plot(fit2$sigma.sq, type = "l")
plot(fit2$tau.sq, type = "l")
```



Posterior summary: compare with MLE

```
par(mfrow = c(2, 2))
theta.true <- aggregate(weight ~ area, pop, mean)[,
    2]
theta.naive <- aggregate(value ~ index, samp, mean)[,
    2]
theta.hat <- apply(fit2$theta, 2, median)
lim <- range(c(theta.true, theta.naive, theta.hat))
plot(theta.true, theta.naive, xlab = "Truth", ylab = "MLE",
    xlim = lim, ylim = lim)
abline(c(0, 1))
plot(theta.true, theta.hat, xlab = "Truth", ylab = "Smoothed",
    xlim = lim, ylim = lim)
abline(c(0, 1))
plot(theta.naive, theta.hat, xlab = "MLE", ylab = "Smoothed",
    xlim = lim, ylim = lim)
abline(c(0, 1))
```

Posterior summary: compare with MLE