Networks in Context Lab, 2024

Very Short Introduction to Bayesian Statistics

Barum Park
Department of Sociology
Cornell University

March 15, 2024

Outline

Very short introduction to Bayesian statistics using `Stan`

Outline

Very short introduction to Bayesian statistics using `Stan`

1. Very short introduction to difference between Bayesian and Frequentist approaches

Outline

Very short introduction to Bayesian statistics using `Stan`

1. Very short introduction to difference between Bayesian and Frequentist approaches

2. Very short introduction to benefits of Bayesian approach

Outline

Very short introduction to Bayesian statistics using `Stan`

1. Very short introduction to difference between Bayesian and Frequentist approaches

2. Very short introduction to benefits of Bayesian approach

3. Very short introduction to using `Stan`'s No-U-Turn sampler to obtain samples from posterior distribution

What is Bayesian Statistics?

Parametric models describe the probability of some data y as a function of a parameter θ (and possibly some other data, x):

$$y \sim f(\theta, x)$$

Parametric models describe the probability of some data y as a function of a parameter θ (and possibly some other data, x):

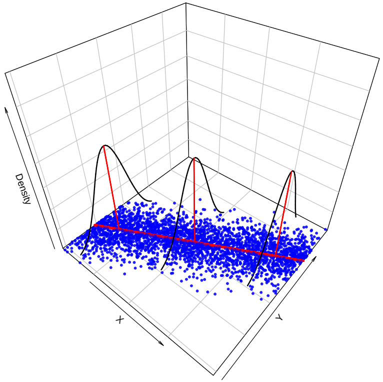
$$y \sim f(\theta, x)$$

For example, in linear regression, we assume

$$y_i \sim \text{Normal}(x_i\beta, \sigma_\epsilon)$$

where $\theta = \{\beta, \sigma_\epsilon\}$.

The objective of the analysis to estimate θ or some function of it



In Frequentist or MLE approaches, inference centers around some *point estimate* of θ. For example

In Frequentist or MLE approaches, inference centers around some *point estimate* of θ. For example

1. What is the most likely value of θ that generated the data? (MLE)

In Frequentist or MLE approaches, inference centers around some *point estimate* of θ. For example

1. What is the most likely value of θ that generated the data? (MLE)
2. Assume $H_0 : \theta = 0$ what would be the distribution of $g(y)$? Does the observed value of $g(y)$ look like it came from that distribution? (NHST)

In Frequentist or MLE approaches, inference centers around some *point estimate* of θ. For example

1. What is the most likely value of θ that generated the data? (MLE)
2. Assume $H_0 : \theta = 0$ what would be the distribution of $g(y)$? Does the observed value of $g(y)$ look like it came from that distribution? (NHST)

The basic assumption is that θ is some unknown *fixed* quantity

(We often talk about *the* “true” θ that generated the data. Talking about $p(\theta)$ or $p(H_0)$ doesn’t make sense.)

In Frequentist or MLE approaches, inference centers around some *point estimate* of θ. For example

1. What is the most likely value of θ that generated the data? (MLE)
2. Assume $H_0 : \theta = 0$ what would be the distribution of $g(y)$? Does the observed value of $g(y)$ look like it came from that distribution? (NHST)

The basic assumption is that θ is some unknown *fixed* quantity

(We often talk about *the* "true" θ that generated the data. Talking about $p(\theta)$ or $p(H_0)$ doesn't make sense.)

The only thing that is random are the data, where randomness arises due to some sampling process.

The Bayesian approach assumes that *θ is a random variable itself*—hence, it has a distribution.

The Bayesian approach assumes that *θ is a random variable itself*—hence, it has a distribution.

So, it *does* make sense to talk about $p(\theta)$ or $p(H_0)$

The Bayesian approach assumes that *θ is a random variable itself*—hence, it has a distribution.

So, it *does* make sense to talk about $p(\theta)$ or $p(H_0)$

The analysis centers around updating the distribution of θ based on newly observed data.

The Bayesian approach assumes that *θ is a random variable itself*—hence, it has a distribution.

So, it *does* make sense to talk about $p(\theta)$ or $p(H_0)$

The analysis centers around updating the distribution of θ based on newly observed data.

1. we have some *prior* belief about θ, represented as a distribution

   (Note: this belief might be "we have no idea at all")

The Bayesian approach assumes that *θ is a random variable itself*—hence, it has a distribution.

So, it *does* make sense to talk about $p(\theta)$ or $p(H_0)$

The analysis centers around updating the distribution of θ based on newly observed data.

1. we have some *prior* belief about θ, represented as a distribution

   (Note: this belief might be "we have no idea at all")

2. We observe some data

The Bayesian approach assumes that *θ is a random variable itself*—hence, it has a distribution.

So, it *does* make sense to talk about $p(\theta)$ or $p(H_0)$

The analysis centers around updating the distribution of θ based on newly observed data.

1. we have some *prior* belief about θ, represented as a distribution

   (Note: this belief might be "we have no idea at all")
2. We observe some data
3. We obtain a new distribution of θ by updating our beliefs based on the data

   (Of course, we might be interested in some statistic of this distribution, such as the mean or median...)

Updating is done via Bayes Rule.

Updating is done via Bayes Rule. Glossing over technical details, recall for random variables v and w, we have

$$p(v \mid w) = \frac{p(v \text{ and } w)}{p(w)} = \frac{p(w \mid v)p(v)}{p(w)}$$

Updating is done via Bayes Rule. Glossing over technical details, recall for random variables v and w, we have

$$p(v \mid w) = \frac{p(v \text{ and } w)}{p(w)} = \frac{p(w \mid v)p(v)}{p(w)}$$

So, let $v = \theta$ and $w = y$, which gives the updating formula

$$p(\theta \mid y) = \frac{p(y \mid \theta)p(\theta)}{p(y)}$$

$$\boxed{p(\theta \mid y) = \frac{p(y \mid \theta)p(\theta)}{p(y)}}$$

Notice that

$$\boxed{p(\theta \mid y) = \frac{p(y \mid \theta)p(\theta)}{p(y)}}$$

Notice that

1. $p(\theta \mid y)$ is the probability of θ *after observing the data*—i.e., the posterior distribution

$$\boxed{p(\theta \mid y) = \frac{p(y \mid \theta)p(\theta)}{p(y)}}$$

Notice that

1. $p(\theta \mid y)$ is the probability of θ *after observing the data*—i.e., the posterior distribution

2. $p(y \mid \theta)$ is the probability of *observing the data* given a particular value of θ—i.e., *likelihood* of the data

$$\boxed{p(\theta \mid y) = \frac{p(y \mid \theta)p(\theta)}{p(y)}}$$

Notice that

1. $p(\theta \mid y)$ is the probability of θ *after observing the data*—i.e., the posterior distribution

2. $p(y \mid \theta)$ is the probability of *observing the data* given a particular value of θ—i.e., *likelihood* of the data

3. $p(\theta)$ is the probability of θ *before observing the data*—i.e, the *prior distribution* of θ

$$\boxed{p(\theta \mid y) = \frac{p(y \mid \theta)p(\theta)}{p(y)}}$$

Notice that

1. $p(\theta \mid y)$ is the probability of θ *after observing the data*—i.e., the posterior distribution

2. $p(y \mid \theta)$ is the probability of *observing the data* given a particular value of θ—i.e., *likelihood* of the data

3. $p(\theta)$ is the probability of θ *before observing the data*—i.e, the *prior distribution* of θ

4. $p(y) = \int_\Theta p(y \mid \theta)p(\theta)d\theta$ is a constant that doesn't depend on θ and is often not of theoretical interest

So, we often write

$$\boxed{p(\theta \mid y) \propto p(y \mid \theta) p(\theta)}$$

i.e., “the posterior is proportional to the likelihood times the prior”

Once we have $p(\theta \mid y)$, it's easy to make any kind of inference regarding θ.

Once we have $p(\theta \mid y)$, it's easy to make any kind of inference regarding θ.

When we are interested in a *point estimate*, we can use the *maximum a posteriori (MAP)* estimate:

$$\mathrm{MAP}(\theta) = \arg\max_{\theta \in \Theta} p(y \mid \theta) p(\theta)$$

(other possiblities are the posterior mean, posterior median, etc...)

Once we have $p(\theta \mid y)$, it's easy to make any kind of inference regarding θ.

When we are interested in a *point estimate*, we can use the *maximum a posteriori (MAP)* estimate:

$$\mathrm{MAP}(\theta) = \arg\max_{\theta \in \Theta} p(y \mid \theta) p(\theta)$$

(other possiblities are the posterior mean, posterior median, etc...)

Comparing this to the maximum likelihood estimate(MLE)

$$\mathrm{MLE}(\theta) = \arg\max_{\theta \in \Theta} p(y \mid \theta)$$

we see that the only difference is the prior.

Once we have $p(\theta \mid y)$, it's easy to make any kind of inference regarding θ.

When we are interested in a *point estimate*, we can use the *maximum a posteriori (MAP)* estimate:

$$\text{MAP}(\theta) = \arg\max_{\theta \in \Theta} p(y \mid \theta) p(\theta)$$

(other possiblities are the posterior mean, posterior median, etc...)

Comparing this to the maximum likelihood estimate(MLE)

$$\text{MLE}(\theta) = \arg\max_{\theta \in \Theta} p(y \mid \theta)$$

we see that the only difference is the prior.

Further, *the likelihood will dominate the prior as n grows large.*

Once we have $p(\theta \mid y)$, it's easy to make any kind of inference regarding θ.

When we are interested in a *point estimate*, we can use the *maximum a posteriori (MAP)* estimate:

$$\mathrm{MAP}(\theta) = \arg\max_{\theta \in \Theta} p(y \mid \theta) p(\theta)$$

(other possiblities are the posterior mean, posterior median, etc...)

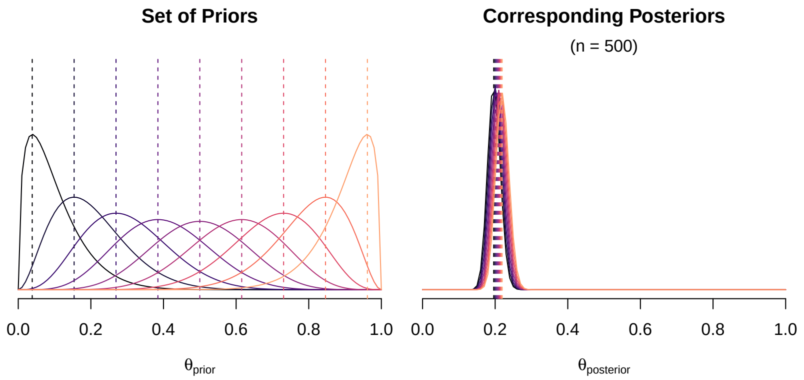
Comparing this to the maximum likelihood estimate(MLE)

$$\mathrm{MLE}(\theta) = \arg\max_{\theta \in \Theta} p(y \mid \theta)$$

we see that the only difference is the prior.

Further, *the likelihood will dominate the prior as n grows large.* So, in large samples, the MAP will approach the MLE

A similar logic applies to other models as well:



(Notes: Beta-Binomial model with $n = 500$ observations and MLE$(\theta) = .2$)

So, why bothering to use Bayesian analysis at all?

Why You Should Be a Bayesian?

There are multiple arguments for using Bayesian statistics. Among others:

1. It's more intuitive

   (95% confidence vs credible intervals...)

2. Once you get the posterior, you can make valid inference about any function of θ

   (e.g., What is $p(\beta_1/\beta_2 \leq \beta_3 \mid y)$?)

3. It works better for "weakly" or non-identified models

   e.g., "Hessian is not positive definite" situations, perfect separation in logistic regression

4. It's a natural way to regualize inference

Let's discuss 4. a bit more in detail...

How does the prior regularize inference?

How does the prior regularize inference?

To get some intuition, it's quite informative to compare the MAP with the MLE.

How does the prior regularize inference?

To get some intuition, it's quite informative to compare the MAP with the MLE.

In linear regression with normal likelihood,

1. Normal prior:

   $\mathrm{MAP}(\theta) = \mathrm{Ridge}(\theta)$ (i.e., L^2 regularization)

2. Double-Exponential or Laplace prior:

   $\mathrm{MAP}(\theta) = \mathrm{LASSO}(\theta)$ (L^1 regularization)

How does the prior regularize inference?

To get some intuition, it's quite informative to compare the MAP with the MLE.

In linear regression with normal likelihood,

1. Normal prior:

   $\mathrm{MAP}(\theta) = \mathrm{Ridge}(\theta)$ (i.e., L^2 regularization)

2. Double-Exponential or Laplace prior:

   $\mathrm{MAP}(\theta) = \mathrm{LASSO}(\theta)$ (L^1 regularization)

But you get more than just a point estimate: the whole *posterior distribution*

You can think of the prior as adding some (pseudo-) observations to your dataset, where *they make sense*

You can think of the prior as adding some (pseudo-) observations to your dataset, where *they make sense*

- *If your dataset is large*, these added observations will have no effect on your inference (likelihood dominates!)

You can think of the prior as adding some (pseudo-) observations to your dataset, where *they make sense*

- ▷ *If your dataset is large*, these added observations will have no effect on your inference (likelihood dominates!)
- ▷ *But if your dataset is small*, they make sure that your estimate is not too far off from a reasonable value

You can think of the prior as adding some (pseudo-) observations to your dataset, where *they make sense*

- ▷ *If your dataset is large*, these added observations will have no effect on your inference (likelihood dominates!)
- ▷ *But if your dataset is small*, they make sure that your estimate is not too far off from a reasonable value

So, you introduce a bit of a bias with the prior in exchange for reduce variance of your inference

Does this mean that Bayesian approaches give you nothing in large datasets?

Does this mean that Bayesian approaches give you nothing in large datasets?

Does this mean that Bayesian approaches give you nothing in large datasets?

- Even in large datasets, you'll often find *niches* (so to say) in covariate space with little data

Does this mean that Bayesian approaches give you nothing in large datasets?

- Even in large datasets, you'll often find *niches* (so to say) in covariate space with little data
- The prior will regularize inference on those niches, while not influencing much the inference in other regards

Does this mean that Bayesian approaches give you nothing in large datasets?

- Even in large datasets, you'll often find *niches* (so to say) in covariate space with little data
- The prior will regularize inference on those niches, while not influencing much the inference in other regards
- This is why random effects models (RE) perform better in prediction tasks than fixed-effect models (FE)

  (REs can be understood as FEs with a prior on the group-level intercepts; conversely, FEs can be understood as REs with infinite-variance priors)

Bayesian Inference in Practice (a.k.a. MCMC)

In Bayesian statistics, we want to make inference based on $p(\theta \mid y)$

In Bayesian statistics, we want to make inference based on $p(\theta \mid y)$

But how do we do this in practice?

In Bayesian statistics, we want to make inference based on $p(\theta \mid y)$

But how do we do this in practice?

1. If cases where $p(\theta \mid y)$ can be calculated analytically, we are done

In Bayesian statistics, we want to make inference based on $p(\theta \mid y)$

But how do we do this in practice?

1. If cases where $p(\theta \mid y)$ can be calculated analytically, we are done

2. In cases where we cannot calculate the posterior analytically, but can *sample* directly from the posterior, we can approximate $p(\theta \mid y)$ using Monte Carlo

In Bayesian statistics, we want to make inference based on $p(\theta \mid y)$

But how do we do this in practice?

1. If cases where $p(\theta \mid y)$ can be calculated analytically, we are done

2. In cases where we cannot calculate the posterior analytically, but can *sample* directly from the posterior, we can approximate $p(\theta \mid y)$ using Monte Carlo

3. In cases we cannot calculate $p(\theta \mid y)$ directly nor sample directly from it (most of the time), there are still ways approximate it (MCMC, HMC, VB, etc.)

In Bayesian statistics, we want to make inference based on $p(\theta \mid y)$

But how do we do this in practice?

1. If cases where $p(\theta \mid y)$ can be calculated analytically, we are done

2. In cases where we cannot calculate the posterior analytically, but can *sample* directly from the posterior, we can approximate $p(\theta \mid y)$ using Monte Carlo

3. In cases we cannot calculate $p(\theta \mid y)$ directly nor sample directly from it (most of the time), there are still ways approximate it (MCMC, HMC, VB, etc.)

In this (very) short introduction, we'll focus on `Stan`, which implements an HMC algorithm (called the No-U-turn sampler)

So, what is Hamiltonian Monte Carlo (HMC)?

So, what is Hamiltonian Monte Carlo (HMC)?

It's an *efficient* Markov Chain Monte Carlo (MCMC) method

So, what is Hamiltonian Monte Carlo (HMC)?

It's an *efficient* Markov Chain Monte Carlo (MCMC) method

But what is MCMC?

So, what is Hamiltonian Monte Carlo (HMC)?

It's an *efficient* Markov Chain Monte Carlo (MCMC) method

But what is MCMC?

Recall that we are in a situatuon cannot sample directly from $p(\theta \mid y)$ ☹

It turns out that it's somtime possible to define a *Markov Chain* on Θ that has $p(\theta \mid y)$ as it's *unique limiting distribution* ☺

So, what is Hamiltonian Monte Carlo (HMC)?

It's an *efficient* Markov Chain Monte Carlo (MCMC) method

But what is MCMC?

Recall that we are in a situatuon cannot sample directly from $p(\theta \mid y)$ ☹

It turns out that it's somtime possible to define a *Markov Chain* on Θ that has $p(\theta \mid y)$ as it's *unique limiting distribution* ☺

So, starting from $\theta^{(0)}$, we might let $\theta^{(s)}$ evolve according to the Markov Process until some large S and take $\{\theta^{(s)}\}_{s>S}$ as samples from $p(\theta \mid y)$.

So, what is Hamiltonian Monte Carlo (HMC)?

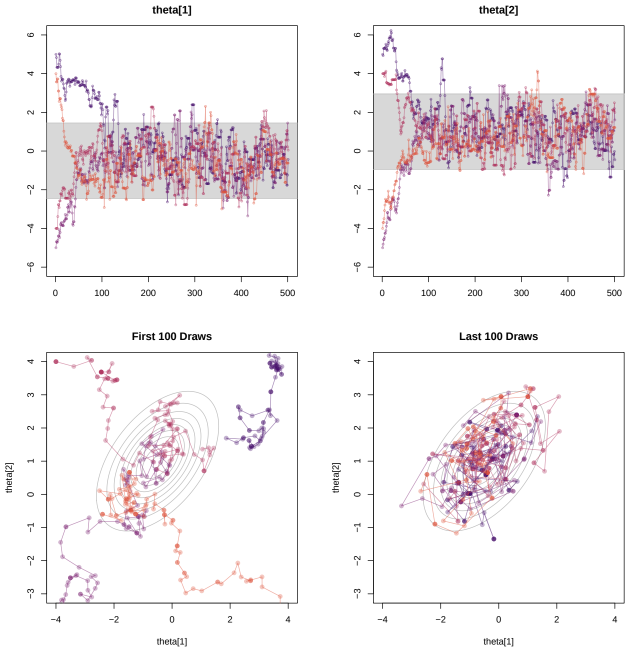
It's an *efficient* Markov Chain Monte Carlo (MCMC) method

But what is MCMC?

Recall that we are in a situatuon cannot sample directly from $p(\theta \mid y)$ ☹

It turns out that it's somtime possible to define a *Markov Chain* on Θ that has $p(\theta \mid y)$ as it's *unique limiting distribution* ☺

So, starting from $\theta^{(0)}$, we might let $\theta^{(s)}$ evolve according to the Markov Process until some large S and take $\{\theta^{(s)}\}_{s>S}$ as samples from $p(\theta \mid y)$. This is MCMC


theta[1]
theta[2]
First 100 Draws
Last 100 Draws
theta[1]
theta[2]
theta[1]
theta[2]

Pause...Questions so far?

Bayesian Models in `Stan`

The sampler implemented in `Stan`, called the *No-U-Turn* sampler, is, again, an improvement on basic HMC.

The sampler implemented in `Stan`, called the *No-U-Turn* sampler, is, again, an improvement on basic HMC. But we won't dwell on the details here.

The sampler implemented in `Stan`, called the *No-U-Turn* sampler, is, again, an improvement on basic HMC. But we won't dwell on the details here.

Instead, let's focus on how to get `Stan` to work!

The sampler implemented in `Stan`, called the *No-U-Turn* sampler, is, again, an improvement on basic HMC. But we won't dwell on the details here.

Instead, let's focus on how to get `Stan` to work!

In a nutshell,

The sampler implemented in `Stan`, called the *No-U-Turn* sampler, is, again, an improvement on basic HMC. But we won't dwell on the details here.

Instead, let's focus on how to get `Stan` to work!

In a nutshell,

1. We specify the (log) posterior distribution up to a constant (i.e., we specify prior and likelihood)

The sampler implemented in `Stan`, called the *No-U-Turn* sampler, is, again, an improvement on basic HMC. But we won't dwell on the details here.

Instead, let's focus on how to get `Stan` to work!

In a nutshell,

1. We specify the (log) posterior distribution up to a constant (i.e., we specify prior and likelihood)
2. `Stan` creates posterior samples for us

The sampler implemented in `Stan`, called the *No-U-Turn* sampler, is, again, an improvement on basic HMC. But we won't dwell on the details here.

Instead, let's focus on how to get `Stan` to work!

In a nutshell,

1. We specify the (log) posterior distribution up to a constant (i.e., we specify prior and likelihood)
2. `Stan` creates posterior samples for us
3. We make inference based on these samples

The sampler implemented in `Stan`, called the *No-U-Turn* sampler, is, again, an improvement on basic HMC. But we won't dwell on the details here.

Instead, let's focus on how to get `Stan` to work!

In a nutshell,

1. We specify the (log) posterior distribution up to a constant (i.e., we specify prior and likelihood)
2. `Stan` creates posterior samples for us
3. We make inference based on these samples

Simple and Beautiful!

Here are some example data that we are going to analyze

```
> data = read.csv(here("example", "data.csv")) # data.frame object
> head(data)
          y date topic
1 0.8346688    1     1
2 0.9251793    2     1
3 0.8998521    3     1
4 0.9033230    4     1
5 0.9124415    5     1
6 0.8928756    6     1
> data$topic |> unique() |> length()
[1] 40
> data$date |> unique() |> length()
[1] 20
```

Here are some example data that we are going to analyze

```
> data = read.csv(here("example", "data.csv")) # data.frame object
> head(data)
          y date topic
1 0.8346688    1     1
2 0.9251793    2     1
3 0.8998521    3     1
4 0.9033230    4     1
5 0.9124415    5     1
6 0.8928756    6     1
> data$topic |> unique() |> length()
[1] 40
> data$date |> unique() |> length()
[1] 20
```

- `y`: A measure of ideological separability of comment threads ensuing a post
- `date`: the date of the post
- `topic`: topic of the post

We'll consider two models: simple linear regression and a multilevel model

We'll consider two models: simple linear regression and a multilevel model

The *simple linear regression* might be formulated as

$$y_{it} = \alpha + \beta x_{it} + \epsilon_{it}, \quad \epsilon_{it} \overset{\text{iid}}{\sim} \text{N}(0, \sigma_{\epsilon}^2)$$

where i denotes the topic and t the date of the data entry.

We'll consider two models: simple linear regression and a multilevel model

The *simple linear regression* might be formulated as

$$y_{it} = \alpha + \beta x_{it} + \epsilon_{it}, \quad \epsilon_{it} \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma_\epsilon^2)$$

where i denotes the topic and t the date of the data entry.

Notice

We'll consider two models: simple linear regression and a multilevel model

The *simple linear regression* might be formulated as

$$y_{it} = \alpha + \beta x_{it} + \epsilon_{it}, \quad \epsilon_{it} \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma_\epsilon^2)$$

where i denotes the topic and t the date of the data entry.

Notice

1. We don't model heterogenity across topics—i.e., we pool across all topics

We'll consider two models: simple linear regression and a multilevel model

The *simple linear regression* might be formulated as

$$y_{it} = \alpha + \beta x_{it} + \epsilon_{it}, \quad \epsilon_{it} \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma_\epsilon^2)$$

where i denotes the topic and t the date of the data entry.

Notice

1. We don't model heterogenity across topics—i.e., we pool across all topics
2. the parameters of interest are $\theta = \{\alpha, \beta, \sigma_\epsilon\}$

We'll consider two models: simple linear regression and a multilevel model

The *simple linear regression* might be formulated as

$$y_{it} = \alpha + \beta x_{it} + \epsilon_{it}, \quad \epsilon_{it} \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma_\epsilon^2)$$

where i denotes the topic and t the date of the data entry.

Notice

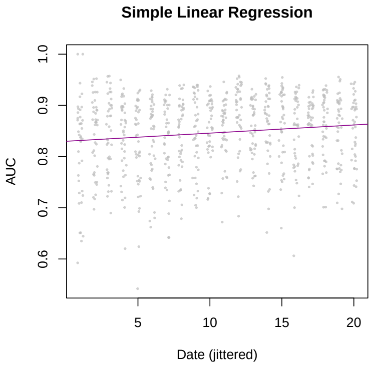
1. We don't model heterogenity across topics—i.e., we pool across all topics
2. the parameters of interest are $\theta = \{\alpha, \beta, \sigma_\epsilon\}$
3. Assign priors to the parmeters:

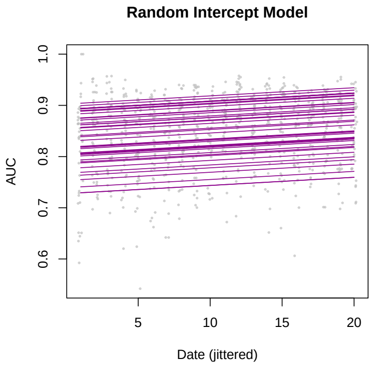
   $\alpha, \beta \sim \mathrm{N}(0, 1), \quad \sigma_\epsilon \sim \mathrm{Exp}(1)$

In the multilevel (random intercept) model, we assume that

In the multilevel (random intercept) model, we assume that

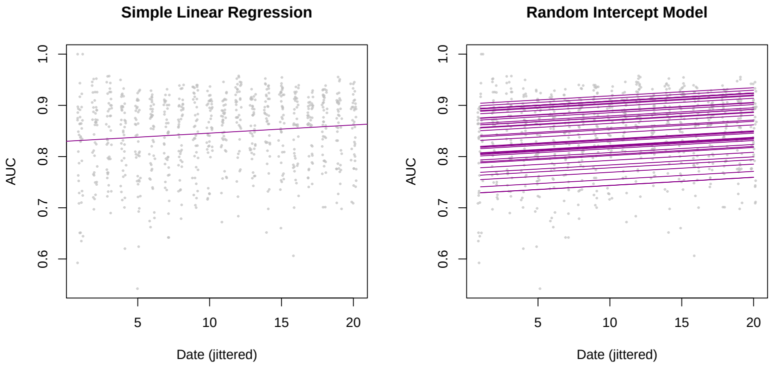
1. Each topic has it's own intercept





In the multilevel (random intercept) model, we assume that

1. Each topic has it's own intercept



2. The intercepts come from the same distribution (e.g., Normal distribution, but not necessarily Normal...)

Formally, we write

$$\begin{aligned} y_{it} = \alpha_i + \beta x_{it} + \epsilon_{it}, \qquad & \epsilon_{it} \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma_\epsilon^2) \\ & \alpha_i \overset{\text{iid}}{\sim} \mathrm{N}(\mu_\alpha, \sigma_\alpha^2) \end{aligned}$$

Formally, we write

$$\begin{aligned} y_{it} = \alpha_i + \beta x_{it} + \epsilon_{it}, \qquad & \epsilon_{it} \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma_\epsilon^2) \\ & \alpha_i \overset{\text{iid}}{\sim} \mathrm{N}(\mu_\alpha, \sigma_\alpha^2) \end{aligned}$$

Notice $\alpha_i \overset{\text{iid}}{\sim} \mathrm{N}(\mu_\alpha, \sigma_\alpha^2)$ is just assigning a prior to the random intercepts

Formally, we write

$$y_{it} = \alpha_i + \beta x_{it} + \epsilon_{it}, \qquad \epsilon_{it} \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma_\epsilon^2)$$
$$\alpha_i \overset{\text{iid}}{\sim} \mathrm{N}(\mu_\alpha, \sigma_\alpha^2)$$

Notice $\alpha_i \overset{\text{iid}}{\sim} \mathrm{N}(\mu_\alpha, \sigma_\alpha^2)$ is just assigning a prior to the random intercepts

Now we have $\theta = \{\beta, \sigma_\epsilon, \alpha_1, ..., \alpha_K, \mu_\alpha, \sigma_\alpha\}$

Formally, we write

$$y_{it} = \alpha_i + \beta x_{it} + \epsilon_{it}, \qquad \epsilon_{it} \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma_\epsilon^2)$$
$$\alpha_i \overset{\text{iid}}{\sim} \mathrm{N}(\mu_\alpha, \sigma_\alpha^2)$$

Notice $\alpha_i \overset{\text{iid}}{\sim} \mathrm{N}(\mu_\alpha, \sigma_\alpha^2)$ is just assigning a prior to the random intercepts

Now we have $\theta = \{\beta, \sigma_\epsilon, \alpha_1, ..., \alpha_K, \mu_\alpha, \sigma_\alpha\}$

Model is completed by assigning priors

Formally, we write

$$y_{it} = \alpha_i + \beta x_{it} + \epsilon_{it}, \qquad \epsilon_{it} \overset{\text{iid}}{\sim} \text{N}(0, \sigma_\epsilon^2)$$
$$\alpha_i \overset{\text{iid}}{\sim} \text{N}(\mu_\alpha, \sigma_\alpha^2)$$

Notice $\alpha_i \overset{\text{iid}}{\sim} \text{N}(\mu_\alpha, \sigma_\alpha^2)$ is just assigning a prior to the random intercepts

Now we have $\theta = \{\beta, \sigma_\epsilon, \alpha_1, ..., \alpha_K, \mu_\alpha, \sigma_\alpha\}$

Model is completed by assigning priors

1. Recall α_i's prior are already specified

Formally, we write

$$y_{it} = \alpha_i + \beta x_{it} + \epsilon_{it}, \qquad \epsilon_{it} \overset{\text{iid}}{\sim} \text{N}(0, \sigma_\epsilon^2)$$
$$\alpha_i \overset{\text{iid}}{\sim} \text{N}(\mu_\alpha, \sigma_\alpha^2)$$

Notice $\alpha_i \overset{\text{iid}}{\sim} \text{N}(\mu_\alpha, \sigma_\alpha^2)$ is just assigning a prior to the random intercepts

Now we have $\theta = \{\beta, \sigma_\epsilon, \alpha_1, ..., \alpha_K, \mu_\alpha, \sigma_\alpha\}$

Model is completed by assigning priors

1. Recall α_i's prior are already specified
2. Add $\beta \sim \text{N}(0, 1)$ and $\sigma_\epsilon \sim \text{Exp}(1)$ as before

Formally, we write

$$y_{it} = \alpha_i + \beta x_{it} + \epsilon_{it}, \qquad \epsilon_{it} \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma_\epsilon^2)$$
$$\alpha_i \overset{\text{iid}}{\sim} \mathrm{N}(\mu_\alpha, \sigma_\alpha^2)$$

Notice $\alpha_i \overset{\text{iid}}{\sim} \mathrm{N}(\mu_\alpha, \sigma_\alpha^2)$ is just assigning a prior to the random intercepts

Now we have $\theta = \{\beta, \sigma_\epsilon, \alpha_1, ..., \alpha_K, \mu_\alpha, \sigma_\alpha\}$

Model is completed by assigning priors

1. Recall α_i's prior are already specified
2. Add $\beta \sim \mathrm{N}(0, 1)$ and $\sigma_\epsilon \sim \mathrm{Exp}(1)$ as before
3. New stuff: $\mu_\alpha \sim \mathrm{N}(0, 1)$ and $\sigma_\alpha \sim \mathrm{Exp}(1)$

Formally, we write

$$y_{it} = \alpha_i + \beta x_{it} + \epsilon_{it}, \qquad \epsilon_{it} \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma_\epsilon^2)$$
$$\alpha_i \overset{\text{iid}}{\sim} \mathrm{N}(\mu_\alpha, \sigma_\alpha^2)$$

Notice $\alpha_i \overset{\text{iid}}{\sim} \mathrm{N}(\mu_\alpha, \sigma_\alpha^2)$ is just assigning a prior to the random intercepts

Now we have $\theta = \{\beta, \sigma_\epsilon, \alpha_1, ..., \alpha_K, \mu_\alpha, \sigma_\alpha\}$

Model is completed by assigning priors

1. Recall α_i's prior are already specified
2. Add $\beta \sim \mathrm{N}(0, 1)$ and $\sigma_\epsilon \sim \mathrm{Exp}(1)$ as before
3. New stuff: $\mu_\alpha \sim \mathrm{N}(0, 1)$ and $\sigma_\alpha \sim \mathrm{Exp}(1)$

Done!

Fitting Bayesian Models in `R` using `brms`

Fitting Bayesian regression models is extremely easy using the `brms` (or `rstanarm`) package

Fitting Bayesian regression models is extremely easy using the `brms` (or `rstanarm`) package

The syntax is basically the same as the `base::glm` or `lme4::lmer` functions

Fitting Bayesian regression models is extremely easy using the `brms` (or `rstanarm`) package

The syntax is basically the same as the `base::glm` or `lme4::lmer` functions

For the simple linear regression model:

```
slr_brms = brms::brm(
    formula = y ~ date, data = data, family = gaussian(),
    warmup = 500, iter = 1500, refresh = 1000, chains = 4,
    cores = 4, seed = 123
)
```

will do the job for us: creating the appropriate `Stan` code, and compile it in `C++`, and run the sampler

```
> slr_brms = brms::brm(...)
> print(slr_brms, digits = 4)
Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI   Rhat Bulk_ESS Tail_ESS
Intercept   0.8300    0.0053   0.8195   0.8402 1.0010     3728     3137
date        0.0016    0.0004   0.0007   0.0025 1.0020     4686     3137

Family Specific Parameters:
      Estimate Est.Error l-95% CI u-95% CI   Rhat Bulk_ESS Tail_ESS
sigma   0.0720    0.0019   0.0683   0.0758 1.0049      861      978
```

```
> slr_brms = brms::brm(...)
> print(slr_brms, digits = 4)
Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI   Rhat Bulk_ESS Tail_ESS
Intercept   0.8300    0.0053   0.8195   0.8402 1.0010     3728     3137
date        0.0016    0.0004   0.0007   0.0025 1.0020     4686     3137

Family Specific Parameters:
      Estimate Est.Error l-95% CI u-95% CI   Rhat Bulk_ESS Tail_ESS
sigma   0.0720    0.0019   0.0683   0.0758 1.0049      861      978
```

But what about the priors?

```
> slr_brms = brms::brm(...)
> print(slr_brms, digits = 4)
Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI   Rhat Bulk_ESS Tail_ESS
Intercept   0.8300    0.0053   0.8195   0.8402 1.0010     3728     3137
date        0.0016    0.0004   0.0007   0.0025 1.0020     4686     3137

Family Specific Parameters:
      Estimate Est.Error l-95% CI u-95% CI   Rhat Bulk_ESS Tail_ESS
sigma   0.0720    0.0019   0.0683   0.0758 1.0049      861      978
```

But what about the priors?

```
> prior_summary(slr_brms)
                prior     class coef group resp dpar nlpar lb ub       source
               (flat)         b                                       default
               (flat)         b date                             (vectorized)
 student_t(3, 0.9, 2.5) Intercept                                     default
   student_t(3, 0, 2.5)     sigma                          0          default
```

Setting custom priors is easy as well:

```r
slr_brms_w_prior = brms::brm(
    formula = y ~ date,
    data = data,
    family = gaussian(),
    prior = c(
        set_prior("normal(0, 1)", class = "Intercept"),
        set_prior("normal(0, 1)", class = "b", coef = "date"),
        set_prior("exponential(1)", class = "sigma")
    ),
    warmup = 500,
    iter = 1500,
    refresh = 1000,
    chains = 4,
    cores = 4,
    seed = 123
)
```

Notice that the results are almost identical:

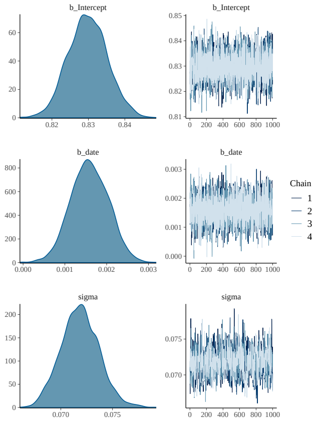
```
> print(slr_brms_w_prior, digits = 4)
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: y ~ date
   Data: data (Number of observations: 800)
  Draws: 4 chains, each with iter = 1500; warmup = 500; thin = 1;
         total post-warmup draws = 4000

Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI   Rhat Bulk_ESS Tail_ESS
Intercept   0.8298    0.0054   0.8196   0.8407 1.0009     3906     2517
date        0.0016    0.0005   0.0007   0.0025 1.0023     4510     2545

Family Specific Parameters:
      Estimate Est.Error l-95% CI u-95% CI   Rhat Bulk_ESS Tail_ESS
sigma   0.0719    0.0018   0.0684   0.0756 1.0051      789      999

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

And we have the priors we want:

```
> prior_summary(slr_brms_w_prior)
          prior     class coef group resp dpar nlpar lb ub  source
         (flat)         b                                  default
   normal(0, 1)         b date                                user
   normal(0, 1) Intercept                                     user
 exponential(1)     sigma                            0        user
```

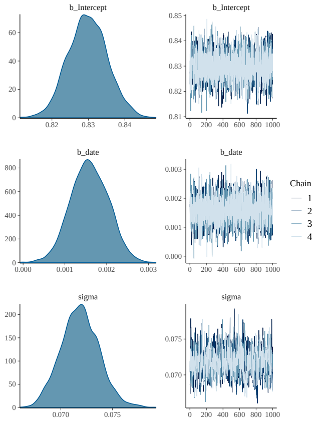
Easy to check the trace- and density-plots using the `plot` method

```
> plot(slr_brms_w_prior)
```



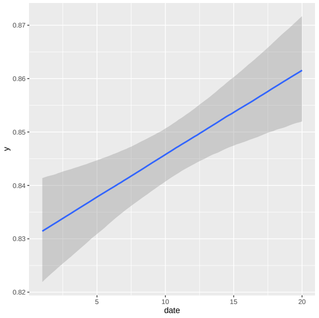
Easy to check the trace- and density-plots using the `plot` method

```
> plot(slr_brms_w_prior)
```



Or look into the conditional predictions with their credible intervals

```
> conditional_effects(
    slr_brms_w_prior,
    effect = "date"
  )
```



Fitting multilevel models are equally easy. We use the same formula syntax as `lme4`:

```
mlm_brms_w_prior = brms::brm(
    formula = y ~ date + (1 | topic), data = data,
    prior = c(
        set_prior("normal(0, 1)", class = "Intercept"),
        set_prior("normal(0, 1)", class = "b", coef = "date"),
        set_prior("exponential(1)", class = "sigma"),
        set_prior("exponential(1)", class = "sd")
    ),
    family = gaussian(), warmup = 1000, iter = 2000, refresh = 1000,
    chains = 4, cores = 4, seed = 123
)
```

```
> print(mlm_brms_w_prior, digits = 4)
Group-Level Effects:
~topic (Number of levels: 40)
              Estimate Est.Error l-95% CI u-95% CI   Rhat Bulk_ESS Tail_ESS
sd(Intercept)   0.0541    0.0066   0.0429   0.0690 1.0152      339      828

Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI   Rhat Bulk_ESS Tail_ESS
Intercept   0.8297    0.0099   0.8098   0.8497 1.0135      161      417
date        0.0016    0.0003   0.0010   0.0022 1.0002     3812     2746

Family Specific Parameters:
      Estimate Est.Error l-95% CI u-95% CI   Rhat Bulk_ESS Tail_ESS
sigma   0.0502    0.0013   0.0479   0.0528 1.0024     3328     2852
```

It’s also possible to extract the posterior draws directly:

```
> psamples = as_draws(mlm_brms_w_prior) # note: Extract draws

> class(psamples) # note: this is basically a ‘list‘ object
[1] "draws_list" "draws"      "list"

> length(psamples) # note: length is equal to the number of chains
[1] 4

> class(psamples[[1]]) # note: each element is again a ‘list’
[1] "list"

> names(psamples[[1]]) # note: which contains all the parameters
[1] "b_Intercept"           "b_date"                "sd_topic__Intercept"
[4] "sigma"                 "r_topic[1,Intercept]"  "r_topic[2,Intercept]"
...
[43] "r_topic[39,Intercept]" "r_topic[40,Intercept]" "lprior"
[46] "lp__"

> psamples[[1]][["b_date"]] # note: this would extract the samples
                                    of the regression coefficient from
                                    the first chain
   [1] 0.0017325622 0.0012639667 0.0018935499 ...
   [6] 0.0016589144 0.0020239859 0.0011239522 ...
   ...
 [991] 0.0014328793 0.0009396389 0.0021512139 ...
 [996] 0.0018722834 0.0016737117 0.0018758484 ...
```

Fitting Bayesian Models in `R` using `cmdstanr`

While `brms` and `rstanarm` are great packages, sometimes we need to code directly in `Stan`

While `brms` and `rstanarm` are great packages, sometimes we need to code directly in `Stan`

This happens most often when you are trying to fit a "non-standard" model

While `brms` and `rstanarm` are great packages, sometimes we need to code directly in `Stan`

This happens most often when you are trying to fit a "non-standard" model

There are two packages that let you directly interact with the `Stan` language from `R`: `rstan` and `cmdstanr`

While `brms` and `rstanarm` are great packages, sometimes we need to code directly in `Stan`

This happens most often when you are trying to fit a "non-standard" model

There are two packages that let you directly interact with the `Stan` language from `R`: `rstan` and `cmdstanr`

A `Stan` program consists of 7 code blocks:

1. `functions{}`
2. `data{}`
3. `transformed data{}`
4. `parameters{}`
5. `transformed parameters{}`
6. `model{}`
7. `generated quantities{}`

A `Stan` program consists of 7 code blocks:

1. `functions{}`
2. `data{}`
3. `transformed data{}`
4. `parameters{}`
5. `transformed parameters{}`
6. `model{}`
7. `generated quantities{}`

So, a typical `Stan` program will look like:

```
functions {

    <some user-defined functions here>

}

data {

    <data specifications here>

}

...

model {

    <model definition here>

}

generated quantities {

    <calculate some extra stuff here>

}
```

A `Stan` program consists of 7 code blocks:

1. `functions{}`
2. `data{}`
3. `transformed data{}`
4. `parameters{}`
5. `transformed parameters{}`
6. `model{}`
7. `generated quantities{}`

So, a typical `Stan` program will look like:

```
functions {

    <some user-defined functions here>

}

data {

    <data specifications here>

}

...

model {

    <model definition here>

}

generated quantities {

    <calculate some extra stuff here>

}
```

We will *not* deal with 1., 3. and 7. today

Let's return to the simple linear regression model:

$$y_i = \alpha + \beta x_i + \epsilon, \quad \sigma_\epsilon \text{ Normal}(0, \sigma_\epsilon)$$

or equivalently

$$y_i \sim \text{Normal}(\alpha + \beta x_i, \sigma_\epsilon)$$

Let's return to the simple linear regression model:

$$y_i = \alpha + \beta x_i + \epsilon, \quad \sigma_\epsilon \text{ Normal}(0, \sigma_\epsilon)$$

or equivalently

$$y_i \sim \text{Normal}(\alpha + \beta x_i, \sigma_\epsilon)$$

The data consists of two vectors:

1. the outcome $y = [y_1, ..., y_i, ..., y_N]^\top$; and
2. the predictor $x = [x_1, ..., x_i, ..., x_N]^\top$

Let's return to the simple linear regression model:

$$y_i = \alpha + \beta x_i + \epsilon, \quad \sigma_\epsilon \text{ Normal}(0, \sigma_\epsilon)$$

or equivalently

$$y_i \sim \text{Normal}(\alpha + \beta x_i, \sigma_\epsilon)$$

The data consists of two vectors:

1. the outcome $y = [y_1, ..., y_i, ..., y_N]^\top$; and
2. the predictor $x = [x_1, ..., x_i, ..., x_N]^\top$

We specify this in the data `data{}` block.

Let's return to the simple linear regression model:

$$y_i = \alpha + \beta x_i + \epsilon, \quad \sigma_\epsilon \text{ Normal}(0, \sigma_\epsilon)$$

or equivalently

$$y_i \sim \text{Normal}(\alpha + \beta x_i, \sigma_\epsilon)$$

The data consists of two vectors:

1. the outcome $y = [y_1, ..., y_i, ..., y_N]^\top$; and
2. the predictor $x = [x_1, ..., x_i, ..., x_N]^\top$

We specify this in the data `data{}` block.

```
data {

    int N;          // no. of obs.
    vector[N] x;    // predictor
    vector[N] y;    // outcome

}
```

The only parameters in the model are

The only parameters in the model are

1. $\alpha \in \mathbb{R}$: the intercept

The only parameters in the model are

1. $\alpha \in \mathbb{R}$: the intercept
2. $\beta \in \mathbb{R}$: the slope coefficient

The only parameters in the model are

1. $\alpha \in \mathbb{R}$: the intercept
2. $\beta \in \mathbb{R}$: the slope coefficient
3. $\sigma_\epsilon \in \mathbb{R}_+$, the residual standard deviation.

The only parameters in the model are

1. $\alpha \in \mathbb{R}$: the intercept
2. $\beta \in \mathbb{R}$: the slope coefficient
3. $\sigma_\epsilon \in \mathbb{R}_+$, the residual standard deviation.

We can declare them in the `parameters{}` block:

```
parameters {

    real alpha;
    real beta;
    real<lower = 0> sigma_epsilon;

}
```

Notice that we specified `real<lower = 0>` to indicate that σ_ϵ has to be positive

Lastly, we specify the (log) posterior density (up to a constant.)

Lastly, we specify the (log) posterior density (up to a constant.)This is equivalent to specifying the likelihood and the prior

Lastly, we specify the (log) posterior density (up to a constant.)This is equivalent to specifying the likelihood and the prior

Let us use the following weakly informative priors

$$\begin{aligned} \alpha &\sim \text{Normal}(0, 2) \\ \beta &\sim \text{Normal}(0, 1) \\ \sigma_\epsilon &\sim \text{Exponential}(1) \end{aligned}$$

As for the likelihood, notice that the model

$$y_i = \alpha + x_i\beta + \epsilon_i$$

implies that

$$y_i \sim \text{Normal}(\alpha + \beta x_i, \sigma_\epsilon)$$

As for the likelihood, notice that the model

$$y_i = \alpha + x_i\beta + \epsilon_i$$

implies that

$$y_i \sim \text{Normal}(\alpha + \beta x_i, \sigma_\epsilon)$$

Hence, the likelihood is

$$\begin{aligned} p(y \mid \alpha, \beta, \sigma_\epsilon) &= \prod_{i=1}^{N} \text{Normal}(\alpha + \beta x_i, \sigma_\epsilon) \\ &= \prod_{i=1}^{N} \text{Normal}(\hat{y}_i, \sigma_\epsilon), \end{aligned}$$

where $\hat{y}_i = \alpha + \beta x_i$.

We code this up in the `model` block as follows:

```
model {

    // linear predictor (local variable)
    vector[N] yhat;

    for (n in 1:N)
        yhat[n] = alpha + beta * x[n];

    // priors
    alpha ~ normal(0, 2);
    beta ~ normal(0, 1);
    sigma_epsilon ~ exponential(1);

    // vectorized likelihood
    y ~ normal(yhat, sigma_epsilon);

}
```

So, in sum, the Stan code will look like

```
data {

    int N;          // no. of obs.
    vector[N] x;    // predictor
    vector[N] y;    // outcome

}

parameters {

    real alpha;
    real beta;
    real<lower = 0> sigma_epsilon;

}

model {

    // linear predictor (local variable)
    vector[N] yhat;

    ...

    y ~ normal(yhat, sigma_epsilon);

}
```

Suppose that this file is saved in a file named `slr.stan`

Suppose that this file is saved in a file named `slr.stan`

From within `R`, we can compile the code with the `cmdstanr` package as follows:

Suppose that this file is saved in a file named `slr.stan`

From within `R`, we can compile the code with the `cmdstanr` package as follows:

```
> library("cmdstanr")
> mod = cmdstan_model("slr.stan")
```

Suppose that this file is saved in a file named `slr.stan`

From within `R`, we can compile the code with the `cmdstanr` package as follows:

```
> library("cmdstanr")
> mod = cmdstan_model("slr.stan")
```

After compiling the model, we need to provide it with data to generate posterior samples.

Suppose that this file is saved in a file named `slr.stan`

From within `R`, we can compile the code with the `cmdstanr` package as follows:

```
> library("cmdstanr")
> mod = cmdstan_model("slr.stan")
```

After compiling the model, we need to provide it with data to generate posterior samples. Usually, you provide the data as a `list` object:

Suppose that this file is saved in a file named `slr.stan`

From within `R`, we can compile the code with the `cmdstanr` package as follows:

```
> library("cmdstanr")
> mod = cmdstan_model("slr.stan")
```

After compiling the model, we need to provide it with data to generate posterior samples. Usually, you provide the data as a `list` object:

```
> standata = list(
    N = nrow(dat),
    x = dat$date,
    y = dat$y
  )
```

Sampling is then done by providing the `cmdstanr` object with the data and options:

```
fit = mod$sample(
    data = standata,
    chains = 4,
    parallel_chains = 4,
    iter_warmup = 1000,
    iter_sampling = 1000,
    refresh = 1000
)
```

Sampling is then done by providing the `cmdstanr` object with the data and options:

```
fit = mod$sample(
    data = standata,
    chains = 4,
    parallel_chains = 4,
    iter_warmup = 1000,
    iter_sampling = 1000,
    refresh = 1000
)
```

We can, thereafter, get summaries and extract the posterior samples with

```
> fit$summary()
# A tibble: 4 × 10
variable           mean  median      sd     mad      q5     q95  rhat ess_bulk
<chr>             <num>   <num>   <num>   <num>   <num>   <num> <num>    <num>
1 lp__           1.70e+3 1.70e+3 1.24e+0 1.01e+0 1.70e+3 1.71e+3  1.00    1283.
2 alpha          8.30e-1 8.30e-1 5.34e-3 5.29e-3 8.21e-1 8.39e-1  1.00    1533.
3 beta           1.59e-3 1.59e-3 4.45e-4 4.37e-4 8.54e-4 2.32e-3  1.00    1779.
4 sigma_epsilon  7.19e-2 7.19e-2 1.80e-3 1.77e-3 6.90e-2 7.49e-2  1.00    1377.
> psamples2 = fit$draws()
```

which returns, as before, a `draws` object.

One nice thing about the `cmdstanr` package is that all cutting-edge `Stan` algorithms are available

One nice thing about the `cmdstanr` package is that all cutting-edge `Stan` algorithms are available

For example:

```
# Auto-diff variational Bayes
vb = mod$variational(data = standata)

# penalized MLE (L-BFGS)
pmle = = mod$optimize(data = standata)

# pathfinder approximation
pfinder = mod$pathfinder(data = standata)

# laplace approximation
laplace = mod$laplace(data = standata)
```

The `Stan` program for the random intercept model is a bit more complicated

The `Stan` program for the random intercept model is a bit more complicated

We start with the data-structure we need to provide `Stan`

```
standat_mlm = list(
    N     = nrow(dat), # total obs.
    J     = length(unique(dat$topic)) # no of topics
    topic = dat$topic,
    x     = dat$date,
    y     = dat$y
)
```

The Stan program for the random intercept model is a bit more complicated

We start with the data-structure we need to provide Stan

```
standat_mlm = list(
    N     = nrow(dat), # total obs.
    J     = length(unique(dat$topic)) # no of topics
    topic = dat$topic,
    x     = dat$date,
    y     = dat$y
)
```

Similarly, the data{} block in our Stan code is expanded:

```
data {

    int N;              // no. of obs.
    int J;              // no. of topics.
    array[N] int topic; // topic indicator
    vector[N] x;        // predictor
    vector[N] y;        // outcome

}
```

The array[N] int object is an array (vector) of integers (you can think of it as std::vector<int> and vector[N] as Eigen::VectorXd)

The `parameter` block would need 3 new elements:

The `parameter` block would need 3 new elements:

1. The mean of the random intercepts, μ_α

The `parameter` block would need 3 new elements:

1. The mean of the random intercepts, μ_α
2. The standard deviation of the random intercepts, σ_α

The `parameter` block would need 3 new elements:

1. The mean of the random intercepts, μ_α
2. The standard deviation of the random intercepts, σ_α
3. A length-J vector of random intercepts, $[\alpha_1, ..., \alpha_J]^\top$

The `parameter` block would need 3 new elements:

1. The mean of the random intercepts, μ_α
2. The standard deviation of the random intercepts, σ_α
3. A length-J vector of random intercepts, $[\alpha_1, ..., \alpha_J]^\top$

Here it becomes complicated...

The `parameter` block would need 3 new elements:

1. The mean of the random intercepts, μ_α
2. The standard deviation of the random intercepts, σ_α
3. A length-J vector of random intercepts, $[\alpha_1, ..., \alpha_J]^\top$

Here it becomes complicated...While we can simply "sample" $\alpha_i \sim \mathrm{N}(\mu_\alpha, \sigma_\alpha)$, `Stan` works much better when sampling from $\mathrm{N}(0, 1)$.

The `parameter` block would need 3 new elements:

1. The mean of the random intercepts, μ_α
2. The standard deviation of the random intercepts, σ_α
3. A length-J vector of random intercepts, $[\alpha_1, ..., \alpha_J]^\top$

Here it becomes complicated...While we can simply "sample" $\alpha_i \sim \mathrm{N}(\mu_\alpha, \sigma_\alpha)$, `Stan` works much better when sampling from $\mathrm{N}(0, 1)$.

So, we'll use the "Matt trick" and sample $\alpha_i^{\text{raw}} \sim \mathrm{N}(0, 1)$, and calculate

$$\alpha_i = \mu_\alpha + \sigma_\alpha * \alpha_i^{\text{raw}},$$

which *induces*

$$\alpha_i \sim \mathrm{N}(\mu_\alpha, \sigma_\alpha)$$

The `parameter` block would need 3 new elements:

1. The mean of the random intercepts, μ_α
2. The standard deviation of the random intercepts, σ_α
3. A length-J vector of random intercepts, $[\alpha_1, ..., \alpha_J]^\top$

Here it becomes complicated...While we can simply "sample" $\alpha_i \sim \mathrm{N}(\mu_\alpha, \sigma_\alpha)$, `Stan` works much better when sampling from $\mathrm{N}(0, 1)$.

So, we'll use the "Matt trick" and sample $\alpha_i^{\mathrm{raw}} \sim \mathrm{N}(0, 1)$, and calculate

$$\alpha_i = \mu_\alpha + \sigma_\alpha * \alpha_i^{\mathrm{raw}},$$

which *induces*

$$\alpha_i \sim \mathrm{N}(\mu_\alpha, \sigma_\alpha)$$

This will make us use the `transformed parameter` block...

Hence, the Stan code looks like:

```
parameters {

    // regression coef
    real beta;
    // resid std. dev.
    real<lower = 0> sigma_epsilon;

    // grand mean of random intercepts
    real mu_alpha;
    // std. dev. of random intercepts
    real<lower = 0> sigma_alpha;
    // aux var for efficient samping
    vector[J] alpha_raw;

}

transformed parameters {

    // random intercepts
    // note: alpha ~ Normal(mu_alpha, sigma_alpha^2)
    vector[J] alpha = alpha_raw * sigma_alpha + mu_alpha;

}
```

The `model` block remains almost the same:

```
model {

    // linear predictor (local variable)
    vector[N] yhat;

    for (n in 1:N)
        yhat[n] = alpha[topic[n]] + beta * x[n];

    // priors
    beta ~ normal(0, 1);
    sigma_epsilon ~ exponential(1);

    mu_alpha ~ normal(0, 2);
    sigma_alpha ~ exponential(1);
    alpha_raw ~ normal(0, 1);

    // vectorized likelihood
    y ~ normal(yhat, sigma_epsilon);

}
```

Assuming the `Stan` code is stored in the file `re.stan`, we can compile and obtain posterior draws as before.

Assuming the Stan code is stored in the file re.stan, we can compile and obtain posterior draws as before.

This time, let's try out the Pathfinder algorithm

```
> mlm = cmdstan_model(here("example", "re.stan"))
> pf = mlm$pathfinder(data = standata_mlm)
> pf$print(digits = 5)
     variable       mean     median      sd     mad         q5        q95
lp_approx__     21.43907   21.54090 6.61650 5.20230    4.55942   32.03380
lp__          1195.49274 1197.07000 6.14990 5.12980 1179.50000 1204.89250
beta           0.00150    0.00150 0.00042 0.00047    0.00084    0.00214
sigma_epsilon  0.04941    0.04940 0.00109 0.00087    0.04750    0.05133
mu_alpha       0.83045    0.83125 0.00349 0.00226    0.82258    0.83693
sigma_alpha    0.05272    0.04976 0.00686 0.00390    0.04561    0.06513
alpha[1]       0.89406    0.89628 0.00915 0.00861    0.87887    0.90943
alpha[2]       0.83116    0.83402 0.01069 0.01407    0.81564    0.84480
...
alpha[39]      0.90629    0.90522 0.01037 0.01258    0.88813    0.91947
alpha[40]      0.87823    0.87935 0.00962 0.00686    0.85935    0.89172
```

These results are very close to the (gold-standard) HMC results!

(As before, we could anlayze the results further using mlm$draws() to obtain the posterior darws)