Prior Predictive Checks: Grammaticality Judgment Example

Bayesian Mixed Effects Models with brms for Linguists

Workshop Materials

2025-11-11

Table of contents

1 Prior Predictive Checks for Binary Data

This document demonstrates how to validate priors for a Bayesian binary model (grammaticality judgments) before fitting to data.

1.1 Setup

```r
library(brms)
library(tidyverse)
library(bayesplot)
library(posterior)  # For as_draws_df() - following Kurz's approach

# Create example grammaticality judgment data
set.seed(42)
n_subj <- 25
n_trials <- 40
n_items <- 30

gram_data <- expand.grid(
  trial = 1:n_trials,
  subject = 1:n_subj,
  item = 1:n_items
) %>%
  filter(row_number() <= n_subj * n_trials * 2) %>%
  mutate(
    condition = rep(c("A", "B"), length.out = n()),
    p_correct = plogis(0.2 + (condition == "B") * 0.4),
    correct = rbinom(n(), size = 1, prob = p_correct)
  )

# Define priors for binary model
gram_priors <- c(
  prior(normal(0, 1.5), class = Intercept),
  prior(normal(0, 1), class = b),
  prior(exponential(1), class = sd),
  prior(lkj(2), class = cor)
)
```

1.2 Fitting Prior Only Model

```r
# Create fits directory if it doesn't exist
if (!dir.exists("fits")) dir.create("fits")

# Fit model with priors only
# Using file argument to cache the fitted model and speed up re-runs
```

```r
prior_pred_gram <- brm(
  correct ~ condition + (1 + condition | subject) + (1 | item),
  data = gram_data,
  family = bernoulli(),
  prior = gram_priors,
  sample_prior = "only",
  chains = 4, iter = 1000,  # 4 chains, fewer iterations for prior checks
  cores = 4,                 # Use 4 cores for parallel sampling
  verbose = FALSE,
  refresh = 0,
  file = "fits/prior_pred_gram",  # Cache model - only refits if data/formula/priors change
  file_refit = "on_change"         # Only refit when necessary
)
```

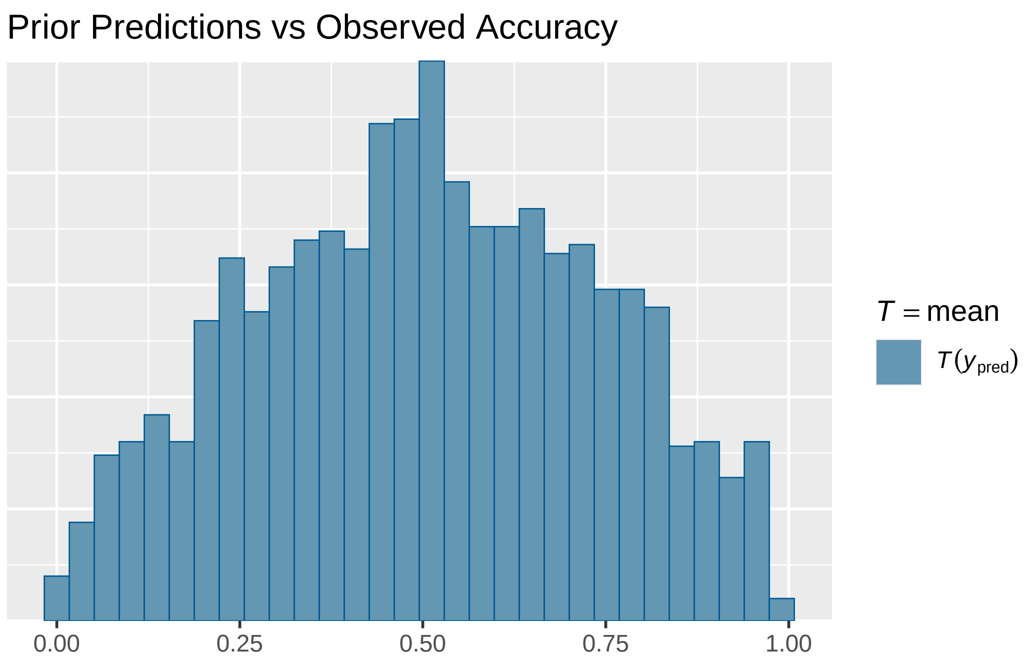
1.3 Prior Predictive Checks

1.3.1 Visual Checks

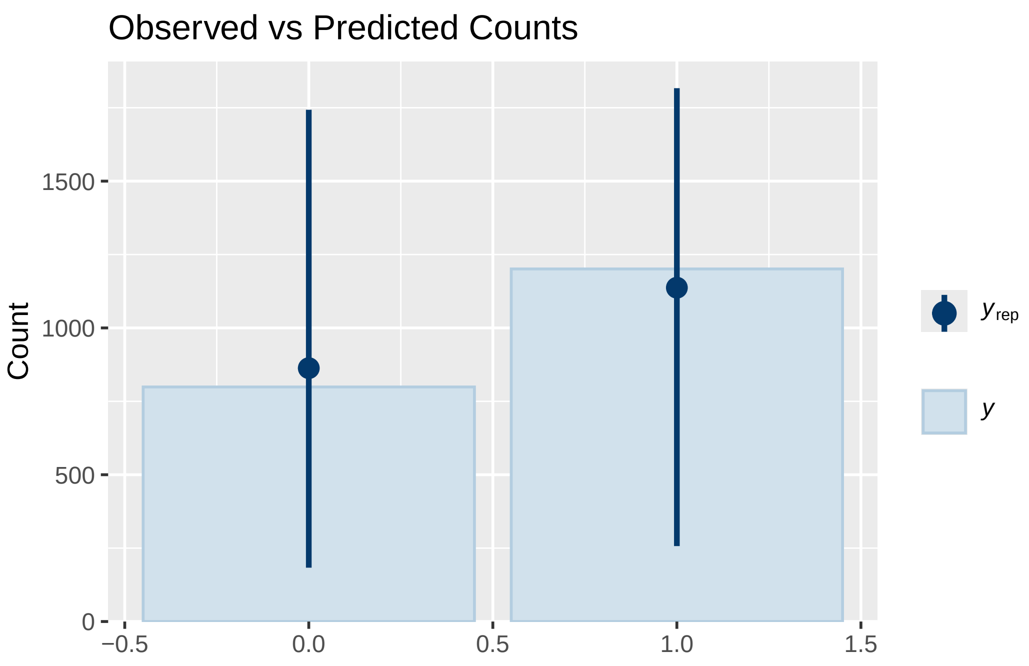
```r
# Mean comparison (proportion correct)
pp_check(prior_pred_gram, type = "stat", stat = "mean", prefix = "ppd") +
  labs(title = "Prior Predictions vs Observed Accuracy")
```

```
Using all posterior draws for ppc type 'stat' by default.
```

```
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```

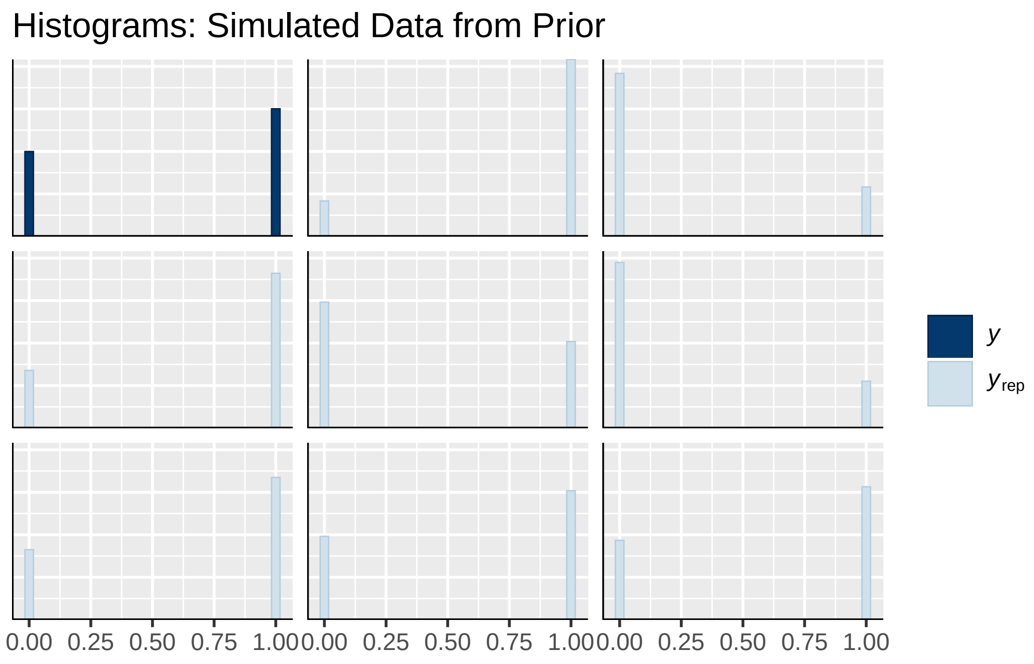


```
# Bar plot
pp_check(prior_pred_gram, type = "bars", ndraws = 100) +
  labs(title = "Observed vs Predicted Counts")
```



```r
# Histogram of simulated datasets
pp_check(prior_pred_gram, type = "hist", ndraws = 8) +
  labs(title = "Histograms: Simulated Data from Prior")
```

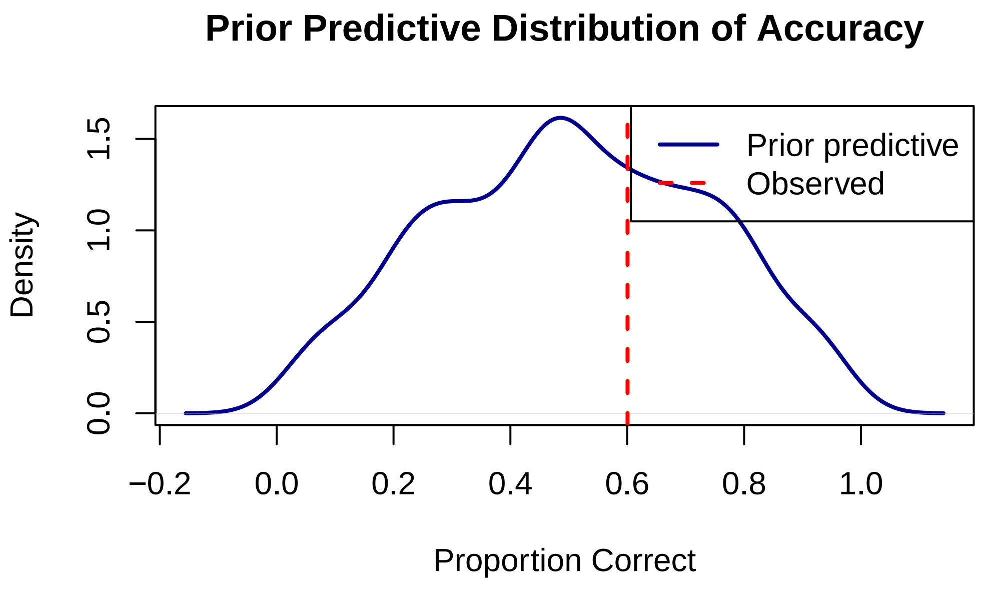
```
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



1.3.2 Prior Predictive Accuracy Distribution

```r
prior_pred_samples <- posterior_predict(prior_pred_gram, ndraws = 1000)
obs_acc <- mean(gram_data$correct)
pred_acc <- apply(prior_pred_samples, 1, mean)

plot(density(pred_acc),
     main = "Prior Predictive Distribution of Accuracy",
     xlab = "Proportion Correct",
     lwd = 2, col = "darkblue")
abline(v = obs_acc, col = "red", lwd = 2, lty = 2)
legend("topright", c("Prior predictive", "Observed"),
       col = c("darkblue", "red"), lwd = 2, lty = c(1, 2))
```



1.4 Prior Distributions

Following Kurz's approach, we extract prior samples using `as_draws_df()` from the posterior package.

```r
# Extract prior draws as a data frame (Kurz's method)
# This works seamlessly with tidyverse and gives us a proper data frame
prior_samples <- as_draws_df(prior_pred_gram)
```

1.4.1 Intercept Prior

```r
# Now we can access columns directly from the data frame
intercept_vals <- prior_samples$b_Intercept

cat("Intercept prior (log-odds scale):\n")
```

```
Intercept prior (log-odds scale):
```

```r
intercept_logodds <- quantile(intercept_vals, c(0.025, 0.5, 0.975), na.rm = TRUE)
print(intercept_logodds)
```

```
      2.5%         50%       97.5% 
-2.98728595  0.04194952  3.19327008 
```

```
cat("\nIntercept prior (probability scale):\n")
```

```

Intercept prior (probability scale):
```

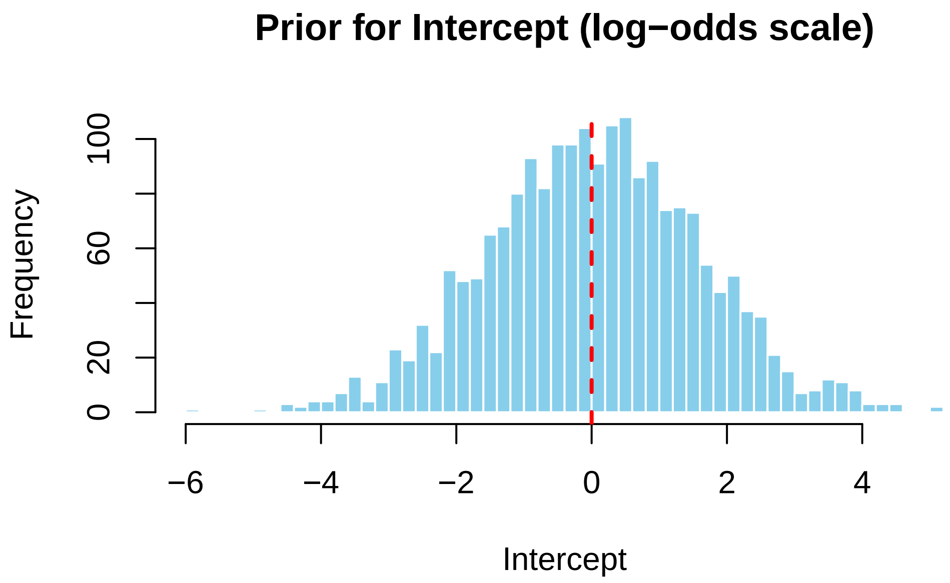
```
intercept_prob <- plogis(intercept_logodds)
print(intercept_prob)
```

```
      2.5%        50%      97.5% 
0.04800357 0.51048584 0.96058023 
```

```
cat("Prior expects", round(intercept_prob[1]*100), "-",
    round(intercept_prob[3]*100), "% baseline accuracy (median",
    round(intercept_prob[2]*100), "%)\n")
```

```
Prior expects 5 - 96 % baseline accuracy (median 51 %)
```

```
# Visualization
hist(intercept_vals,
     main = "Prior for Intercept (log-odds scale)",
     xlab = "Intercept",
     breaks = 50, col = "skyblue", border = "white")
abline(v = 0, col = "red", lwd = 2, lty = 2)
```



1.4.2 Effect Size Prior

```r
effect_vals <- prior_samples$b_conditionB

cat("Condition effect prior (log-odds scale):\n")
```

```
Condition effect prior (log-odds scale):
```

```r
effect_logodds <- quantile(effect_vals, c(0.025, 0.5, 0.975), na.rm = TRUE)
print(effect_logodds)
```

```
       2.5%         50%       97.5% 
-1.93961957 -0.01510226  1.96317214 
```

```r
cat("\nCondition effect (odds multiplier):\n")
```

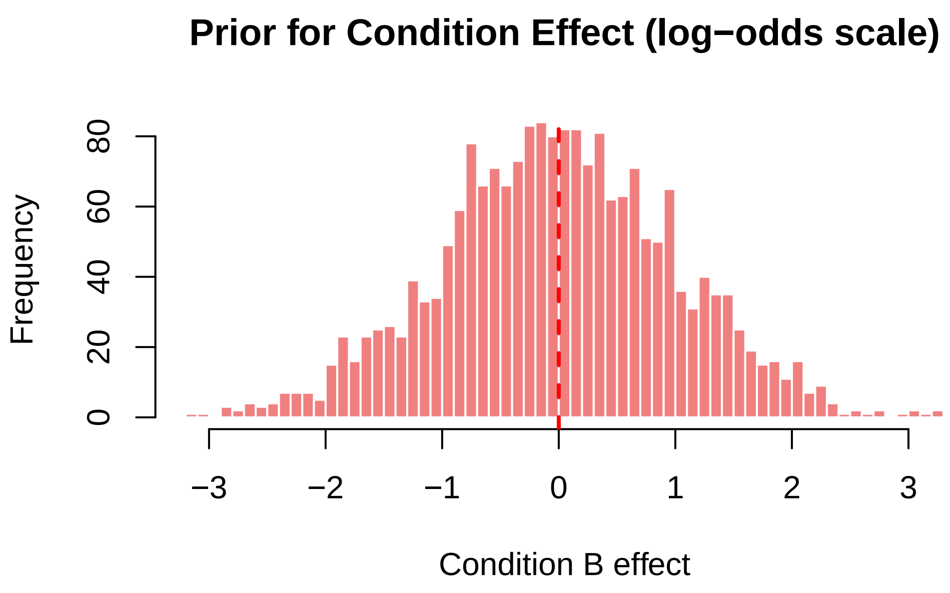
```

Condition effect (odds multiplier):
```

```r
print(exp(effect_logodds))
```

```
     2.5%       50%     97.5% 
0.1437586 0.9850112 7.1218829 
```

```r
# Visualization
hist(effect_vals,
     main = "Prior for Condition Effect (log-odds scale)",
     xlab = "Condition B effect",
     breaks = 50, col = "lightcoral", border = "white")
abline(v = 0, col = "red", lwd = 2, lty = 2)
```



1.5 Random Effects Distributions

For prior predictive checks, we examine the **implied distribution** of subject-specific parameters by extracting the hyperprior SDs and simulating random effects.

1.5.1 Subject Random Intercepts

```r
# Extract hyperprior SDs from prior samples
sd_subject_intercept <- prior_samples$sd_subject__Intercept

cat("Subject random intercept SD prior:\n")
```

```
Subject random intercept SD prior:
```

```r
print(quantile(sd_subject_intercept, c(0.025, 0.5, 0.975)))
```

```
      2.5%        50%      97.5% 
0.02900055 0.68343780 3.86970012 
```

```r
# Simulate random effects from this prior
set.seed(123)
n_sims <- 1000
simulated_intercepts <- rnorm(n_sims, mean = 0, sd = median(sd_subject_intercept))

cat("\nImplied subject random intercepts (log-odds scale):\n")
```

```
Implied subject random intercepts (log-odds scale):
```

```r
subject_int <- quantile(simulated_intercepts, c(0.025, 0.5, 0.975))
print(subject_int)
```

```
        2.5%          50%        97.5%
-1.326931618  0.006294215  1.392768861
```

```r
cat("\nImplied subject-specific accuracy (probability scale):\n")
```

```
Implied subject-specific accuracy (probability scale):
```

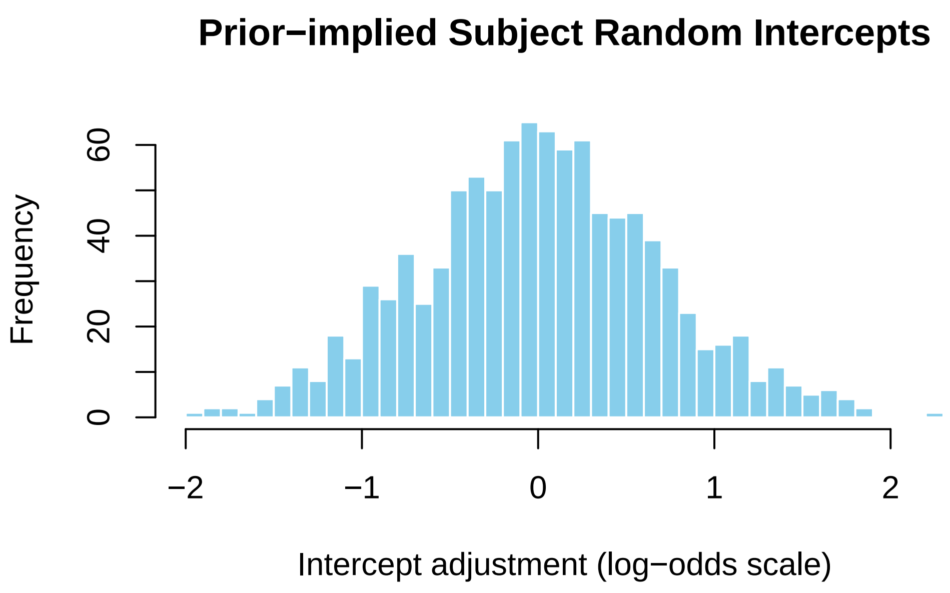
```r
# Combine with typical intercept value from prior
intercept_median <- median(intercept_vals)
subject_prob <- plogis(intercept_median + subject_int)
print(subject_prob)
```

```
     2.5%       50%     97.5%
0.2167034 0.5120586 0.8076354
```

```r
cat("Prior implies subject accuracy ranges from",
    round(subject_prob[1]*100), "% to",
    round(subject_prob[3]*100), "%\n")
```

```
Prior implies subject accuracy ranges from 22 % to 81 %
```

```r
# Visualization
hist(simulated_intercepts,
     main = "Prior-implied Subject Random Intercepts",
     xlab = "Intercept adjustment (log-odds scale)",
     breaks = 30, col = "skyblue", border = "white")
```



1.5.2 Subject Random Slopes

```
# Extract hyperprior SDs for slopes
sd_subject_slope <- prior_samples$sd_subject__conditionB

cat("Subject random slope SD prior:\n")
```

```
Subject random slope SD prior:
```

```
print(quantile(sd_subject_slope, c(0.025, 0.5, 0.975)))
```

```
      2.5%        50%      97.5%
0.02717447 0.68593886 3.63193775
```

```
# Simulate random slope effects
simulated_slopes <- rnorm(n_sims, mean = 0, sd = median(sd_subject_slope))

cat("\nImplied subject random slopes (log-odds scale):\n")
```

```

Implied subject random slopes (log-odds scale):
```

```
subject_slope <- quantile(simulated_slopes, c(0.025, 0.5, 0.975))
print(subject_slope)
```

```
       2.5%         50%       97.5% 
-1.36609138  0.03762538  1.30733868 
```

```
cat("\nImplied subject-specific effects (odds multipliers):\n")
```

```

Implied subject-specific effects (odds multipliers):
```

```
print(exp(subject_slope))
```

```
     2.5%       50%     97.5% 
0.2551021 1.0383422 3.6963235 
```

```
cat("Weak effect subjects (2.5%): ", round(exp(subject_slope[1]), 2), "× odds multiplier\n")
```

```
Weak effect subjects (2.5%):  0.26 × odds multiplier
```

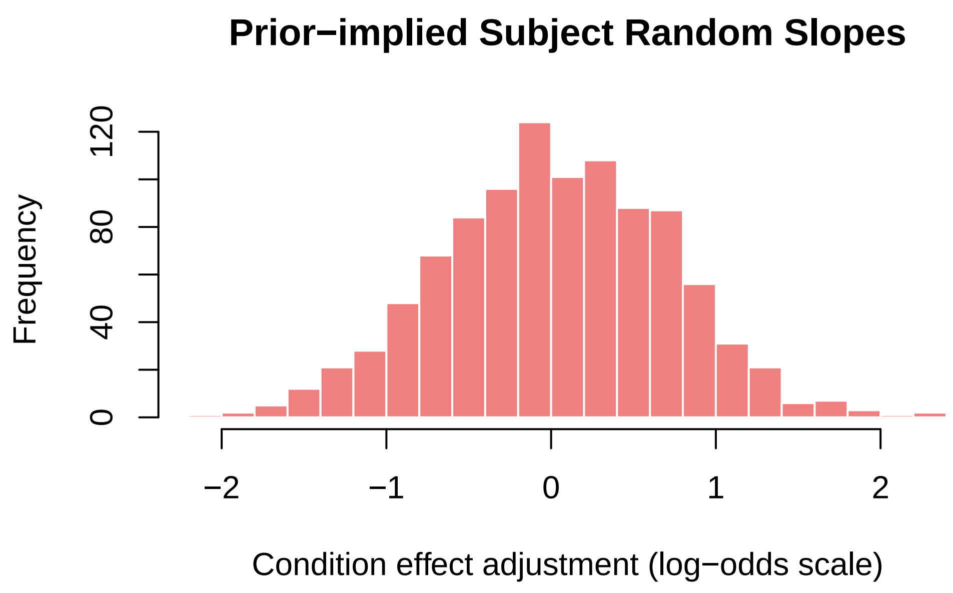
```
cat("Average effect subjects (50%): ", round(exp(subject_slope[2]), 2), "× odds multiplier\n
```

```
Average effect subjects (50%):  1.04 × odds multiplier
```

```
cat("Strong effect subjects (97.5%): ", round(exp(subject_slope[3]), 2), "× odds multiplier\
```

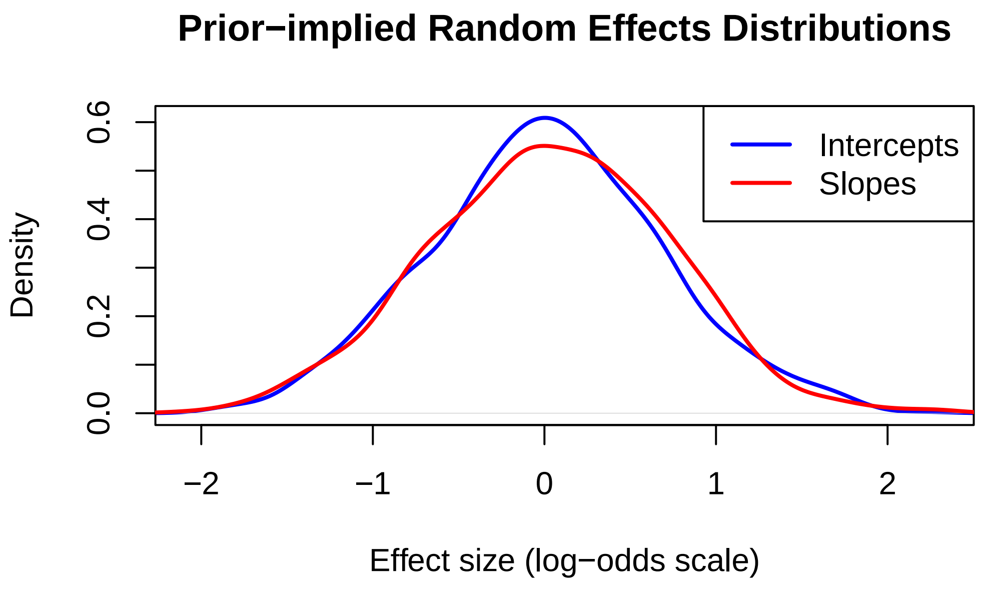
```
Strong effect subjects (97.5%):  3.7 × odds multiplier
```

```
# Visualization
hist(simulated_slopes,
     main = "Prior-implied Subject Random Slopes",
     xlab = "Condition effect adjustment (log-odds scale)",
     breaks = 30, col = "lightcoral", border = "white")
```



1.5.3 Random Effects Comparison

```
# Density plot comparing intercepts and slopes
plot(density(simulated_intercepts),
     main = "Prior-implied Random Effects Distributions",
     xlab = "Effect size (log-odds scale)",
     col = "blue", lwd = 2, xlim = range(c(simulated_intercepts, simulated_slopes)),
     ylim = c(0, max(density(simulated_intercepts)$y, density(simulated_slopes)$y)))
lines(density(simulated_slopes),
      col = "red", lwd = 2)
legend("topright", c("Intercepts", "Slopes"), col = c("blue", "red"), lwd = 2)
```



1.6 Interpretation

1.6.1 Good Signs (Prior is Reasonable)

- Prior generates ~50% baseline accuracy (Intercept   0)
- Condition effect varies but is typically moderate
- Between-subject accuracy ranges 30-70% or similar
- No impossible values (all between 0 and 1)

1.6.2 Problems to Watch For

- Mean accuracy » 90%: intercept prior too high
- All subjects 50% ± 1%: intercept SD too small
- Very weak prior predictions: slopes prior too small

1.7 Summary

Before fitting your model to actual data, always validate that your priors: 1. Generate reasonable predictions for your domain 2. Allow sufficient flexibility for the data to inform the posterior 3. Respect constraints (e.g., 0-1 for probabilities) 4. Account for variation across subjects/items

Adjust priors as needed and rerun these checks.