dirichletprocess Package GSoC-25 Tasks Analysis

Priyanshu Tiwari

2025-03-09

Contents

Introduction

This notebook is the analysis of the three tasks asked for the project "Improving the performance of multivariate normal models in dirichletprocess".

1. **Easy**: Fit normal mixture models to standard datasets
2. **Medium**: Analyze the effect of alpha priors on a lognormal mixture
3. **Hard**: Implement a custom Gamma mixture model

Installing the required packages:

```r
if (!require("dirichletprocess")) {
  install.packages("dirichletprocess")
}

library(dirichletprocess)
library(ggplot2)
library(coda)
```

Task 1: Normal Mixture Models

In this task, we'll fit Dirichlet process mixture models with Gaussian kernels to:

- The `faithful` dataset (univariate)
- The `iris` dataset (multivariate)

Univariate Model with Faithful Dataset

```r
data(faithful)
head(faithful)
```

```
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55
```

```r
faithful_scaled <- scale(faithful$waiting)

set.seed(123)
dp_faithful <- DirichletProcessGaussian(faithful_scaled)
dp_faithful <- Fit(dp_faithful, 1000)
```

Examining the results:

```r
print(dp_faithful)
```

```
## Dirichlet process object run for 1000 iterations.
##
##   Mixing distribution          normal
##   Base measure parameters  0, 1, 1, 1
##   Alpha Prior parameters         2, 4
##   Conjugacy                 conjugate
##   Sample size                     272
##
##   Mean number of clusters        4.41
##   Median alpha                   0.50
```

```r
# number of clusters found
num_clusters <- length(unique(dp_faithful$clusterLabels))
cat("Number of clusters found:", num_clusters, "\n")
```

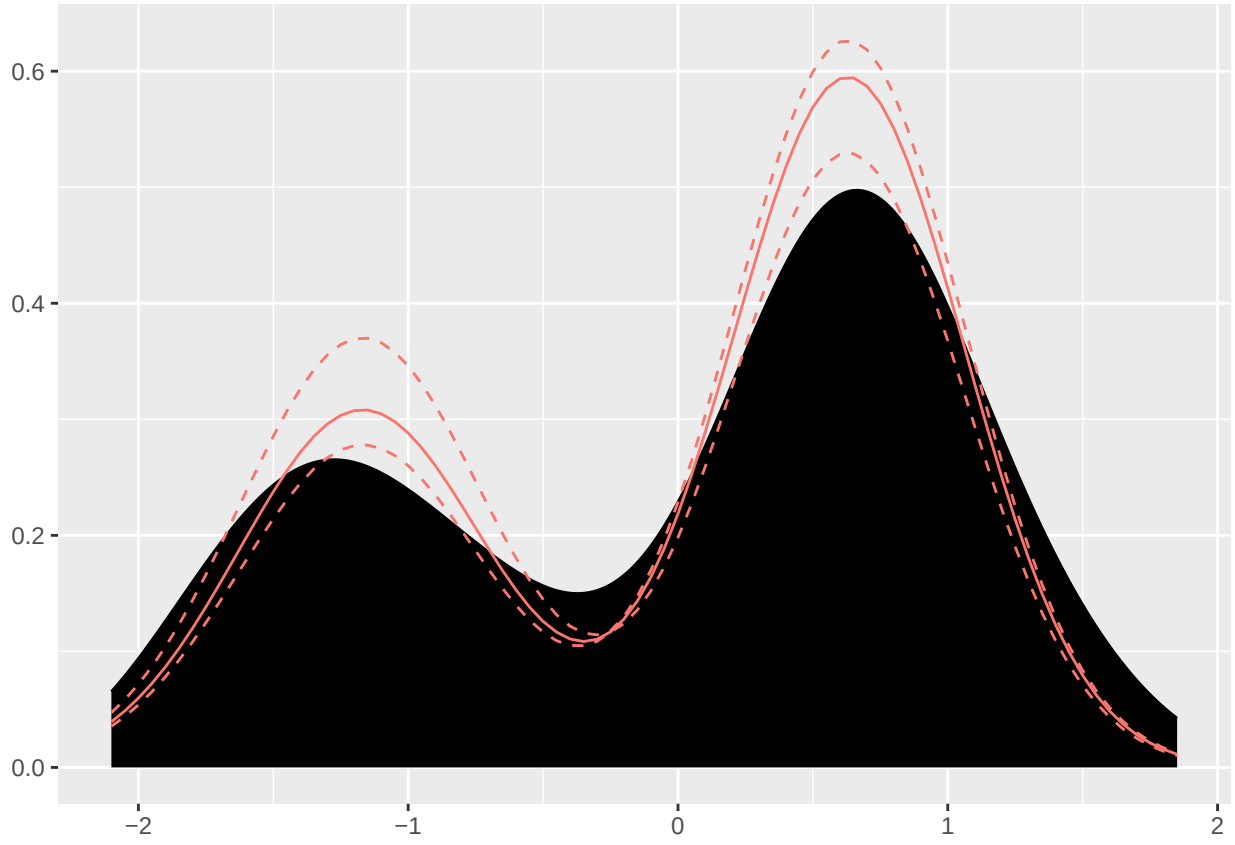
```
## Number of clusters found: 2
```

```r
# cluster sizes
table(dp_faithful$clusterLabels)
```

```
##
##   1   2
## 174  98
```

Density plot:

```r
plot(dp_faithful)
```



Multivariate Model with Iris Dataset

We fit a multivariate normal model to the `iris` dataset which contains measurements of four features for three species of iris flowers.

```r
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```r
iris_scaled <- scale(iris[, 1:4])

set.seed(456)
dp_iris <- DirichletProcessMvnormal(iris_scaled)
dp_iris <- Fit(dp_iris, 1000)
```

Results:

```r
print(dp_iris)
```

```
## Dirichlet process object run for 1000 iterations.
##
##   Mixing distribution                                                                                  mvnormal
##   Base measure parameters  c(0, 0, 0, 0), c(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1), 4, 4
##   Alpha Prior parameters                                                                                  2, 4
##   Conjugacy                                                                                          conjugate
##   Sample size                                                                                              150
##
##   Mean number of clusters                                                                                 1.17
##   Median alpha                                                                                            0.20
```

```r
# number of clusters found
unique_clusters <- length(unique(dp_iris$clusterLabels))
cat("Number of clusters found in iris data:", unique_clusters, "\n")
```

```
## Number of clusters found in iris data: 2
```

```r
# cluster sizes
table(dp_iris$clusterLabels)
```

```
##
##   1   2
##  50 100
```

We plot the clustering. For multivariate data, this will show the first two dimensions with points colored by their cluster assignments:

```r
plot(dp_iris)
```

Task 2: Lognormal Mixture and Alpha Prior Effects

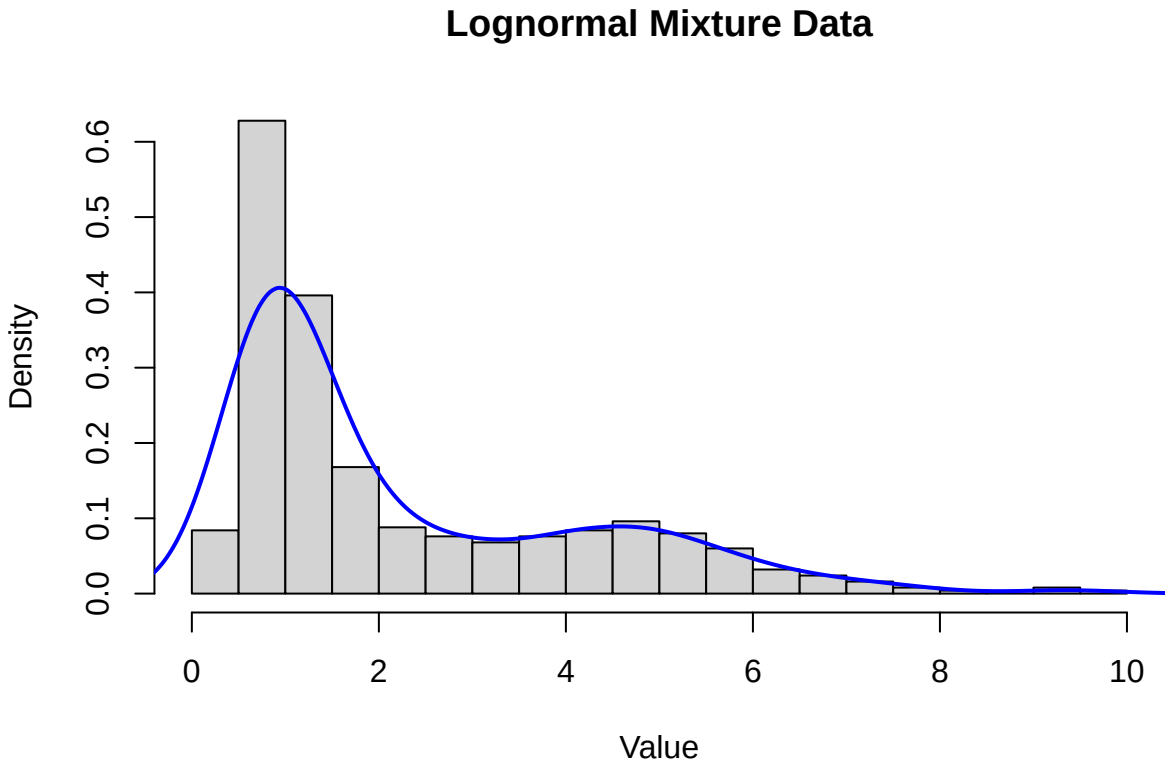
In this task, we'll:

- Generate data from a lognormal mixture model
- Fit a Dirichlet process with different alpha priors
- Sample from the posterior
- Analyze how the alpha prior affects clustering

Generate Data from a Lognormal Mixture

```r
set.seed(123)
n <- 500
component1 <- rlnorm(n * 0.7, meanlog = 0, sdlog = 0.5)  # 70% from component1
component2 <- rlnorm(n * 0.3, meanlog = 1.5, sdlog = 0.3)  # 30% from component2
lnorm_data <- c(component1, component2)
```

```r
# Histogram of the original data
hist(lnorm_data, breaks = 30, freq = FALSE,
     main = "Lognormal Mixture Data", xlab = "Value")
lines(density(lnorm_data), col = "blue", lwd = 2)
```



Fit Models with Different Alpha Priors

We'll fit three models with different alpha priors to see how they affect the clustering:

```r
# Scale the data
lnorm_data_scaled <- scale(lnorm_data)

# 1. Default prior: alphaPriors = c(2, 4)
dp_lnorm1 <- DirichletProcessGaussian(lnorm_data_scaled)
dp_lnorm1 <- Fit(dp_lnorm1, 2000)

# 2. Prior encouraging fewer clusters
dp_lnorm2 <- DirichletProcessGaussian(lnorm_data_scaled, alphaPriors = c(1, 2))
dp_lnorm2 <- Fit(dp_lnorm2, 2000)

# 3. Prior encouraging more clusters
dp_lnorm3 <- DirichletProcessGaussian(lnorm_data_scaled, alphaPriors = c(5, 1))
dp_lnorm3 <- Fit(dp_lnorm3, 2000)
```

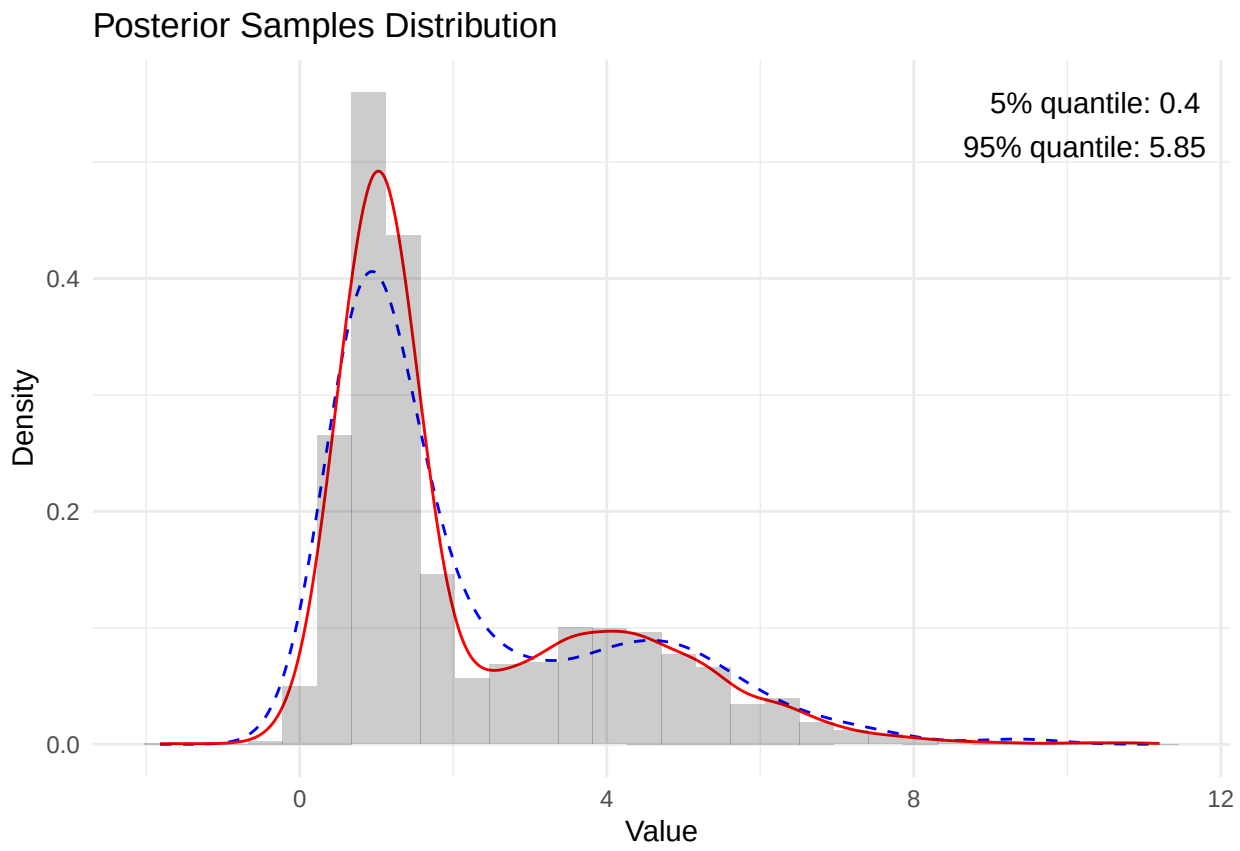
Sample from the Posterior

We sample from the posterior of the first model and calculate the 5% and 95% quantiles:

```
# Posterior density estimates
x_grid <- seq(min(lnorm_data_scaled), max(lnorm_data_scaled), length.out = 1000)
posterior_frame <- PosteriorFrame(dp_lnorm1, x_grid, ndraws=1000)

# Generating actual posterior samples
n_samples <- 5000
cluster_probs <- dp_lnorm1$pointsPerCluster / sum(dp_lnorm1$pointsPerCluster)
selected_clusters <- sample(1:dp_lnorm1$numberClusters, n_samples, replace=TRUE, prob=cluster_probs)

# For each selected cluster, generating a sample from the Gaussian
# with that cluster's parameters
posterior_samples <- numeric(n_samples)
for (i in 1:n_samples) {
  cluster <- selected_clusters[i]
  mu <- dp_lnorm1$clusterParameters[[1]][,,cluster]
  sigma <- dp_lnorm1$clusterParameters[[2]][,,cluster]
  posterior_samples[i] <- rnorm(1, mu, sigma)
}

# Transform back to original scale
posterior_samples_original <- posterior_samples * sd(lnorm_data) + mean(lnorm_data)

# Calculating quantiles from the generated samples
quantiles <- quantile(posterior_samples_original, c(0.05, 0.95))
cat("5% and 95% quantiles of posterior samples:", quantiles[1], "and", quantiles[2], "\n")
```

```
## 5% and 95% quantiles of posterior samples: 0.402768 and 5.848588
```

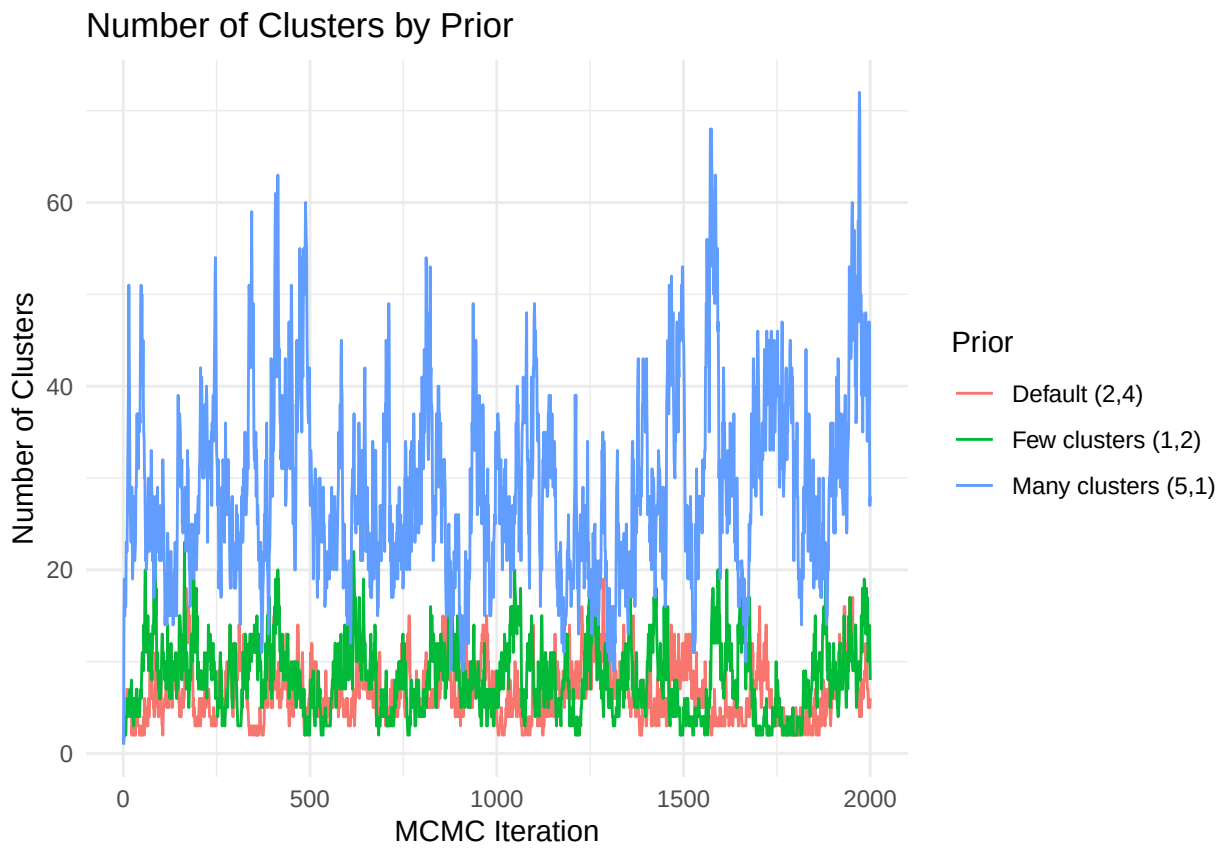
```
# Plot the posterior samples and density
ggplot() +
  # Original data density
  geom_density(data = data.frame(x = lnorm_data), aes(x = x),
               color = "blue", linetype = 2) +
  # Posterior samples density
  geom_density(data = data.frame(x = posterior_samples_original),
               aes(x = x), color = "red") +
  # Histogram of posterior samples
  geom_histogram(data = data.frame(x = posterior_samples_original),
                 aes(x = x, y = after_stat(density)), bins = 30,
                 alpha = 0.2, fill = "black") +
  labs(title = "Posterior Samples Distribution",
       x = "Value", y = "Density") +
  theme_minimal() +
  annotate("text", x = Inf, y = Inf,
           label = paste("5% quantile:", round(quantiles[1], 2),
                         "\n95% quantile:", round(quantiles[2], 2)),
           hjust = 1.1, vjust = 1.5, size = 4)
```



Analyze the Effect of Alpha Priors on Clustering

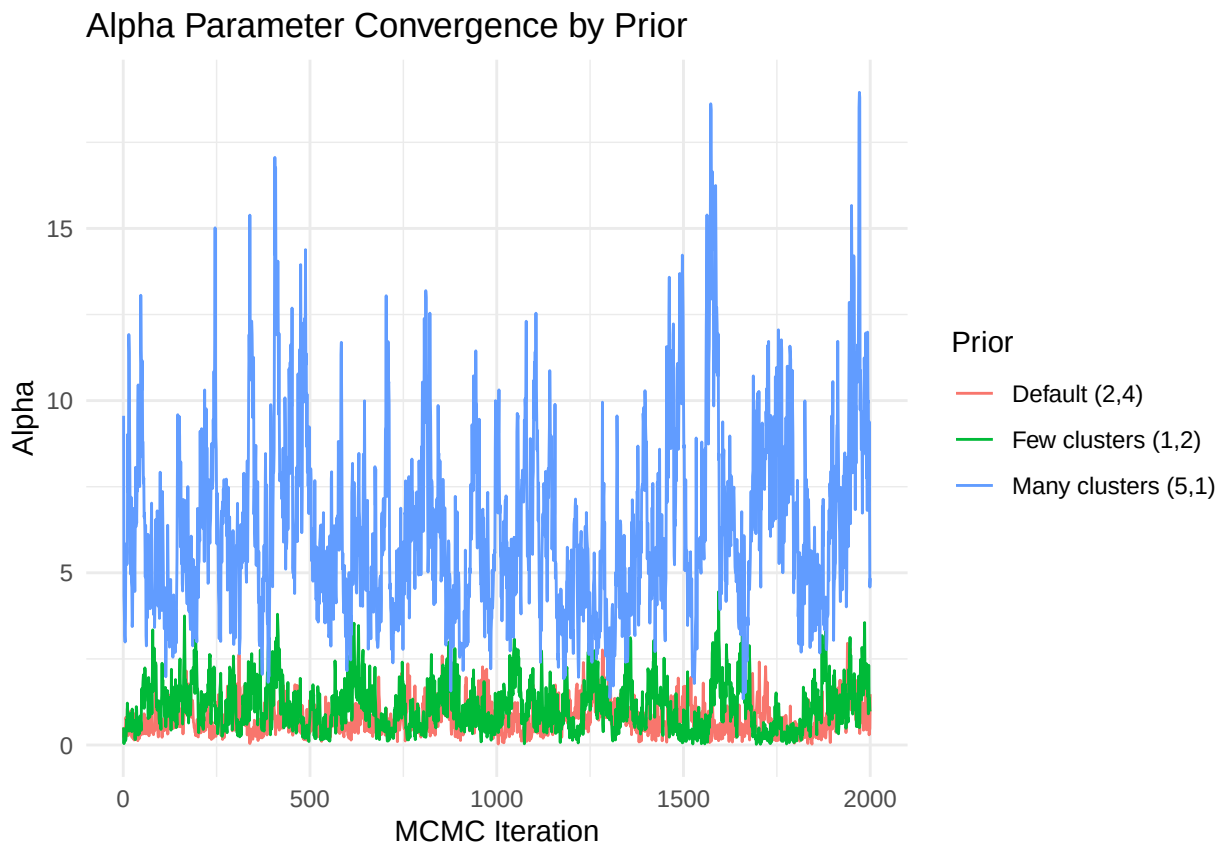
We compare the number of clusters found by each model:

```
n_clusters1 <- sapply(dp_lnorm1$labelsChain, function(x) length(unique(x)))
n_clusters2 <- sapply(dp_lnorm2$labelsChain, function(x) length(unique(x)))
n_clusters3 <- sapply(dp_lnorm3$labelsChain, function(x) length(unique(x)))

cluster_df <- data.frame(
  Iteration = rep(1:length(n_clusters1), 3),
  Clusters = c(n_clusters1, n_clusters2, n_clusters3),
  Prior = rep(c("Default (2,4)", "Few clusters (1,2)", "Many clusters (5,1)"),
              each = length(n_clusters1))
)

# Plot number of clusters over iterations
ggplot(cluster_df, aes(x = Iteration, y = Clusters, color = Prior)) +
  geom_line() +
  labs(title = "Number of Clusters by Prior",
       x = "MCMC Iteration", y = "Number of Clusters") +
  theme_minimal()
```



Now, we examine the alpha parameter chains to assess convergence:

```
alpha_df <- data.frame(
  Iteration = rep(1:length(dp_lnorm1$alphaChain), 3),
  Alpha = c(dp_lnorm1$alphaChain, dp_lnorm2$alphaChain, dp_lnorm3$alphaChain),
  Prior = rep(c("Default (2,4)", "Few clusters (1,2)", "Many clusters (5,1)"),
              each = length(dp_lnorm1$alphaChain))
)

# Plot alpha over iterations
ggplot(alpha_df, aes(x = Iteration, y = Alpha, color = Prior)) +
  geom_line() +
  labs(title = "Alpha Parameter Convergence by Prior",
       x = "MCMC Iteration", y = "Alpha") +
  theme_minimal()
```



Summarizing the average number of clusters and convergence metrics:

```
# average number of clusters in the second half of the chain
avg_clusters1 <- mean(n_clusters1[1000:2000])
avg_clusters2 <- mean(n_clusters2[1000:2000])
avg_clusters3 <- mean(n_clusters3[1000:2000])

cat("Average number of clusters (second half of chain):\n")
```

```
## Average number of clusters (second half of chain):
```

```
cat("Default prior (2,4):", avg_clusters1, "\n")
```

```
## Default prior (2,4): 6.526474
```

```
cat("Prior for fewer clusters (1,2):", avg_clusters2, "\n")
```

```
## Prior for fewer clusters (1,2): 8.087912
```

```
cat("Prior for more clusters (5,1):", avg_clusters3, "\n")
```

```
## Prior for more clusters (5,1): 29.91409
```

```r
# Calculate convergence metrics
burnin <- 1000
effective_samples1 <- effectiveSize(dp_lnorm1$alphaChain[burnin:2000])
effective_samples2 <- effectiveSize(dp_lnorm2$alphaChain[burnin:2000])
effective_samples3 <- effectiveSize(dp_lnorm3$alphaChain[burnin:2000])

cat("Effective sample sizes after burn-in:\n")
```

```
## Effective sample sizes after burn-in:
```

```r
cat("Default prior:", effective_samples1, "\n")
```

```
## Default prior: 22.04478
```

```r
cat("Prior for fewer clusters:", effective_samples2, "\n")
```

```
## Prior for fewer clusters: 40.70079
```

```r
cat("Prior for more clusters:", effective_samples3, "\n")
```

```
## Prior for more clusters: 28.85655
```

Task 3: Custom Gamma Mixture Model

In this task, we'll implement a custom mixture model using the Gamma distribution. The Gamma distribution is useful for modeling positive continuous data with different shapes.

Mathematical Framework

The Gamma mixture model can be represented as a mixture of Gamma distributions:

$$p(x) = \sum_{j=1}^{\infty} w_j \cdot \text{Gamma}(x|\alpha_j, \beta_j)$$

where:

- w_j are the mixture weights that sum to 1
- α_j is the shape parameter for the j-th component
- β_j is the rate parameter for the j-th component

$$\text{Gamma}(x|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$$

is the Gamma PDF

In our nonparametric Bayesian approach, we place a Dirichlet process prior on the distribution of the parameter pairs (α, β):

$$(\alpha_j, \beta_j) \sim G$$
$$G \sim \text{DP}(\gamma, G_0)$$

where:

- G is a random distribution drawn from the Dirichlet process
- γ is the concentration parameter controlling the diversity of components
- G_0 is the base measure representing our prior belief about the parameters

For the prior on the shape parameter, we use a transformed Beta distribution:

$$u \sim \text{Beta}(a_0, b_0)$$
$$\alpha = -s \cdot \log(u)$$

where s is a scale parameter. For the rate parameter, we use a Gamma prior:

$$\beta \sim \text{Gamma}(c_0, d_0)$$

For posterior sampling, we use a combination of Gibbs sampling for the rate parameter and Metropolis-Hastings for the shape parameter.

Implementation

```
# Likelihood function for Gamma distribution
Likelihood.gamma <- function(mdobj, x, theta) {
  shape <- theta[[1]]
  rate <- theta[[2]]
  return(as.numeric(dgamma(x, shape = shape, rate = rate)))
}

# Prior draw function for Gamma parameters
PriorDraw.gamma <- function(mdobj, n) {

  a0 <- mdobj$priorParameters[1]  # shape
  b0 <- mdobj$priorParameters[2]  # shape
  c0 <- mdobj$priorParameters[3]  # rate
  d0 <- mdobj$priorParameters[4]  # rate

  # Draw shape parameter (using a transform of Beta)
  shape_scale <- mdobj$priorParameters[5]  # Scale factor for shape
  u <- rbeta(n, a0, b0)
  shape <- -shape_scale * log(u)  # Transform to get positive values with appropriate scale

  # Draw rate from Gamma
  rate <- rgamma(n, shape = c0, rate = d0)
```

```
  theta <- list(
    array(shape, dim = c(1, 1, n)),
    array(rate, dim = c(1, 1, n))
  )
  return(theta)
}

# Posterior draw function using sufficient statistics and conjugacy
PosteriorDraw.gamma <- function(mdobj, x, n = 1) {

  a0 <- mdobj$priorParameters[1]
  b0 <- mdobj$priorParameters[2]
  c0 <- mdobj$priorParameters[3]
  d0 <- mdobj$priorParameters[4]
  shape_scale <- mdobj$priorParameters[5]

  n_obs <- length(x)

  # Sufficient statistics
  sum_x <- sum(x)
  sum_log_x <- sum(log(x))

  # For the gamma shape parameter, we need to use MCMC since
  # there's no direct conjugate update
  shape_samples <- numeric(n)
  rate_samples <- numeric(n)

  for(i in 1:n) {
    # Initial values
    alpha_current <- 1  # Starting value for shape

    # Since we know the posterior mode for rate given shape
    # we can use Gibbs sampling
    n_iter <- 500
    n_burnin <- 300

    for(j in 1:n_iter) {
      # Update rate given shape (conjugate update)
      beta_posterior <- rgamma(1, shape = c0 + n_obs * alpha_current,
                               rate = d0 + sum_x)

      # Update shape using Metropolis-Hastings
      alpha_proposal <- exp(rnorm(1, log(alpha_current), 0.1))

      # Log posterior ratio for shape
      log_ratio <- (alpha_proposal - alpha_current) * sum_log_x -
        n_obs * (lgamma(alpha_proposal) - lgamma(alpha_current)) +
        n_obs * alpha_proposal * log(beta_posterior) -
        n_obs * alpha_current * log(beta_posterior) +
        (a0 - 1) * (log(1 - exp(-alpha_proposal/shape_scale)) -
                    log(1 - exp(-alpha_current/shape_scale))) +
        (alpha_current - alpha_proposal)/shape_scale
```

```
      # Accept/reject shape proposal
      if(log(runif(1)) < log_ratio) {
        alpha_current <- alpha_proposal
      }

      # Store samples after burn-in
      if(j > n_burnin) {
        shape_samples[i] <- alpha_current
        rate_samples[i] <- beta_posterior
      }
    }
  }

  theta <- list(
    array(shape_samples, dim = c(1, 1, n)),
    array(rate_samples, dim = c(1, 1, n))
  )

  return(theta)
}

# Predictive function for Gamma
Predictive.gamma <- function(mdobj, x) {
  # For positive data
  pred <- numeric(length(x))

  # Monte Carlo approximation with samples from the prior
  n_samples <- 1000

  # Draw samples from the prior
  prior_samples <- PriorDraw.gamma(mdobj, n_samples)
  shape_samples <- prior_samples[[1]][1, 1, ]
  rate_samples <- prior_samples[[2]][1, 1, ]

  # For each x, compute the predictive density
  for(i in seq_along(x)) {
    if(x[i] <= 0) {
      pred[i] <- 0  # Gamma density is 0 for x <= 0
    } else {
      # Compute Gamma density for each prior sample
      densities <- dgamma(x[i], shape = shape_samples, rate = rate_samples)
      # Average over samples
      pred[i] <- mean(densities)
    }
  }

  return(pred)
}
```

Testing the Model

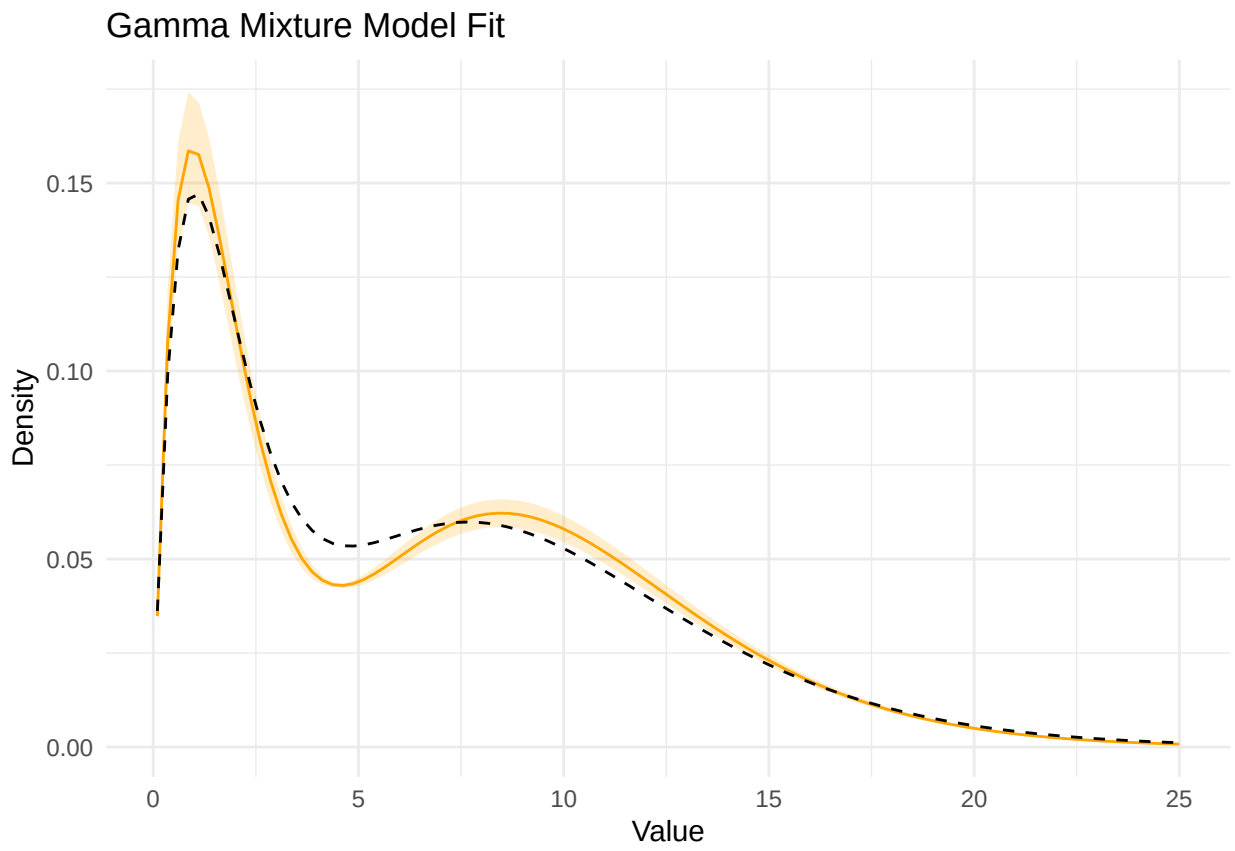
We generate data from a mixture of two Gamma distributions and fit our custom model:

```r
# Gamma mixing distribution
gammaMd <- MixingDistribution(distribution = "gamma",
                              priorParameters = c(2, 2, 2, 0.5, 5),  # a0, b0, c0, d0, shape_scale
                              conjugate = "conjugate")

# simulated data from a mixture of two Gamma distributions
set.seed(123)
y <- c(rgamma(200, shape = 2, rate = 1),   # Shape=2, Mean=2
       rgamma(300, shape = 5, rate = 0.5)) # Shape=5, Mean=10


dp <- DirichletProcessCreate(y, gammaMd)
dp <- Initialise(dp)
dp <- Fit(dp, 1000)
```

Results

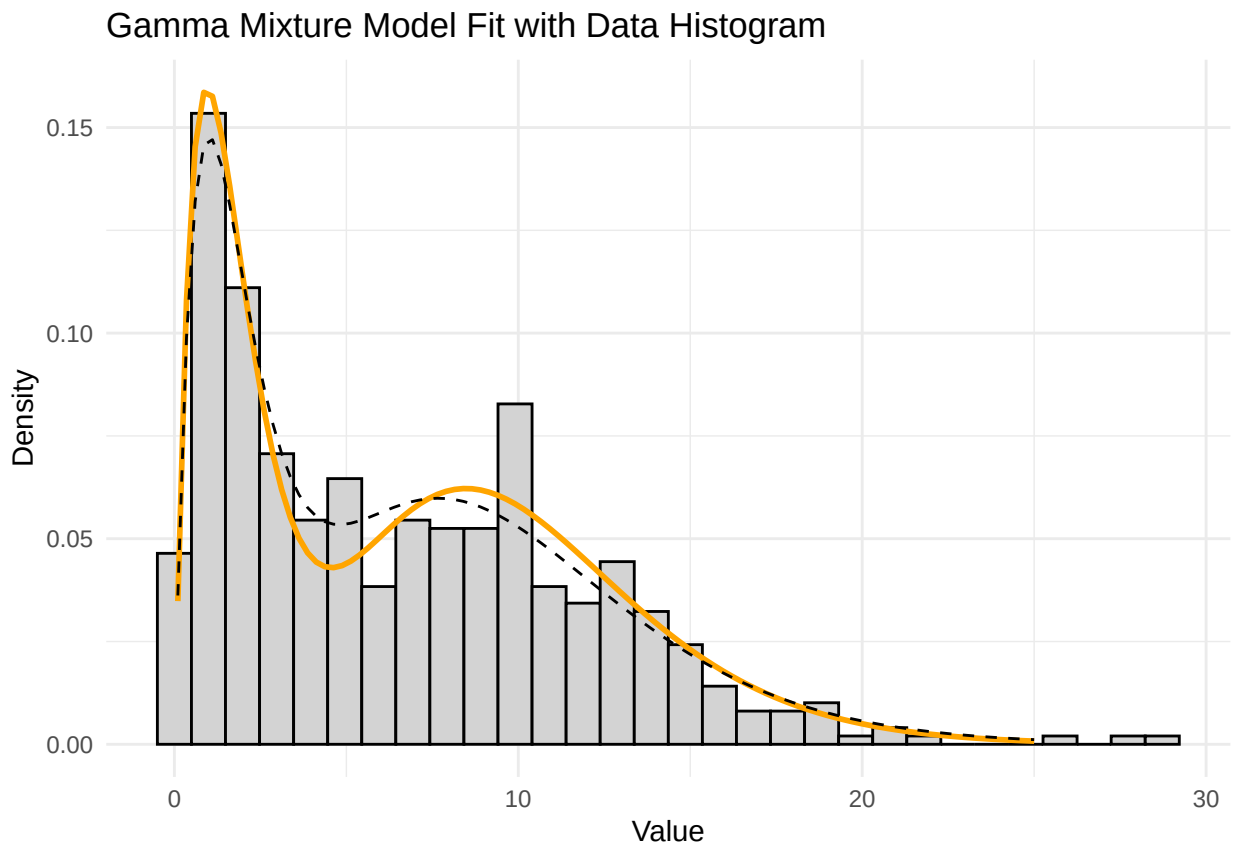
Plot of the true density and the estimated posterior:

```r
# Generate posterior predictive frame
x_range <- seq(0.1, 25, length.out = 100)
pf <- PosteriorFrame(dp, x_range, 1000)

trueFrame <- data.frame(
  x = x_range,
  y = 0.4 * dgamma(x_range, shape = 2, rate = 1) +
    0.6 * dgamma(x_range, shape = 5, rate = 0.5)
)

ggplot() +
  geom_ribbon(data = pf,
              aes(x = x, ymin = X5., ymax = X95.),
              colour = NA,
              fill = "orange",
              alpha = 0.2) +
  geom_line(data = pf, aes(x = x, y = Mean), colour = "orange") +
  geom_line(data = trueFrame, aes(x = x, y = y), linetype = "dashed") +
  labs(title = "Gamma Mixture Model Fit",
       x = "Value",
       y = "Density") +
  theme_minimal()
```



Plot of the data histogram with the fitted density:

```
ggplot() +
  geom_histogram(aes(x = y, y = after_stat(density)), bins = 30, fill = "lightgray", color = "black") +
  geom_line(data = pf, aes(x = x, y = Mean), colour = "orange", size = 1) +
  geom_line(data = trueFrame, aes(x = x, y = y), linetype = "dashed") +
  labs(title = "Gamma Mixture Model Fit with Data Histogram",
       x = "Value",
       y = "Density") +
  theme_minimal()
```



Examine the clusters found by the model:

```
# Number of clusters found
n_clusters <- length(unique(dp$clusterLabels))
cat("Number of clusters found:", n_clusters, "\n")
```

```
## Number of clusters found: 3
```

```
# Get the parameters for each cluster
cluster_params <- data.frame(
  Cluster = 1:dp$numberClusters,
  Shape = sapply(1:dp$numberClusters, function(i) dp$clusterParameters[[1]][,,i]),
  Rate = sapply(1:dp$numberClusters, function(i) dp$clusterParameters[[2]][,,i]),
  Size = dp$pointsPerCluster,
  Proportion = dp$pointsPerCluster / sum(dp$pointsPerCluster)
)

print(cluster_params)
```

```
##   Cluster    Shape      Rate Size Proportion
## 1       1 6.174073 0.6039547  298      0.596
## 2       2 2.280872 1.1533397  186      0.372
## 3       3 2.185196 2.6859146   16      0.032
```