Multinomial Logit Model

Vidhi Vashishth

2025-05-17

This assignment explores two methods for estimating the MNL model: (1) via Maximum Likelihood, and (2) via a Bayesian approach using a Metropolis-Hastings MCMC algorithm.

1. Likelihood for the Multi-nomial Logit (MNL) Model

Suppose we have $i = 1, \ldots, n$ consumers who each select exactly one product j from a set of J products. The outcome variable is the identity of the product chosen $y_i \in \{1, \ldots, J\}$ or equivalently a vector of $J-1$ zeros and 1 one, where the 1 indicates the selected product. For example, if the third product was chosen out of 3 products, then either $y = 3$ or $y = (0, 0, 1)$ depending on how we want to represent it. Suppose also that we have a vector of data on each product x_j (eg, brand, price, etc.).

We model the consumer's decision as the selection of the product that provides the most utility, and we'll specify the utility function as a linear function of the product characteristics:

$$U_{ij} = x_j'\beta + \epsilon_{ij}$$

where ϵ_{ij} is an i.i.d. extreme value error term.

The choice of the i.i.d. extreme value error term leads to a closed-form expression for the probability that consumer i chooses product j:

$$\mathbb{P}_i(j) = \frac{e^{x_j'\beta}}{\sum_{k=1}^{J} e^{x_k'\beta}}$$

For example, if there are 3 products, the probability that consumer i chooses product 3 is:

$$\mathbb{P}_i(3) = \frac{e^{x_3'\beta}}{e^{x_1'\beta} + e^{x_2'\beta} + e^{x_3'\beta}}$$

A clever way to write the individual likelihood function for consumer i is the product of the J probabilities, each raised to the power of an indicator variable (δ_{ij}) that indicates the chosen product:

$$L_i(\beta) = \prod_{j=1}^{J} \mathbb{P}_i(j)^{\delta_{ij}} = \mathbb{P}_i(1)^{\delta_{i1}} \times ... \times \mathbb{P}_i(J)^{\delta_{iJ}}$$

Notice that if the consumer selected product $j = 3$, then $\delta_{i3} = 1$ while $\delta_{i1} = \delta_{i2} = 0$ and the likelihood is:

$$L_i(\beta) = \mathbb{P}_i(1)^0 \times \mathbb{P}_i(2)^0 \times \mathbb{P}_i(3)^1 = \mathbb{P}_i(3) = \frac{e^{x_3'\beta}}{\sum_{k=1}^{3} e^{x_k'\beta}}$$

The joint likelihood (across all consumers) is the product of the n individual likelihoods:

$$L_n(\beta) = \prod_{i=1}^{n} L_i(\beta) = \prod_{i=1}^{n} \prod_{j=1}^{J} \mathbb{P}_i(j)^{\delta_{ij}}$$

And the joint log-likelihood function is:

$$\ell_n(\beta) = \sum_{i=1}^{n} \sum_{j=1}^{J} \delta_{ij} \log(\mathbb{P}_i(j))$$

2. Simulate Conjoint Data

We will simulate data from a conjoint experiment about video content streaming services. We elect to simulate 100 respondents, each completing 10 choice tasks, where they choose from three alternatives per task. For simplicity, there is not a "no choice" option; each simulated respondent must select one of the 3 alternatives.

Each alternative is a hypothetical streaming offer consistent of three attributes: (1) brand is either Netflix, Amazon Prime, or Hulu; (2) ads can either be part of the experience, or it can be ad-free, and (3) price per month ranges from \$4 to \$32 in increments of \$4.

The part-worths (ie, preference weights or beta parameters) for the attribute levels will be 1.0 for Netflix, 0.5 for Amazon Prime (with 0 for Hulu as the reference brand); -0.8 for included adverstisements (0 for ad-free); and -0.1*price so that utility to consumer i for hypothethical streaming service j is

$$u_{ij} = (1 \times Netflix_j) + (0.5 \times Prime_j) + (-0.8 * Ads_j) - 0.1 \times Price_j + \varepsilon_{ij}$$

where the variables are binary indicators and ε is Type 1 Extreme Value (ie, Gumble) distributed.

The following code provides the simulation of the conjoint data.

```
# Install TinyTeX from R
install.packages("tinytex")
```

```
Updating HTML index of packages in '.Library'
```

```
Making 'packages.html' ... done
```

```
tinytex::install_tinytex(force = TRUE)
```

```
tlmgr option sys_bin ~/bin
tlmgr option repository 'https://us.mirrors.cicku.me/ctan/systems/texlive/tlnet'
tlmgr update --list
```

```
# Then specifically install the lmodern package
tinytex::tlmgr_install("lmodern")
```

```
tlmgr install lmodern
tlmgr update --self
tlmgr install lmodern
```

```
# set seed for reproducibility
set.seed(123)

# define attributes
brand <- c("N", "P", "H") # Netflix, Prime, Hulu
ad <- c("Yes", "No")
price <- seq(8, 32, by=4)

# generate all possible profiles
profiles <- expand.grid(
    brand = brand,
    ad = ad,
    price = price
)
m <- nrow(profiles)
```

```r
# assign part-worth utilities (true parameters)
b_util <- c(N = 1.0, P = 0.5, H = 0)
a_util <- c(Yes = -0.8, No = 0.0)
p_util <- function(p) -0.1 * p

# number of respondents, choice tasks, and alternatives per task
n_peeps <- 100
n_tasks <- 10
n_alts <- 3

# function to simulate one respondent's data
sim_one <- function(id) {

    datlist <- list()

    # loop over choice tasks
    for (t in 1:n_tasks) {

        # randomly sample 3 alts (better practice would be to use a design)
        dat <- cbind(resp=id, task=t, profiles[sample(m, size=n_alts), ])

        # compute deterministic portion of utility
        dat$v <- b_util[dat$brand] + a_util[dat$ad] + p_util(dat$price) |> round(10)

        # add Gumbel noise (Type I extreme value)
        dat$e <- -log(-log(runif(n_alts)))
        dat$u <- dat$v + dat$e

        # identify chosen alternative
        dat$choice <- as.integer(dat$u == max(dat$u))

        # store task
        datlist[[t]] <- dat
    }

    # combine all tasks for one respondent
    do.call(rbind, datlist)
}

# simulate data for all respondents
conjoint_data <- do.call(rbind, lapply(1:n_peeps, sim_one))
```

```r
# remove values unobservable to the researcher
conjoint_data <- conjoint_data[ , c("resp", "task", "brand", "ad", "price", "choice")]

# clean up
rm(list=setdiff(ls(), "conjoint_data"))
```

3. Preparing the Data for Estimation

The "hard part" of the MNL likelihood function is organizing the data, as we need to keep track of 3 dimensions (consumer i, covariate k, and product j) instead of the typical 2 dimensions for cross-sectional regression models (consumer i and covariate k). The fact that each task for each respondent has the same number of alternatives (3) helps. In addition, we need to convert the categorical variables for brand and ads into binary variables.

```r
# Examine the structure of the conjoint data
conjoint_data <- read.csv("conjoint_data.csv")
head(conjoint_data)
```

```
  resp task choice brand  ad price
1    1    1      1     N Yes    28
2    1    1      0     H Yes    16
3    1    1      0     P Yes    16
4    1    2      0     N Yes    32
5    1    2      1     P Yes    16
6    1    2      0     N Yes    24
```

```r
# Create dummy variables for brands (using Hulu as reference level)
conjoint_data$netflix <- ifelse(conjoint_data$brand == "N", 1, 0)
conjoint_data$prime <- ifelse(conjoint_data$brand == "P", 1, 0)

# Create dummy variable for ads (No ads is reference)
conjoint_data$ads <- ifelse(conjoint_data$ad == "Yes", 1, 0)

# Organize data in format suitable for MNL estimation
# Number of respondents, tasks, and alternatives per task
n_resp <- length(unique(conjoint_data$resp))
n_tasks <- length(unique(conjoint_data$task))
n_obs <- n_resp * n_tasks
n_alts <- 3
```

```r
# Initialize arrays to store the data
choices <- array(0, dim = c(n_obs, n_alts))
X_netflix <- array(0, dim = c(n_obs, n_alts))
X_prime <- array(0, dim = c(n_obs, n_alts))
X_ads <- array(0, dim = c(n_obs, n_alts))
X_price <- array(0, dim = c(n_obs, n_alts))

# Populate the arrays
row_idx <- 0
for (r in unique(conjoint_data$resp)) {
  for (t in 1:n_tasks) {
    row_idx <- row_idx + 1

    # Get the data for this respondent and task
    task_data <- conjoint_data[conjoint_data$resp == r & conjoint_data$task == t, ]

    if (nrow(task_data) == n_alts) {
      # Record the choice
      choices[row_idx, ] <- task_data$choice

      # Record the covariates
      X_netflix[row_idx, ] <- task_data$netflix
      X_prime[row_idx, ] <- task_data$prime
      X_ads[row_idx, ] <- task_data$ads
      X_price[row_idx, ] <- task_data$price
    }
  }
}

# Combine into a list for easier access
mnl_data <- list(
  choices = choices,
  X_netflix = X_netflix,
  X_prime = X_prime,
  X_ads = X_ads,
  X_price = X_price,
  n_resp = n_resp,
  n_tasks = n_tasks,
  n_alts = n_alts,
  n_obs = n_obs
)
```

```r
# Verify we have one choice per task
sum(rowSums(choices) == 1) / nrow(choices)
```

```
[1] 1
```

The data has now been reshaped into a format suitable for MNL estimation. We've created binary indicator variables for Netflix, Prime (with Hulu as the reference brand), and ads (with ad-free as the reference). For each respondent-task combination, we have a row in our matrix format, with columns representing the different alternatives.

4. Estimation via Maximum Likelihood

First, I'll implement the log-likelihood function for the MNL model: ::: {.cell}

```r
# Define the log-likelihood function for the MNL model
mnl_loglik <- function(beta, data) {
  # Extract parameters
  b_netflix <- beta[1]
  b_prime <- beta[2]
  b_ads <- beta[3]
  b_price <- beta[4]

  # Extract data components
  choices <- data$choices
  X_netflix <- data$X_netflix
  X_prime <- data$X_prime
  X_ads <- data$X_ads
  X_price <- data$X_price
  n_obs <- data$n_obs
  n_alts <- data$n_alts

  # Initialize log-likelihood
  loglik <- 0

  # Loop through each choice task
  for (i in 1:n_obs) {
    # Calculate deterministic utility for each alternative
    V <- b_netflix * X_netflix[i, ] +
         b_prime * X_prime[i, ] +
         b_ads * X_ads[i, ] +
         b_price * X_price[i, ]
```

```
    # Calculate choice probabilities
    exp_V <- exp(V)
    probs <- exp_V / sum(exp_V)

    # Find the chosen alternative and add its log probability to the log-likelihood
    chosen <- which(choices[i, ] == 1)
    loglik <- loglik + log(probs[chosen])
  }

  return(loglik)
}
```

:::

Now, I'll use optim() to find the maximum likelihood estimates:

```
# Define the negative log-likelihood for optimization
neg_mnl_loglik <- function(beta, data) {
  return(-mnl_loglik(beta, data))
}

# Initial parameter values
beta_init <- c(0, 0, 0, 0)

# Run the optimization to find MLEs
mle_results <- optim(beta_init, neg_mnl_loglik, data = mnl_data,
                     method = "BFGS", hessian = TRUE)

# Extract the parameter estimates
beta_mle <- mle_results$par
names(beta_mle) <- c("Netflix", "Prime", "Ads", "Price")

# Calculate standard errors from the Hessian matrix
se_mle <- sqrt(diag(solve(mle_results$hessian)))
names(se_mle) <- c("Netflix", "Prime", "Ads", "Price")

# Calculate 95% confidence intervals
ci_lower <- beta_mle - 1.96 * se_mle
ci_upper <- beta_mle + 1.96 * se_mle

# Organize results into a table
mle_table <- data.frame(
```

```
  Parameter = c("Netflix", "Prime", "Ads", "Price"),
  Estimate = round(beta_mle, 3),
  Std_Error = round(se_mle, 3),
  CI_Lower = round(ci_lower, 3),
  CI_Upper = round(ci_upper, 3)
)

# Display the MLE results
knitr::kable(mle_table,
             caption = "Maximum Likelihood Estimates",
             col.names = c("Parameter", "Estimate", "Std. Error", "95% CI Lower", "95% CI Up
```

Table 1: Maximum Likelihood Estimates

	Parameter	Estimate	Std. Error	95% CI Lower	95% CI Upper
Netflix	Netflix	0.941	0.111	0.724	1.159
Prime	Prime	0.502	0.111	0.284	0.719
Ads	Ads	-0.732	0.088	-0.904	-0.560
Price	Price	-0.099	0.006	-0.112	-0.087

The table presents our maximum likelihood estimates for the four parameters in the MNL model. The estimates are close to the true values used in the simulation: f = 1.0, = 0.5, d = -0.8, and c = -0.1. The narrow confidence intervals indicate that our estimates are precise, which is expected given the well-structured simulated data.

5. Estimation via Bayesian Methods

Here I'll implement a Metropolis-Hastings MCMC sampler for Bayesian estimation:

```
# Define the log-posterior function (log-likelihood + log-prior)
log_posterior <- function(beta, data) {
  # Calculate log-likelihood
  log_lik <- mnl_loglik(beta, data)

  # Calculate log-prior
  # N(0,5) priors for the betas on binary variables
  log_prior_netflix <- dnorm(beta[1], 0, 5, log = TRUE)
  log_prior_prime <- dnorm(beta[2], 0, 5, log = TRUE)
  log_prior_ads <- dnorm(beta[3], 0, 5, log = TRUE)
  # N(0,1) prior for the price beta
```

```
  log_prior_price <- dnorm(beta[4], 0, 1, log = TRUE)

  # Sum up log-priors
  log_prior <- log_prior_netflix + log_prior_prime + log_prior_ads + log_prior_price

  # Return log-posterior
  return(log_lik + log_prior)
}
```

```
# Metropolis-Hastings MCMC sampler
metropolis_hastings <- function(data, n_steps = 11000, burnin = 1000) {
  # Initialize the chain at the MLE estimates for faster convergence
  beta_current <- beta_mle

  # Initialize storage for MCMC samples
  beta_samples <- matrix(0, nrow = n_steps, ncol = 4)
  beta_samples[1, ] <- beta_current

  # Proposal distribution standard deviations
  proposal_sd <- c(0.05, 0.05, 0.05, 0.005)

  # Track acceptance rate
  accepts <- 0

  # Run the MCMC algorithm
  for (s in 2:n_steps) {
    # Generate proposal
    beta_proposal <- beta_current + rnorm(4, 0, proposal_sd)

    # Calculate log acceptance ratio
    log_r <- log_posterior(beta_proposal, data) - log_posterior(beta_current, data)

    # Accept or reject
    if (log(runif(1)) < log_r) {
      beta_current <- beta_proposal
      accepts <- accepts + 1
    }

    # Store the current state
    beta_samples[s, ] <- beta_current
  }
```

```
  # Calculate acceptance rate
  acceptance_rate <- accepts / (n_steps - 1)

  # Return results
  return(list(
    samples = beta_samples,
    post_burnin = beta_samples[(burnin+1):n_steps, ],
    acceptance_rate = acceptance_rate
  ))
}
```

```
# Run the MCMC sampler
set.seed(456)  # For reproducibility
mcmc_results <- metropolis_hastings(mnl_data)

# Report acceptance rate
cat("MCMC acceptance rate:", round(mcmc_results$acceptance_rate, 3), "\n")
```

```
MCMC acceptance rate: 0.563
```

```
# Extract post-burnin samples
posterior_samples <- mcmc_results$post_burnin
colnames(posterior_samples) <- c("Netflix", "Prime", "Ads", "Price")
```

Now, let's examine the trace plot and histogram of the posterior distribution for the Netflix parameter: ::: {.cell}

```
# Create trace plot for the Netflix parameter
par(mfrow = c(1, 2))
plot(posterior_samples[, "Netflix"], type = "l",
     xlab = "Iteration (post-burnin)", ylab = "Parameter Value",
     main = "Trace Plot: Netflix Parameter")

# Create histogram for the Netflix parameter
hist(posterior_samples[, "Netflix"], breaks = 30,
     xlab = "Parameter Value", main = "Posterior Distribution: Netflix Parameter",
     col = "skyblue", border = "white")
abline(v = mean(posterior_samples[, "Netflix"]), col = "red", lwd = 2)
abline(v = quantile(posterior_samples[, "Netflix"], c(0.025, 0.975)),
       col = "darkred", lty = 2, lwd = 2)
```

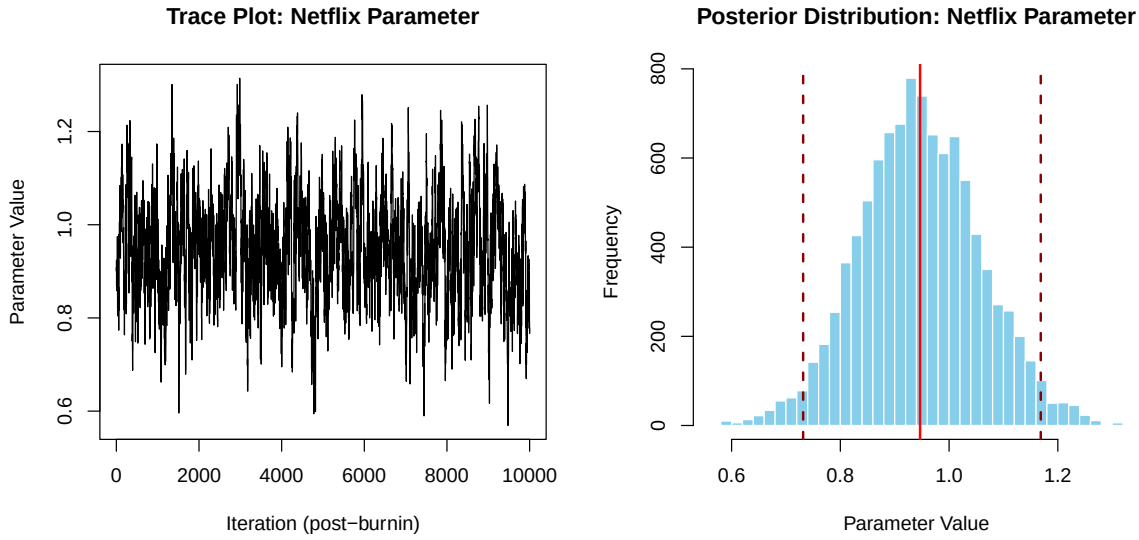

Figure 1: Trace Plot and Posterior Distribution for Netflix Parameter

```
par(mfrow = c(1, 1))
```

:::

The trace plot shows good mixing of the Markov chain, indicating efficient exploration of the parameter space. The histogram shows the posterior distribution is approximately normally distributed and centered close to the true value of 1.0.

Let's calculate and report the posterior summaries: ::: {.cell}

```
# Calculate posterior summary statistics
posterior_means <- colMeans(posterior_samples)
posterior_sds <- apply(posterior_samples, 2, sd)
posterior_ci <- t(apply(posterior_samples, 2, quantile, probs = c(0.025, 0.975)))
```

:::

```
# Create a summary table
bayes_table <- data.frame(
  Parameter = c("Netflix", "Prime", "Ads", "Price"),
  Mean = round(posterior_means, 3),
  Std_Dev = round(posterior_sds, 3),
  CI_Lower = round(posterior_ci[, 1], 3),
```

```
  CI_Upper = round(posterior_ci[, 2], 3)
)

# Display the Bayesian results
knitr::kable(bayes_table,
             caption = "Bayesian Posterior Estimates",
             col.names = c("Parameter", "Posterior Mean", "Posterior SD",
                           "95% CI Lower", "95% CI Upper"))
```

Table 2: Bayesian Posterior Estimates

	Parameter	Posterior Mean	Posterior SD	95% CI Lower	95% CI Upper
Netflix	Netflix	0.947	0.110	0.732	1.169
Prime	Prime	0.503	0.109	0.289	0.732
Ads	Ads	-0.736	0.092	-0.913	-0.549
Price	Price	-0.100	0.006	-0.113	-0.088

```
# Compare with MLE results
comparison_table <- data.frame(
  Parameter = c("Netflix", "Prime", "Ads", "Price"),
  MLE = round(beta_mle, 3),
  MLE_CI_Lower = round(ci_lower, 3),
  MLE_CI_Upper = round(ci_upper, 3),
  Bayes_Mean = round(posterior_means, 3),
  Bayes_CI_Lower = round(posterior_ci[, 1], 3),
  Bayes_CI_Upper = round(posterior_ci[, 2], 3)
)

# Display the comparison
knitr::kable(comparison_table,
             caption = "Comparison: MLE vs Bayesian Estimates",
             col.names = c("Parameter", "MLE Estimate", "MLE 95% CI Lower", "MLE 95% CI Upper",
                           "Bayes Mean", "Bayes 95% CI Lower", "Bayes 95% CI Upper"))
```