Multinomial Logit Model

Zhutong Zhang

2025-05-17

This assignment expores 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 $i = 1, \dots, n$ consumers who each select exactly one product j from a set of J products. The outcome variable is the identity of the product chosen $y_i \in \{1, \dots, J\}$ 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.

> **Note**

```r
# 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)

# 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)
```

3. Preparing the Data for Estimation

```
import pandas as pd
import numpy as np

# Load simulated conjoint data
conjoint_data = pd.read_csv("conjoint_data.csv")

# One-hot encode categorical variables
X = pd.get_dummies(conjoint_data, columns=["brand", "ad"], drop_first=True)


# Convert IDs and choice flag to integer
for col in ['resp', 'task', 'choice']:
    X[col] = X[col].astype(int)

# Add unique choice set identifier
X['choice_set'] = X['resp'].astype(str) + "_" + X['task'].astype(str)

# Sort for readability (optional)
X = X.sort_values(by=['resp', 'task']).reset_index(drop=True)

# Preview cleaned dataset
X.head()
```

4. Estimation via Maximum Likelihood

```python
import numpy as np
from scipy.optimize import minimize
from scipy.special import logsumexp

features = ['brand_N', 'brand_P', 'ad_Yes', 'price']
X_data = X[features].astype(float).values

y_data = X['choice'].astype(int).values

group_ids = X['choice_set'].astype(str).values


def neg_log_likelihood(beta, X, y, group_ids):
    """
    beta: parameter vector (length 4)
    X: (n_obs, n_features)
    y: (n_obs,)  binary indicator (1 if chosen)
    group_ids: choice set ID (same for each set of alternatives)
    """
    Xb = X @ beta
    df = pd.DataFrame({'group': group_ids, 'xb': Xb})

    # Compute log-denominator for each group using logsumexp
    denom = df.groupby('group')['xb'].transform(lambda x: logsumexp(x))

    log_probs = Xb - denom
    ll = np.sum(y * log_probs)

    return -ll  # negative log-likelihood for minimization
```

```
init_params = np.zeros(X_data.shape[1])  # e.g. [0, 0, 0, 0]

result = minimize(
    fun=neg_log_likelihood,
    x0=init_params,
    args=(X_data, y_data, group_ids),
    method='BFGS',
    options={'disp': True}
)


beta_hat = result.x
hessian_inv = result.hess_inv  # approx. inverse Hessian
standard_errors = np.sqrt(np.diag(hessian_inv))


z_95 = 1.96
lower = beta_hat - z_95 * standard_errors
upper = beta_hat + z_95 * standard_errors


summary_df = pd.DataFrame({
    'Coefficient': features,
    'Estimate': beta_hat,
    'Std. Error': standard_errors,
    '95% CI Lower': lower,
    '95% CI Upper': upper
})

summary_df.round(4)
```

5. Estimation via Bayesian Methods

-
-
  -
  -

```
import numpy as np
import pandas as pd
from scipy.special import logsumexp
import matplotlib.pyplot as plt

# Prepare data
X_data = X[['brand_N', 'brand_P', 'ad_Yes', 'price']].astype(float).values
y_data = X['choice'].astype(int).values
group_ids = X['choice_set'].astype(str).values
```

```
# Log-prior: N(0, 5^2) for binaries; N(0, 1^2) for price
def log_prior(beta):
    lp = -0.5 * (beta[0:3]**2 / 25).sum()  # binary features
    lp += -0.5 * (beta[3]**2 / 1)          # price
    return lp

# Posterior = log-likelihood + log-prior
```

```
def log_posterior(beta):
    return -neg_log_likelihood(beta, X_data, y_data, group_ids) + log_prior(beta)
```

```
# Metropolis-Hastings sampler with fixed proposal sd
def metropolis_sampler(log_post_fn, start, steps=11000, proposal_sd=None):
    if proposal_sd is None:
        proposal_sd = np.array([0.05, 0.05, 0.05, 0.005])

    draws = np.zeros((steps, len(start)))
    draws[0] = start
    current_lp = log_post_fn(start)

    for t in range(1, steps):
        proposal = draws[t - 1] + np.random.normal(0, proposal_sd)
        proposal_lp = log_post_fn(proposal)
        log_accept_ratio = proposal_lp - current_lp

        if np.log(np.random.rand()) < log_accept_ratio:
            draws[t] = proposal
            current_lp = proposal_lp
        else:
            draws[t] = draws[t - 1]

    return draws
```

```
np.random.seed(42)
start_beta = np.zeros(4)

samples = metropolis_sampler(log_posterior, start=start_beta, steps=11000)
posterior = samples[1000:]  # burn-in

# Acceptance rate
accepted = np.sum(np.any(samples[1:] != samples[:-1], axis=1))
accept_rate = accepted / (samples.shape[0] - 1)
print(f"Acceptance rate: {accept_rate:.3f}")
```

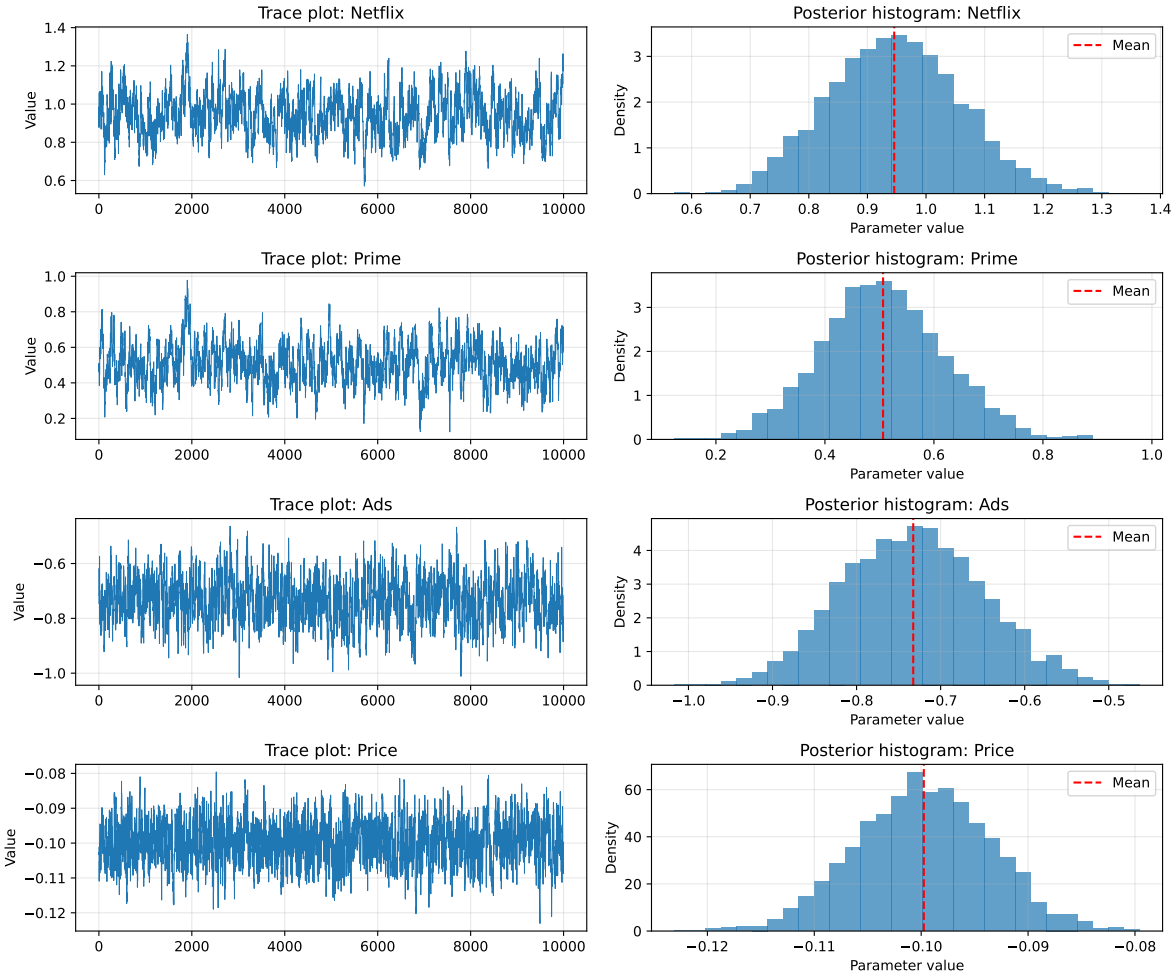
Trace plot of the algorithm & Histogram of the posterior distribution

```
param_names = [r"Netflix", r"Prime", r"Ads", r"Price"]

fig, axes = plt.subplots(4, 2, figsize=(12, 10))
for i in range(4):
    # Trace plot
    axes[i, 0].plot(posterior[:, i], linewidth=0.7)
    axes[i, 0].set_title(f"Trace plot: {param_names[i]}")
    axes[i, 0].set_ylabel("Value")
    axes[i, 0].grid(alpha=0.3)

    # Histogram
    axes[i, 1].hist(posterior[:, i], bins=30, density=True, alpha=0.7)
    axes[i, 1].axvline(posterior[:, i].mean(), color='red', linestyle='--', label="Mean")
    axes[i, 1].set_title(f"Posterior histogram: {param_names[i]}")
    axes[i, 1].set_xlabel("Parameter value")
    axes[i, 1].set_ylabel("Density")
    axes[i, 1].grid(alpha=0.3)
    axes[i, 1].legend()

plt.tight_layout()
plt.show()
```



Report

```
summary = pd.DataFrame({
    'Parameter': ['Netflix', 'Prime', 'Ads', 'Price'],
    'Mean': posterior.mean(axis=0),
    'Std': posterior.std(axis=0),
    '2.5%': np.percentile(posterior, 2.5, axis=0),
    '97.5%': np.percentile(posterior, 97.5, axis=0)
}).round(4)

summary
```

Comparison and Conclusion

6. Discussion

Interpreting Parameter Estimates in Real-World Contexts

Moving Toward Hierarchical (Random-Parameter) MNL Models

To simulate hierarchical data:

-
  -
  -
  -
  -

To estimate such a model:

-
  -
  -
-
-

Why use hierarchical models?

-
-
-