498818_SDS475_Summary4

2024-11-06

Markov Chains

- **Definition**: A Markov Chain is a sequence where each state depends only on the previous state.
- **Ergodic Theorem**: This theorem ensures that as the number of iterations increases, the distribution of samples approaches the target distribution f, the stationary distribution of the chain.
- **Burn-in Period**: Initial samples may not represent the target distribution well, so a "burn-in" period (discarding initial samples) is used for better approximation.

Common MCMC Algorithms

Metropolis-Hastings Algorithm

Objective

The Metropolis-Hastings algorithm is used to sample from complex probability distributions by constructing a Markov chain that converges to the target distribution.

Process

1. **Choose a Proposal Distribution** $k : S \times S \to \mathbb{R}$:
    - This distribution $k(Y|X)$ is used to suggest potential next states based on the current state.
    - The proposal can be symmetric (e.g., $k(Y|X) = k(X|Y)$) or asymmetric.
2. **Generate a Proposal**:
    - Starting from the current state X_{t-1}, use the proposal distribution to draw a candidate state Y.
3. **Acceptance Decision**:
    - Calculate an acceptance probability $\alpha(X_{t-1}, Y) = \min\left(1, \frac{f(Y)k(Y|X_{t-1})}{f(X_{t-1})k(X_{t-1}|Y)}\right)$.
    - Draw a random number $U \sim \text{Uniform}(0, 1)$:
        - **Accept** Y if $U \leq \alpha(X_{t-1}, Y)$, setting $X_t = Y$.
        - Otherwise, **reject** Y and keep the current state, setting $X_t = X_{t-1}$.
4. **Limitations**:
    - **Convergence Speed**: May be slow, especially in high-dimensional spaces.
    - **Sensitivity to Proposal Distribution**: A poor choice can lead to inefficiency.
    - **Risk of Getting Stuck**: If the proposal leads to local trapping, the chain may not explore the entire space well.
5. **Alternative**: The **Gibbs Sampler**, which is often preferred when the conditional distributions are known and can be sampled directly.

Gibbs Sampler

Objective

A special case of Metropolis-Hastings, the Gibbs Sampler sequentially updates each variable by sampling from its full conditional distribution, often providing faster convergence when applicable.

Process

1. **Sequentially Update Variables**:

- Draw each variable conditional on all other variables in the sequence.
- The process can proceed through:
  - **Systematic Scan**: Updates each variable in a sequential order per iteration, which may provide faster convergence when dependencies between variables are weak.
  - **Random Scan**: Updates one randomly selected variable per iteration, offering better mixing properties, especially in high-dimensional spaces or when variables are strongly dependent.

2. **Applications**:
   - Suitable when conditional distributions are known and easy to sample from.
   - Common in high-dimensional problems and Bayesian inference.

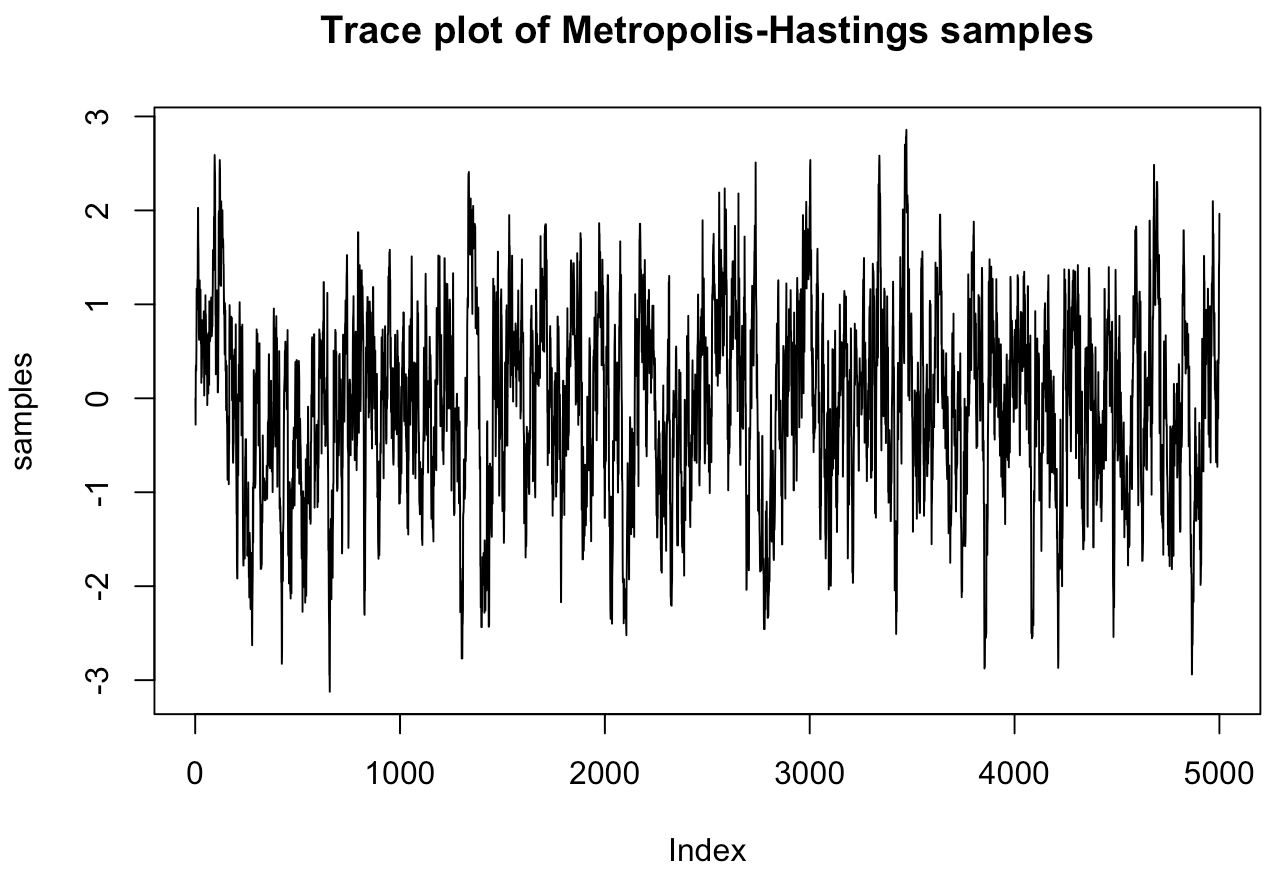
Comparison of Systematic vs. Random Scan

Feature	Systematic-Scan Gibbs Sampler	Random-Scan Gibbs Sampler
Order of Updates	Sequentially updates all variables	Randomly selects one variable to update
Updates per Iteration	Updates all variables	Updates only one variable per iteration
Convergence	May converge faster with weak dependencies	Better mixing for strong dependencies
Efficiency	More efficient when conditional distributions are simple	May require more iterations but explores state space better
Conditional Distribution	Depends on recently updated values	Depends on previous iteration values

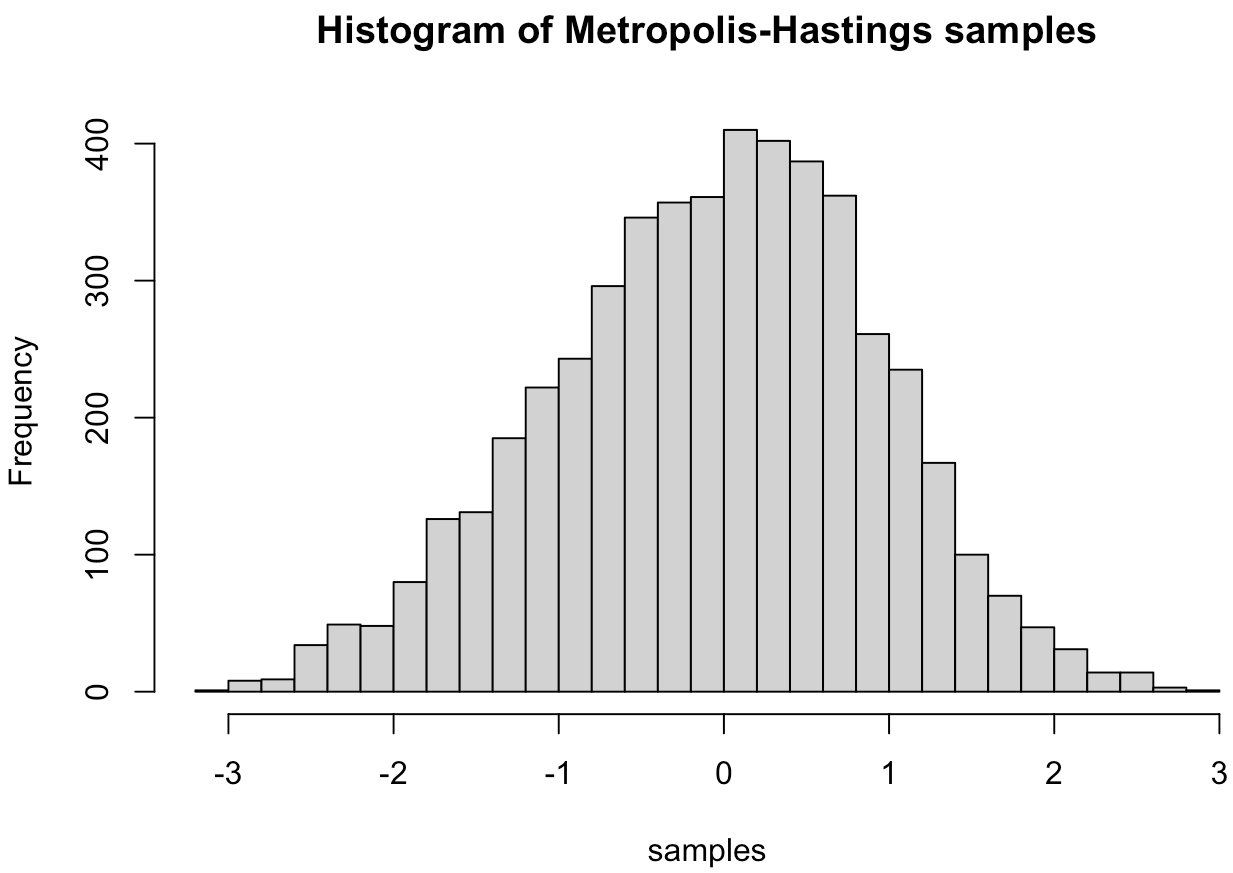
Example Code: Implementing Metropolis-Hastings in R

```r
# Define the target distribution (example: normal distribution)
target_dist <- function(x) {
  dnorm(x, mean = 0, sd = 1)
}

# Define the proposal distribution (e.g., normal distribution centered at current x)
proposal_dist <- function(x) {
  rnorm(1, mean = x, sd = 0.5)
}

# Metropolis-Hastings Algorithm
metropolis_hastings <- function(start, iter = 1000) {
  x <- numeric(iter)
  x[1] <- start

  for (i in 2:iter) {
    proposal <- proposal_dist(x[i - 1])
    accept_prob <- min(1, target_dist(proposal) / target_dist(x[i - 1]))
    if (runif(1) < accept_prob) {
      x[i] <- proposal
    } else {
      x[i] <- x[i - 1]
    }
  }
  return(x)
}
```

```r
# Running the algorithm
set.seed(123)
samples <- metropolis_hastings(start = 0, iter = 5000)

# Plotting the trace of samples
plot(samples, type = "l", main = "Trace plot of Metropolis-Hastings samples")
```




```
hist(samples, breaks = 30, main = "Histogram of Metropolis-Hastings samples")
```



Convergence Diagnostics

Gelman-Rubin Method Overview

The Gelman-Rubin method is a key convergence diagnostic used in Markov Chain Monte Carlo (MCMC) to evaluate whether a chain has converged to its target distribution. It is particularly useful in assessing convergence by comparing variance within and between multiple chains.

Purpose of the Gelman-Rubin Diagnostic

1. **Monitor Convergence**: Helps determine if each chain has reached the target distribution.
2. **Compare Variances**:
    - **Within-chain variance**: Measures variation within each individual chain.
    - **Between-chain variance**: Measures variation between different chains.

   By comparing these variances, we can assess if all chains are sampling from the same distribution.

Why Use Multiple Starting Points?

- **Diversity in Exploration**: Starting chains at different points ensures that various regions of the parameter space are explored.
- **Convergence Evidence**: If chains starting from different points converge to a similar distribution, it suggests that the chains are sampling from the target distribution.

Calculating the Gelman-Rubin Statistic

The Gelman-Rubin diagnostic calculates the **Potential Scale Reduction Factor (PSRF)**, also referred to as $\hat{R}$. This factor provides a measure of how much each chain differs from others in the group, giving insight into convergence.

1. **Run Multiple Chains**: Typically, two or more chains with different starting points.
2. **Calculate Between-Chain and Within-Chain Variances**:
    - **Within-chain variance** W: Average of variances within each chain.
    - **Between-chain variance** B: Variance across the means of each chain.
3. **Calculate** $\hat{R}$ **(PSRF)**:
    - Using the formula:

$$\hat{R} = \sqrt{\frac{\hat{V}}{W}}$$

    - Where $\hat{V}$ is the weighted average of W and B, computed as:

$$\hat{V} = \frac{n-1}{n}W + \frac{B}{n}$$

    - Here, n is the number of samples in each chain.
4. **Interpretation**:

- $\hat{R} \approx 1$: Convergence is typically indicated when $\hat{R}$ values are close to 1. A common threshold is $\hat{R} < 1.1$, suggesting that within-chain and between-chain variances are similar, indicating the chains are likely sampling from the same distribution.
- $\hat{R} > 1.1$: Indicates lack of convergence, suggesting further iterations or adjustments may be needed.

Advantages of the Gelman-Rubin Diagnostic

- **Robust to Initial Values**: By using multiple chains, it avoids issues with single chain initialization.
- **Quantitative Measure**: Provides a numeric value for convergence, making it easier to assess convergence rigorously.

Practical Implementation in R

```r
# Example implementation of Gelman-Rubin Diagnostic

# Load required package
library(coda)

# Simulate MCMC chains
set.seed(123)
chain1 <- rnorm(1000, mean = 5, sd = 2)
chain2 <- rnorm(1000, mean = 5, sd = 2)
chain3 <- rnorm(1000, mean = 5, sd = 2)

# Combine chains into an mcmc.list object
mcmc_chains <- mcmc.list(as.mcmc(chain1), as.mcmc(chain2), as.mcmc(chain3))

# Calculate Gelman-Rubin diagnostic
gelman_diag <- gelman.diag(mcmc_chains, autoburnin = FALSE)
print(gelman_diag)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]          1          1
```

```r
# Interpretation of Rhat
cat("Gelman-Rubin diagnostic (Rhat):", gelman_diag$psrf[, "Point est."])
```

```
## Gelman-Rubin diagnostic (Rhat): 1.000334
```

Bootstrap Methods

Motivation

The bootstrap method, introduced by Brad Efron in 1979, is a powerful, nonparametric Monte Carlo method. It allows for estimating the sampling distribution of a statistic by resampling with replacement from observed data. Bootstrap methods are widely used when the population distribution is unknown or the sample size is small, providing flexibility in statistical inference without relying on parametric assumptions.

- **Applications**:
  - **Bias Estimation**: Helps identify and measure the systematic deviation of an estimator from the true population parameter.
  - **Confidence Interval Construction**: Useful for estimating intervals when the exact distribution of a statistic is unknown.
  - **Prediction Error Estimation**: Bootstrap provides a straightforward way to assess model prediction accuracy.
  - **Sampling Variability Assessment**: Essential when studying the variation in an estimate due to random sampling effects.

Key Definitions

1. **Estimator**: A rule or function that calculates an estimate of a population parameter from the sample data, such as the sample mean as an estimator of the population mean.
2. **Estimate**: The computed value obtained by applying an estimator to a specific dataset, representing our best guess for the population parameter.
3. **Parameter**: A fixed quantity describing the entire population, such as the population mean or variance.
4. **Function**: In statistical modeling, a rule assigning elements from one set to elements in another set.
5. **Predictive Model**: A model used to predict future data points based on patterns identified in historical data.
6. **Decision Boundary**: In classification, this is the threshold or surface that separates different classes based on model predictions.

Types of Uncertainty

1. **Bias**: Systematic deviation from the true parameter. High bias implies an estimator consistently overshoots or undershoots the target.
    - *Example*: Using an estimator that consistently underestimates the population mean demonstrates bias.
2. **Sampling Variability**: Variation in a statistic across different random samples drawn from the same population.
3. **Confidence Interval**: A range of values within which a population parameter is likely to fall, given a specified level of confidence (e.g., 95%).
4. **Prediction Error**: The discrepancy between observed values and values predicted by a model, encompassing the effects of assumptions and variability.

Detailed Bootstrap Procedure

1. **Draw B Bootstrap Samples**: For each of the B bootstrap iterations, resample with replacement from the original data to form a new "bootstrap sample."
2. **Calculate the Statistic for Each Sample**: For each bootstrap sample, compute the statistic of interest, such as the mean or median.
3. **Use the Bootstrap Distribution**: With the bootstrap estimates, calculate properties such as bias, variance, and construct confidence intervals.

Each bootstrap sample approximates the original dataset, treating it as a pseudo-population to study how sample statistics behave across random samples.

Confidence Intervals Using Bootstrap

There are multiple methods to construct bootstrap confidence intervals, each with specific assumptions and advantages. Here's a breakdown of each method with examples of when to use them:

1. **Standard Normal Bootstrap**: Assumes that the bootstrap distribution of the statistic is approximately normal.
    - *Use Case*: Effective when the bootstrap distribution of the statistic appears symmetric and normal-like.
    - *Example*: Constructing confidence intervals for large sample means or differences between means.
2. **Bootstrap t Confidence Interval**: Uses the t-distribution, which adjusts for small sample sizes, making it ideal for limited data.
    - *Use Case*: Useful for small sample sizes or when the population variance is unknown.
    - *Example*: Estimating the mean difference between two small samples.
3. **Basic Bootstrap**: Directly adjusts percentiles of the bootstrap distribution to form an interval.
    - *Use Case*: Appropriate when the bootstrap distribution is asymmetric, but normality isn't assumed.

- *Example*: Calculating a confidence interval for the median in a skewed data distribution.

4. **Percentile Bootstrap**: Constructs confidence intervals by directly using empirical percentiles (e.g., 2.5% and 97.5%) from the bootstrap sample distribution.
   - *Use Case*: When a distribution-free, straightforward method is needed, and symmetry is not essential.
   - *Example*: Creating a confidence interval for a robust statistic like the median in non-normal data.
5. **Bias-Corrected and Accelerated (BCa) Bootstrap**: Adjusts for both bias and skewness in the bootstrap distribution, providing a more refined interval for skewed data.
   - *Use Case*: Best for heavily skewed distributions or when bias in the estimate is suspected.
   - *Example*: Constructing intervals for a non-linear estimator, such as a ratio or complex regression coefficient.

Practical Implementation

Example Code for Bootstrap in R

This example demonstrates a bootstrap procedure to calculate a statistic (e.g., mean) and construct confidence intervals.

```
# Load necessary library
library(boot)

# Generate example data
set.seed(123)
data <- rnorm(100)

# Define a function to calculate the statistic (e.g., mean)
stat_func <- function(data, indices) {
  return(mean(data[indices]))
}

# Perform bootstrap
bootstrap_results <- boot(data = data, statistic = stat_func, R = 1000)
print(bootstrap_results)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = data, statistic = stat_func, R = 1000)
##
##
## Bootstrap Statistics :
##       original      bias    std. error
## t1* 0.09040591 0.002019936  0.08894492
```

```
# Calculate confidence intervals using different methods
boot.ci(bootstrap_results, type = c("norm", "basic", "perc", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bootstrap_results, type = c("norm", "basic",
##     "perc", "bca"))
##
## Intervals :
## Level      Normal              Basic
## 95%   (-0.0859,  0.2627 )   (-0.0906,  0.2626 )
##
## Level     Percentile            BCa
## 95%   (-0.0818,  0.2715 )   (-0.0842,  0.2681 )
## Calculations and Intervals on Original Scale
```

Summary

The key topics covered were:

- **Markov Chain Monte Carlo (MCMC) and Metropolis-Hastings Algorithm**: We explored the Metropolis-Hastings algorithm, a method for sampling from complex probability distributions by constructing a Markov chain that converges to the target distribution. This process involves a proposal distribution and an acceptance criterion, allowing the algorithm to approximate the target distribution over time.

Additionally, we reviewed the Gibbs Sampler, a variation that updates variables conditionally; it can proceed in a **systematic scan** (sequentially updating each variable) or **random scan** (updating one randomly selected variable per iteration). These approaches vary in efficiency and convergence properties depending on dependencies between variables.

- **Convergence Diagnostics (Gelman-Rubin Method)**: Convergence diagnostics are crucial in MCMC to verify if the sampling has reached the target distribution. The Gelman-Rubin diagnostic, through the Potential Scale Reduction Factor (PSRF), compares within-chain and between-chain variances to assess convergence. Running multiple chains with different starting points provides robust evidence that the chains have converged to the same distribution, ensuring reliable results.
- **Bootstrap Methods**: The bootstrap is a nonparametric approach to statistical inference, valuable for scenarios where the population distribution is unknown. By resampling with replacement from observed data, bootstrap methods allow us to estimate bias, sampling variability, and prediction error.

Bootstrap confidence intervals, including Standard Normal, Percentile, and Bias-Corrected and Accelerated (BCa) intervals, provide versatile tools for constructing intervals across varying data distributions.

These tools — MCMC for approximating complex distributions, Gelman-Rubin for assessing convergence, and bootstrap methods for inference — form a comprehensive foundation for robust statistical analysis, ensuring accuracy and reliability in both research and practical applications.