Gibbs Sampling in LDA

Promothesh Chatterjee*

Gibbs Sampling for LDA: A Step-by-Step Example

Gibbs Sampling is a key component of the inference process in Latent Dirichlet Allocation (LDA). LDA is a generative probabilistic model used for discovering the underlying topics in a collection of documents. Here's a detailed explanation of where and how Gibbs Sampling fits into the LDA framework:

Overview of LDA

LDA assumes the following generative process for a document:

1. **Choose a distribution over topics** for each document m, denoted by m, from a Dirichlet distribution with parameter .
2. **For each word** in the document:
    - Choose a topic Zmn from the multinomial distribution m.
    - Choose a word Wm from the multinomial distribution zmn, where k is the word distribution for topic k drawn from a Dirichlet distribution with parameter .

Inference in LDA

The goal of inference in LDA is to estimate the hidden topic structure that generated the observed collection of documents. Specifically, we want to determine:

- The distribution of topics in each document (m).
- The distribution of words in each topic (k).
- The topic assignment for each word in each document (zmn).

Role of Gibbs Sampling

Gibbs Sampling is used to perform approximate inference by sampling from the posterior distribution of the hidden variables given the observed data.

Step 1: Start with Random Assignments Imagine we start by randomly guessing which topic each word belongs to. For example, we might randomly assign "room" to the "food" topic, "clean" to the "service" topic, and so on.

Step 2: Refine the Guesses

1. **Look at Each Word**: We look at each word in each review one by one. Let's say we look at the word "clean" in Review 1.
2. **Temporarily Forget the Current Topic**: We temporarily remove "clean" from its current topic. It's like saying, "Let's forget our initial guess for this word."
3. **Calculate Probabilities**: We calculate how likely "clean" is to belong to each possible topic:
    - **Rooms**: We check how common "clean" is in the "rooms" topic across all reviews.
    - **Food**: We check how common "clean" is in the "food" topic.
    - **Service**: We check how common "clean" is in the "service" topic. We also consider how common each topic is in Review 1.
4. **Make a New Guess**: Based on these probabilities, we make a new guess about which topic "clean" belongs to. We assign "clean" to a new topic according to the calculated probabilities.

Step 3: Repeat Many Times We repeat this process for every word in every review. For example, we next look at "comfortable" in Review 1, then "quiet", and so on. After we go through all words in all reviews, we start again. Each time we do this, our guesses get better.

Final Outcome

After repeating this process many times, we end up with:

- **Topic-Word Distributions**: A list showing the probability of each word belonging to each topic. For example, "clean" might have a high probability of being in the "rooms" topic.
- **Document-Topic Distributions**: A list showing the probability of each topic being present in each review. For example, Review 1 might be 80% about "rooms," 10% about "food," and 10% about "service".
- **Topic Assignments**: A final assignment of topics to each word in each review. This helps us understand which words are grouped together under each topic.

Gibbs Sampling for LDA: A Step-by-Step Example with Calculations

Dataset and Initial Setup Let's use a synthetic dataset of hotel reviews, where each review contains words related to rooms, food, and service.

Synthetic Dataset: 1. **Review 1**: "room", "clean", "comfortable", "quiet" 2. **Review 2**: "food", "delicious", "breakfast", "fresh" 3. **Review 3**: "service", "friendly", "helpful", "staff" 4. **Review 4**: "room", "spacious", "clean", "view" 5. **Review 5**: "food", "tasty", "dinner", "variety" 6. **Review 6**: "service", "courteous", "professional", "staff" 7. **Review 7**: "room", "quiet", "clean", "comfortable" 8. **Review 8**: "food", "fresh", "lunch", "delicious" 9. **Review 9**: "service", "helpful", "staff", "courteous" 10. **Review 10**: "room", "view", "spacious", "clean"

Initial Setup: - **Topics**: Rooms, Food, Service - **Words**: "room", "clean", "comfortable", "quiet", "food", "delicious", "breakfast", "fresh", "service", "friendly", "helpful", "staff", "spacious", "view", "tasty", "dinner", "variety", "courteous", "professional", "lunch"

Step 1: Start with Random Assignments

Imagine we start by randomly guessing which topic each word belongs to. For example, we might randomly assign "room" to the "food" topic, "clean" to the "service" topic, and so on.

Assumptions for Initial Counts:

- "clean" in "Rooms" topic: 3
- "clean" in "Food" topic: 1
- "clean" in "Service" topic: 1
- Total words in "Rooms" topic: 30
- Total words in "Food" topic: 20
- Total words in "Service" topic: 20
- Words in Review 1 assigned to "Rooms" topic: 2
- Words in Review 1 assigned to "Food" topic: 1
- Words in Review 1 assigned to "Service" topic: 1
- Total words in Review 1: 4

Hyperparameters: - Alpha ( ): 0.1 - Beta ( ): 0.01

Calculating Probabilities

Let's walk through calculating the probabilities for the word "clean" in Review 1.

Temporarily Remove the Word First, we temporarily remove "clean" from its current topic assignment. This means we pretend "clean" doesn't belong to any topic for a moment.

Calculate the Probability for Each Topic We calculate how likely it is for "clean" to belong to each of the three topics: Rooms, Food, and Service.

Factor 1: Probability of Word Given Topic

- **Rooms**:
  - Count of "clean" in "Rooms" topic: 3
  - Total words in "Rooms" topic: 30
  - Smoothed count: `3 + 0.01 = 3.01`
  - Smoothed total: `30 + 20 * 0.01 = 30.2`
  - `P(clean | Rooms) = 3.01 / 30.2  0.0997`
- **Food**:
  - Count of "clean" in "Food" topic: 1
  - Total words in "Food" topic: 20
  - Smoothed count: `1 + 0.01 = 1.01`
  - Smoothed total: `20 + 20 * 0.01 = 20.2`
  - `P(clean | Food) = 1.01 / 20.2  0.0500`
- **Service**:
  - Count of "clean" in "Service" topic: 1
  - Total words in "Service" topic: 20
  - Smoothed count: `1 + 0.01 = 1.01`
  - Smoothed total: `20 + 20 * 0.01 = 20.2`
  - `P(clean | Service) = 1.01 / 20.2  0.0500`

Factor 2: Probability of Topic Given Document

- **Rooms**:
  - Words in Review 1 assigned to "Rooms" topic: 2
  - Total words in Review 1: 4
  - Smoothed count: `2 + 0.1 = 2.1`
  - Smoothed total: `4 + 3 * 0.1 = 4.3`
  - `P(Rooms | Review1) = 2.1 / 4.3  0.4884`
- **Food**:
  - Words in Review 1 assigned to "Food" topic: 1
  - Total words in Review 1: 4
  - Smoothed count: `1 + 0.1 = 1.1`
  - Smoothed total: `4 + 3 * 0.1 = 4.3`
  - `P(Food | Review1) = 1.1 / 4.3  0.2558`
- **Service**:
  - Words in Review 1 assigned to "Service" topic: 1
  - Total words in Review 1: 4
  - Smoothed count: `1 + 0.1 = 1.1`
  - Smoothed total: `4 + 3 * 0.1 = 4.3`
  - `P(Service | Review1) = 1.1 / 4.3  0.2558`

Combined Probabilities To get the combined probability of assigning "clean" to each topic, we multiply the probabilities from Factor 1 and Factor 2:

- **Rooms**:
  - `P(Rooms) = P(clean | Rooms) * P(Rooms | Review1)`
  - `= 0.0997 * 0.4884`
  - `0.0487`
- **Food**:
  - `P(Food) = P(clean | Food) * P(Food | Review1)`
  - `= 0.0500 * 0.2558`
  - `0.0128`
- **Service**:
  - `P(Service) = P(clean | Service) * P(Service | Review1)`
  - `= 0.0500 * 0.2558`
  - `0.0128`

Normalize the Probabilities Finally, we normalize these probabilities so that they sum to 1:

- Total probability: `0.0487 + 0.0128 + 0.0128 = 0.0743`
- Normalized probabilities:
  - `P'(Rooms) = 0.0487 / 0.0743  0.6556`
  - `P'(Food) = 0.0128 / 0.0743  0.1722`
  - `P'(Service) = 0.0128 / 0.0743  0.1722`

Interpretation

- The normalized probability `P'(Rooms)  0.6556` means that there is a 65.56% chance that "clean" should belong to the "Rooms" topic.
- The normalized probability `P'(Food)  0.1722` means that there is a 17.22% chance that "clean" should belong to the "Food" topic.
- The normalized probability `P'(Service)  0.1722` means that there is a 17.22% chance that "clean" should belong to the "Service" topic.

Using these probabilities, we would reassign the word "clean" to the most likely topic, which in this case is "Rooms". Use the normalized probabilities to make a new guess about the topic for each word. This is done by sampling from the probability distribution. For example, if a word has a higher probability of belonging to Topic 1 compared to Topic 2, it is more likely to be assigned to Topic 1 in the next iteration.

Repeat this process for all words in all reviews multiple times. Each iteration refines the topic assignments based on the updated guesses from the previous step. Through many iterations, the algorithm converges towards a stable set of topic assignments, effectively uncovering the hidden topics within the documents. This iterative refinement is key to the effectiveness of Gibbs Sampling in LDA.

Summary

In Gibbs Sampling for LDA, we calculate the probability of assigning each word to each topic by considering: 1. **How often the word appears in each topic across all reviews (with smoothing)**. 2. **How common each topic is in the current review (with smoothing)**.

We then combine these probabilities, normalize them, and use them to make a new guess about the topic for each word. This process is repeated for all words in all reviews many times, refining our guesses each time to better identify the hidden topics.

Gibbs Sampling using R code

Step 1: Create a Synthetic Dataset Let's make up some hotel reviews. Each review talks about different things like rooms, food, or service. The variable 'documents' in the code below can be thought of as tokenized reviews after pre-processing such as removing stop-words etc.

```r
# Load necessary libraries
library(tidyverse)
library(data.table)

# Create a synthetic dataset of hotel reviews
documents <- list(
  c("room", "clean", "comfortable", "quiet"),
  c("food", "delicious", "breakfast", "fresh"),
  c("service", "friendly", "helpful", "staff"),
  c("room", "spacious", "clean", "view"),
  c("food", "tasty", "dinner", "variety"),
  c("service", "courteous", "professional", "staff"),
  c("room", "quiet", "clean", "comfortable"),
  c("food", "fresh", "lunch", "delicious"),
  c("service", "helpful", "staff", "courteous"),
  c("room", "view", "spacious", "clean")
)

# Define vocabulary and topics
vocab <- c("room", "clean", "comfortable", "quiet", "spacious", "view",
           "food", "delicious", "breakfast", "fresh", "tasty", "dinner", "variety",          "lunch",
K <- 3  # Number of topics
V <- length(vocab)  # Vocabulary size
D <- length(documents)  # Number of documents
```

We have 10 reviews (documents), and each review contains words related to topics 'rooms', 'food', or 'service'. The vocabulary is a list of all unique words used in the reviews.

Step 2: Initialize Parameters

```r
# Randomly assign initial topics to words
set.seed(123)
z <- lapply(documents, function(doc) sample(1:K, length(doc), replace = TRUE))
```

- `set.seed(123)`: This sets the random seed to 123, ensuring that the random assignments are reproducible. This means if you run the code again, you will get the same random assignments.
- `z <- lapply(documents, function(doc) sample(1:K, length(doc), replace = TRUE))`: This line assigns a random topic to each word in each document.
  - `lapply(documents, function(doc) ...)` applies the function to each document in the `documents` list.
  - `sample(1:K, length(doc), replace = TRUE)` samples a number between 1 and `K` (the number of topics) for each word in the document. `length(doc)` specifies how many samples to draw (one for each word), and `replace = TRUE` allows the same topic to be assigned to multiple words.

```r
# Count matrices
nw <- matrix(0, nrow = K, ncol = V)  # word-topic counts
nd <- matrix(0, nrow = D, ncol = K)  # document-topic counts
nwsum <- rep(0, K)  # total word count for each topic
ndsum <- sapply(documents, length)  # total word count for each document
```

- `nw <- matrix(0, nrow = K, ncol = V)`: This creates a `K` x `V` matrix filled with zeros, where `K` is the number of topics and `V` is the size of the vocabulary. This matrix will count how many times each word is assigned to each topic.
- `nd <- matrix(0, nrow = D, ncol = K)`: This creates a `D` x `K` matrix filled with zeros, where `D` is the number of documents. This matrix will count how many times each topic is assigned to words in each document.
- `nwsum <- rep(0, K)`: This creates a vector of zeros with length `K`. This vector will count the total number of words assigned to each topic.
- `ndsum <- sapply(documents, length)`: This creates a vector containing the total number of words in each document. `sapply(documents, length)` applies the `length` function to each document in the `documents` list, returning the number of words in each document.

```r
# Initialize counts
for (d in 1:D) {
  for (n in 1:length(documents[[d]])) {
    topic <- z[[d]][n]
    word <- match(documents[[d]][n], vocab)
    nw[topic, word] <- nw[topic, word] + 1
    nd[d, topic] <- nd[d, topic] + 1
    nwsum[topic] <- nwsum[topic] + 1
  }
}
```

- `for (d in 1:D) { ... }`: This outer loop iterates over each document `d` from 1 to `D`.
- `for (n in 1:length(documents[[d]])) { ... }`: This inner loop iterates over each word `n` in document `d`.
- `topic <- z[[d]][n]`: This retrieves the topic assigned to the `n`-th word in the `d`-th document from the `z` list.
- `word <- match(documents[[d]][n], vocab)`: This finds the index of the `n`-th word in the `d`-th document within the vocabulary list `vocab`.
- `nw[topic, word] <- nw[topic, word] + 1`: This increments the count of the `word` in the `topic` by 1 in the `nw` matrix.
- `nd[d, topic] <- nd[d, topic] + 1`: This increments the count of the `topic` in the `d`-th document by 1 in the `nd` matrix.
- `nwsum[topic] <- nwsum[topic] + 1`: This increments the total word count for the `topic` by 1 in the `nwsum` vector.

```r
# Hyperparameters
alpha <- 0.1
beta <- 0.01
```

- `alpha <- 0.1`: This sets the hyperparameter `alpha` to 0.1. `alpha` controls the distribution of topics in each document. A smaller `alpha` makes the document more likely to be dominated by a few topics.
- `beta <- 0.01`: This sets the hyperparameter `beta` to 0.01. `beta` controls the distribution of words in each topic. A smaller `beta` makes the topics more likely to be dominated by a few words.

This setup prepares the data and parameters for running the Gibbs Sampling algorithm in the next step. #### Step 3: Gibbs Sampling Iterations

Sure! Let's break down each line of Step 3 in detail.

Step 3: Gibbs Sampling Iterations

```
# Gibbs sampling function
gibbs_sampler <- function(iterations) {
  for (iter in 1:iterations) {
    for (d in 1:D) {
      for (n in 1:length(documents[[d]])) {
        word <- match(documents[[d]][n], vocab)
        topic <- z[[d]][n]
```

- `gibbs_sampler <- function(iterations) { ... }`: This defines a function `gibbs_sampler` that takes the number of iterations as an argument. The function runs Gibbs Sampling for the specified number of iterations.
- `for (iter in 1:iterations) { ... }`: This outer loop runs the Gibbs Sampling process for the specified number of iterations.
- `for (d in 1:D) { ... }`: This loop iterates over each document `d` from 1 to `D`.
- `for (n in 1:length(documents[[d]])) { ... }`: This loop iterates over each word `n` in document `d`.
- `word <- match(documents[[d]][n], vocab)`: This finds the index of the `n`-th word in the `d`-th document within the vocabulary list `vocab`.
- `topic <- z[[d]][n]`: This retrieves the topic currently assigned to the `n`-th word in the `d`-th document from the `z` list.

```
        # Decrease counts
        nw[topic, word] <- nw[topic, word] - 1
        nd[d, topic] <- nd[d, topic] - 1
        nwsum[topic] <- nwsum[topic] - 1
```

- `nw[topic, word] <- nw[topic, word] - 1`: This decreases the count of the `word` in the `topic` by 1 in the `nw` matrix. We do this because we are going to reassign the topic, so we need to temporarily remove this word from the counts.
- `nd[d, topic] <- nd[d, topic] - 1`: This decreases the count of the `topic` in the `d`-th document by 1 in the `nd` matrix.
- `nwsum[topic] <- nwsum[topic] - 1`: This decreases the total word count for the `topic` by 1 in the `nwsum` vector.

```
        # Compute probabilities for each topic
        p <- (nw[, word] + beta) / (nwsum + V * beta) * (nd[d, ] + alpha) / (ndsum[d] + K * alpha)
        p <- p / sum(p)
```

- `p <- (nw[, word] + beta) / (nwsum + V * beta) * (nd[d, ] + alpha) / (ndsum[d] + K * alpha)`: This calculates the probability of assigning each topic to the `n`-th word in the `d`-th document.
  - `nw[, word] + beta`: This is the smoothed count of the word in each topic.
  - `nwsum + V * beta`: This is the smoothed total count of words in each topic.
  - `(nw[, word] + beta) / (nwsum + V * beta)`: This is the probability of the word given the topic.

  - `nd[d, ] + alpha`: This is the smoothed count of the topic in the document.
  - `ndsum[d] + K * alpha`: This is the smoothed total count of words in the document.
  - `(nd[d, ] + alpha) / (ndsum[d] + K * alpha)`: This is the probability of the topic given the document.
  - The product of these two probabilities gives the joint probability of the word and topic given the document.

- `p <- p / sum(p)`: This normalizes the probabilities so that they sum to 1.

```
        # Sample new topic
        new_topic <- sample(1:K, 1, prob = p)
```

- `new_topic <- sample(1:K, 1, prob = p)`: This samples a new topic for the `n`-th word in the `d`-th document based on the computed probabilities `p`.

```
        # Increase counts with new topic
        z[[d]][n] <- new_topic
        nw[new_topic, word] <- nw[new_topic, word] + 1
        nd[d, new_topic] <- nd[d, new_topic] + 1
        nwsum[new_topic] <- nwsum[new_topic] + 1
      }
    }
  }
}
```

- `z[[d]][n] <- new_topic`: This assigns the new topic to the `n`-th word in the `d`-th document.
- `nw[new_topic, word] <- nw[new_topic, word] + 1`: This increases the count of the `word` in the new topic by 1 in the `nw` matrix.
- `nd[d, new_topic] <- nd[d, new_topic] + 1`: This increases the count of the new topic in the `d`-th document by 1 in the `nd` matrix.
- `nwsum[new_topic] <- nwsum[new_topic] + 1`: This increases the total word count for the new topic by 1 in the `nwsum` vector.

Summary

- **Define Gibbs Sampler Function**: We define a function `gibbs_sampler` to run the Gibbs Sampling process for a specified number of iterations.
- **Iterate Over Documents and Words**: We loop through each word in each document.
- **Decrease Current Topic Counts**: Temporarily remove the word from its current topic assignment to recalculate probabilities.
- **Compute Topic Probabilities**: Calculate the probability of each topic for the current word based on the current state of the model.
- **Sample a New Topic**: Assign a new topic to the word based on the computed probabilities.
- **Increase New Topic Counts**: Update the count matrices with the new topic assignment.

This step refines the topic assignments for each word in each document iteratively, improving the model's understanding of the underlying topics. #### Step 4: Inspect Results

Sure! Let's break down each line of Step 4 in detail.

Step 4: Inspect Results

```r
# Topic-word distribution
phi <- (nw + beta) / (rowSums(nw) + V * beta)
rownames(phi) <- paste("Topic", 1:K)
colnames(phi) <- vocab
phi
```

- `phi <- (nw + beta) / (rowSums(nw) + V * beta)`: This calculates the topic-word distribution matrix `phi`.
  - `nw + beta`: This adds the smoothing parameter `beta` to each element of the word-topic count matrix `nw`.
  - `rowSums(nw) + V * beta`: This calculates the total number of words in each topic, with smoothing.
  - `(nw + beta) / (rowSums(nw) + V * beta)`: This normalizes the counts to get the probabilities of each word given each topic.
- `rownames(phi) <- paste("Topic", 1:K)`: This sets the row names of the `phi` matrix to "Topic 1", "Topic 2", etc.
- `colnames(phi) <- vocab`: This sets the column names of the `phi` matrix to the words in the vocabulary.
- `phi`: Display the topic-word distribution matrix `phi`.

This matrix `phi` shows the probability of each word being associated with each topic.

```r
# Document-topic distribution
theta <- (nd + alpha) / (rowSums(nd) + K * alpha)
rownames(theta) <- paste("Document", 1:D)
theta
```

- `theta <- (nd + alpha) / (rowSums(nd) + K * alpha)`: This calculates the document-topic distribution matrix `theta`.
  - `nd + alpha`: This adds the smoothing parameter `alpha` to each element of the document-topic count matrix `nd`.
  - `rowSums(nd) + K * alpha`: This calculates the total number of words in each document, with smoothing.
  - `(nd + alpha) / (rowSums(nd) + K * alpha)`: This normalizes the counts to get the probabilities of each topic given each document.
- `rownames(theta) <- paste("Document", 1:D)`: This sets the row names of the `theta` matrix to "Document 1", "Document 2", etc.
- `theta`: Display the document-topic distribution matrix `theta`.

This matrix `theta` shows the probability of each topic being present in each document.

```r
# Final topic assignments for each document
z
```

- `z`: This displays the final topic assignments for each word in each document. The `z` list contains the topic assignments for each word in each document after running the Gibbs Sampling iterations.

Summary

- **Calculate Topic-Word Distribution**: The matrix `phi` shows the probability of each word given each topic, providing insights into which words are associated with which topics.
- **Calculate Document-Topic Distribution**: The matrix `theta` shows the probability of each topic given each document, indicating the topic composition of each document.
- **Display Final Topic Assignments**: The list `z` contains the final topic assignments for each word in each document, showing how the words are grouped into topics.

These results give us a clear understanding of the topics discovered by the LDA model and how the words and documents are associated with these topics. ### Conclusion

In this example, we used LDA with Gibbs Sampling to find hidden topics in a set of hotel reviews. We started with random guesses and used Gibbs Sampling to refine our guesses over many iterations. The final result shows us which words are associated with which topics and helps us understand what people are talking about in the reviews.