UNIVERSITY OF CALIFORNIA

Los Angeles

Comparison of Different Hierarchical Dirichlet Process Implementations

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Statistics

by

Peichen Wu

2020

ABSTRACT OF THE THESIS

Comparison of Different Hierarchical Dirichlet Process Implementations

by

Peichen Wu
Master of Science in Statistics
University of California, Los Angeles, 2020
Professor Arash Amini, Chair

The Hierarchical Dirichlet Process (HDP) is an important Bayesian nonparametric model for grouped data, such as corpus or document collections. It can be very useful in an NLP setting where we are trying to classify documents in a corpus. A great advantage of HDP is its flexibility: we do not need to specify the number of components (or topics) we want and can instead let the data decide. Like other Bayesian nonparametric models, exact posterior inference is intractable, instead we can use Monte Carlo Markov Chain (MCMC) methods to estimate the posterior distribution, and different MCMC methods can affect the performance of the HDP implementation. In this thesis, we will compare four different HDP samplers by applying them to a set of simulated data and a set of real data, and we will do this by comparing the mixing time of their NMI (normalized mutual information, which can be considered as the "amount of information" obtained about one variable by observing the other variable) and perplexity.

The thesis of Peichen Wu is approved.

University of California, Los Angeles

2020

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

In Natural Language Processing (NLP), a problem we often encounter is the bag-of-words problem, where we are given a corpus with a number of documents, each containing varying number of words, and we want to classify the documents into different groups. A machine learning model designed for this purpose is a topic model, a type of statistical model for discovering the abstract "topics" for a corpus, or a collection of documents. A topic model assigns a "topic" to each document by looking at hidden semantic structures in a text body. Intuitively, if a document is on a specific topic, then there are certain words that are likely to appear with higher or lower frequencies, for example a document about computers are more likely to contain words such as "CPU" and "software" while a document about automobiles can have words "speed" and "transmission" more frequently.

There are several widely-used topic modeling methods, one of which is the Latent Dirichlet Allocation (LDA). This is a generative model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. But for LDA, we would have to specify the number of groups for the model. In real life, we usually do not know the "true" number of groups, so we want a non-parametric method that would "learn" the number of groups on its own. A good choice would be the Hierarchical Dirichlet Process. Unlike LDA, HDP determines the "optimal" number of groups through posterior sampling and classify each document accordingly.

But is this good enough? HDP does have a major improvement over the older LDA, but what if we want the algorithm to be faster? There are several alternative posterior sampling methods proposed for HDP to increase speed, including a split-merge method and a slice sampler algorithm. To the best of our knowledge, no literature has compared the performance of these methods yet, so this is what we will do in this thesis. More specifically, we will perform the following tasks:

- Generating simulated corpus data and test the performance of different HDP packages and implementations. I will use the Normalized Mutual Information (NMI) as a measure of "accuracy", and mixing time will be measured by the number of iterations it takes for the NMI to converge. I am also interested in how the number of documents in the corpus would affect the mixing time.
- Finding and implementing a reliable change point detection method to estimate the mixing time of the NMI curve.
- Finding and implementing an algorithm to estimate the perplexity of the models as a different measure of model performances so that we don't need to know the true labels to evaluate the model.

1.1 Hierarchical Dirichlet Process

Hierarchical Dirichlet Process was proposed by Teh et. al. in [Ta2005] as a non-parametric Bayesian approach to clustering grouped data. The paper uses the Chinese restaurant franchise process to describe the formulation of the HDP model and discusses its posterior sampling process. The paper also includes experiments comparing the performance of HDP and LDA models using perplexity as the evaluation metric and showed that the non-parametric HDP is as good as the best possible LDA. Since we have to pre-specify the number of groups for a LDA model, to find the "best possible" LDA, the experiment trained the corpus using LDA models with different pre-specified number of topics, and the LDA model with the best perplexity is the "best possible" LDA. In comparison, HDP automatically achieves the "optimal" perplexity without the need of parameter-tuning. A shorter version of the paper can be found at [Tb2005].

As for most nonparametric Bayesian models for grouped data, exact posterior inference is intractable for HDP, so we need to use a Monte Carlo Markove Chain sampling method for posterior sampling. The HDP formulation by Teh et. al. uses Gibbs sampler for posterior sampling, but Gibbs sampler can be slow to mix since it only allows changing the topic status of one observed word at a time. To counter this drawback, several alternative implementations are proposed, including the split-merge sampler and the slice sampler.

The split-merge MCMC sampler implementation is proposed by Wang et. al. in [W2012]. While mostly based on the Gibbs sampler, the split-merge sampler has an additional split-merge step: two observations are picked at random, if the observations are in the same component then a "split" operation is proposed, otherwise a "merge" operation is proposed, and whether the resulting state after the "split" or "merge" operation is accepted is determined by Metropolis-Hasting. According to the paper, the implementation is more effective for DP mixtures when the mixture models have overlapping clusters.

Amini et. al. proposed another variation of the HDP model with slice sampler for posterior sampling in [AA2019]. This implementation is based on [G2009] where Slice Sampler is used for Dirichlet process mixture models. Slice sampler has the desirable properties of fast mixing time and potential for parallelization, so in theory the proposed implementation should mix faster than previous HDP implementations.

1.2 Evaluating Topic Models

Now we need to decide how do we want to evaluate and compare these different implementations of HDP. First, we need a measurement of "accuracy" for the HDP models. For a simulated corpus that has a "true" topic for each document, we can look at how closely the predicted topics matches the "true" topics. One such measurement is the normalized mutual information (NMI), which can be thought of as a measurement of mutual dependence of two variables. For real data, we do not know the true topics of each document, so instead we will use held-out perplexity, which is a statistical measure of how well a probability model predicts a sample. The main idea is to calculate how likely it is for each word to be in the document in the testing set given all other words in the document and the topic-word assignment matrix from the training set. Wallach et. al. listed some common approaches for estimating perplexity in [W2009], including the Harmonic Means method and Chib-style estimator, and proposed some original methods, such as the Left-to-Right method. We will go over some of these methods in more details in later sections.

After determining the measurement of "accuracy", we want to observe how the "accuracy" of the HDP models change with each iteration of posterior sampling. Let's say we are working with a simulated corpus and use NMI as out measurement of "accuracy". In theory, NMI should increase with each iteration and converge to a "final value", and what we are most interested in is at which iteration does the NMI converge, or "mix". This is equivalent to finding a single change point in the NMI series.

A widely used method for change point detection is binary segmentation, which is a consistent estimation of the number and locations of multiple change-points in data. The main idea of binary segmentation is to search for a single change point in the data series using a CUSUM-like procedure, and when a change point is detected, the series is split into two segments by the change point, and a similar search is performed in both segments, possibly leading to further splits, until a certain criteria is satisfied. A variant of such method called wild binary segmentation is proposed by Fryzlewicz et. al. in [F2014]. Instead of taking the global CUSUM, wild binary segmentation randomly draw a number of subsamples and take the CUSUM of each subsample. This method works better than binary segmentation when the spacing between change points is short. Another version of binary segmentation for change point detection is proposed by Padilla et. al. in [P2019].

CHAPTER 2

Hierarchical Dirichlet Process

Hierarchical Dirichlet process (HDP) is a non-parametric Bayesian model for clustering problems involving several groups of data, with the number of classes inferred automatically by the model. An example would be classifying each document in a copra. The hierarchical structure ensures that different documents can share the same component distribution so that dependencies between groups can be modeled. This chapter will talk about the basic concepts of hierarchical Dirichlet process and different posterior sampling methods, including posterior sampling with slice sampler.

2.1 Basic Concepts

To get started, we will first go over some basic concepts that build the foundation of the hierarchical Dirichlet process. This section will give some technical definitions of Dirichlet process and talk about stick-breaking process and Chinese restaurant process as representations of HDP.

2.1.1 Dirichlet Process

Dirichlet process is widely used in non-parametric Bayesian models. Let (Θ, B) be a measurable space, G_0 be a probability measure on the space, let α_0 be a positive real number and let $(A_1, A_2...A_r)$ be any finite partition of Θ. If G is a random probability measure distributed according to a Dirichlet process, which is written as $G \sim DP(\alpha_0, G_0)$, we have the following:

$$(G(A_1), ...G(A_r)) \sim Dir(\alpha_0 G_0(A_1), ...\alpha_0 G_0(A_r)) \tag{2.1}$$

where Dir stands for the Dirichlet Distribution.

Dirichlet process is important because it is the foundation of HDP mixture models. Consider a set of data $x = (x_1, ...x_n)$ and assume each data point is exchangeable. We draw $G \sim DP(\alpha_0, G_0)$, then draw n latent factors independently: $\theta_i \sim G$ for $i = 1, ...n$. Then we draw n data points $x_i \sim F(\theta_i)$ where F is a distribution for $i = 1, ...n$. The Dirichlet process has an important clustering property, and if there are m distinct values of θ_i, we would get a Dirichlet process mixture model with m components.

2.1.2 Hierarchical Dirichlet Process

Now let's move on to hierarchical Dirichlet process. As its name suggests, HDP uses a hierarchy of several layers of Dirichlet processes. For simplicity we will only talk about the two-layer HDP model. From the first layer of DP we can sample parameters for a second layer of DP mixture models, one DP mixture model for each group of data (or document). This ensures that the distribution of parameters is shared among all groups.

Now let's write out the equations. We have a global probability measure G_0, and for each group j we also have a probability measure G_j. The global measure G_0 is distributed as $DP(\gamma, H)$ where H is a base distribution and γ is a concentration parameter. We then sample G_j for each group: $G_j \sim DP(\alpha_0, G_0)$. For each group, we sample a certain number of factors $\theta_{ji} \sim G_j$ and then sample the individual data $x_{ji} \sim F(\theta_{ji})$. We can write out the equations as conditional equations:

$$G_0 \mid \gamma, H \sim DP(\gamma, H) \tag{2.2}$$

$$G_j \mid \alpha_0, G_0 \sim DP(\alpha_0, G_0) \tag{2.3}$$

$$\theta_{ji} \mid G_j \sim G_j \tag{2.4}$$

$$x_{ji} \mid \theta_{ji} \sim F(\theta_{ji}) \tag{2.5}$$

For better understanding, we can think about a topic modeling scenario where we have a corpus of k documents and each document has n_j number of words. We are given a prior distribution H and a concentration parameter γ, and we will sample the global probability measure G_0 from $DP(\gamma, H)$. We can consider G_0 as a distribution of probabilities, as shown by the stick-breaking construction in the next section. Then for $j = 1, ..., k$, we sample a probability measure G_j for each document from $DP(\alpha_0, G_0)$. For document j we will sample n_j words, with each word denoted as x_{ji} for $i = 1, ..., n_j$, by first sampling θ_{ji} from G_j and then sample $x_{ji} \sim F(\theta_{ji})$.

2.1.3 The stick-breaking construction

The stick-breaking process shows that measures drawn from a Dirichlet process are discrete, and it is used in the construction of G_0 and G_j. Suppose we have independent sequences of i.i.d. random variables $(\pi'_k)_{k=1}^{\infty}$ and $(\phi_k)_{k=1}^{\infty}$:

$$\pi'_k \mid \alpha_0, G_0 \sim Beta(1, \alpha_0) \tag{2.6}$$

$$\phi_k \mid \alpha_0, G_0 \sim G_0 \tag{2.7}$$

$$\pi_k = \pi'_k \prod_{l=1}^{k-1} (1 - \pi'_l) \tag{2.8}$$

And define random measure G as:

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\phi_k} \tag{2.9}$$

where δ_ϕ is a probability measure concentrated at ϕ. For convenience we will write $\pi \sim GEM(\alpha_0)$. We can thus represent the global measure G_0 and each G_j as:

$$G_0 = \sum_{k=1}^{\infty} \beta_k \delta_{\phi_k}$$
$$G_j = \sum_{k=1}^{\infty} \pi_{jk} \delta_{\phi_k}$$

where $\phi_k \sim H$ and $(\beta_k)_{k=1}^{\infty} \sim GEM(\gamma)$.

If we want to express the above equations in words using G_j as an example, we would say that each factor θ_{ji} is distributed according to G_j and takes on the value ϕ_k with probability π_{jk}. Let $\phi = (\phi_k)_{k=1}^{\infty}$ and let z_{ji} be an indicator so that $\theta_{ji} = \phi_{z_{ji}}$. We can write another representation of hierarchical Dirichlet process model based on the stick-breaking process:

$$\beta \mid \gamma \sim GEM(\gamma) \tag{2.10}$$
$$\pi_j \mid \alpha_0, \beta \sim DP(\alpha_0, \beta) \tag{2.11}$$
$$\phi_k \mid H \sim H \tag{2.12}$$
$$z_{ji} \mid \pi_j \sim \pi_j \tag{2.13}$$
$$x_{ji} \mid z_{ji}, \phi \sim F(\phi_{z_{ji}}) \tag{2.14}$$

2.1.4 Chinese Restaurant Franchise

Now we have a general idea of the basic components of the HDP, but how exactly do we generate a HDP model? Here we will describe HDP in terms of a Chinese restaurant franchise. The CRF view explains how to generate θ_{ji} for each group j, and it can directly lead us to an efficient Gibbs sampler for the HDP mixture models.

Consider a Chinese restaurant with an unbounded number of tables and a menu with K dishes. The first customer sits at the first table, and subsequent customers will sit at an occupied table with a probability proportional to the number of customers already at that table or at the next empty table with probability proportional to α_0. Each occupied table is assigned a dish from the menu. This is the Chinese restaurant process.

Now let's consider a franchise of Chinese restaurants, where the franchise can be thought of as a corpus with each restaurant representing a single document and each dish represents a topic. All restaurants in the franchise shares the same menu, which allows the documents to share the same set of topics. Let t_{ji} denote the table that customer i in restaurant j is sitting at, and let ψ_{jt} denote the dish at table t of restaurant j, where ψ_{jt} is drawn from G_0. λ_l for $l = 1, ...K$ denote the dishes on the menu.

Let us write out some equations. Let t_{ji} denote the table that customer i in restaurant j sitting at and let n_{jt} be the number of customers already at table t, the conditional distribution is as follows:

$$t_{ji} \mid t_{j1}, ...t_{ji-1}, \alpha_0 \sim \sum_t \frac{n_{jt}}{\sum_{t'} n_{jt'} + \alpha_0} \delta_t + \frac{\alpha_0}{\sum_{t'} n_{jt'} + \alpha_0} \delta_{t^{new}} \tag{2.15}$$

δ_t is the indicator that the customer sits at table t and δ_{new} is the indicator that the customer sits at a new table. Now we want to assign each factor θ_{ji} to a dish $\psi_{jt_{ji}}$. Note that ψ_{jt} is sampled from G_0 and is thus distributed according to $DP(\gamma, H)$, so we can perform another CRP process here. Let k_{jt} be the table associated with the customer having dish ψ_{jt}, let m_{jt} be the number of tables serving dish k in restaurant j, and we have a conditional distribution:

$$k_{jt} \mid k_{11}...k_{1n_1}, k_{21}...k_{jt-1}, \gamma_0 \sim \sum_t \frac{m_k}{\sum_{k'} m_{jk'} + \gamma_0} \delta_k + \frac{\alpha_0}{\sum_{k'} m_{k'} + \alpha_0} \delta_{k^{new}} \tag{2.16}$$

Finally, we draw λ_k as the topic assigned to table k, giving us $\psi_{jt} = \lambda_{k_{jt}}$, thus completing our generative process.

2.2 Posterior Inference

In this section we will discuss three Markov Chain Monte Carlo posterior sampling methods that can be used in posterior sampling of HDP models. The first method is a Gibbs sampler based on the Chinese restaurant franchise representation. This method is straightforward and not difficult to implement, but it can also be slow to mix. The second method is the split-merge algorithm which introduces a set of "split" and "merge" operations in addition to the Gibbs sampler. The third method is the slice sampler.

Before we get started, recall that x_{ji} are our observed data and θ_{ji} are our factors. Let θ_{ji} be associated with table t_{ji}, in other words: $\theta_{ji} = \psi_{jt_{ji}}$. Recall that ψ_{jt} is an instance of the mixture component k_{jt}: $\psi_{jt} = \phi_{k_{jt}}$. Let $z_{ji} = k_{jt_{ji}}$ denote the mixture component associated with x_{ji}.

Let $f(\cdot \mid \theta)$ and h be the density functions for $F(\theta)$ and H respectively. For notation, when a set of variable or a count variable has a superscript with the minus sign attached, it means the variable corresponding to the superscripted index is removed from the set or count. For example, $x^{-ji} = x \setminus x_{ji}$. We will denote the conditional density of x_{ji} under mixture component k given all data except x_{ji} as:

$$f_k^{-x_{ji}}(x_{ji}) = \frac{\int f(x_{ji} \mid \phi_k) \prod_{j'i' \neq ji, z_{j'i'}=k} f(x_{j'i'} \mid \phi_k) h(\phi_k) d\phi_k}{\int \prod_{j'i' \neq ji, z_{j'i'}=k} f(x_{j'i'} \mid \phi_k) h(\phi_k) d\phi_k} \tag{2.17}$$

2.2.1 Gibbs sampler posterior sampling

What's nice about the CRF representation is that this view directly leads us to a nice Gibbs sampler for posterior sampling of the HDP mixture model. But rather than sampling θ_{ji} and ψ_{ji} directly, sampling their indexes t_{ji} and k_{ji} is more efficient. For notations, let n_{jtk} denote the number of customers in restaurant j table t with dish k, and let m_{jk} denote the number of tables in restaurant j serving dish k. $f(\cdot|\theta)$ and h will be the density functions for $F(\theta)$ and H respectively.

Sampling t. The conditional probability of t_{ji} is

$$p(t_{ji} = t \mid t^{-ji}, k) \propto \begin{cases} \alpha_0 f(x_{ji} \mid t^{-ji}, t_{ji} = t^{new}, k) & \text{if } t = t^{new} \\ n_{jt}^{-ji} f_{k_{jt}}^{-x_{ji}}(x_{ji}) & \text{if t currently used} \end{cases} \tag{2.18}$$

where

$$p(x_{ji} \mid t^{-ji}, t_{ji} = t^{new}, k) = \sum_{k=1}^{K} \frac{m_k}{m+\gamma} f_k^{-x_{ji}}(x_{ji}) + \frac{\gamma}{m+\gamma} f_{k^{new}}^{-x_{ji}}(x_{ji}) \tag{2.19}$$

Sampling k. The conditional probability of k_{jt} is

$$p(k_{jt} = k \mid t, k^{-jt}) \propto \begin{cases} \gamma f_{k^{new}}^{-x_{jt}}(x_{jt}) & \text{if } k = k^{new} \\ m_k^{-jt} f_k^{-x_{jt}}(x_{jt}) & \text{if t currently used} \end{cases} \tag{2.20}$$

A more detailed algorithm and derivation can be found at [Ta2005].

2.2.2 Split-Merge posterior sampling

A split-merge algorithm is proposed for HDP by Wang et. al. in [W2012]. The Gibbs sampler is embellished with a split-merge operation. Two observations, which are tables in the CRF representation, are picked at random. If the observations are in the same component then a "split" is proposed: all the observations associated with that observation are divided into two new components If the observations are instead in different components, a "merge" is proposed: the observations from the two components are placed into the same component. Then Metropolis-Hasting is used to determined whether the proposed split or merge operation is accepted.

Now let's get into more detail by looking at a split case. Let $\mathbf{c}$ be the current state and $\mathbf{c}_{split}$ be the split state, and let (j, t) denote table t in document j, and k_{jt} is the topic of the specified table. Let $S_1 = \{(j_1, t_1)\}$, $S_2 = \{(j_2, t_2)\}$, and $k_{j_1t_1} = k_1$, $k_{j_2t_2} = k_2$. Let m_{k_1} and m_{k_2} be the number of tables in S_1 and S_2. S_1 and S_2 will be receiving tables form S_c assigned to k_1 and k_2, S_c is defined in algorithm 2. We then use the *sequential allocation restricted Gibbs sampling* to sample k_{jt} from a uniformly permuted $S_{\mathbf{c}}$ where (j, t) are successive table indexes:

$$p(k_{jt} = k_l \mid S_1, S_2) \propto m_{kl} f_{kl}^{-x_{jt}}(x_{jt}), l = 1, 2$$

This is a one-pass Gibs sampling restricted over k_1 and k_2. If $k = k_1$, then (j, t) will be placed in S_1, otherwise they will be placed in S_2. The process is repeated until all tables in $S_{\mathbf{c}}$ are visited. Let the realization of k_{jt} be $k_{jt}(r)$, then the transition probability from $\mathbf{c}$ to $\mathbf{c}_{split}$ is

$$q(\mathbf{c} \to \mathbf{c}_{split}) = \prod_{(j,t)\in S} p(k_{jt} = k_{jt}(r) \mid S_1, S_2)$$

On the other hand, we have

$$q(\mathbf{c}_{split} \to \mathbf{c}) = 1$$

This is because there is only one way to merge k_1 and k_2.

Whether the split or merge operation is accepted is determined by Metropolis-Hastings, and the probability of accepting the operation is

$$A = \frac{P(\mathbf{c}_{split})}{P(\mathbf{c})} \frac{L(\mathbf{c}_{split})}{L(\mathbf{c})} \frac{q(\mathbf{c}_{split} \to \mathbf{c})}{q(\mathbf{c} \to \mathbf{c}_{split})} \tag{2.21}$$

$\frac{P(\mathbf{c}_{split})}{P(\mathbf{c})}$ is the prior ratio and can be calculated as:

$$\frac{P(\mathbf{c}_{split})}{P(\mathbf{c})} = \gamma \frac{(m_{k_1}-1)!(m_{k_2}-1)!}{(m_k-1)!}$$

$\frac{L(\mathbf{c}_{split})}{L(\mathbf{c})}$ is the likelihood ratio and can be calculated as:

$$\frac{L(\mathbf{c}_{split})}{L(\mathbf{c})} = \frac{f_{k_1}(\{x_{ji} : z_{ji} = k_1\}) f_{k_2}(\{x_{ji} : z_{ji} = k_2\})}{f_k(\{x_{ji} : z_{ji} = k\})}$$

where z_{ji} is th topic index for word x_{ji}, and

$$f_k(\{x_{ji} : z_{ji} = k\}) = \frac{\Gamma(V\eta)}{\Gamma(n_k + V\eta)} \frac{\prod_V \Gamma(n_k^V + \eta)}{\Gamma^V(\eta)}$$

V is the size of vocabulary, and n_k is the number of words assigned to topic k in the corpus.

Algorithm 1 Split-Merge Algorithm

```
1: Choose two distinct tables $(j_1, t_1)$ and $(j_2, t_2)$ at random uniformly
2: if $k_{j_1t_1} = k_{j_2t_2} = k$, we perform the Split Case, as in algorithm 2
3: if $(k_{j_1t_1} = k_1) \neq (k_{j_2t_2} = k_2)$, we perform the Merge Case, as in algorithm 3
4: Sample $u \sim Unif(0,1)$, accept the move if $u < A$, otherwise reject it.
```

```
Algorithm 2 Split Case
1: Let $S_{\mathbf{c}}$ be the set of tables whose topic is k excluding tables $(j_1, t_1)$ and $(j_2, t_2)$ in state $\mathbf{c}$.
2: Assign $k_{j_1t_1} = k_1$ and $k_{j_2t_2} = k_2$. Randomly permute $S_{\mathbf{c}}$, then run the sequential allocation restricted Gibbs algorithm to assign tables in $S_{\mathbf{c}}$ to $k_1$ or $k_2$ to obtain the split state $\mathbf{c}_{split}$. Calculate $q(\mathbf{c} \to \mathbf{c}_{split})$
3: Calculate acceptance ratio:
   $$A = \frac{P(\mathbf{c}_{split})}{P(\mathbf{c})} \frac{L(\mathbf{c}_{split})}{L(\mathbf{c})} \frac{q(\mathbf{c}_{split} \to \mathbf{c})}{q(\mathbf{c} \to \mathbf{c}_{split})}$$
   note that $q(\mathbf{c} \to \mathbf{c}_{split}) = 1$, since there is only one way to merge topics from $\mathbf{c}_{split}$ to $\mathbf{c}$.
```

```
Algorithm 3 Merge Case
1: Let $S_{\mathbf{c}}$ be the set of tables whose topic is either $k_1$ or $k_2$ excluding tables $(j_1, t_1)$ and $(j_2, t_2)$ in state $\mathbf{c}$.
2: Randomly permute $S_{\mathbf{c}}$, then run the sequential allocation restricted Gibbs algorithm to assign tables in $S_{\mathbf{c}}$ to $k_1$ or $k_2$ to reach the original state $\mathbf{c}$. Calculate $q(\mathbf{c}_{merge} \to \mathbf{c})$
3: Assign $k_{j_1t_1} = k_{j_2t_2} = k$ and $k_{jt} = k$ for $(j, t) \in S_{\mathbf{c}}$ to obtain merge state $\mathbf{c}_{merge}$.
4: Calculate acceptance ratio:
   $$A = \frac{P(\mathbf{c}_{merge})}{P(\mathbf{c})} \frac{L(\mathbf{c}_{merge})}{L(\mathbf{c})} \frac{q(\mathbf{c}_{merge} \to \mathbf{c})}{q(\mathbf{c} \to \mathbf{c}_{merge})}$$
   note that $q(\mathbf{c} \to \mathbf{c}_{merge}) = 1$, since there is only one way to merge topics from $\mathbf{c}$ to $\mathbf{c}_{merge}$.
```

2.2.3 Slice Sampler posterior sampling

Exact slice sampler for HDP is proposed by Amini et. al. in [AA2019], and should mix faster than the conventional HDP with Gibbs sampler. Let's recall the basic idea of slice sampling: to sample from $f(x)$, we introduce a nonnegative variable u and look at the joint density $g(x, u) = 1\{u \leq f(x)\}$ whose marginal over x is $f(x)$, then we perform Gibbs sampling on g, and in the end we only keep the samples of x and discard those of u.

Now let's carry the idea over to HDP. To make the implementation of slice sampler easier, we need to make some changes to the original formulation:

$$\beta \mid \gamma_0 \sim GEM(\gamma_0),\ \beta = (\beta_k) \tag{2.22}$$

$$\gamma_j \mid \alpha_0 \sim GEM(\alpha_0),\ \gamma_j = (\gamma_{jt}) \tag{2.23}$$

$$k_{jt} \mid \beta \sim \beta \tag{2.24}$$

$$t_{ji} \mid \gamma_j \sim \gamma_j,\ i = 1, ...n_j \tag{2.25}$$

$$z_{ji} \mid t_j, k_j = k_{j,t_{ji}} \tag{2.26}$$

γ_{jt} represents the fraction of customers in restaurant j that sits at table t, and z_{ji} is a simpler representation of $\psi_{jt_{ji}}$, the dish for customer i in restaurant j. The overall joint density of the model is given below:

$$p(y, f, t, k, \gamma', \beta') = \prod_j (\prod_i [f_{k_j,t_{ji}}(y_{ji})\gamma_{j,t_{ji}}] \prod_t b_{\alpha_0}(\gamma'_{k_{jt}}) \prod_t \beta_{k_{jt}}) \prod_k [b_{\gamma_0}(\beta'_k) F(f_k)] \tag{2.27}$$

We augment the model by adding variables $u_j = u_{ji}$ and $v_j = v_{jt}$. The new density becomes:

$$p(y, f, t, k, \gamma', \beta') = \prod_j (\prod_i f_{k_j,t_{ji}}(y_{ji}) 1\{u_{ji} \leq \gamma_{j,t_{ji}}\} \prod_t b_{\alpha_0}(\gamma'_{k_{jt}}) \prod_t 1\{v_{jt} \leq \beta_{k_{jt}}\}) \prod_k [b_{\gamma_0}(\beta'_k) F(f_k)] \tag{2.28}$$

For sampling of (γ', u), we first sample

$$u_{ji} \mid \gamma', t, k, \beta', v \sim Unif(0, \gamma_{j,t_{ji}}) \tag{2.29}$$

Then we sample γ' from

$$\gamma'_{jt} \mid \gamma'_{-jt}, t, k, \beta' \sim Beta(n_t(t_j) + 1, n_{>t}(t_j) + \alpha_0) \tag{2.30}$$

Similarly, for sampling of (β', v), we first sample

$$v_{jt} \mid \beta', t, k, \gamma', u \sim Unif(0, \beta_{k_{jt}}) \tag{2.31}$$

Then we sample γ' from

$$\beta'_k \mid \gamma'_{-k}, t, k, \gamma' \sim Beta(n_k(k)+1, n_{>k}(k)+\gamma) \tag{2.32}$$

The sampling of t and k are shown below.

$$p(t_{ji}=t \mid t \setminus t_{ji}, k, \theta, \gamma', \beta', u, v) \propto f_{k_{jt}}(y_{ji}) 1\{u_{ji} \leq \gamma_{jt}\} \tag{2.33}$$

$$p(k_{jt}=k \mid t, k \setminus k_{jt}, \theta, \gamma', \beta', u, v) \propto 1\{v_{jt} \leq \beta_k\} \prod_{i:t_{ji}=t} f_k(y_{ji}) \tag{2.34}$$

More details about the slice sampler can be found in [AA2019].

CHAPTER 3

Evaluating Topic Models

Before we start our experiment, we need to find a way to evaluate HDP so that we can compare the performance of different models. We will need to evaluate the models from two aspects: first we want to know the best accuracy of the model, which is how close the predicted classification of each document is to the real label. Second, we want to see how fast the model can get us to the "optimal" accuracy, and by fast we mean in less iterations. We also need to consider that in most cases we do not know the "real" label for each document in real world scenarios, so we will discuss a different metric to evaluate performance on real data, which is perplexity, and different ways to calculate it.

3.1 Measurement of Performance

In this thesis we will be using normalized mutual information as a measurement of model performance. In probability and information theory, the mutual information (MI) of two random variables is a measure of the mutual dependence, or agreement, between the two variables. It quantifies the "amount of information" obtained about one random variable through observing the other random variable. Let (X, Y) be a pair of random variables, with joint distribution $P_{X,Y}$ and marginal distributions P_X and P_Y. The MI is defined as $I(X, Y) = D_{KL}(P_{X,Y} || P_X \otimes P_Y)$ where D_{KL} is the Kullback–Leibler divergence. Mutual information can go from 0 to ∞, so to be able to compare mutual information of different sets, we will want to use normalized mutual information (NMI), which is always between 0 and 1.

3.2 Change Point Detection

We want to find out how many iterations it takes for our NMI curve to converge. This is equivalent to finding a single change point in the series. A commonly used method for change point detection is binary segmentation. Binary segmentation searches for a single change point in the data series using a CUSUM-like procedure, and when a change point is detected, the series is split into two segments by the change point (hence the name "binary"), and a similar search is performed in both segments, possibly leading to further splits. The splitting process is terminated when a certain criteria is satisfied. A limitation of the binary segmentation is that there is a minimum distance requirement between change points. A solution to this problem is the wild binary segmentation (WBS). Instead of taking the global CUSUM from the entire data, Wild binary segmentation randomly draw a number of subsamples and take the CUSUM of each subsample. We then maximise each CUSUM, choose the largest maximiser over the entire collection of CUSUMs, and take it to be the first change-point candidate to be tested against a certain threshold. If the change point is considered to be significant, the same procedure is then repeated recursively on the left and right side.

This method works better than binary segmentation when the spacing between change points is short, and it is computationally fast. However, since we am only looking for a single most significant change point, we decided to implement the Binary Segmentation method instead of the WBS. To estimate mixing time, we can first find all significant change points using binary segmentation, then pick the one that satisfies our requirements by imposing conditions. Here we set two thresholds, one is a upper bound on the variance of the segment to the right of the change point, the idea is that the NMI converges after the change point so the variance to the right of the change point should be low enough. Another threshold is the difference between the variance to the left and to the right of the change point, because we believe the change point should bring a sharp change in variance.

The pseudo code for binary segmentation [F2014] is shown below. Notations: T is a threshold for CUSUM, s is the starting point, and e is the ending point. $X^b_{s,e}$ is the CUSUM at point b with starting point s and ending point e. $var(a, b)$ is the variance of the sequence from point a to point b.

Algorithm 4 Binary Segmentation

```
1: procedure BiSeg($s$, $e$, $T$):
2:     if $e - s < 2$ then
3:         stop
4:     else
5:         $b_0 = argmax_{b \in \{s, \ldots e-1\}} |X^b_{s,e}|$
6:         if $|X^{b_0}_{s,e}| > T$ then
7:             add $b_0$ to the set of change points
8:             $d_{b_0} = |var(s, b_0) - var(b_0 + 1, e)|$ is the corresponding difference-in-variance
   statistic for point $b_0$
9:             $BiSeg(s, b_0)$
10:            $BiSeg(b_0 + 1, e)$
11:        else
12:            stop
```

Usually we set $s = 1$ and $e =$ *index of last element*. To ensure that the NMI is starting to converge at change point p, we impose a constraint T_{var} on variance so that $var(p, e) < T_{var}$. Any change point that does not satisfy the variance constraint will be discarded. For each change point we get from binary segmentation, we find the corresponding d_p statistic. The mixing time occurs at $p_0 = argmax_{p \in all\ change\ points}(d_p)$ with all p satisfying the variance constraint.

There are some downsides to our current implementation: for this to work, the sequence after the change point needs to converge so that the variance is small enough. If the sequence after the change point does not converge nicely, the accuracy will drop significantly. The accuracy is also low if the convergence transition is smooth. As stated earlier, my implementation of binary segmentation contains a threshold for the difference in variance between the two segments divided by the change point, so our function is more likely to correctly locate a change point with a sharp change in NMI trend. For NMI of slice sampler, there is a distinct point where NMI suddenly stops climbing and stabilizes, while the change points of the other two implementations are usually nor as sharp, so our function has a better estimation of the slice sampler outputs.

3.3 Estimating Perplexity for Real Data

For simulated data, since we have a "true" label for each generated document, we are able to calculate the accuracy. But for real data, we usually do not know the real topic of documents, so we will need another metric to measure the performance of the model. One such metric is perplexity.

Perplexity is probably the most widely-used metric in evaluating topic modeling. In most cases, we calculate the perplexity of some held-out documents: that is, we divide the data into a training set and a testing set. Then we train a HDP model on the training set and uses the topic-word distribution from the training set to estimate the perplexity, or likelihood, of the testing set. To put this in equation, perplexity of a held-out abstract with words $w_1, ...w_I$ is defined to be:

$$\exp\big(-\frac{1}{I} \log\ p(w_1, ...w_I \mid \text{training corpus})\big) \tag{3.1}$$

The likelihood for a single document can be written as:

$$p(w, z, \theta \mid \Phi, \alpha m) = p(w \mid z, \Phi, \alpha m) p(z \mid \theta) p(\theta) \tag{3.2}$$

By marginalizing θ, we have the probability of obtaining topic z for each document w as shown in the equation below:

$$P(z_i = t \mid w, z_{-i}, \Phi, \alpha m) \propto P(w_i \mid z_i = t, \Phi) P(z_i = t \mid z_{-i}, \alpha m) \tag{3.3}$$

$$\propto \phi_{w_i \mid t} \frac{\{N_t\}_{-i} + \alpha m_t}{N - 1 + \alpha} \tag{3.4}$$

where $\{N_t\}_{-i}$ is the number of times topic t occurs in the current document excluding position i and αm is the Dirichlet prior parameters for the data. If we run Gibbs sampler until it mixes, we can have access to the samples from $P(z \mid w, \Phi, m)$.

For the predictive density when given training set, let's denote words from training set as w' and words from the held-out set as w, and the same goes for z' and θ'. The predictive density can be written as:

$$P(w, z, \theta \mid w') = \int P(w, z, \theta \mid \Phi, \alpha m) P(\Phi, \alpha m \mid w') d\Phi d\alpha dm \tag{3.5}$$

where Φ is the word-topic distribution. Usually we use $m = \mathbf{1}_T / T$ where T is the number of topics and $\mathbf{1}_T$ is a vector of 1 with length T. We can marginalize this equation further to obtain

$$P(w, z \mid w') = \int P(w, z \mid \Phi, \alpha m) P(\Phi, \alpha m \mid w') d\Phi d\alpha dm$$
$$P(w \mid w') = \int P(w \mid \Phi, \alpha m) P(\Phi, \alpha m \mid w') d\Phi d\alpha dm$$

Finding $P(w \mid w')$ is the key to estimating perplexity. We have access to $P(\Phi, \alpha m \mid w')$ from the training set, the difficulty is finding $P(w \mid \Phi, \alpha m)$. Wallach et. al. listed several methods for estimating $P(w \mid \Phi, \alpha m)$ to evaluate perplexity for LDA topic models in [W2009]. For the rest of this section, we will refer to the current document as $\mathbf{w}$, its latent topic assignment as $\mathbf{z}$, and its document-specific topic distribution as θ.

But the methods and equations listed by Wallach et. al. are meant for LDA models. If we want to find the perplexity of HDP models, we will have to make some changes. For the HDP models we will use π instead of θ, and the likelihood equation is shown below:

$$p(w, z, \pi \mid \beta, \Phi, \alpha m) = p(w \mid z, \Phi, \alpha m) p(z \mid \pi) p(\pi \mid \beta)$$

Note that we have $p(\pi \mid \beta)$ instead of $p(\theta)$, which adds to the complexity. We want to figure out the density of $p(\pi \mid \beta)$.

Remember that while LDA has only a fixed number of topics, HDP can generate up to infinite number of topics, so we will need to truncate β to its first K elements for a large enough K. By the definition of Dirichlet process and stick-breaking process, we have

$$(\pi_1, \pi_2, ...\pi_{K-1}, \sum_{k=K}^{\infty} \pi_k) \mid \beta \sim Dir(\alpha_0\beta_1, ...\alpha_0\beta_{K-1}, \alpha_0 \sum_{k=K}^{\infty} \beta_k)$$

Let's redefine $\theta = (\pi_1, \pi_2, ...\pi_{K-1}, \sum_{k=K}^{\infty} \pi_k)$ and let $\mathbf{m} = (m_k)$ where

$$m_k = \beta_k \text{ for } k = 1, ...K-1, m_K = \sum_{k=K}^{\infty} \beta_k$$

Then we have $\theta \mid \beta \sim Dir(\alpha_0\mathbf{m})$ and this is similar to LDA. That is, we can approximate HDP by the LDA by merging all clusters with $k \leq K$ for $\mathbf{m}$.

But in most cases we do not have access to β, so we can approximate β using the property that $E[\pi] = \beta$ for HDP models. Remember that π is the document-topic distribution. So we can use π' from training data:

$$\beta \approx \frac{1}{J} \sum_{j=1}^{J} [\pi']_j \tag{3.6}$$

Note that since the HDP output is naturally truncated and has only a finite number of topics, we do not need to truncate β.

In this paper, the harmonic means method is implemented to estimate perplexity. Besides the harmonic means methods, we will also talk about some of the other methods listed in [W2009].

3.3.1 Harmonic Means

Harmonic means method is based on the following unbiased estimator:

$$\frac{1}{P(w)} = \sum_z \frac{P(z \mid w)}{P(w \mid z)} \approx \frac{1}{S} \sum_s \frac{1}{P(w \mid z^{(s)})} \tag{3.7}$$

where $z^{(s)}$ is drawn from $P(z|w)$. And we get the following:

$$P(w \mid \Phi, \alpha m) \approx \frac{1}{\frac{1}{S}\sum_s \frac{1}{P(w \mid z^{(s)}, \Phi)}} \tag{3.8}$$

$$= HM(\{P(w \mid z^{(s)}, \Phi)\}_{s=1}^{S}) \tag{3.9}$$

In the equation above, $z^{(s)} \sim P(z \mid w, \Phi, \alpha m)$ and HM() is the harmonic means.

Many have expressed reservations regarding the harmonic means, nor is it the fastest way to find likelihood. But harmonic means is still widely used due to its ease of implementation and relative computational efficiency. We also decided to use harmonic means for estimating perplexity in my thesis. The algorithm to implement harmonic means is shown in 5. For the notations, *words* is a vector of words from the testing set, *topics* is the topic-word matrix from the training model, *prior* is a vector of prior probabilities for the topics, and α s a smoothing term. There are T_0 topics and V_0 words n total, and N_d is the number of words in the document.

The algorithm listed above gives us the likelihood of a single document at one iteration. To find the likelihood of the whole corpus, we simply add up the log likelihood of each document. This means if we want to find out how perplexity change over time, we have to iterate Harmonic means over all iterations and for each iteration we also have to iterate over all documents, which can be time consuming.

3.3.2 Chib-style Estimation

For any set of "special" latent topic assignments z^*, the following holds by Bayes' rule:

$$P(w \mid \Phi, \alpha m) = \frac{P(z^*, w \mid \Phi, \alpha m)}{P(z^* \mid w, \Phi, \alpha m)} \tag{3.10}$$

Chib-style estimation introduces a family of estimators that first pick a z^* and then estimate the denominator and numerator respectively. Any choice of "special state" z^* is valid. z^* is set by iteratively maximizing 3.3 for positions $1, \ldots, N$ after a few iterations of Gibbs sampling. The algorithm is listed in Algorithm 6.

Algorithm 5 Harmonic Means

```
1: procedure HM($words$, $topics$, $prior$, ITERS, $\alpha$):
2:     initialize $N_z$, a vector to keep track of topics
3:     initialize $z_z$, a vector to keep track of words in the document
4:     for each word in document do
5:         probability of each topic = the column of $topics$ corresponding to current word × $prior$
6:         A topic is picked according to the calculated probabilities
7:         $N_z$[topic picked] = $N_z$[topic picked] + 1
8:     for each iteration do
9:         $loglik = 0$
10:        for each word in document do
11:            $N_z$[topic of word] = $N_z$[topic of word] + 1
12:            probability of each topic = the column of $topics$ corresponding to current word × $prior$
13:            A topic is picked according to the calculated probabilities
14:            $N_z$[new topic] = $N_z$[new topic] + 1
15:            $loglik = loglik + \log(topics[word, newtopic])$
16:    $perplexity = -\log(\sum$ exp(loglik for each iteration))
```

Algorithm 6 Chib-style Estimation

```
1: initialize $z^*$ to a high posterior probability state
2: sample s uniformly form 1,...S
3: sample $z^{(s)} \sim \tilde{T}(z^{(s)} \leftarrow z^*)$
4: for $s' = (s+1) : S$ do
5:     sample $z^{(s')} \sim T(z^{(s')} \leftarrow z^{(s'-1)})$
6: for $s' = (s-1) : -1 : 1$ do
7:     sample $z^{(s')} \sim \tilde{T}(z^{(s')} \leftarrow z^{(s'+1)})$
8: $P(w \mid \Phi, \alpha m) \approx P(w, z^* \mid \Phi, \alpha m) / \frac{1}{S} \sum_{s'} T(z^* \leftarrow z^{(s)})$
```

Here T is any Markov chain operator, including the Gibbs sampler. We refer to T as the forward operation transition and it satisfies

$$P(z^* \mid w, \Phi, \alpha m) = \sum_{z} T(z^* \leftarrow z) P(z \mid w, \Phi, \alpha m) \tag{3.11}$$

We can think of T as sequentially applying (3.3) for positions 1 to N, and the reserve operator $\tilde{T}$ can be constructed by applying (3.3) backwards, and the definition is as follows:

$$\tilde{T}(z^* \leftarrow z) = T(z \leftarrow z^*) \frac{P(z^*)}{P(z)} \tag{3.12}$$

3.3.3 "Left-to-Right" Algorithm

This approach is proposed by Wallach et. al. The main idea is to decompose $P(w \mid \Phi, \alpha m)$ as:

$$P(w \mid \Phi, \alpha m) = \prod_{n} P(w_n \mid w_{<n}, \Phi, \alpha m) \tag{3.13}$$

$$= \prod_{n} \sum_{z_{\leq z}} P(w_n, z_{\leq z} \mid w_{<n}, \Phi, \alpha m) \tag{3.14}$$

Each sum over $z_{\leq z}$ can be approximated using an approach inspired by sequential Monte Carlo methods as detailed in Algorithm 7:

Algorithm 7 Left-to-Right

```
1: initialize $l := 0$
2: for each position n in w do
3:     initialize $p_n := 0$
4:     for each particle $r = 1 : R$ do
5:         for $n' < n$ do
6:             sample $z_{n'}^{(r)} \sim P(z_{n'}^{(r)}|w_{n'}, \{z_{<n}^{(r)}\}_{-n'}, \Phi, \alpha m)$
7:         $p_n := p_n + \sum_t P(w_n, z_n^{(r)} = t|z_{<n}^{(r)}, \Phi, \alpha m)$
8:         sample $z_{<n}^{(r)} \sim P(z_n^{(r)}|w_n, z_{<n}^{(r)}, \Phi, \alpha m)$
9:     $p_n := \frac{p_n}{R}$
10:    $l := l + log(p_n)$
11: $log(P(w \mid \Phi, \alpha m)) \approx l$
```

CHAPTER 4

Experiments

We want to compare the performance of HDP with slice sampler to existing methods. In this chapter, we will talk about the different packages we used and how to run them in R. Then we will test the packages on simulated data and real data.

4.1 Packages and Implementations

We will briefly talk about the packages we have used for comparison and discuss how to run them. Since there lack packages for HDP in R or python, the packages we are using are mostly taken directly from Github. These code are mostly not perfect and can be difficult to compile or run, so in this section we will go over the implementations and packages I have used and briefly talk about how to use them. I am using a Windows laptop with Ubuntu terminal installed.

4.1.1 Exact Slice Sampler

This package is written by Amini et. al. and implements the algorithm proposed in [AA2019]. The code can be found at [AA]. The code is written in R and relatively easy to run. The function *hdp-slice-samplerC* runs the posterior inference. The input is a list object where each element is a vector representing the words in each document. The vectors can be numeric or characters. The output is also a list of equal length, where each element is a vector representing the topic assigned to each word for each document. This allows for easy computation of NMI and perplexity.

4.1.2 Nicola Roberts

The package implements the HDP from [Ta2005] and can be found at [NR], along with instructions on installation. For posterior inference, we will need to first run *hdp-quick-init* to create a *Chain* object, then run *hdp-posterior* with the *Chain* object as input, and we will get a *Posterior* object. The input for *hdp-quick-init* is a matrix object with each row representing a document and each column representing a word. To find the topic assigned to each document, we need to use the *clust-dp-counts* attribute of the *Posterior* class, which is a table with each row as a document and each column as topic. Note that the first row of the table does not represent any document and will not be evaluated.

A word of caution is that *hdp-posterior* function can only run in a Linux environment, such as Mac OS or Ubuntu. If run in a Windows environment, the function will cause Rstudio to abort the R session. Also, the *posterior* class object does not provide information on the topic assignment for each single word in the trained model, so we cannot obtain the topic-word matrix for each iteration and thus cannot calculate the perplexity of Nicola Roberts models.

4.1.3 Split-Merge Algorithm

This package is written by Chong Wang et. al. implementing the Split-Merge algorithm from [W2012] and can be found at [DB]. The code is written mostly in C++ and requires the GSL or GNU scientific library, and it has to be compiled and run under a Linux environment. I used the Ubuntu terminal on Windows to run this package. The package files contain a *hdp* folder which includes all necessary code to run HDP and a Makefile for compiling the code. To compile the code on my laptop, I added the command *.PHONY: hdp* as the first line of the Makefile.

The input files have to be in *ldac* format, and we can transform txt files to ldac file using the python file *text2ldac.py*. This file is not included in the package and can be found at [TL]. First we need to create a folder as corpus with each txt file representing a document in the corpus. The txt file includes all words in the document. We then pass the folder to *text2ldac.py*, which gives us the desired *ldac* format input in the *.dat* file. The command used is *python text2ldac.py input-folder*.

Then we want to train our HDP model using the *ldac* file. This can be done using the *./hdp –algorithm train –data input.dat –directory training-directory* command where *–data* is the input file in *ldac* format and *–directory* is the directory we want to save the output files. Additional arguments of the training function include *–max iter* that allows us to set the maximum iterations of posterior MCMC samplings to perform and *–save lag* that specifies the number of iterations lapsed before the output for a iteration is saved into our pre-specified directory. For example, if *–save lag* is set to be 2, then only the output of every other iteration will be saved. Another important argument is *–split merge*, which allows us to choose whether to do the Split-Merge sampling or just the normal Gibbs sampling.

For the output, the package outputs three files for each iteration: a *topics.dat* file, a *word-assignments.dat* file, and a *.bin* file. The *word-assignments.dat* file is the most useful for us. The file contains a table with 4 attributes for every single word in the corpus: document id, word id, table id and topic id. We can infer from this table the topic assigned to each document. There is also a *state.log* file from the output that records the likelihood for each iteration.

There is one major problem with the package: the output files and input files does not have the same order for words and documents. For example, the first input document (file) in the folder might be indexed 11 in the output *word-assignments.dat* file. If we cannot reorder the output file index to match with the input, we won't be able to calculate NMI. To do that, we can match the input and output files by looking at the frequencies of words in each document. An algorithm is proposed below in Algoritm 8:

Algorithm 8 Word match

```
1: procedure REMATCH($l_{in}$, $l_{out}$, $T$):
2:     for 1 to 5 do
3:         for n from 1 to 15 do
4:             match words with unique frequency from $T_{in}(n)$ and $T_{out}(n)$
5:             remove matched words from $l_{in}$ and $l_{out}$
6:     for 1 to 5 do
7:         for each document in $l_{in}$ do
8:             if only one unmatched word in document then
9:                 match word by matching corresponding document in $l_{out}$
```

Luckily, the ordering is not totally random. For example, when the number of documents and words are fixed, the ordering pattern is the same, so we do not have to perform the matching process for every single corpus.

4.1.4 DAI Python Implementation

The DAI python code also implements HDP from [Ta2005] using direct assignment inference, and can be found at [DAI]. The code are contained in just two files, *HDPcodeup.py* and *vocabulary.py*, and to run the code we only need to download the two files and does not require compilation. The python implementation reads a txt file as input where the txt file represents a corpus, and each row of words represents a document. The output is a CSV files, where the row represents iterations and the column represents each document. The entry is thus the topic assigned to each document at each iteration.

To run the code in R, you will need the *reticulate* package, but some functions from the *reticulate* package, such as *source.python*, can change some functionalities in R and result in errors, so we decide to not use this implementation for testing.

4.2 Simulated Data

Now we want to generate some simulated data sets to test the performance of our models. We are not only interested in comparing different implementations, we also want to see if changing the number of documents in the corpus will affect the mixing time of HDP.

4.2.1 Design

We generate corpus with different number of documents, denoted J, and W different words choices, with each word choice representing a different topic. Each document in the corpus contains 300 words, and each word is generated so that it has a true "latent" topic and a observed topic. The observed topic can be the same as the true topic. The topic of each document is determined by majority vote of the true word topics in the document.

We choose to test the performance on data with $J = 100, 75, 50, 25, 20$. For each value of J, we generate 10 different sets of corpus, and for each corpus we perform 5 posterior samplings and find the mean NMI curve. We then apply our Binary Segmentation algorithm to find the mixing time for each corpus and compare the distribution of mixing time for different J or W using box plots.

4.2.2 Tuning Parameter for Slice Sampler

In chapter 2, we discussed an alternative formulation of the HDP model for ease of implementation of slice sampler for posterior sampling. As shown in equation (2.22), the parameters α_0 and γ_0 will affect the distributions of β and γ_j respectively. Before we start, we want to know how changing the shape of distributions of β and γ_j can affect the shape of the NMI curves.

Let's do a quick recap of the structure of an HDP model. The HDP is characterized by a hierarchy of Dirichlet Processes, and in our case it is a two-layer structure. In equations

(2.22), the Dirichlet processes are characterized by stick-breaking processes, and we can see that the first layer of stick-breaking process generates a discrete distribution for β and the second layer generates discrete distributions γ_j for each document j. In the original HDP formulation from [Ta2005] as shown in equation (2.10), β is the base distribution for the Dirichlet process of the second layer, whereas in the formulation for slice sampling β becomes a distribution for k_{jt}, the dish at table t from restaurant in the CRF view, and γ_j is the distribution of t_{ji}, the table that customer i from restaurant j is sitting at.

Now that we know that γ_0 and α_0 does in the model, we want to know how tuning these two parameters will affect the performance of HDP. In our case, we want to see how different combinations of γ_0 and α_0 will affect the NMI curve of the HDP classifications. For this purpose we perform the following experiment where we generate 6 corpus as described in the previous subsection with $J = 50, W = 20$. For each combination of $\gamma_0 = 0.5, 1, 3, 5$ and $\alpha_0 = 0.5, 1, 3, 5$, we will perform the HDP with slice sampler on each corpus and calculate the mean NMI curve.

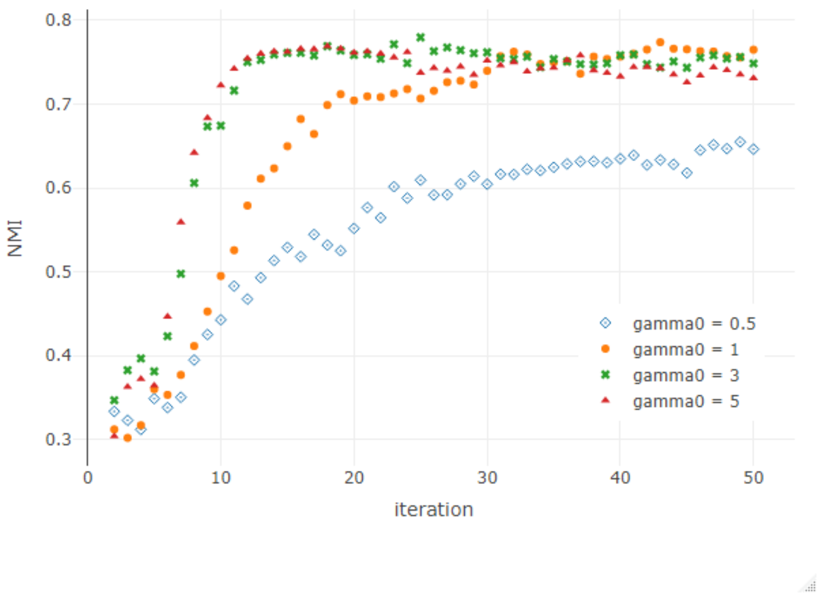

Figure 4.1: NMI curve for different γ_0 using slice sampler with $\alpha_0 = 3, J = 50$

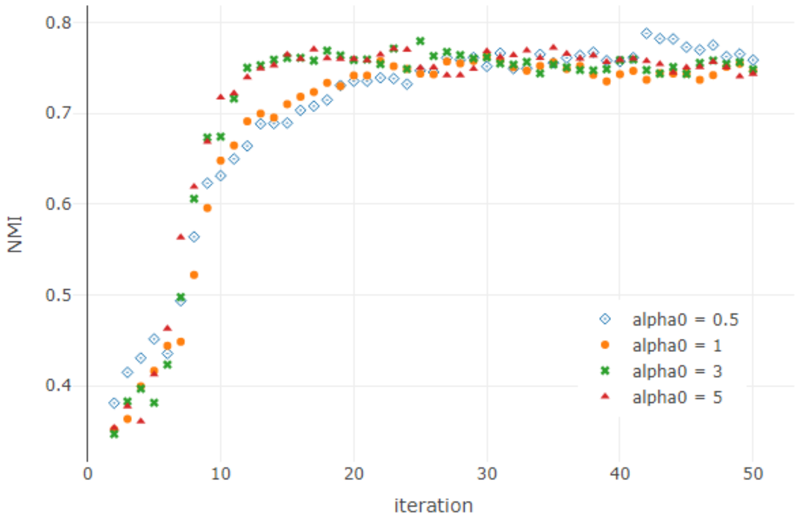

Figure 4.2: NMI curve for different α_0 using slice sampler with $\gamma_0 = 3, J = 50$

Our results show that changing γ_0 has a relatively large effect on the NMI curve, as shown in Figure 4.1. It is clear that NMI curves with smaller γ_0 value lies below the ones with higher γ_0 value, which means lower performance. When γ_0 is less than 3, a smaller γ_0 will result in lower NMI and a longer mixing time, but increasing γ_0 while $\gamma_0 \geq 3$ does not result in significant improvements in the NMI curve. On the other hand, changing α_0 does not result in much difference in the NMI curve given that γ_0 remains the same. This is shown in Figure 4.2, where we set $\gamma_0 = 3$. We can see that the NMI curve of different α_0 does not differ significantly. So in our experiments later on, we set $\gamma_0 = 3$ and $\alpha_0 = 3$ for our HDP model with slice sampler.

4.2.3 Mixing Time and Number of Words

First we want to explore how changing W, the number of different word topics, will affect mixing time. We set $J = 30$, which is 30 documents in the corpus, and set $W = 10, 20, 30, 50, 70, 100$, then we use the Nicola Roberts package and slice sampler HDP for posterior inference. The resulting box plots for the Nicola Roberts package is shown in Figure 4.3, and the box plots for slice sampler HDP is shown in Figure 4.4.

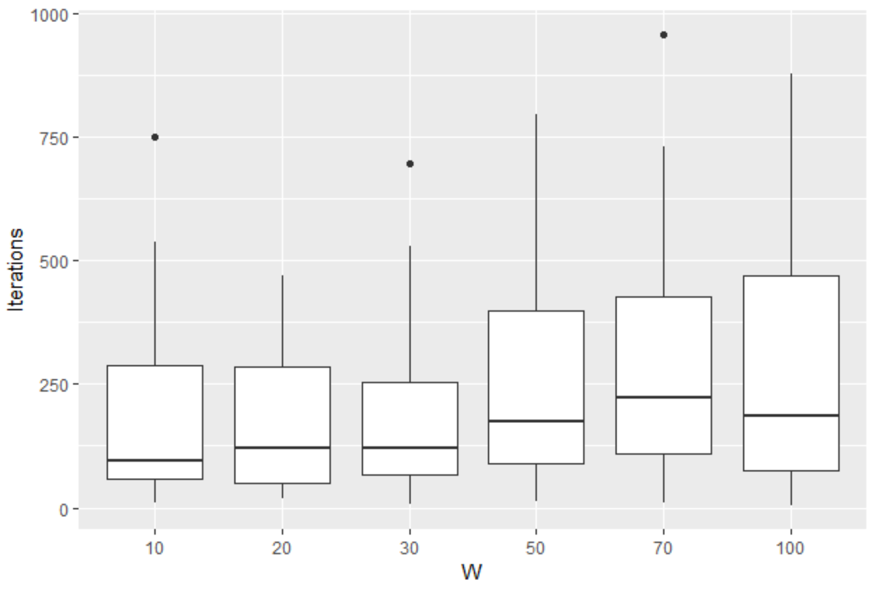

Figure 4.3: Mixing time for different W with Nicola Roberts package, J = 50

Here are some observations:

First, the slice sampler implementation mixes significantly faster. We will get more time to talk about this later. Second, there does not seem to be a significant change in mixing time for the Nicola Roberts package as W increases. The mixing time for the slice sampler implementation does seem to have a slight upward trend as W increases.

Let us try another set of data. This time we set $J = 50$. The box plots for the Nicola Roberts package is shown in Figure 4.5, and the box plots for slice sampler HDP is shown in Figure 4.6.

There does not seem to be any trends for either implementation. The mixing time for slice sampler might seem to increase slightly with W, but that is because the mixing time or $W = 10$ is significantly lower than other W. We can conclude that increasing the number of word topics does not lead to significant changes in mixing time, so we will only focus on

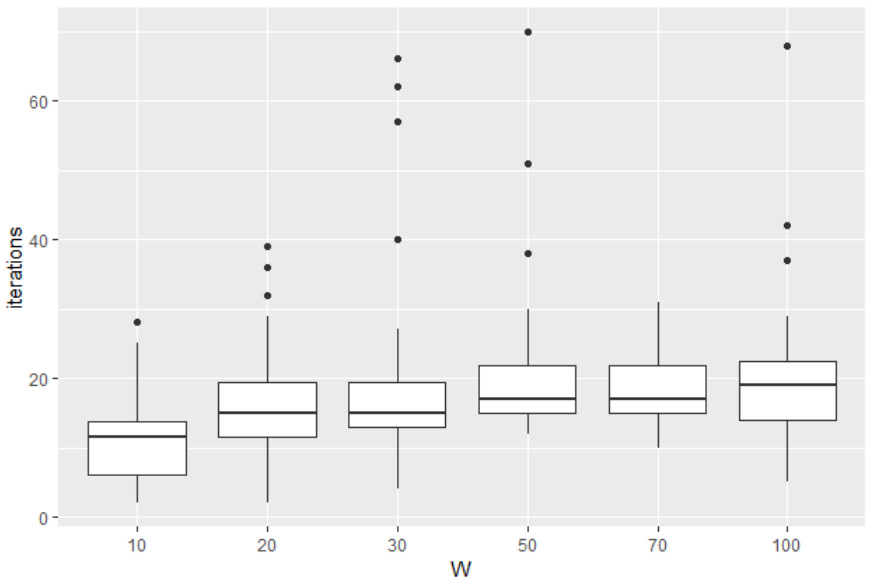

Figure 4.4: Mixing time for different W with slice sampler HDP, J = 50

changing J in the following sections.

4.2.4 Mixing Time and Number of Topics

Now we want to see how changing the number of documents in the corpus can affect the mixing time. We set $W = 20$, meaning that there are in theory 20 topics for each document. The estimated mixing time for the Nicola Roberts package and slice sampler are shown in Figure 4.8, but since it is hard to read the distribution of mixing time for slice sampler, Figure 4.7 is made separately for slice sampler.

Figure 4.8 shows that the mixing time steadily decreases as J increases, and begins to stabilize at around $J = 50$. The stabilizes mixing time is around 120 iterations. The mixing time distribution for smaller J, such as $J = 5, 10$, have larger variance. This is largely due to the drawbacks of out Binary segmentation mentioned before. Also, the NMI curve becomes more unstable with small J and occasionally the NMI would not converge. In fact, we do not think it's a good idea to apply the Nicola Roberts package to corpus with less than 20

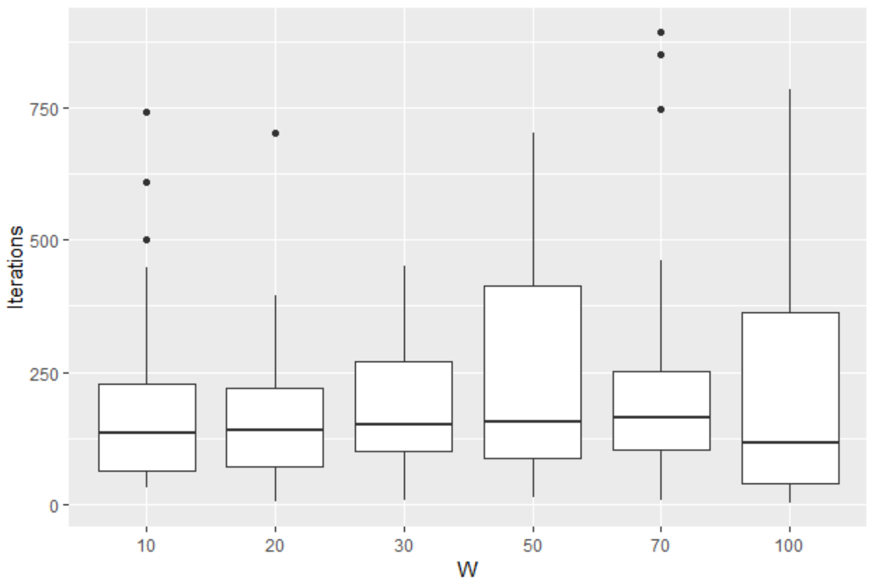

Figure 4.5: Mixing time for different W with Nicola Roberts package, J = 50

documents due to the poor performance.

Figure 4.7 shows that the mixing time increases as J increases, and stabilizes when $J = 50$. The final mixing time is around 17 iterations. Unlike the Nicola Roberts package, the slice sampler performs well on corpus with small number of documents.

Now let's add the implementations from Blei's package to the comparison, which includes an implementation using just the Gibbs sampler and an implementation of the split-merge algorithm. The mixing times of the Blei package is compared with those of Nicola Roberts package and slice sampler HDP in Figure 4.8.

We can see that the slice sampler does have shorter mixing time for all J. The two implementations from the Blei package, labeled "non-split" for normal Gibbs sampler and "split" for Split-Merge algorithm in the plot, have roughly the same performance as the Nicola Roberts package, and the mixing time also decreases as J increases and stabilizes at around $J = 25$. It seems that the main purpose of the Split-Merge algorithm is not to improve the mixing time. Also note that the graph seems to suggest that the Blei package implementa-

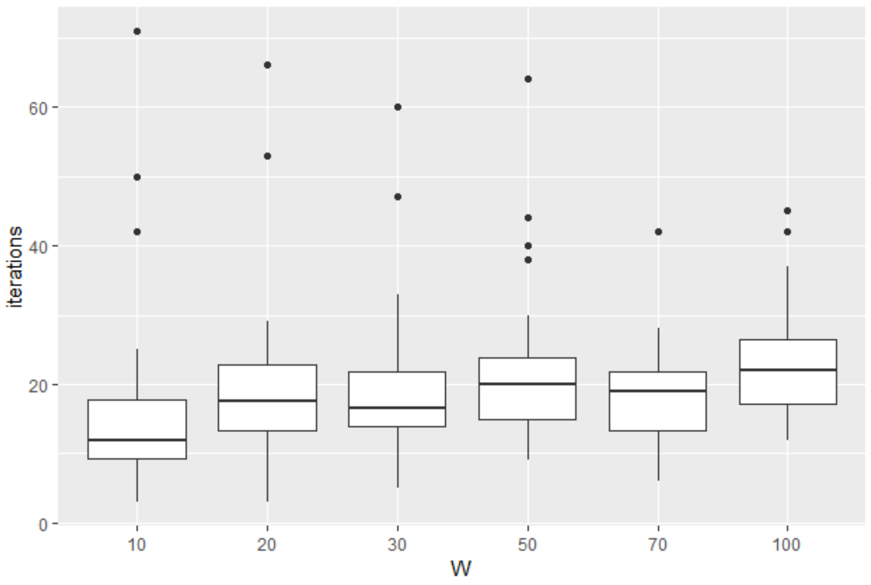

Figure 4.6: Mixing time for different W with slice sampler HDP, J = 50

tions perform better than the Nicola Roberts package in terms of mixing time variance for smaller J, but looking at the original NMI output suggests that the Blei implementations does not perform well for small number of documents either. Also, it is worth noting that the NMI have high starting positions for the Blei implementations. For Nicola Roberts package and slice sampler HDP, the NMI would start at arounf 0.4 to 0.5, and then converge between 0.9 to 1. The Blei implementations would have NMI starting at around 0.7 or even as high as 0.8, and this is more significant with small J. So here we express reservations on the Blei package as well as our estimated mixing time for both split and non-split.

4.3 Real Data

We are not satisfied with only testing on the simulated data. We want to see how HDP with slice sampler and other posterior sampling methods perform on real data by comparing their perplexity. In this section we will compare the perplexity of slice sampler and the two Blei methods (one with split-merge, the other with normal Gibbs sampler). We are unable

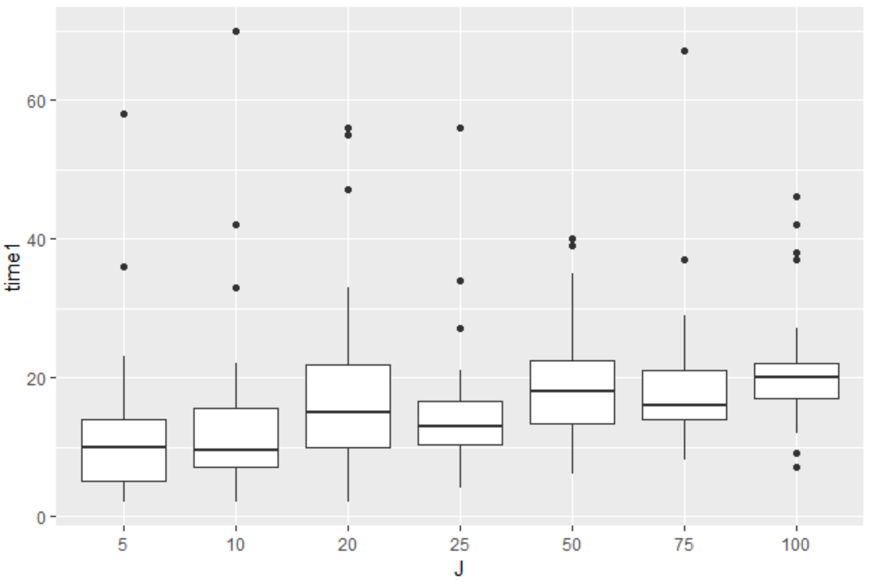

Figure 4.7: Mixing time for different J with slice sampler HDP

to evaluate the output of the Nicola-Roberts package since it does not give the assignment of topic to each word, which are needed for building doc-topic assignment matrices for estimating perplexity.

4.3.1 Design

The data we will be using is a corpus of NSF research award abstracts from 1990 to 1994 and can be found in [BoW]. The corpus consists of 49074 documents, each representing an abstract from a award-wining paper, and 30797 distinct words. However, as mentioned earlier, we have to match the document and word from the Blei package outputs to the original input, and our matching algorithm cannot handle too many documents or words, so we have to take a subset of the data for evaluation. First we want to cut down on the number of words, and from our observation words that are less than 4 characters are either trivial words that can be considered as stop words or words that does not make sense (such as *aaa*), so we drop all words less than 4 characters from the data. Then we take a subset

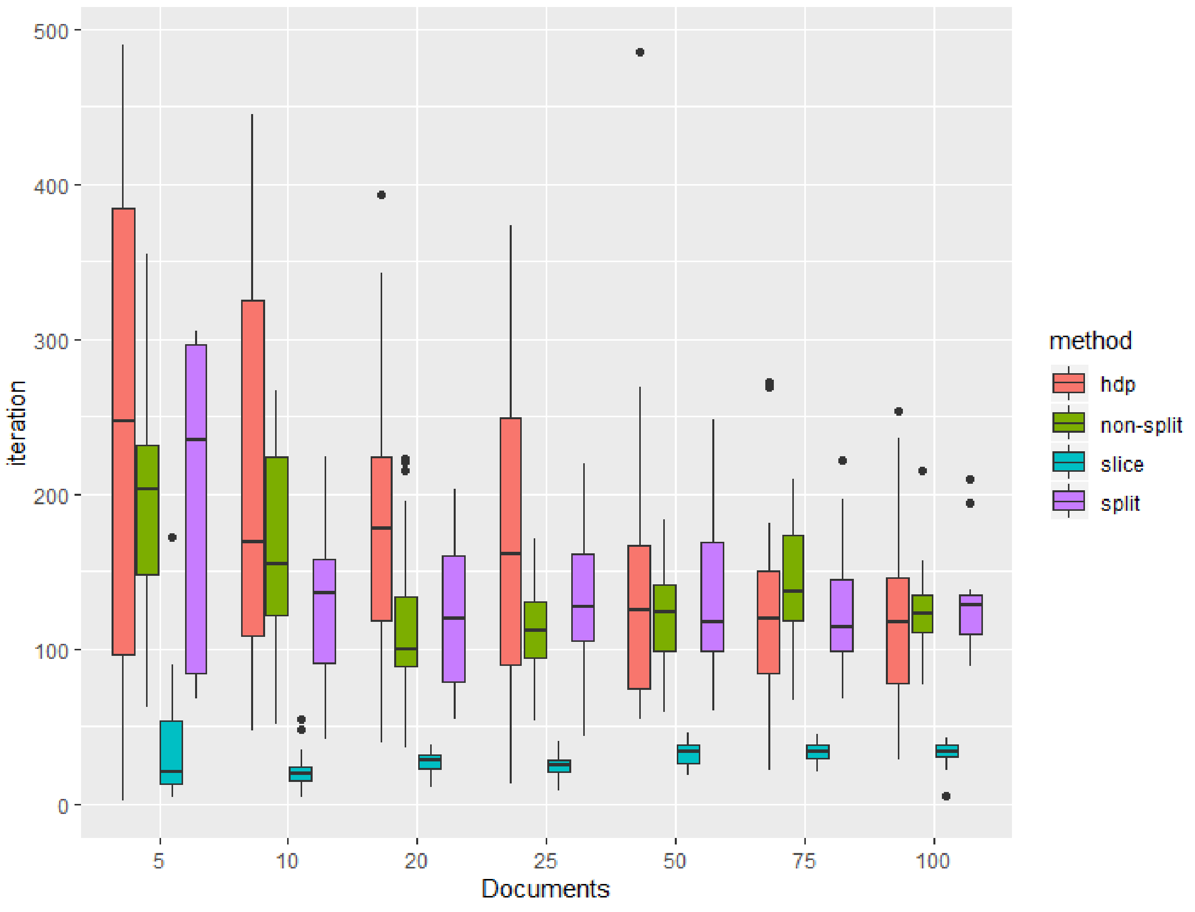

Figure 4.8: Mixing time for different J with all 4 samplers

of 2000 words, and sample 200 documents from the data. This gives us a corpus with 200 documents and 377 distinct words.

We will do a cross-validation to estimate perplexity. The corpus is divided into 4 partitions, each time we train our HDP model using 3 partitions and estimate the perplexity using the remaining partition as the test data. Then we will take the average of the 4 perplexity curves as our final perplexity curve.

4.3.2 Perplexity vs NMI

Before we apply our methods to real data, we want to make sure that our implementation of Harmonic Means for estimating perplexity is accurate. To test that, we will perform a cross validation on out simulated data using Slice Sampler HDP, estimate the perplexity for each cross validation, find the mixing time by finding the change point in the perplexity curve and then compare the perplexity mixing time to the NMI mixing time.

Here's how we will perform our cross validation: for each corpus, we split the documents into 5 sets, that means we will perform 5 cross validations and each time we use 4 sets of documents to train our HDP model and obtain the word-topic distribution, while the 1 set of documents left will be used as testing set for perplexity estimation. We will use the 10 corpus generated earlier for $J = 100, 75, 50, 25, 20$, and for each corpus we calculate the mean perplexity for mixing time estimation. We compare the mixing time estimation using NMI curves to those estimated using perplexity in Figure 4.9.

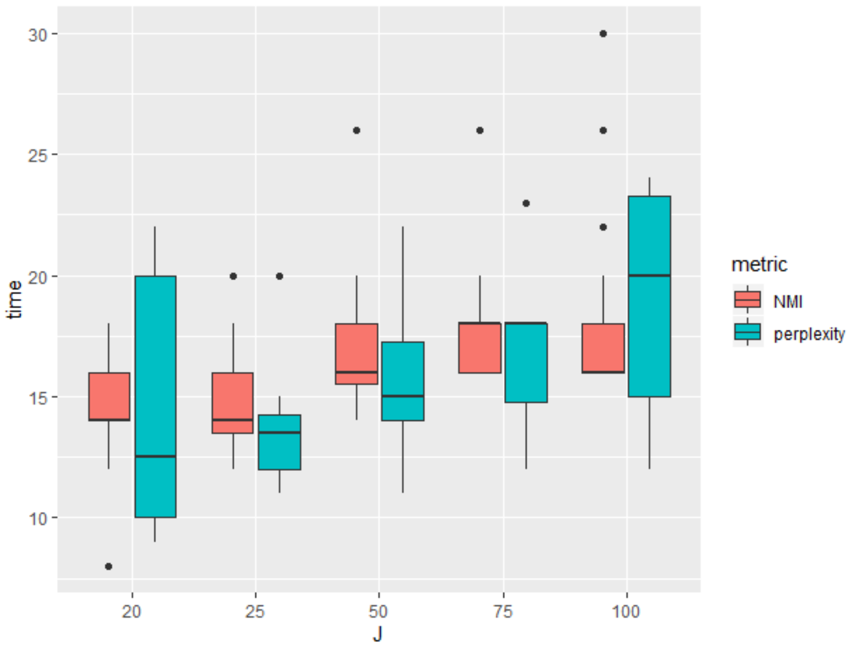

Figure 4.9: Mixing time of NMI and Perplexity for slice sampler

We can see that although there are some variations, the mixing time for both methods does not have significant differences for each J value. This should be able to show that our Harmonic means implementation for perplexity estimation is reasonable and safe to use.

4.3.3 Perplexity Comparison

The perplexity curves of the different HDP models are displayed in Figure 4.10. For better visual, the perplexity curve of the Blei methods are pictured in Figure 4.11. Contrary to our conclusion by comparing NMI curves, slice sampler does not mix significantly faster than the Blei implementations, all three methods mix at about the 20th iteration.

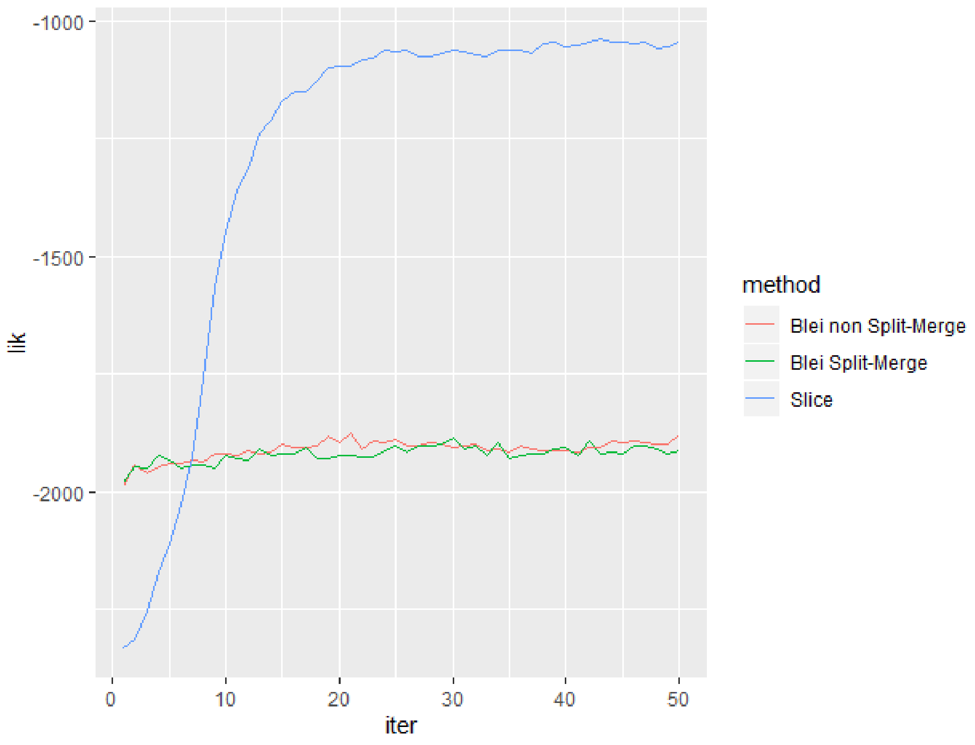

Figure 4.10: Mean perplexity of real data

However, it is interesting how the perplexity for slice sampler converges to a much bigger value than the perplexity of the Blei methods. The likelihood of slice sampler converges at around -11000, and the likelihood of both Blei models converges at around -19000. There

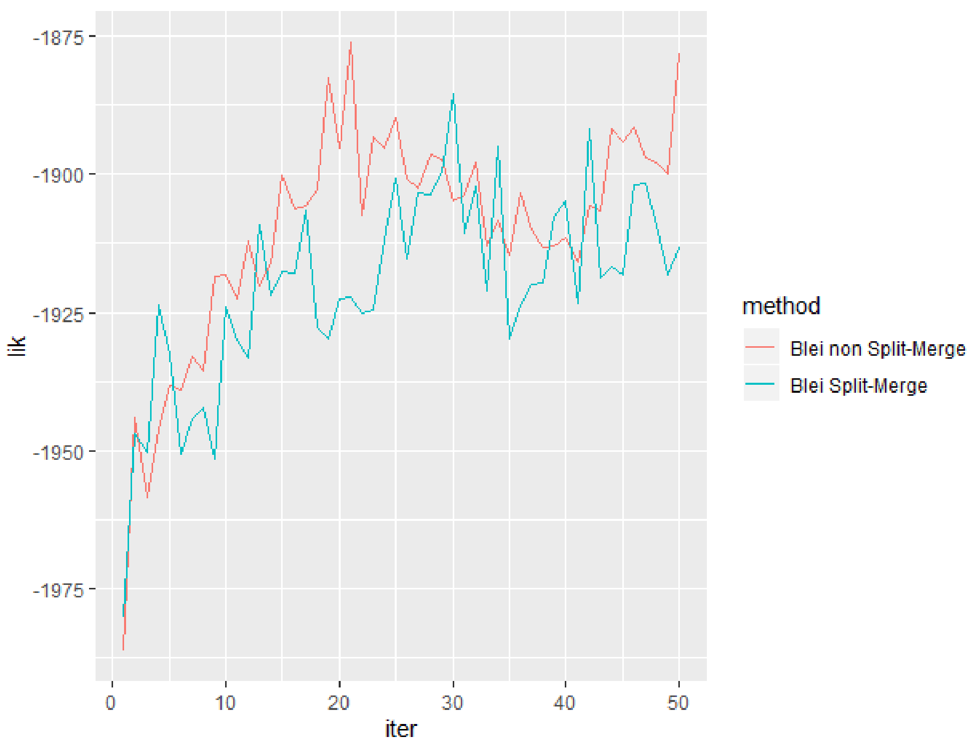

Figure 4.11: Mean perplexity of real data for Blei methods

is almost no difference between perplexity curve of the split-merge and non split-merge methods from the Blei package. Also note that the change in likelihood is much larger for slice sampler. I think it is fair to say that by comparing the perplexity, the slice sampler performs better, but there is no evidence showing that it mixes faster than other methods.

CHAPTER 5

Conclusion

In this thesis, I have discussed hierarchical Dirichlet process and its different posterior sampling methods, including the original Gibbs sampling method, the split-merge method which can be considered as an extension of the Gibbs method, and the Slice sampling method. I then tested three different packages, the Nicola Roberts package that implements Gibbs sampling method, the Blei package that implements both Gibbs sampling and split-merge method, and Amini's package for slice sampling, on simulated corpus with different number of documents. The performance of the different implementations are evaluated by finding and comparing the distribution of the mixing times of the NMI curves, and our result shows that slice sampler mixes faster than both the Gibbs sampling method and the split-merge method. We also found that as the number of documents increases, the mixing time for Gibbs sampling method and split-serge method will decrease and the mixing time for slice sampling method will increase, and both will stabilizing after the number of documents reach 50-75.

I also compared the performance of slice sampling method and the two methods (split-merge and Gibbs) from Blei package by estimating and comparing their held-out perplexity curve. The Nicola-Roberts package is not used since it does not produce the necessary outputs needed for perplexity estimation. The slice sampler method has better perplexity but shows no evidence of faster mixing, contrary to the results we get from the simulated data. This can be due to the fact that there are more words than documents in our real corpus: in out simulated data, there are always more words than documents in the corpus.

So we cannot make the conclusion that HDP with slice sampler mixes faster than other HDP implementations.

There are several parts of this work that can be improved in future studies. First, I am assuming that the packages are implementing the HDP algorithms correctly. Also, the Blei package has NMI that starts off at high level, so I do express reservations on the estimated mixing time from the Blei package. The last thing is that our data generation process is not very straightforward: a better way might be specifying the topic of the document first and then generate the words in the document.

REFERENCES

[AA2019] AA Amini, M Paez, L Lin, ZS Razaee ”Exact slice sampler for Hierarchical Dirichlet Processes” arXiv preprint arXiv:1903.08829, 2019

[Ta2005] Y.W. Teh, M.I. Jordan, M.J. Beal and D.M. Blei. ”Hierarchical Dirichlet Process” JASA 101(476):1566-1581, 2006.

[Tb2005] YW Teh, MI Jordan, MJ Beal, DM Blei ”Sharing clusters among related groups: Hierarchical Dirichlet processes” Advances in neural information processing systems, 2005

[W2012] C Wang, DM Blei ”A split-merge MCMC algorithm for the hierarchical Dirichlet process” arXiv preprint arXiv:1201.1657, 2012

[H2011] Gregor Heinrich. “Infinite LDA” – Implementing the HDP with minimum code complexity. Technical note, TN2011/1, 18 February 2011

[F2014] P Fryzlewicz ”Wild binary segmentation for multiple change-point detection” The Annals of Statistics, 2014

[P2019] OHM Padilla, Y Yu, D Wang, A Rinaldo ”Optimal nonparametric change point detection and localization” arXiv preprint arXiv:1905.10019, 2019

[W2009] Wallach, H. M., Murray, I., Salakhutdinov, R., and Mimno, D. (2009) ”Evaluation Methods for Topic Models”, in Proceedings of the 26th Annual International Conference on Machine Learning, ACM.

[G2009] M Kalli, JE Griffin, SG Walker ”Slice sampling mixture models” Statistics and computing 21 (1), 93-105

[AA] Amini, Arash ”aaamini/hdpslicer” *Github*, https://github.com/aaamini/hdpslicer

[NR] Roberts, Nicola *Github*, https://github.com/nicolaroberts/hdp

[DB] C Wang, D Blei *Github*, https://github.com/blei-lab/hdp

[DAI] Amoualian, Hesam *Github*, https://github.com/Hesamalian/HDP

[TL] Knopp, Johannes *Github*, https://github.com/JoKnopp/text2ldac

[BoW] NSF Research Awards Abstracts 1990-2003, The UCI KDD Archive, Information and Computer Science 18 Nov. 2003, kdd.ics.uci.edu/databases/nsfabs/nsfawards.html.