technische universität
dortmund

Master Thesis

Fast Latent Dirichlet Allocation with Background Topics

Stefan Joschko - van Ackern
September, 2020

Supervisors:
Prof. Dr. Erich Schubert
Prof. Dr. Jörg Rahnenführer

Technical University Dortmund
Department of Computer Science
Chair for Artificial Intelligence (Chair 8)
https://www-ai.cs.tu-dortmund.de

In cooperation with:
Department of Statistics
Statistical Methods in Genetics and Chemometry
https://www.statistik.tu-dortmund.de

Contents

Chapter 1

Introduction

Topic modeling is a part of natural language processing that can be used to solve a wide variety of problems in computer science. The ability to discover abstract topics from unlabeled text data makes topic models an excellent choice for many commercial and academic applications. In the last 20 years a lot of research has evolved around topic models and while they were original intended for text analysis, they can be adapted for other problems such as image classification [26], audio classification [21] or bio informatics [15]. To date, *Latent Dirichlet Allocation (LDA)* [5] is arguably the most common topic model and has many extensions that adapt it for different use cases, like the commonly used models *Author-Topic LDA* [36] and *Seeded LDA* [24]. A recurring problem when using LDA on text data is, that it tends to assign high probability to terms that appear very often. When not dealt with, such terms can pervade the learned topics, partially hiding the terms of actual interest. For this reason it is common practice to filter out stop-words before applying topic models to a text corpus. While this often yields decent results, it is an *ad hoc* measure that rests on a non-coherent theoretical basis. There are of course several other approaches to deal with stop-words in topic models like using word weighting techniques or using the syntactical dependencies between words. Another approach is to introduce a separate topic called *Background Topic (BT)*, which explains frequent terms and stop-words. This way, other topics should be more likely to reflect the terms of actual interest.

1.1 Goals

A BT has been proposed and used by several works in the past, however it is usually not explained in grate detail. The goal of this thesis is to investigate the use of a BT in detail and how it can be used to improve the performance of LDA, especially on datasets where stop-words are not filtered out as a preprocessing step. Two new extensions of LDA that include a BT are developed and tested. The *20Newsgroups* dataset which includes

English news articles as well as a collection of news articles from the German newspaper *Frankfurter Allgemeine Zeitung (FAZ)* are used for this purpose. This thesis further investigates if incorporating prior knowledge about the distribtuion of stop-words can improve the resulting topics. Two different approaches are to either use a BT with fixed term probabilities or to choose asymmetric priors by utilizing word weighting schemes like term frequency.
Another common problem with LDA is that it needs a considerable amount of computation time and resources to be executed on large datasets. Including a BT and not removing frequently used words only worsens this circumstance. It is therefore desirable to incorporate the BT into state of the art sampling approaches like *LightLDA* [47] and *F+LDA* [46] to speed up computation. This thesis will therefore lay another emphasis on techniques to reduce sampling complexity for the two new developed LDA extensions.

1.2 Structure

This thesis will introduce the commonly used topic models *Latent Semantic Analysis (LSI)* [12], *Probabilistic Latent Semantic Analysis (PLSA)* [22] and the already mentioned LDA, including their likelihood function, inference methods and possible extensions, in section 2. It further describes different methods to evaluate the quality of produced topics. Section 3 first discusses different approaches that can be used to deal with stop-words, including background topics. Afterwards two new LDA extensions are developed that modify the generative model to incorporate a BT. One model lays emphasis on choosing fixed term probabilities proportional to different word weighting schemes, while the other uses asymmetric priors to encourage the BT to learn stop-words on its own. Methods to accelerate the sampling process of such models are discussed in section 4. It further covers details about the developed *Java* implementation. Both models are applied to the 20Newsgroups dataset as well as a collection of articles from the FAZ newspaper. The results and evaluation of those experiments are described in section 5, where the main focus is on cluster performance and coherence. Section 6 concludes this thesis and gives an outlook to future research.

Chapter 2

Topic Models

Topic models are a frequently used method in text-mining and provide a way to discover abstract topics in a collection of documents. A topic is intuitively understood by humans, but not easy to define formally. It could also be regarded as the theme or subject of the given text. Depending on the granularity, a topic can tell something about a particular sentence, paragraph or book. The granularity deepens on what single document is expected to be. Assume a collection of news articles needs to be analyzed, where each article is considered to be a document. One article might be about artificial intelligence, while another one is about cryptocurrency. The first will most likely include terms like *learning* and *classification*, while the latter will contain terms like *currency* and *trade*. Terms like *internet* or *regulation* however are likely to appear in both articles. Therefore, a document is usually a mixture of multiple topics, while topics represent a probability distribution over the vocabulary. However, one problem that arises when comparing documents directly is that of synonymy and polysemy. Often the same word has multiple meanings or multiple different words refer to the same thing. An exact search will not find such dependency's. Topic models address this problem by aiming to reflect the hidden semantic structures (the latent space) of a text corpus. Usually, they try to group terms that often appear together, while simultaneously grouping documents that are similar. Figure 2.1 shows an example of such.

2.1 Basics

There are several things that all topic models in this chapter have in common. The input for each topic model is a collection of text documents containing words. The total number of documents is donated by D. Each document d contains one or more words and its length is donated by n_d. The set of all existing terms is called *vocabulary* and its size is donated by V. This thesis will make a distinction between *words* (or more specifically *word tokens*) and *terms*. A word token refers to a specific position w_{di} inside a document d, while a

term is a specific initialization of that word token. So two different words can have the same term assigned to them, but they are nonetheless two different words. Lastly a topic model deals with exactly T topics. With variable T usually being a hyperparameter that can be chosen as desired.

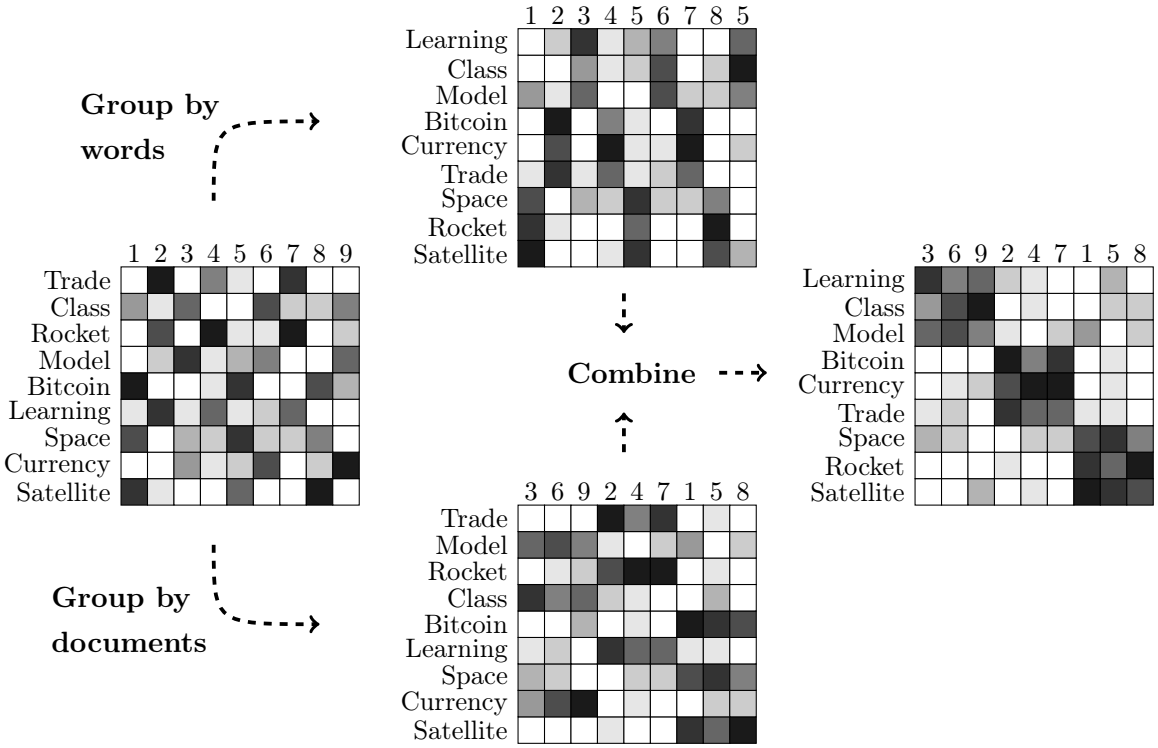


Figure 2.1: Shown is the general idea of topic models. The left matrix shows the frequency of 9 different terms in 9 documents in a corpus. A darker shading indicates a higher term frequency in the corresponding document. It is possible to group similar documents together by reordering the rows, as shown in the bottom matrix. Or similar terms can be grouped together by reordering the columns, as shown in the top matrix. Topic models aim to group together similar words and similar documents at the same time. The resulting matrix on the right shows that documents 3,6,9 are about machine learning, while documents 2,4,7 are about cryptocurrency and 1,5,8 are about rocket science.

2.2 Latent Semantic Analysis (LSA)

Latent Semantic Analysis (LSA) [12] (also called *Latent Semantic Indexing* (LSI)) is one of the first proposed methods to retrieve topics from a document-term matrix. Its goal is to find a low-rank approximation of the document-term matrix by using *singular value decomposition (SVD)*. SVD is a technique in linear algebra that factorizes any matrix A into the product of 3 separate matrices $A = U \times \Sigma \times V^T$. If the document-term matrix A represents the documents as columns and the terms as rows (just like in figure 2.1), then

matrix U represents the connection between terms and topics. Matrix V^T represents the connection between topics and documents and Σ holds the strength of each topic on the diagonal axis. By truncating Σ until only the highest T values remain on the diagonal axis, the best (least-squares) approximation is obtained. An example of this is shown in figure 2.2. While term frequencies are the standard way to represent the corpus document-term matrix, it is also possible to use other representation like *term frequency inverse document frequency (TFIDF)* or *Entropy* [14]. Since they usually decrease the importance of frequent terms they can be an effective way to deal with stop-words in LSA.

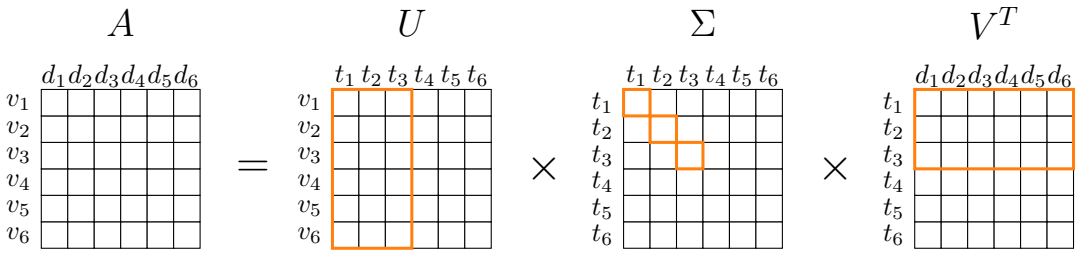


Figure 2.2: Shown is an example of the SVD that is performed by LSA on a simple corpus with documents $d_1, \ldots, d_6$ and terms $v_1, \ldots, v_6$. The document-term matrix A is factorized into the tree matrices $U \times \Sigma \times V^T$. In the shown example, the three topics t_1, t_2, t_3 are selected and topics t_4, t_5, t_6 are discarded. The parts inside the orange boxes represent the approximation achieved by LSA.

2.3 Probabilistic Latent Semantic Analysis (pLSA)

Probabilistic Latent Semantic Analysis (pLSA) [22] (also called Probabilistic Latent Semantic Indexing (pLSI)) is a mixture model based on a generative process. It is inspired by LSA (described in section 2.2) but instead of using SVD for dimension reduction, it is based on the likelihood principle and defines a generative model. The core of pLSA is a latent variable model, which the authors call *aspect model*. This model can be thought of as a Bayesian network, which is shown in plate notation in figure 2.3. It introduces a

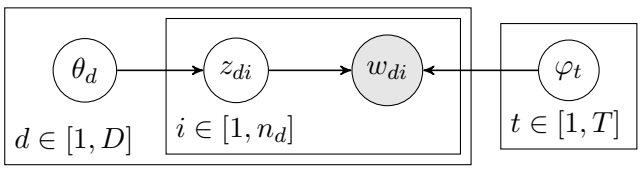


Figure 2.3: The plate diagram of pLSI.

unobserved class variable z_{di} for each word w_{di} in each document d. This latent variable indicates to which topic the corresponding word w_{di} is assigned to. Each document is therefore a mixture of the T topics. Hence, a multinomial distribution θ_d is defined for each document d over all T topics, which describes the mixture of topics in document d.

The set of those distributions is defined as $\Theta = \{\theta_d\}_{d=1}^{D}$. Each topic $t \in [1, T]$ is represented as a multinomial distribution φ_t over all terms in the vocabulary. Thus φ_t indicates how important each term is for topic t. The set of all φ_t is defined as $\Phi = \{\varphi_t\}_{t=1}^{T}$. [1]
The idea of the generative model approach is to come up with a bayesian model that could have generated the given data. The likelihood can be increased by tweaking the unobserved variables of the generative model. Table 2.1 lists all important quantities of pLSA.

T	number of topics
D	number of documents in the corpus
V	number of terms in the vocabulary
n_d	length of document d
θ_d	document-topic distribution for document d
Θ	set of all document-topic distributions, $\Theta = \{\theta_d\}_{d=1}^{D}$
φ_t	topic-term distribution for topic t
Φ	set of all topic-term distributions, $\Phi = \{\varphi_t\}_{t=1}^{T}$
z_{di}	topic assignment for the ith word token in document d
$\mathbf{z}$	set of all topic assignments, $\mathbf{z} = \{\{z_{di}\}_{i=1}^{n_d}\}_{d=1}^{D}$
w_{di}	term assignment for the ith word token in document d
$\mathbf{w}$	set of all term assignments, $\mathbf{w} = \{\{w_{di}\}_{i=1}^{n_d}\}_{d=1}^{D}$

Table 2.1: Quantities for pLSA and LDA.

[1] The original pLSA paper [22] does not define the multinomial distributions θ_d and φ_t. Instead it only introduces the latent class variable z_{di}. In this case, z_{di} is the only unobserved variable, while the word w_{di} and the document d are observed. Therefore the topic coverage for each document (which is defined as θ_d) is tied to the actual observed document. Hence pLSA is a generative model for words given a document, but it can not generate new documents. LDA, which is described in section 2.4, does not suffer from this problem. However θ_d and φ_t are still defined in this section, since it makes the switch from pLSA to LDA much easier.

2.3.1 Likelihood of pLSA

The likelihood that a single word w_{di} instantiates term v is defined as:

$$p(w_{di} = v|\theta_d, \Phi) = \sum_{t=1}^{T} p(z_{di} = t|\theta_d)\, p(w_{di} = v|\varphi_t) = \sum_{t=1}^{T} \theta_{d,t}\, \varphi_{t,v} \tag{2.1}$$

Hence, the probability for each topic t in the given document d (represented by $\theta_{d,t}$) is multiplied with the probability of term v in the topic t (represented by $\varphi_{t,v}$). To get the likelihood of document d, the likelihoods of all its containing words are added up. To help with that a count function $c(v, d)$ is defined, which returns the number of times a term v appears in document d. The likelihood of a document d is defined as:

$$p(w_d|\theta_d, \Phi) = \sum_{v=1}^{V} \sum_{t=1}^{T} c(v, d)\, \theta_{d,t}\, \varphi_{t,v} \tag{2.2}$$

The likelihood of the whole corpus $\mathbf{w}$ is defined by summing up the likelihoods of each document:

$$p(\mathbf{w}|\Theta, \Phi) = \sum_{d=1}^{D} \sum_{v=1}^{V} \sum_{t=1}^{T} c(v, d)\, \theta_{d,t}\, \varphi_{t,v} \tag{2.3}$$

Thus, the document-topic distributions Θ and topic-term distributions Φ need to be optimized in order to increase the likelihood of the model. The set Θ contains a multinomial distribution of every document d with size T, which gives $D \times T$ values to optimize. Set Φ contains T multinational distributions that spread over all V terms, which yields another $T \times V$ values to optimize. One method that can be used for that is the *Expectation Maximization (EM)* algorithm [49].

2.4 Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation (LDA) is an extension of the, in section 2.3 discussed, pLSI topic model. The original paper [5] is one of the most cited papers in the topic modeling research area and the algorithm has since been applied to many different fields such as image classification [26], audio classification [21] or bio informatics [15].

LDA extends the generative model of pLSI such that it lets the user impose prior knowledge about the topics and their coverage over the set of documents. While pLSI works completely unsupervised with no extra effort required, it is often beneficial to use extra knowledge about the topics and documents. To address this, LDA introduces two new hyperparameters α and β. By changing α, one can impose prior knowledge about the document-topic distributions θ_d. This way it is possible to control which topics are covered by a document and which are not. Similarly by changing β, one can impose prior knowledge about the topic-term distributions φ_t. This makes it possible to control how the topics should look like. Figure 2.4 shows the Bayesian network for the generative process

of LDA as a plate diagram. It is mostly the same Bayesian network as the one for pLSI (figure 2.3), except that it now includes the two hyperparameters α and β. While in pLSI all document-topic distributions θ_d and topic-term distributions φ_t are chosen randomly, LDA samples those distribution from an *Dirichlet* distribution, which uses α and β as hyperparameters respectively.

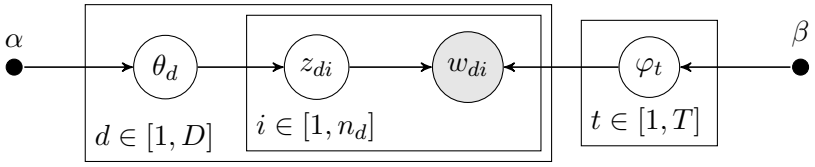


Figure 2.4: The plate diagram of LDA.

2.4.1 Dirichlet Distribution

The Dirichlet density function for order $K \geq 2$ is defined as following:

$$f(x_1, \ldots, x_k; \alpha_1, \ldots, \alpha_K) = \frac{1}{\mathrm{B}(\alpha)} \prod_{i=1}^{K} x_i^{\alpha_i - 1} \tag{2.4}$$

B() is the *beta function* which is defined as:

$$\mathrm{B}(\alpha_1, \ldots, \alpha_K) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{K} \alpha_i)} \tag{2.5}$$

$\Gamma()$ is the *gamma function* which is defined as:

$$\Gamma(n) = (n-1)! \tag{2.6}$$

If α is symmetric i. e. all elements of α have the same value, it can be represented as a scalar and the dirichlet density function becomes:

$$f(x_1, \ldots, x_k; \alpha) = \frac{\Gamma(K\alpha)}{\Gamma(\alpha)^k} \prod_{i=1}^{K} x_i^{\alpha_i - 1} \tag{2.7}$$

However in the following thesis a symmetric prior, say $\alpha = 0.1$, will be donated as $\alpha = \langle 0.1_{\times T} \rangle$ which in this case indicates that the value 0.1 is repeated T times. The main idea of using the Dirichlet distribution is to control the shape of the multinomial distributions θ_d and φ_t by altering the hyperparameters α and β, which therefore lets one impose prior knowledge. Figure 2.5 illustrates how the Dirichlet distribution behaves. A good baseline configuration that yields heuristically good model quality is to choose $\alpha = \langle (\frac{50}{T})_{\times T} \rangle$ and $\beta = \langle 0.1_{\times V} \rangle$ as symmetric priors [16]. When choosing symmetric priors one controls the sparseness of the generated distributions, but without favoring any specific topics or terms.

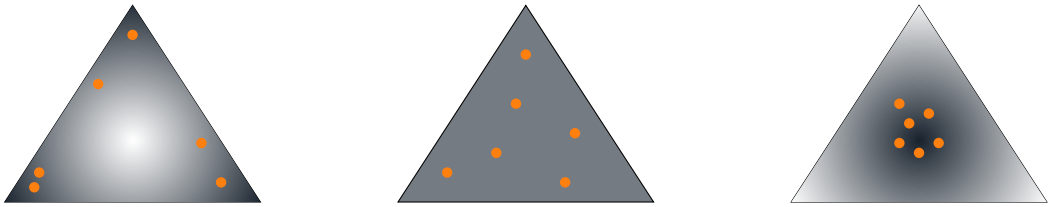


Figure 2.5: Assume that the goal is to create several distributions that represent a three sided dice i.e. multinomial distributions of order 3. This can be achieved by sampling from a dirichlet distribution with hyperparameter $\alpha = (10, 10, 10)$, which yields dices that are mostly fair, but sometimes slightly unfair. If instead a hyperparameter $\alpha = (100, 100, 100)$ is used, the chance of sampling a unfair dice is lower than before, since higher numbers mean lower variance. With $\alpha = (1, 1, 1)$ the function yields dices that are randomly distributed, with no special tendency in any direction. Choosing values smaller than 1 yields multinomial distributions that are sparse. With $\alpha = (0.1, 0.1, 0.1)$ the resulting dices strongly favor one side. Since α is symmetric in this example, it is arbitrary which side the coins favor more.

2.4.2 Generative Process

The full generative algorithm of LDA is shown in algorithm 1. At first the T topic-term distributions are sampled from a dirichlet distribution with parameter β. Next, for each document a document-topic distribution is sampled from a dirichlet distribution with parameter α. A *poisson distribution* with parameter ξ is used to sample a document length n_d, which is useful to make LDA a full generative process. However when LDA is used on existing data, this hyperparameter is usually overlooked since given documents already have a fixed length. For each word token in the document a topic assignment $z_{d,i}$ is sampled from the corresponding document-topic distribution θ_d. Lastly, a term assignment is sampled from the corresponding topic-term distribution (indicated by $\varphi_{z_{di}}$).

1 **for** each topic $t \in [1, T]$ **do** } topic plate
2 | Sample distribution over vocabulary $\varphi_t \sim \text{Dirichlet}(\beta)$
3 **for** each document $d \in [1, D]$ **do** } document plate
4 | Sample distribution over topics $\theta_d \sim \text{Dirichlet}(\alpha)$
5 | Sample document length $n_d \sim \text{Poisson}(\xi)$
6 | **for** each position $i \in [1, n_d]$ in document d **do** } word plate
7 | | Sample a topic assignment $z_{di} \sim \text{Multinomial}(\theta_d)$
8 | | Sample a term $w_{di} \sim \text{Multinomial}(\varphi_{z_{di}})$

Algorithm 1 : The generative process of LDA. The brackets on the right reference the LDA plate diagram shown in figure 2.4. They indicate how different parts of the algorithm correspond to the different plates in the plate diagram.

2.4.3 Likelihood of LDA

The likelihood that a single word w_{di} instantiates term v is the same for LDA and pLSA (shown in equation 2.1):

$$p(w_{di} = v|\theta_d, \Phi) = \sum_{t=1}^{T} \theta_{d,t}\, \varphi_{t,v} \tag{2.8}$$

However, the likelihood of a document is different (compared to equation 2.2), since θ_d is now dependent on the hyperparameter α:

$$p(w_d|\alpha, \Phi) = \int \sum_{v=1}^{V} \sum_{t=1}^{T} c(v,d)\, \theta_{d,t}\, \varphi_{t,v}\ \underbrace{p(\theta_d|\alpha)\, \mathrm{d}\theta_d}_{\substack{\text{because} \\ \theta_d \sim \text{Dirichlet}(\alpha)}} \tag{2.9}$$

It is necessary to integrate over $p(\theta_d|\alpha)$ to account for all possible configurations of θ_d. Also recall that $c(v,d)$ is a function that counts the number of times term v appears in document d. Moreover when considering the likelihood of the whole corpus $\mathbf{w}$ it needs to be taken into account that φ_t is dependent on hyperparameter β:

$$p(\mathbf{w}|\alpha, \beta) = \int \sum_{d=1}^{D} \int \sum_{v=1}^{V} \sum_{t=1}^{T} c(v,d)\, \theta_{d,t}\, \varphi_{t,v}\ \underbrace{p(\theta_d|\alpha)\, \mathrm{d}\theta_d}_{\substack{\text{because} \\ \theta_d \sim \text{Dirichlet}(\alpha)}}\ \underbrace{\prod_{t=1}^{T} p(\varphi_t|\beta)\, \mathrm{d}\varphi_1 \ldots \mathrm{d}\varphi_T}_{\substack{\text{because} \\ \varphi_t \sim \text{Dirichlet}(\beta)}} \tag{2.10}$$

Compared to the likelihood of pLSA (equation 2.3), the basic idea is still the same, with the difference being the integration over $p(\theta_d|\alpha)$ and $p(\varphi_t|\beta)$. Which makes a lot of sense, since the main difference between pLSA and LDA is to condition θ_d and φ_t on α and β by sampling them from a dirichlet distribution.

2.4.4 Inference

Section 2.4.2 defined a generative process for LDA. However the main goal is not to generate new documents, but to learn from existing data. Hence it is necessary to reverse the generative process and learn the posterior distributions of the latent variables $\Theta, \Phi, \mathbf{z}$ given the observed data. This task is known as *posterior inference*. The equation that needs to be solved is the intractable probability $p(\Theta, \Phi, \mathbf{z}|\alpha, \beta, \mathbf{w})$. While it is not practical to compute it exactly, there are ways to approximate it. Since the invention of LDA, many approximate inference algorithms have been proposed. Some of which are Expectation propagation [31], (Collapsed) Variational inference [5] [38] and (Collapsed) Gibbs sampling [18] [16]. Each technique has its benefits and disadvantages. However Gibbs sampling is arguably the most widely used technique for LDA today, since it usually yields relatively simple algorithms with good performance. The remainder of this section describes the general Gibbs sampling approach and then constructs a collapsed Gibbs sampler specific to LDA.

Collapsed Gibbs Sampling

Gibbs sampling is a *Marcov Chain Monte Carlo (MCMC)* technique that can emulate high-dimensional probability distributions by the stationary behavior of a Markov chain. It is the special case where each dimension of a distribution is sampled conditioned on all other values. To understand this concept, consider a joint distribution $p(x) = (x_1, x_2, \ldots, x_m)$ that needs to be approximated. A function $g(x^{(t)})$ is defined that makes a probabilistic choice about which state $x^{(t+1)}$ to visit next. Since a iterative probabilistic choices is made about the posterior distribution, Gibbs sampling is a Monte Carlo method. The function $g(x^{(t)})$ makes decisions according to a transition probability $P_{\text{trans}}(x^{(t+1)}|x^{(0)}, x^{(1)}, \ldots, x^{(t)})$, which tells how likely a new state $x^{(t+1)}$ is given all the previous states. The process simulates a Marcov Chain since the next state that is visited depends only on the current state such that:

$$P_{\text{trans}}(x^{(t+1)}|x^{(0)}, x^{(1)}, \ldots, x^{(t)}) = P_{\text{trans}}(x^{(t+1)}|x^{(t)}) \tag{2.11}$$

As an example, to sample x from a three dimensional joint distribution $p(x) = (x_1, x_2, x_3)$ the sampling would proceed as following:

$$x_1^{(t+1)} \sim P(x_1|x_2^{(t)}, x_3^{(t)})$$

$$x_2^{(t+1)} \sim P(x_2|x_1^{(t+1)}, x_3^{(t)})$$

$$x_3^{(t+1)} \sim P(x_3|x_1^{(t+1)}, x_2^{(t+1)})$$

By repeating this process the stationary distribution is reached eventually. Notice that for sampling $x_2^{(t+1)}$ the new value $x_1^{(t+1)}$ is used while also using the current value $x_3^{(t)}$. However Gibbs sampling also has some disadvantages. First there is no obvious way of telling if and when the algorithm converges i. e. when it reaches the stationary distribution. Hence in practice one has to define a stopping criterion. This can either be a fixed number of iterations or a threshold for the increase of likelihood between iterations. A second problem is that the algorithm can get stuck in a local maximum that is not the global maximum, due to the random initial position. This can however be approached by executing the algorithm several times.

$n_{d,t}$	number of times topic t is assigned inside document d
$n_{d,\cdot}$	set of $n_{d,t}$ for each topic in document d, $n_{d,\cdot} = \{n_{d,t}\}_{t=1}^{T}$
n_d	length of document d (note that $n_d = \lvert n_{d,\cdot}\rvert = \sum_{t=1}^{T} n_{d,t}$)
$n_{t,v}$	number of times topic t is assigned to term v
$n_{t,\cdot}$	set of $n_{t,v}$ for each term for topic t, $n_{t,\cdot} = \{n_{t,v}\}_{v=1}^{V}$
n_t	total assignments of topic t (note that $n_t = \lvert n_{t,\cdot}\rvert = \sum_{v=1}^{V} n_{t,v}$)

Table 2.2: Notation of the count variables used in LDA equations.

Collapsed Gibbs Sampler for LDA

The goal of a Gibbs sampler for LDA is to infer the three hidden variables $\Theta, \Phi, \mathbf{z}$. So a straightforward solution would be to construct a sampler for all three variables. However an important observation about Θ, Φ is that they can be derived from $\mathbf{z}$ with the following two equations:

$$\theta_{d,t} = \frac{n_{d,t} + \alpha_t}{\sum_{t=1}^{T} n_{d,t} + \alpha_t} \tag{2.12}$$

$$\varphi_{t,v} = \frac{n_{t,v} + \beta_v}{\sum_{v=1}^{V} n_{t,v} + \beta_v} \tag{2.13}$$

With $n_{d,t}$ counting how many words of document d are assigned to topic t and $n_{t,v}$ counting how many times term v is associated with topic t. Both $n_{d,t}$ and $n_{t,v}$ can be easily derived from $\mathbf{z}$. Table 2.2 shows an overview of all relevant count variables that are used in the following equations. A simpler Gibbs sampler can be constructed by integrating out Θ, Φ and just sampling z_i, which is then called *collapsed Gibbs sampler*. In this case z_i refers to the topic assignment of any word in any document i. e. any position in $\mathbf{z}$. The resulting posterior according to the Gibbs sampling algorithm is $p(z_i|\mathbf{z}^{(-i)}, \mathbf{w}, \alpha, \beta)$, where $\mathbf{z}^{(-i)}$ means that z_i is removed from $\mathbf{z}$. With the rules of conditional probability it is possible to reshape this posterior in the following way to arrive at the joint distribution of $\mathbf{z}, \mathbf{w}$ given the hyperparameter α, β:

$$p(z_i|\mathbf{z}^{(-i)}, \mathbf{w}, \alpha, \beta) = \frac{p(z_i, \mathbf{z}^{(-i)}, \mathbf{w}|\alpha, \beta)}{p(\mathbf{z}^{(-i)}, \mathbf{w}|\alpha, \beta)} \tag{2.14}$$

$$\propto p(z_i, \mathbf{z}^{(-i)}, \mathbf{w}|\alpha, \beta) = p(\mathbf{z}, \mathbf{w}|\alpha, \beta) \tag{2.15}$$

When looking again at the plate diagram of LDA in figure 2.4 it is seen that $\mathbf{z}$ is only dependent on α, while $\mathbf{w}$ is dependent on $\mathbf{z}$ and β. The joint probability of $\mathbf{z}, \mathbf{w}$ can therefore be rewritten (with help of the *chain rule*) as:

$$p(\mathbf{z}, \mathbf{w}|\alpha, \beta) = p(\mathbf{z}|\alpha)p(\mathbf{w}|\mathbf{z}, \beta) \tag{2.16}$$

Now both parts of the product in 2.16 can be handled separately. By incorporating Θ into the first part $p(\mathbf{z}|\alpha)$ it becomes:

$$p(\mathbf{z}|\alpha) = \int p(\mathbf{z}|\Theta)p(\Theta|\alpha)\mathrm{d}\Theta \tag{2.17}$$

Similarly, incorporating Φ into the second part $p(\mathbf{w}|\mathbf{z}, \beta)$ results in:

$$p(\mathbf{w}|\mathbf{z}, \beta) = \int p(\mathbf{w}|\mathbf{z}, \Phi)p(\Phi|\beta)\mathrm{d}\Phi \tag{2.18}$$

Both equations 2.17 and 2.18 basically are multinomial distributions with a dirichlet prior. The dirichlet distribution is a *conjugate prior* of the multinomial distribution. Which

means that the conjugate prior (dirichlet distribution) of a probability distribution (multinomial distribution) results in a posterior distribution with the same functional form as the prior (but with different parameters). This effect can be seen when resolving equation 2.17 by multiplying the multinomial distributions $p(\mathbf{z}|\Theta)$ with the dirichlet distributions [2] $p(\Theta|\alpha)$:

$$\int p(\mathbf{z}|\Theta)p(\Theta|\alpha)\mathrm{d}\Theta = \int \prod_{d=1}^{D}\prod_{t=1}^{T}\theta_{d,t}^{n_{d,t}} \prod_{d=1}^{D}\frac{1}{\mathrm{B}(\alpha)}\prod_{t=1}^{T}\theta_{d,t}^{\alpha_t-1}\mathrm{d}\theta_d \tag{2.19}$$

$$= \prod_{d=1}^{D}\frac{1}{\mathrm{B}(\alpha)}\int \prod_{t=1}^{T}\theta_{d,t}^{n_{d,t}+\alpha_t-1}\mathrm{d}\theta_d \tag{2.20}$$

$$= \prod_{d=1}^{D}\frac{\mathrm{B}(n_{d,\cdot}+\alpha)}{\mathrm{B}(\alpha)}, \quad n_{d,\cdot} = \{n_{d,t}\}_{t=1}^{T} \tag{2.21}$$

Where $n_{d,\cdot} = \{n_{d,t}\}_{t=1}^{T}$ is a vector that counts the assignments $n_{d,t}$ of each topic t in document d. Similarly, equation 2.18 can be resolved as following:

$$\int p(\mathbf{w}|\mathbf{z},\Phi)p(\Phi|\beta)\mathrm{d}\Phi = \int \prod_{t=1}^{T}\prod_{v=1}^{V}\varphi_{t,v}^{n_{t,v}} \prod_{t=1}^{T}\frac{1}{\mathrm{B}(\beta)}\prod_{v=1}^{V}\varphi_{t,v}^{\beta_t-1}\mathrm{d}\varphi_t \tag{2.22}$$

$$= \prod_{t=1}^{T}\frac{1}{\mathrm{B}(\beta)}\int \prod_{v=1}^{V}\varphi_{t,v}^{n_{t,v}+\beta_t-1}\mathrm{d}\varphi_t \tag{2.23}$$

$$= \prod_{t=1}^{T}\frac{\mathrm{B}(n_{t,\cdot}+\beta)}{\mathrm{B}(\beta)}, \quad n_{t,\cdot} = \{n_{t,v}\}_{v=1}^{V} \tag{2.24}$$

Where $n_{t,\cdot} = \{n_{t,v}\}_{v=1}^{V}$ is a vector that counts the assignments of topic t to each term v in the vocabulary. By inserting equations 2.21 and 2.24 back into equation 2.16, the final form of the joint probability of $\mathbf{z}, \mathbf{w}$ is obtained:

$$p(\mathbf{z},\mathbf{w}|\alpha,\beta) = p(\mathbf{z}|\alpha)p(\mathbf{w}|\mathbf{z},\beta) \tag{2.16}$$

$$= \prod_{d=1}^{D}\frac{\mathrm{B}(n_{d,\cdot}+\alpha)}{\mathrm{B}(\alpha)}\prod_{t=1}^{T}\frac{\mathrm{B}(n_{t,\cdot}+\beta)}{\mathrm{B}(\beta)} \tag{2.25}$$

With help of equation 2.25 the final update rule for single word token is constructed from equation 2.14:

$$p(z_i|\mathbf{z}^{(-i)},\mathbf{w},\alpha,\beta) = \frac{p(\mathbf{z},\mathbf{w}|\alpha,\beta)}{p(\mathbf{z}^{(-i)},\mathbf{w}|\alpha,\beta)} \tag{2.14}$$

$$= \frac{p(\mathbf{z}|\alpha)}{p(\mathbf{z}^{(-i)}|\alpha)}\,\frac{p(\mathbf{w}|\mathbf{z},\beta)}{p(\mathbf{w}^{(-i)}|\mathbf{z}^{(-i)},\beta)p(w_i)} \tag{2.26}$$

$$\propto \prod_{d=1}^{D}\frac{\mathrm{B}(n_{d,\cdot}+\alpha)}{\mathrm{B}(n_{d,\cdot}^{(-i)}+\alpha)}\prod_{t=1}^{T}\frac{\mathrm{B}(n_{t,\cdot}+\beta)}{\mathrm{B}(n_{t,\cdot}^{(-i)}+\beta)} \tag{2.27}$$

[2] Recall the definition of the dirichlet distribution in equation 2.4 if needed.

$$\propto \frac{\Gamma(n_{d,t}+\alpha_t)\Gamma(\sum_{t=1}^{T} n_{d,t}^{(-i)}+\alpha_t)}{\Gamma(n_{d,t}^{(-i)}+\alpha_t)\Gamma(\sum_{t=1}^{T} n_{d,t}+\alpha_t)} \frac{\Gamma(n_{t,v}+\beta_v)\Gamma(\sum_{v=1}^{V} n_{t,v}^{(-i)}+\beta_v)}{\Gamma(n_{t,v}^{(-i)}+\beta_v)\Gamma(\sum_{v=1}^{V} n_{t,v}+\beta_v)} \tag{2.28}$$

$$\propto \frac{n_{d,t}^{(-i)}+\alpha_t}{(\sum_{t=1}^{T} n_{d,t}+\alpha_t)-1} \frac{n_{t,v}^{(-i)}+\beta_v}{\sum_{v=1}^{V} n_{t,v}^{(-i)}+\beta_v} \tag{2.29}$$

Note that the first denominator of equation 2.29 can also be written as following:

$$(\sum_{t=1}^{T} n_{d,t}+\alpha_t)-1 = |n_{d,\cdot}+\alpha|-1 = (n_d-1)+|\alpha| \tag{2.30}$$

Here variable n_d is the length of document d. Furthermore the second denominator of equation 2.29 can be written as:

$$\sum_{v=1}^{V} n_{t,v}^{(-i)}+\beta_v = |n_{t,\cdot}^{(-i)}+\beta| = n_t^{(-i)}+|\beta| \tag{2.31}$$

Here n_t describes the total assignments of topic t in the whole corpus. By incorporating this notation into equation 2.29 a more simple to read equation is created: [3]

$$p(z_i = t|\text{rest}) \propto \frac{n_{d,t}^{(-i)}+\alpha_t}{(n_d-1)+|\alpha|} \frac{n_{t,v}^{(-i)}+\beta_v}{n_t^{(-i)}+|\beta|} \tag{2.32}$$

By omitting the $(-i)$ notation and removing the first denominator (which is constant for each document) the even more easy to read equation is:

$$p(z_i = t|\text{rest}) \propto (n_{d,t}+\alpha_t) \frac{n_{t,v}+\beta_v}{n_t+|\beta|} \tag{2.33}$$

Although it is not trivial to arrive at this final equation, it can be intuitively understood. The first part $(n_{d,t}+\alpha_t)$ describes the proportion of topic t in document d. This basically corresponds to the document-topic distribution θ_d, with the difference being that the corresponding value α_t is added. Which is the reason why α and β are sometimes also called *pseudo counts*. The math works out that way because, as mentioned earlier, the dirichlet distribution is a conjugate prior to the multinomial distribution. It is easy to see how a different initialization of α directly influences the document-topic distribution. If for example α is symmetric and very large, the pseudo counts would smooth out the document-topic distributions and documents would be more likely to have equal probability on all topics. If α however is chosen asymmetric, topics with a high value in α would be favored and vice versa. The second part of the equation $(n_{t,v}+\beta_v)$ basically corresponds to the topic-term distribution φ_t, also with the difference being that the pseudo count β_v is added. The denominator $(n_t+|\beta|)$ makes sure that the probability is normalized.
With help of equation 2.33 the sampling algorithm can build up a probability distribution

[3] If α or β are symmetric and represented as a scalar, than they can also be written as αT and βV, instead of $|\alpha|$ and $|\beta|$.

for word token i by computing $p(z_i = t|\text{rest})$ for each existing topic t. After normalizing the distribution a new topic assignment z_i can be sampled from it. For one full corpus sweep the algorithm iteratively samples a new topic assignment for each word token in each document. Algorithm 2 shows the full collapsed Gibbs sampling algorithm for LDA. Furthermore figure 2.6 shows an example of the general idea of Gibbs sampling.

Input : Corpus $\mathbf{w} = \{\{w_{di}\}_{i=1}^{n_d}\}_{d=1}^{D}$, Number of iterations M
Output : $\Theta, \Phi, \mathbf{z}$

Choose a random topic assignment for each word token $\mathbf{z} = \{\{z_{di}\}_{i=1}^{n_d}\}_{d=1}^{D}$
Initialize count matrices $n_{d,t}$, $n_{t,v}$ and n_t according to $\mathbf{z}$
Allocate memory for array $p \leftarrow \{0\}_{t=1}^{T}$
for M **do**
  for $d = 1$ to D **do**
    for $i = 1$ to n_d **do**
      `decrementCounts`($n_{d,z_{d,i}}$, $n_{z_{d,i},w_{d,i}}$, $n_{z_{d,i}}$)
      for $t = 1$ to T **do**
        $p(t) \leftarrow (n_{d,t} + \alpha_t) \frac{n_{t,v}+\beta_v}{n_t+|\beta|}$
      $z_{d,i} \leftarrow$ `sampleTopicFrom`(p)
      `incrementCounts`($n_{d,z_{d,i}}$, $n_{z_{d,i},w_{d,i}}$, $n_{z_{d,i}}$)
$\Theta \leftarrow$ compute from $\mathbf{z}$ according to equation 2.12
$\Phi \leftarrow$ compute from $\mathbf{z}$ according to equation 2.13
return $(\Theta, \Phi, \mathbf{z})$

Algorithm 2 : The collapsed Gibbs algorithm for LDA.

2.5 Evaluation of Topic Models

The evaluation of produced topics is an open research area and arguably the least understood part of topic modeling. In the paper *Reading Tea Leaves: How Humans Interpret Topic Models* [7] the authors emphasize that intrinsic measures, like for example held-out likelihood, do not correlate well with human judgment. They facilitate that a high likelihood is not a good indicator of semantically meaningful topics. If the aim is to produce topics that can be understood and interpreted by humans then coherence measures should be considered. Those coherence measures are described in section 2.5.2.
However topics models can be also used for dimensionality reduction or text classification. When documents are labeled, it is possible to evaluate the output of a topic model by its cluster performance. Measures for computing the cluster performance are described in section 2.5.3.
But first, section 2.5.1 discusses one of the most common intrinsic measures for topic models called *Perplexity*.

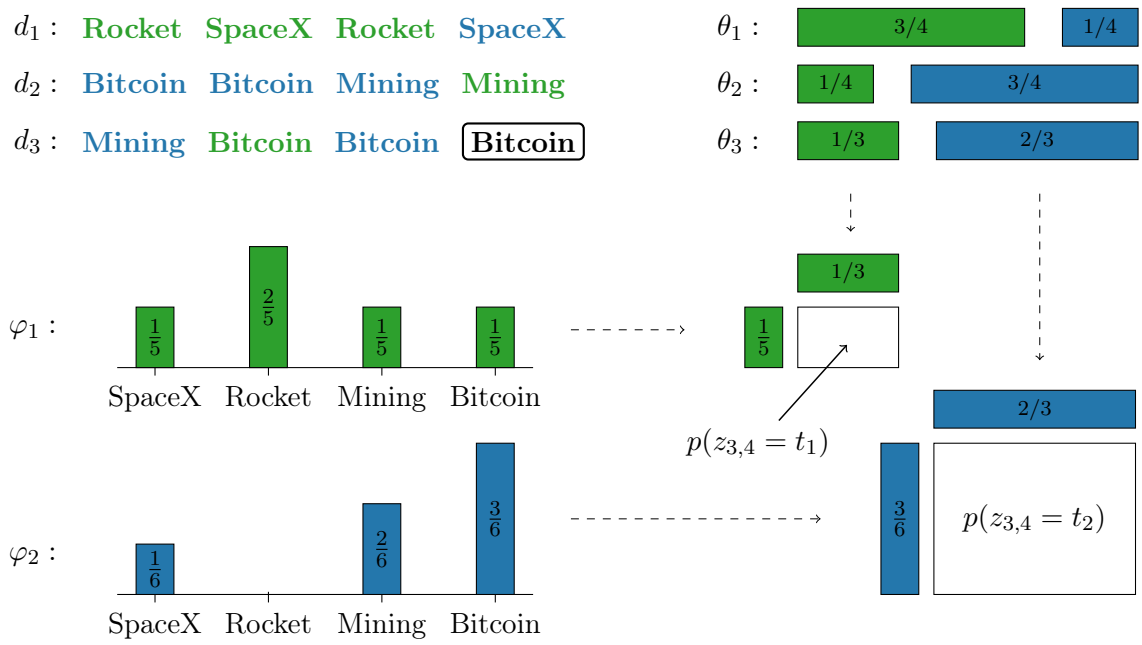


Figure 2.6: Shown is a small example on how Gibbs sampling for LDA works. It does however not consider the influence of the hyperparameters α and β. The corpus (top-left) contains 3 documents, which each contain 4 words. Document d_1 is about SpaceX and Rockets, while d_2 and d_3 are about Bitcoin and Mining. Every word is assigned to either topic t_1 (green) or topic t_2 (blue). The task is to sample a topic assignment for word $w_{3,4}$. It is therefore temporally removed from the corpus. Next the algorithm computes θ_3 (the document-topic distribution of document d_3) which reveals that $1/3$ of the remaining words are assigned to topic t_1 and $2/3$ to topic t_2. It also computes the topic-term distribution φ_1 and φ_2 (it needs however only to consider the probability of the current term). Topic t_1 has a probability of $1/5$ for term *Bitcoin*, while t_2 has $1/2$. To get the probability that $w_{3,4}$ is assigned to topic t_1 (or t_2 respectively) the probability of t_1 in d_3 is multiplied with the probability of term *Bitcoin* in t_1. The final step is to create a multinomial distribution with the two resulting probabilities and sample from it to get the topic assignment for word $w_{3,4}$. However, since those probabilities generally do not sum to 1 an extra normalization step is required.

2.5.1 Perplexity

Perplexity is a common way to evaluate probabilistic models by using the log likelihood on held-out data. This way one can measure how the model adapts to new unseen data. This is usually done by splitting the text data in one part for training $\mathbf{w}^{(train)}$ and one for testing $\mathbf{w}^{(test)}$. The model is trained on $\mathbf{w}^{(train)}$ and the set of topic-term distributions $\Phi^{(train)}$ is obtained from it. The document-topics distributions $\Theta^{(train)}$ are not taken into consideration since they are specific to the documents in the training set. The predictive likelihood of the unseen data is therefore $p(\mathbf{w}^{(test)}|\Phi^{(train)}, \alpha)$. The authors of [42] propose several methods to compute an approximation of the likelihood for unseen documents. The perplexity is then defined as the exponential of the negative normalized log likelihood:

$$\text{perplexity}(\mathbf{w}^{(test)}) = \exp\left(-\frac{\log\, p(\mathbf{w}^{(test)}|\Phi^{(train)}, \alpha)}{\#\text{ tokens of } \mathbf{w}^{(test)}}\right) \tag{2.34}$$

In this case, a lower perplexity suggests a better model, since it is a decreasing function of the predictive likelihood. However since [7] found that human judgment and perplexity are mostly not correlated and sometimes even slightly anti-correlated, it is arguable if perplexity is an appropriate measure for the evaluation of topics.

2.5.2 Coherence Measures

Topic models generally give no guaranty on the interpretability of their output. Generated topics are often flawed to human domain experts. There has been lots of research around measuring how well generated topics align with human evaluations. In [29] the authors worked with human experts that judged the quality of generated topics. They propose a coherence measure based on the co-occurrence of the top terms between topics. Hence, they consider the N top words from each generated topic and sum a confirmation measure over all possible word pairs. Such an confirmation measure can for example be the *pointwise mutual information (PMI)*. Scores computed by this approach have shown to correlate better with human ratings than intrinsic measures like perplexity [7] [29]. Topics that are distinguishable from each other usually score higher than those which are very similar. If a topic model does not deal with stop-words in a meaningful way it is expected that the produced topics score worse, since a lot of words are scattered across all topics. The coherence score can therefore be used to evaluate the performance of the BT, since a higher score is expected when most stop-words are concentrated inside a single topic.
The paper [35] compares a wide variety of different coherence measures and how they relate to human ratings. They propose the C_v score which is a coherence measure that (according to the authors tests) clearly outperforms other common coherence measure like C_{UMass} or C_{UCI}. The set of top terms for each topic t is called W and consist of the top $N = 20$ terms with highest probability in t. Each W is segmented into 20 segments

where each term $w_i \in W$ is tested against the full set W (which in [35] is called *one-set segmentation*). The authors describe a segment as a tuple $(W', W^\star)$ where W' is a set that contains just the single term w_i and $W^\star$ is the full set W. Each tuple is evaluated using an *indirect confirmation measure* by converting both elements of the tuple $(W', W^\star)$ into equally sized vectors and computing the cosine similarity. Compared to using a direct confirmation measure where two terms are compared directly, an indirect confirmation measure takes all top words W into account. The advantage is that if two terms in W appear seldom together but co-occur very frequently with all other terms in W, they can still get a high confirmation. An element, say W', is converted to a vector with function $\vec{v}(W')$ which is defined as follows:

$$\vec{v}(W') = \left\{ \sum_{w_i \in W'} NPMI(w_i, w_j) \right\}_{j=1}^{|W|} \tag{2.35}$$

$$NPMI(w_i, w_j) = \frac{\log \frac{P(w_i, w_j)+\epsilon}{P(w_i) \times P(w_j)}}{-\log(P(w_i, w_j) + \epsilon)} \tag{2.36}$$

Here *NPMI* is a direct confirmation measure that is computed between two terms. The probability of two terms $P(w_i, w_j)$ can be derived from the tested corpus itself or another external reference corpus. For the C_v measure a *boolean sliding window* with size 110 is used. This window is moved over the documents, creating new virtual documents each time. Since it is moved just one word token per step, created virtual documents carry some proximity between word tokens. $P(w_i, w_j)$ is then derived by counting co-occurrences of terms inside these virtual documents. The similarity of both vectors $\vec{u} = \vec{v}(W')$ and $\vec{w} = \vec{v}(W^\star)$ from tuple $(W', W^\star)$ is then computed using the cosine similarity which is defined as follows:

$$S_{cos} = \frac{\sum_{i=1}^{|W|} u_i \times w_i}{\|\vec{u}\|_2 \times \|\vec{w}\|_2} \tag{2.37}$$

The final C_v coherence score is then derived by computing the arithmetic mean of all tuples.

2.5.3 Cluster Performance

The output of LDA are the topic assignments for each word token $\mathbf{z}$ as well as the document-topic distributions Θ and the topic-term distributions Φ. Therefore it does not provide an ad hoc clustering. However, since documents are usually described by just a few topics, a clustering can be recovered from the LDA result by looking at the document-topic distributions Θ. This however only works if the given data is labeled and each document d has a true class $K(d)$ assigned to it. A common approach to group documents with LDA into k cluster is to choose $T = k$ and take for each document d the topic with the highest probability in θ_d as its cluster assignment. This derived cluster assignment is called $C(d)$. It is

however not clear to which label the cluster number assignments correspond to. Matching labels to the corresponding topic number is therefore a separate task and only possible if the predicted clustering is relatively unambiguously. There are multiple measures that can be used to judge the quality of the predicted labels C when the true labels K are known. A selecting that is used for this thesis is discussed in the following.

Adjusted Rand Index

The *Adjusted Rand Index (ARI)* measures the similarity of two cluster assignments. It is bounded by the range $[-1, 1]$, where a higher value indicates a better clustering. It is symmetric and the permutation of the assignments are not taken into account. If two different cluster assignments are a perfect match they get a score of 1, while random, uniform assignments get a score close to 0. The ARI utilizes *pair-counting*, where two documents d_i and d_j are a pair (d_i, d_j) if (and only if) they are in the same cluster. It is computed in the following way:
Let a be the number of pairs (d_i, d_j) where $C(d_i) = C(d_j)$ and $K(d_i) = K(d_j)$ (the true positives). And let d be the number of pairs (d_i, d_j) where $C(d_i) \neq C(d_j)$ and $K(d_i) \neq K(d_j)$ (the true negatives). Further notice that $\binom{D}{2}$ represents the total number of possible pairs without ordering. The (unadjusted) *random index (RI)* is defined as [32]:

$$RI = \frac{a + d}{\binom{D}{2}} \tag{2.38}$$

To guarantee that a random, uniform clustering will get a value close to 0, the ARI is adjusted by the expected RI in the following way [23]:

$$ARI = \frac{RI - E[RI]}{max(RI) - E[RI]} \tag{2.39}$$

Normalized Mutual Information

The *Normalized Mutual Information (NMI)* measures the agreement of two different cluster assignments. Just like the ARI it is symmetric and ignores permutations. If two different cluster assignments are a perfect match they get a score of 1, while assignments that are independent get a score close to 0. It is defined in the following way:
The *Shannon entropy* (or just called *entropy*) describes the average level of information and is defined as [37]:

$$H(C) = -\sum_{t=1}^{|C|} P(t) \log P(t) = -\sum_{t=1}^{|C|} \frac{|C_t|}{D} \log \frac{|C_t|}{D} \tag{2.40}$$

Where C is a clustering with $|C|$ different labels, with $|C_t|$ being the number of documents that are assigned to label t. The (unnormalized) *mutual information (MI)* of two different cluster assignments C and K is defined as [44]:

$$MI(C,K) = \sum_{i=1}^{|C|}\sum_{j=1}^{|K|} P(i,j)\log\frac{P(i,j)}{P(i)P(j)} = \sum_{i=1}^{|C|}\sum_{j=1}^{|K|}\frac{|C_i \cap K_j|}{D}\log\frac{D|C_i \cap K_j|}{|C_i||K_j|} \tag{2.41}$$

Note that $H(C) = MI(C,C)$. The NMI is defined in the following way [44]:

$$NMI(C,K) = \frac{MI(C,K)}{\text{mean}(H(C)+H(K))} = \frac{MI(C,K)+MI(K,C)}{MI(C,C)+MI(K,K)} \tag{2.42}$$

Chapter 3

Background Topic

The generative LDA model assumes that all terms in the corpus are generated from a single language model. If stop-words are not removed then the model has no other choice than to give high probability to those very common terms in order to maximize likelihood. To get rid of those terms, it is possible to design a generative model that assumes common words are coming from a different kind of distribution. This way, the model can consider two different kinds of language models. One for the unknown topics of actual interest and another one that assigns high probabilities to common stop-word like terms. A BT is a discrete distribution over the vocabulary, just like any other topic. A document draws from the normal topics, as well as from the BT. When the BT is designed correctly it should act as a filter which explains most of the common stop-word like terms, making it more likely for the normal topics to reflect terms of actual interest. Furthermore such a BT is not exclusive to stop-words and could be used to filter out all kinds of biases from a given dataset. First section 3.1 discusses the problem of stop-words in topic models and different approaches to deal with them. Section 3.2 defines two new topic models that are heavily inspired by LDA, which also incorporate a BT.

3.1 Handling Stop-Words in Topic Models

Stop-words are terms like *the, of, and, to, a* that are very common in any language and are used trough all domains and subject areas. When not dealt with, such terms can pervade the learned topics, partially hiding the terms of actual interest [10]. Such topics look subjectively less meaningful and model performance can suffer because documents about very different subjects still share many terms. The following of this section discusses several different approaches to deal with this problem.

3.1.1 Using Stop-Word Lists or Term Frequency

The most common approach to deal with stop-words in topic modeling is to simply remove them as a preprocessing step. There are, however, several problems associated with using predefined stop-word lists. First, a stop-word list is language specific. Second, a very general list is often not sufficient and a more domain-dependent list is required. As an alternative to stop-word lists, terms can also be removed by their term frequency (tf) of document frequency (df). One could for example filter out all words that appear in more than 50% of all documents. However this comes at the risk of under-coverage or over-coverage, where either noise remains or terms of interest are accidentally removed. While stop-word removal often yields decent results, it is an ad hoc measure that rests on a non-coherent theoretical basis.

3.1.2 Using Word Weighting

The authors of [43] use term weighting schemes such as *Information Content (IC)* and *Pointwise Mutual Information (PMI)*, to account for very frequent terms and thus improve LDA results. In their implementation of the Gibbs sampler, they integrate these weights into the probability of a word token being assigned to a specific topic $p(z_i = t|\text{rest})$ (compare to equation 2.32) such that terms with a high weight have a greater probability. They show that, for their cross-language retrieval task, word weighting improves results even though stop-words were not removed beforehand.

3.1.3 Using syntactical Dependencies between Words

The authors of [17] argue that terms are used for two different reasons. They either serve a syntactic function (stop-words fall into this category) or provide semantic content. Because function words and content words are used differently in human language they have different dependencies and occurrences. The syntactic dependencies between terms are bounded inside a single sentence, while semantic dependencies are usually bounded by the whole document. The authors use a *hidden Markov model (HMM)* (which compared to LDA is not based on the bag-of-words assumption) that learns the syntactic dependencies between words. Words that are classified as content words by the HMM get emitted to an LDA model which in turn learns the semantic dependency between those words. This however comes at a considerable computational cost compared to just using an LDA model.

3.1.4 Using asymmetric Dirichlet Priors

In [41] the authors lay great emphasis on the fact that choosing symmetric priors for LDA often yields sub-optimal performance. They argue that choosing the right prior for the document-topic distribution α is crucial for a well performing LDA model. On the

other hand, choosing an asymmetric β prior does not improve results according to their experiments. Most text corpora have a power law term distribution, where some terms are used very frequently while more specific terms occur relatively rarely. The assumption of a symmetric α prior that all topics have almost the same size is not well justified. A more realistic assumption (which is supported by the test results of [41]) is that topics vary in size by a lot, where some topics are more general and others are more specific. When choosing an α prior that reflects the document-topic distributions in a more meaningful way, it can be assumed that the biggest topic (or the few biggest topics) explains the very common stop-word like terms, while smaller topics are the ones of actual interest. The authors further speculate if the robustness against very common terms of more advanced models like *Pachinko allocation* [28] and *Wallach's topic-based language mode* [39] might just be the result of a well-chosen α prior.

3.1.5 Using a Background Topic

The book [49] defines a BT for pLSA (described in section 2.3) and discusses it in great detail. It defines an observed variable $\lambda_B \in [0, 1]$ which controls how likely the BT is. Hence a word is either drawn from the BT with probability λ_B or drawn from any of the normal topics with probability $1 - \lambda_B$. By setting $\lambda_B = 0.5$ for example, 50% of all words in the corpus will be assigned to the BT. The authors further assume that the BT itself is known and fixed. Although they give no specific instructions on how to initialize such BT, they state that a good first approach could be, to use the normalized counts for each term.

The authors of [8] define an extension of LDA which they call the *special words with background model* (SWB). This model aims to improve query performance for finding documents when search terms contain some general and specific terms at the same time. A word is generated by either a normal topic, a document specific topic, or a background topic. Each word has a latent variable $x \in \{0, 1, 2\}$ associated with it, that defines from which of the three distributions it is drawn. Here however, the BT is not assumed to be known and fixed. Instead it is (similar to the normal topics in LDA) drawn from a Dirichlet distribution with hyperparameter β_2. The normal topics are controlled by hyperparameter β_1. All three kinds of distributions (normal topics, background topics, document specific topic) are learned by the model. Therefore the difference between the BT and the normal topics is the separate dirichlet prior β_2 and no dependence on hyperparameter α. In their experiments the authors use $T = 200$ and $\beta_1 = \langle 0.0001_{\times V} \rangle$, $\beta_2 = \langle 0.01_{\times V} \rangle$ as symmetric priors. On their used datasets the BT explained roughly $5 - 10\%$ of the words. It is however worth mentioning that standard stop-words were removed from the corpus before training the model.

In [11] the authors create a bayesian statistical model for sentence extraction in query-

focused multi-document summarization, which they call *BAYESUM*. It leverages the common case in which multiple documents are relevant to a single query. Their model assumes that every sentence appears in a document because (a) it is relevant for a query, (b) it provides background information about the document that is not relevant to a known query, (c) it contains useless filler words. Every word is assigned to one of the three sources and thus a sentence is a mixture of those sources. For every sentence s in a document d, there is a vector π_{ds} that controls the mixture of this sentence. In the generative story the latent variable z_{ksn} for every word w_{ksn} is sampled from π_{ds}. If $z_{ksn} = 0$ then the word is considered to be a filler word. The set of all π_{ds} vectors, which is called π, is generated by a Dirichlet distribution with prior α. The hidden variables are then inferred using *expectation propagation* [30]. Their experiments show that their model consistently outperforms contending, state of the art information retrieval models, however they do not explicitly describe which role their BT played in that.
Paper [19] also focuses on multi-document summarization. The authors define a hierarchical LDA-style model which they call *HIERSUM*. They also define a BT (called BACKGROUND) that is drawn from a dirichlet distribution with prior λ_B. They further draw a content distribution (called CONTENT) and a document-specific vocabulary distribution (called DOCSPECIFIC) for each document. For each sentence s, a distribution over topics is drawn in the form of (CONTENT, DOCSPECIFIC, BACKGROUND) from a dirichlet distribution with pseudo-counts $(1, 5, 10)$. It is therefore assumed that $10/16$ of the word tokens contained by a sentence belong to the BT. The hidden variables are then inferred using Gibbs sampling. Although their model performs well it is not mentioned which impact the BT has on achieved results.

3.2 Incorporating a Background Topic into LDA

As discussed in section 3.1.5 there are already several different ways found to initialize and use a BT. However the BT is used for varying tasks and is not always described in great detail. The goal of this thesis is to investigate how a BT can be used effectively on corpora with no stop-word filtering. In the following, two modified LDA models that incorporate a BT are discussed in detail. The task of implementing a BT can roughly be divided into two considerations:

- First consider that the BT will probably be bigger than the other topics, since stopwords are usually very frequent. In some cases the BT might be assigned to more than 50% of all word tokens. It is therefore necessary to increase and control the general probability of the BT.

- The second task is, to only capture the right terms. The BT should only incorporate high frequency terms that are most likely stop-words, while leaving the terms of actual interest untouched. It can therefore be useful to incorporate some prior knowledge about the distribution of stop-words, for example by taking word weighting measures into account.

3.2.1 Static Background Topic

A first approach to implement a BT is to use a static multinomial distribution φ_B which holds a probability for each term in the vocabulary. This model will be called LDA-BT$_{\text{static}}$. Compared to a normal topic, the BT can not change. Furthermore it is expected that stop-words appear in every document regardless of its content. It is therefore also reasonable to assume that the BT should not be dependent on the document-topic distribution θ_d. For example, a stop-word in document d should always have the same probability of being assigned to the BT, no matter if d contains a lot of stop-words or just a few. Figure 3.1 shows a plate diagram of the modified LDA algorithm. The new hyperparameter $\lambda_B \in [0, 1]$

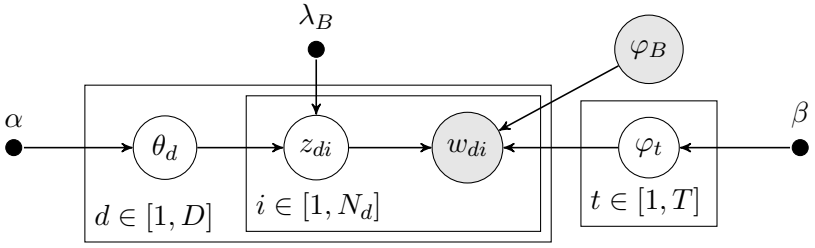


Figure 3.1: The plate diagram of the LDA-BT$_{\text{static}}$ model. Note that the BT distribution φ_B is treated as an observed variable. Hyperparameter λ_B is a weighting factor that controls the general probability of the BT.

is a weighting factor that controls the general probability of the BT. The probability of a single word being assigned to one of the normal topics changes as following:

$$p(z_i = t|\text{rest}) \propto (1 - \lambda_B) \times \frac{n_{d,t} + \alpha_t}{n_d + |\alpha|} \frac{n_{t,v} + \beta_v}{n_t + |\beta|}, \quad \forall t \in [1, T] \tag{3.1}$$

The BT on the other hand is just dependent on λ_B and the static BT distribution φ_B:

$$p(z_i = t_B|\text{rest}) \propto \lambda_B \times \varphi_{B\,v}, \quad t_B = T + 1 \tag{3.2}$$

Notice that in equation 3.1 the document proportion $(n_{d,t} + \alpha_t)$ is divided by the normalization constant $(n_d + |\alpha|)$, which is omitted for the normal LDA model in equation 2.33. In this case however, it becomes necessary since $p(z_i = t_B|\text{rest})$ is independent of the

document size. This thesis will test the following four initializations for the static distribution φ_B:

$$B_{(tf)} = \left\{ \frac{tf(v)}{\sum_{v'=1}^{V} tf(v')} \right\}_{v=1}^{V} \tag{3.3}$$

$$B_{(df)} = \left\{ \frac{df(v)}{\sum_{v'=1}^{V} df(v')} \right\}_{v=1}^{V} \tag{3.4}$$

$$B_{(tf\times df)} = \left\{ \frac{tf(v) \times df(v)}{\sum_{v'=1}^{V} tf(v') \times df(v')} \right\}_{v=1}^{V} \tag{3.5}$$

$$B_{(ic)} = \left\{ \frac{-\log_2 p(v)}{\sum_{v'=1}^{V} -\log_2 p(v')} \right\}_{v=1}^{V}, \quad p(v) = \frac{tf(v)}{\sum_{v'=1}^{V} tf(v')} \tag{3.6}$$

Where $tf(v)$ and $df(v)$ are functions that return the term frequency and document frequency of term v respectively. The initializations $B_{(tf)}$, $B_{(df)}$ and $B_{(tf\times df)}$ are simply initialized proportional to the tf, df and $(tf \times df)$ of each term. $B_{(ic)}$ is initialized proportionally to the Information Content of each term, which is a measure also used in [43] (discussed in section 3.1.2).

3.2.2 Background Topic with individual Beta Prior

As mentioned in section 2.4, LDA already has a build in way to tweak the shape of topics and their distribution among documents by choosing asymmetric α and β priors. They can be used to shape a BT with the desired behavior. First, the probability of the BT is incorporated into the document-topic distribution θ_d. Hyperparameter α and θ_d are then of size $T + 1$. The pseudo count α_{t_B} can be increased while leaving the rest of the values symmetric, to increase the general probability of the BT. It is however not clear in which proportion α needs to be set in order to function properly.

Changing α might already be enough, nonetheless it would be beneficial to also have control over the pseudo counts of the BT. Choosing β as an asymmetric prior however is not sufficient since it affects all topics equally. Therefore a new hyperparameter β_B that is exclusive for the BT is introduced, which can be chosen asymmetric without affecting the other topics. The prior β for the normal topics is renamed to β_T. The resulting model will be called LDA-$\text{BT}_{\text{dynamic}}$ and is shown as a plate diagram in figure 3.2. The probability of a single word being assigned to one of the normal topics remains untouched, compared to the normal LDA model (equation 2.33):

$$p(z_i = t|\text{rest}) \propto (n_{d,t} + \alpha_t) \frac{n_{t,v} + \beta_{Tv}}{n_t + |\beta_T|}, \quad \forall t \in [1, T] \tag{3.7}$$

While the probability of t_B is just slightly different, as it has its own hyperparameter:

$$p(z_i = t_B|\text{rest}) \propto (n_{d,t_B} + \alpha_{t_B}) \frac{n_{t_B,v} + \beta_{Bv}}{n_t + |\beta_B|}, \quad t_B = T + 1 \tag{3.8}$$

What needs to be investigated is how the three hyperparameter α, β_T and β_B need to be chosen to increase model performance.

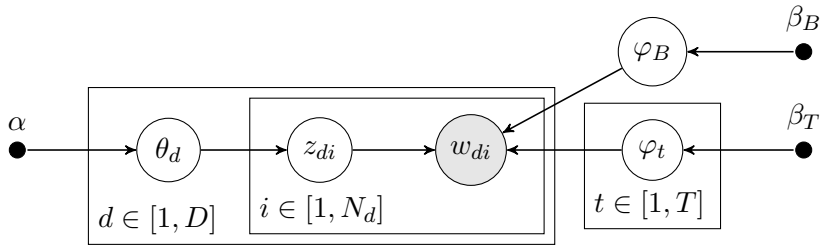


Figure 3.2: The plate diagram of the LDA-$\text{BT}_{\text{dynamic}}$ model. Note that the document-topic distribution θ_d and its prior α have a length of $T + 1$ to accommodate for the background topic φ_B. The topic-term distributions φ_t and φ_B, as well as their corresponding priors β_T and β_B are still of length V.

Chapter 4

Accelerate Gibbs sampling

Gibbs sampling is computationally expensive. When dealing with large corpora that span over thousands or even millions of documents, model inference can take hours or even days. With an increasing number of documents often also comes an increasing number of topics. Since a BT is used and high frequency stop-words are not removed, the models take even longer to compute than models trained on datasets without stop-words. Normal Gibbs sampling is bounded by the size of the corpus (total number of word tokens) times the number of topics. For each word token, the algorithm has to infer a new topic, so there is no other way than to touch each word token once per corpus sweep. However it is possible to accelerate the process of sampling a new topic for a single word token. Since the invention of LDA many different methods have been proposed to speed up the inference process of Gibbs sampling for LDA. Those include *SparseLDA*, *AliasLDA*, *LightLDA*, *WrapLDA* and *F+LDA*. All of those samplers are based on the observation that the update rule is sparse. By rearranging the update rule the algorithm can make use of that sparsity. Section 4.1 describes the SparseLDA sampler in detail while section 4.2 discusses the F+Tree LDA sampler. The following section 4.3 briefly discusses the *Metropolis Hastings* based approaches AliasLDA, LightLDA and WrapLDA. All samplers are further compared in section 4.4. Section 4.5 additionally describes the implementation of the used samplers in the programming language Java.

4.1 SparseLDA

SparseLDA [45] was the first algorithm that studied the sparsity of the LDA update equation. It first rearranges the LDA Gibbs update equation 2.33 in the following way:

$$p(z_i = t|\text{rest}) \propto \frac{\alpha_t \, \beta_v}{l} + \frac{n_{d,t} \, \beta_v}{l} + \frac{(n_{d,t} + \alpha_t) \, n_{t,v}}{l}, \quad l = n_t + |\beta| \tag{4.1}$$

The first term $\alpha_t \, \beta_v$ is constant for all documents. The second term $n_{d,t} \, \beta_v$ is independent of the current word w_i and becomes zero for every topic that is not assigned to at least

one word of the current document. The last term $(n_{d,t} + \alpha_t)\, n_{t,v}$ becomes zero for every topic that is never assigned to term v. The sum over all T topics of equation 4.1 can now be divided into three buckets:

$$s = \sum_{t=1}^{T} \frac{\alpha_t\, \beta_v}{l} \tag{4.2}$$

$$r = \sum_{t=1}^{T} \frac{n_{d,t}\, \beta_v}{l} \tag{4.3}$$

$$q = \sum_{t=1}^{T} \frac{(n_{d,t} + \alpha_t)}{l}\, n_{t,v} \tag{4.4}$$

The sum of all three buckets $(s + r + q)$ is equal to $\sum_{t=1}^{T} p(z_i = t|\text{rest})$, which is called the *normalizing constant*. The buckets can each be computed relatively efficiently:

- s does not change at all, as long as the hyperparameters α and β stay constant.
- r just depends on the current document. It needs to be calculated once each time the sampler starts at a new document. The decrement and increment step that happen for each word token can both be calculated by a single subtraction or addition.
- q changes for every word. However the first part $\sum_{t=1}^{T} \frac{(n_{d,t}+\alpha_t)}{l}$ can also be calculated once at the beginning of a new document for every topic (just like the r-bucket). It then takes a single multiplication for every topic where $n_{t,v} \neq 0$, to fully compute q. However to perform this step efficiently the algorithm needs to have a data structure that offers quick access to such topics. This is achieved by caching a list for each term v in the vocabulary, which is called $tlist_v$. It contains a tuple $(t, n_{t,v})$ for each topic t where $n_{t,v} \neq 0$. Those lists are precomputed once at the start of the algorithm and then get updated each time a new topic is assigned to a word token.

With the normalizing constant $(s+r+q)$ in place the algorithm can now sample a number $u \sim \mathcal{U}(0, s + r + q)$ uniformly between 0 and $(s + r + q)$. Depending on which bucket u falls in, the algorithm has to only search for a topic inside that bucket:

- If $u < s$ the s-bucket is hit. The algorithm then iterates over all topics, adding up $\frac{(\alpha_t\, \beta_v)}{l}$, until a value greater than u is reached.
- If $s < u < (s+r)$ the r-bucket is hit. In this case, the algorithm iterates over all the topics, subtracting $\frac{n_{d,t}\, \beta_v}{l}$ from u until $u < s$ is reached.
- If $u > (s + r)$ the q-bucket is hit. In this case, the algorithm has to only iterate over those topics where $n_{t,v} \neq 0$, which can be substantially less than T. It therefore iterates over the topics found in $tlist_v$, subtracting $\frac{(n_{d,t}+\alpha_t)}{l}\, n_{t,v}$ from u until $u < (s + r)$ is reached.

4.1.1 Incorporating the Background Topic

The approach for incorporating the BT is different for both newly developed models LDA-BT$_{\text{static}}$ and LDA-BT$_{\text{dynamic}}$. For the LDA-BT$_{\text{static}}$ model it is possible to treat the probability of the background topic given the current term v as a fourth bucket. The probability for the BT is defined as $p(z_i = t_B|\text{rest}) \propto \lambda_B \times \varphi_{B\,v}$ (equation 3.2) while the probability for all other topics is multiplied by $(1 - \lambda_B)$ and normalized by $(n_d + |\alpha|)$ (equation 3.1). However SparseLDA is designed to work with the normal LDA update rule defined in equation 2.33. In order to keep this behavior, the probability for the BT can be changed to $p(z_i = t_B|\text{rest}) \propto \frac{\lambda_B}{1-\lambda_B} \times \varphi_{B\,v} \times (n_d + |\alpha|)$, which makes it possible to leave the rest of the algorithm untouched. This new probability can be added onto the normalizing constant. If $u \sim \mathcal{U}(0, s + r + q + \frac{\lambda_B}{1-\lambda_B} \times \varphi_{B\,v} \times (n_d + |\alpha|))$ is bigger than $(s + r + q)$, the BT-bucket is hit.

Another problem is that the original SparseLDA algorithm does not consider an asymmetric β prior. When β is asymmetric then the s-bucket and r-bucket need to be updated for each new word token. This is achieved by multiplying β_v to the s-bucket and r-bucket before the new topic assignment is sampled. Afterwards the buckets are dividing by β_v again. Algorithm 3 shows the full SparseLDA sampling for the LDA-BT$_{\text{static}}$ model in detail.

For the LDA-BT$_{\text{dynamic}}$ model, there is no need to change the normalization constant. With this model it is just a matter of inserting the β_B prior, whenever the current topic is $t = T+1$. The β_T prior is used in any other case. The full algorithm for the LDA-BT$_{\text{dynamic}}$ model is shown in algorithm 4.

4.1.2 Further Improvements by Sorting

To make the sampling process inside the q-bucket even faster, the authors of [45] propose to sort each list $tlist_v$ in a descending order such that the topic with the highest $n_{t,v}$ gets accessed first. That way the algorithm should reach a topic, where $u < (s + r)$, faster. To achieve this, all precomputed lists get sorted in a descending order before the algorithm starts sampling. When a word token gets assigned with a new topic, only two tuples change. Tuple $(t^{(old)}, n_{t,v})$ changes to $(t^{(old)}, n_{t,v} - 1)$ and $(t^{(new)}, n_{t,v})$ changes to $(t^{(new)}, n_{t,v} + 1)$. Maintaining the sorted order after the updates is fast, since the rest of the list is already sorted. The authors further propose to encode a tuple $(t, n_{t,v})$ into a single integer, such that a higher $n_{t,v}$ results in a higher integer value. This way, the encoded list $tlist_v$ is memory efficient and can also be sorted with standard sorting algorithms.

Input : Corpus $\mathbf{w}$, Hyperparameter $(\alpha, \beta, \lambda_B)$, Static BT b, Iterations M
Output : $\Theta, \Phi, \mathbf{z}$

Choose a random topic assignment for each word token $\mathbf{z} = \{\{z_{di}\}_{i=1}^{n_d}\}_{d=1}^{D}$
Initialize count matrices $n_{d,t}$, $n_{t,v}$ and n_t according to $\mathbf{z}$
Create a list $tlist_v$ for each term v with $\{t \in [1,T] \mid n_{t,v} \neq 0\}$
$s_{\text{sum}} \leftarrow \sum_{t=1}^{T} \frac{\alpha_t}{n_t + |\beta|}$
for M **do**
  for $d = 1$ to D **do**
    $r_{\text{sum}} \leftarrow \sum_{t=1}^{T} \frac{n_{d,t}}{n_t + |\beta|}$, $q \leftarrow \left\{ \frac{n_{d,t}\,\alpha_t}{n_t + |\beta|} \right\}_{t=1}^{T}$
    for $i = 1$ to n_d **do**
      $t \leftarrow z_{di}$, $v \leftarrow w_{di}$
      if $t \neq T + 1$ **then**
        $q(t) \leftarrow \frac{n_{d,t}\,\alpha_t}{n_t + |\beta| - 1}$
        $s_{\text{sum}} \leftarrow (s_{\text{sum}} - \frac{\alpha_t}{n_t + |\beta|} + \frac{\alpha_t}{n_t + |\beta| - 1}) \times \beta_v$
        $r_{\text{sum}} \leftarrow (r_{\text{sum}} - \frac{n_{d,t}}{n_t + |\beta|} + \frac{n_{d,t}}{n_t + |\beta| - 1}) \times \beta_v$
      `decrementCounts(`$n_{d,t}$ `,` $n_{t,v}$ `,` n_t`)`
      $tlist_v$`.update(`t`,` $n_{t,v}$`)`
      $q_{\text{sum}} \leftarrow 0$
      for topic t' in $tlist_v$ **do**
        $q_{\text{sum}} \leftarrow q_{\text{sum}} + q(t') \times n_{t',v}$
      $b_{\text{sum}} \leftarrow b(v) \times (n_d + |\alpha| - 1) \times \frac{\lambda_B}{1 - \lambda_B}$
      $u \leftarrow$`randomBetween(0, 1)`$\times (s_{\text{sum}} + r_{\text{sum}} + q_{\text{sum}} + b_{\text{sum}})$
      if $u < s_{sum}$ **then**
        for $t = 1;\ t \geq T;\ t = t + 1$ **do**
          $u \leftarrow u - \frac{\alpha_t \beta_v}{n_t + |\beta|}$
          if $u \leq 0$ **then break**
      else if $u < (s_{\text{sum}} + r_{\text{sum}})$ **then**
        for $t = 1;\ t \geq T;\ t = t + 1$ **do**
          $u \leftarrow u - \frac{n_{d,t}\,\beta_v}{n_t + |\beta|}$
          if $u \leq s_{sum}$ **then break**
      else if $u \leq (s_{\text{sum}} + r_{\text{sum}} + q_{\text{sum}})$ **then**
        for topic t in $tlist_v$ **do**
          $u \leftarrow u - q(t) \times n_{t,v}$
          if $u \leq (s_{sum} + r_{sum})$ **then break**
      else $t \leftarrow T + 1$
      $z_{di} \leftarrow t$
      `incrementCounts(`$n_{d,t}$ `,` $n_{t,v}$ `,` n_t`)`
      $tlist_v$`.update(`t`,` $n_{t,v}$`)`
      if $t \neq T + 1$ **then**
        $q(t) \leftarrow \frac{n_{d,t}\,\alpha_t}{n_t + |\beta|}$
        $s_{\text{sum}} \leftarrow \frac{s_{\text{sum}}}{\beta_v} - \frac{\alpha_t}{n_t + |\beta| - 1} + \frac{\alpha_t}{n_t + |\beta|}$
        $r_{\text{sum}} \leftarrow \frac{r_{\text{sum}}}{\beta_v} - \frac{n_{d,t}}{n_t + |\beta| - 1} + \frac{n_{d,t}}{n_t + |\beta|}$
$\Theta \leftarrow$ compute from $\mathbf{z}$ according to equation 2.12
$\Phi \leftarrow$ compute from $\mathbf{z}$ according to equation 2.13
return $(\Theta, \Phi, \mathbf{z})$

decrement counts and therefore update $q(t)$, s_{sum}, r_{sum} and $tlist_v$

build up q_{sum} by using only the topics in $tlist_v$

decide in which bucket u falls in and sample from it

increment counts and therefore update $q(t)$, s_{sum}, r_{sum} and $tlist_v$

Algorithm 3 : The SparseLDA sampling algorithm for the LDA-BT$_{\text{static}}$ model.

Input : Corpus $\mathbf{w}$, Hyperparameter $(\alpha, \beta_T, \beta_B)$, Iterations M
Output : $\Theta, \Phi, \mathbf{z}$

Choose a random topic assignment for each word token $\mathbf{z} = \{\{z_{di}\}_{i=1}^{n_d}\}_{d=1}^{D}$
Initialize count matrices $n_{d,t}$, $n_{t,v}$ and n_t according to $\mathbf{z}$
Create a list $tlist_v$ for each term v with $\{t \in [1,T] \mid n_{t,v} \neq 0\}$
Define $betaSum(t) = \begin{cases} |\beta_T| & \text{if } t \leq T \\ |\beta_B| & \text{else} \end{cases}$ and $betaV(t,v) = \begin{cases} \beta_{T\,v} & \text{if } t \leq T \\ \beta_{B\,v} & \text{else} \end{cases}$
$s_{\text{sum}} \leftarrow \sum_{t=1}^{T+1} \frac{\alpha_t}{n_t + betaSum(t)}$
for M **do**
  for $d = 1$ to D **do**
    $r_{\text{sum}} \leftarrow \sum_{t=1}^{T+1} \frac{n_{d,t}}{n_t + betaSum(t)}$, $q \leftarrow \left\{ \frac{n_{d,t}\,\alpha_t}{n_t + betaSum(t)} \right\}_{t=1}^{T+1}$
    for $i = 1$ to n_d **do**
      $t \leftarrow z_{di}$, $v \leftarrow w_{di}$
      $q(t) \leftarrow \frac{n_{d,t}\,\alpha_t}{n_t + betaSum(t) - 1}$
      $s_{\text{sum}} \leftarrow (s_{\text{sum}} - \frac{\alpha_t}{n_t + betaSum(t)} + \frac{\alpha_t}{n_t + betaSum(t)-1}) \times betaV(t,v)$
      $r_{\text{sum}} \leftarrow (r_{\text{sum}} - \frac{n_{d,t}}{n_t + betaSum(t)} + \frac{n_{d,t}}{n_t + betaSum(t)-1}) \times betaV(t,v)$
      `decrementCounts(`$n_{d,t}$ `,` $n_{t,v}$ `,` n_t`)`
      $tlist_v$`.update(`t`,` $n_{t,v}$`)`
      $q_{\text{sum}} \leftarrow 0$
      for topic t' in $tlist_v$ **do**
        $q_{\text{sum}} \leftarrow q_{\text{sum}} + q(t') \times n_{t',v}$
      $u \leftarrow$ `randomBetween(0, 1)` $\times (s_{\text{sum}} + r_{\text{sum}} + q_{\text{sum}})$
      if $u < s_{sum}$ **then**
        for $t = 1;\ t \geq T+1;\ t = t+1$ **do**
          $u \leftarrow u - \frac{\alpha_t \times betaV(t,v)}{n_t + betaSum(t)}$
          if $u \leq 0$ **then break**
      else if $u < (s_{\text{sum}} + r_{\text{sum}})$ **then**
        for $t = 1;\ t \geq T+1;\ t = t+1$ **do**
          $u \leftarrow u - \frac{n_{d,t} \times betaV(t,v)}{n_t + betaSum(t)}$
          if $u \leq s_{sum}$ **then break**
      else
        for topic t in $tlist_v$ **do**
          $u \leftarrow u - q(t) \times n_{t,v}$
          if $u \leq (s_{sum} + r_{sum})$ **then break**
      $z_{di} \leftarrow t$
      `incrementCounts(`$n_{d,t}$ `,` $n_{t,v}$ `,` n_t`)`
      $tlist_v$`.update(`t`,` $n_{t,v}$`)`
      $q(t) \leftarrow \frac{n_{d,t}\,\alpha_t}{n_t + betaSum(t)}$
      $s_{\text{sum}} \leftarrow \frac{s_{\text{sum}}}{\beta_v} - \frac{\alpha_t}{n_t + betaSum(t)-1} + \frac{\alpha_t}{n_t + betaSum(t)}$
      $r_{\text{sum}} \leftarrow \frac{r_{\text{sum}}}{\beta_v} - \frac{n_{d,t}}{n_t + betaSum(t)-1} + \frac{n_{d,t}}{n_t + betaSum(t)}$
$\Theta \leftarrow$ compute from $\mathbf{z}$ according to equation 2.12
$\Phi \leftarrow$ compute from $\mathbf{z}$ according to equation 2.13
return $(\Theta, \Phi, \mathbf{z})$

decrement counts and therefore update $q(t)$, s_{sum}, r_{sum} and $tlist_v$

build up q_{sum} by using only the topics in $tlist_v$

decide in which bucket u falls in and sample from it

increment counts and therefore update $q(t)$, s_{sum}, r_{sum} and $tlist_v$

Algorithm 4 : The SparseLDA sampling algorithm for the LDA-BT$_{\text{dynamic}}$ model.

4.1.3 Sampling Complexity

With the above discussed techniques SparseLDA reduces the sampling cost for a single word token to $\mathcal{O}(T_d + T_v)$ (compared to $\mathcal{O}(T)$ when using the normal Gibbs sampling process). Here T_d represents the number of actually instantiated topics in document d, while T_v represents the number of topics occurring for a particular term v. This complexity is achieved because the sampler only iterates over topics where $n_{t,v} \neq 0$ or $n_{d,t} \neq 0$ to find a new topic assignment. Whenever T is larger than the average amount of words per document, it is obvious that T_d is smaller than T. Furthermore T_d should get even smaller with increasing likelihood of the model since a document will be mostly described by just a few topics. For a relatively small corpus it can be assumed that T_v is also smaller than T. Likewise T_v should also get even smaller with increasing likelihood of the model. Therefore the assumption $T_d + T_v \ll T$ is well justified, for a relatively small corpus. However as shown in [25], for a massive dataset with millions of documents T_v approaches T, which diminishes most of the performance gains achieved by SparseLDA.

4.2 F+LDA

F+LDA [46] uses the same decomposition of the Gibbs sampler update rule as AliasLDA [25]:

$$p(z_i = t|\text{rest}) \propto \alpha_t \times q + n_{d,t} \times q\,, \quad q = \frac{n_{t,v} + \beta_v}{n_t + |\beta|} \tag{4.5}$$

This equation can be divided into two buckets, while summing over all T topics:

$$s = \sum_{t=1}^{T} \alpha_t \times q \tag{4.6}$$

$$r = \sum_{t=1}^{T} n_{d,t} \times q \tag{4.7}$$

The sum $(s + r)$ is equal to $\sum_{t=1}^{T} p(z_i = t|\text{rest})$, which is the normalizing constant. Both buckets need to be computed before a new term assignment for a word can be sampled:

- s is reused multiple times by AliasLDA without updating it, while F+LDA uses a data structure called *F+Tree* which updates each iteration. A F+Tree can be constructed in $\mathcal{O}(T + \log T)$ and enables to sample from q in $\mathcal{O}(\log T)$. Additionally a single element can be updated in $\mathcal{O}(\log T)$ and its full probability sum can be retrieved in $\mathcal{O}(1)$. Figure 4.1 shows an example on how a F+Tree looks like, how it is updated and how it is used for sampling. Another beneficial property of a F+Tree is that it can be stored in a single array with size $2T - 1$. To utilize the F+Tree in an

efficient way it is necessary to iterate through the corpus in a term by term manner instead of a document by document manner. [1]

- r is dependent on $n_{d,t}$ and $n_{t,v}$ and therefore dense. However, it only needs to be considered for those topics where $n_{d,t} \neq 0$ in the current document d. Similar to SparseLDA, a list for each document d called $tlist_d$ which stores the topics where $n_{d,t} \neq 0$ for d, can be used. Such lists can be maintained while incrementing and decrementing the count matrices. Furthermore, while iteratively computing r, the algorithm can also build up a list $\text{cumsum}(r) = \{\sum_{t=1}^{t'} n_{d,t} \times q\}_{t'=1}^{T}$. This list contains for each topic t' the cumulative sum of all topics that come before it and will be useful later in the sampling process.

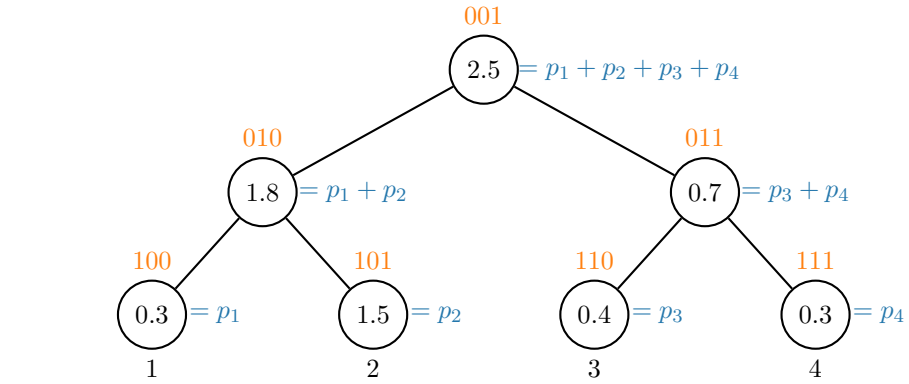
The general approach is to construct a new F+Tree for q each time the sampler arrives at a new term. As long as the term does not change, the F+Tree can be updated in $\mathcal{O}(\log T)$ for each word token. With the normalizing constant $(s + r)$ in place the algorithm then samples a number $u \sim \mathcal{U}(0, s + r)$ uniformly between 0 and $(s + r)$. Depending on which bucket u falls in, the corresponding topic can be derived from this bucket:

- If $u < s$ then u has hit the s-bucket. The topic can then be sampled from the F+Tree in $\mathcal{O}(\log T)$ like shown in figure 4.1(b).
- If $s < u$ then u has hit the r-bucket. Since the algorithm has already build up the cumulative sum for each topic $\text{cumsum}(r)$, the algorithm can find the smallest topic t' that satisfies $(\sum_{t=1}^{t'} n_{d,t} \times q) > u$ in $\mathcal{O}(\log T)$, by using a binary search.

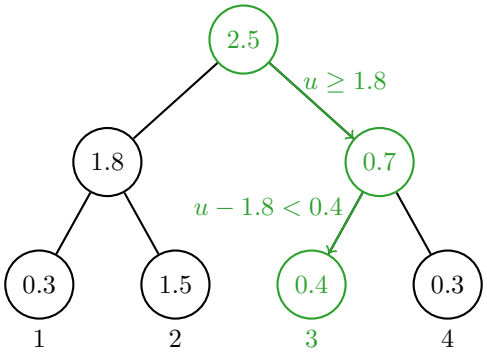
4.2.1 Incorporating the Background Topic

Incorporating the BT into the F+Tree LDA sampler is very similar to incorporating it into SparseLDA (described in section 4.1.1). For the LDA-BT$_{\text{static}}$ model the normalization constant is modified. That way, a sample $u \sim \mathcal{U}(0, s + r + \frac{\lambda_B}{1-\lambda_B} \times \varphi_{B\,v} \times (n_d + |\alpha|))$ can be created. If u is bigger than $(s + r)$, the BT-bucket is hit.
In addition, the original F+Tree sampler does not consider an asymmetric α prior. In the original paper [46] the authors propose that the F+Tree is constructed using just q and the s-bucket is computed by multiplying the symmetric α prior with the probability sum of this F+Tree. However, to integrate an asymmetric α prior the F+Tree needs to be constructed using $\alpha_t \times q$. In this case the probability sum of the F+Tree is already the full s-bucket. The full F+Tree sampler for the LDA-BT$_{\text{static}}$ model is shown in algorithm 5.

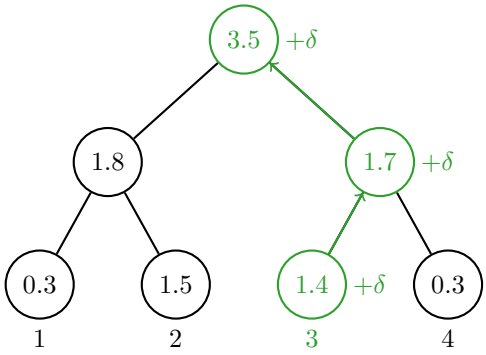
[1] The authors of F+LDA also propose the decomposition $p(z_i = t|\text{rest}) \propto \beta_v \times q + n_{t,v} \times q\,, \quad q = \frac{n_{d,t}+\alpha_t}{n_t+|\beta|}$ where the algorithm would iterate through the corpus in a document by document fashion. However, then the F+Tree has to be recomputed after each document instead of each term. Their experiments show that usually the average occurrences per term are higher than the average document length, which yields to better performance when using the term by term technique.



(a) F+Tree for $p = \{0.3, 1.5, 0.4, 0.3\}$. The orange text represents the binary index of each node, while the blue shows how the node values are calculated.



(b) Sample from the F+Tree with $u = 2.1$.



(c) Update probability p_3 with $\delta = 1.0$.

Figure 4.1: An illustration of how a F+Tree looks like, how it can sampled with logarithmic complexity and how it can be updated with logarithmic complexity. This example is mostly copied from the original F+Tree paper [46], where also the corresponding pseudo code for constructing, sampling and updating the F+Tree can be found. The whole F+Tree for a probability distribution with size T can be stored in a single array with size $2T - 1$.

Input : Corpus $\mathbf{w}$, Hyperparameter $(\alpha, \beta, \lambda_B)$, Static BT b, Iterations M
Output : $\Theta, \Phi, \mathbf{z}$

Choose a random topic assignment for each word token $\mathbf{z} = \{\{z_{di}\}_{i=1}^{n_d}\}_{d=1}^{D}$
Initialize count matrices $n_{d,t}$, $n_{t,v}$ and n_t according to $\mathbf{z}$
Create a list $tlist_d$ for each document d with $\{t \in [1,T] \mid n_{d,t} \neq 0\}$
Create a list $occurrencelist_v$ for each term v with $\{w_{di} \in \mathbf{w} \mid w_{di} = v\}$
Allocate memory for arrays $r \leftarrow \{0\}_{t=1}^{T}$, $q \leftarrow \{0\}_{t=1}^{T}$ and F+Tree f
for M **do**
  for $v = 1$ to V **do**
    for $t = 1$ to T **do**
      `updateSBucket(`f`,` q`,` t`,` v`)`
    } build up s-bucket for term v by updating q and f
    for w_{di} in $occurrencelist_v$ **do**
      $t \leftarrow z_{di}$
      `decrementCounts(`$n_{d,t}$ `,` $n_{t,v}$ `,` n_t`)`
      if $n_{d,t} = 0$ **then** $tlist_d$`.removeTopic(`t`)`
      `updateSBucket(`f`,` q`,` t`,` v`)`
      } decrement counts and therefore update $tlist_d$, q and f
      $r_{\text{sum}} \leftarrow 0$
      for index i, topic t' in $tlist_d$ **do**
        $r_{\text{sum}} \leftarrow r_{\text{sum}} + n_{d,t'} \times q(t')$
        $r(i) \leftarrow r_{\text{sum}}$
      } build up r-bucket by using only the topics in $tlist_d$
      $b_{\text{sum}} \leftarrow b(v) \times (n_d + |\alpha| - 1) \times \frac{\lambda_B}{1-\lambda_B}$
      $u \leftarrow$`randomBetween(0, 1)`$\times(r_{\text{sum}} + f$`.getSum()` $+ b_{\text{sum}})$
      if $u < p_{sum}$ **then**
        $t \leftarrow tlist_d$`(lowerBound(`r`,` $tlist_d$`.size(),` u`))`
      else if $u < (r_{\text{sum}} + f$`.getSum()`$)$ **then**
        $t \leftarrow f$`.sample(`$u - r_{\text{sum}}$`)`
      else $t \leftarrow T + 1$
      } decide in which bucket u falls in and sample from it
      $z_{di} \leftarrow t$
      `incrementCounts(`$n_{d,t}$ `,` $n_{t,v}$ `,` n_t`)`
      if $n_{d,t} = 1$ **then** $tlist_d$`.addTopic(`t`)`
      `updateSBucket(`f`,` q`,` t`,` v`)`
      } increment counts and therefore update $tlist_d$, q and f
$\Theta \leftarrow$ compute from $\mathbf{z}$ according to equation 2.12
$\Phi \leftarrow$ compute from $\mathbf{z}$ according to equation 2.13
return $(\Theta, \Phi, \mathbf{z})$

Function `updateSBucket(F+Tree` f`, Array` q`, Topic` t`, Term` v`):`
  if $t \neq T + 1$ **then**
    $q(t) \leftarrow \frac{n_{t,v}+\beta_v}{n_t+|\beta|}$
    f`.update(`t`,` $\alpha_t \times q(t)$`)`

Algorithm 5 : The F+LDA sampling algorithm for the LDA-BT$_{\text{static}}$ model.

Input : Corpus $\mathbf{w}$, Hyperparameter $(\alpha, \beta_T, \beta_B)$, Iterations M
Output : $\Theta, \Phi, \mathbf{z}$

Choose a random topic assignment for each word token $\mathbf{z} = \{\{z_{di}\}_{i=1}^{n_d}\}_{d=1}^{D}$
Initialize count matrices $n_{d,t}$, $n_{t,v}$ and n_t according to $\mathbf{z}$
Create a list $tlist_d$ for each document d with $\{t \in [1, T+1] \mid n_{d,t} \neq 0\}$
Create a list $occurrencelist_v$ for each term v with $\{w_{di} \in \mathbf{w} \mid w_{di} = v\}$
Allocate memory for arrays $r \leftarrow \{0\}_{t=1}^{T+1}$, $q \leftarrow \{0\}_{t=1}^{T+1}$ and F+Tree f
for M **do**
  for $v = 1$ to V **do**
    for $t = 1$ to $T+1$ **do**
      `updateSBucket(`f`, `q`, `t`, `v`)`
    } build up s-bucket for term v by updating q and f
    for w_{di} in $occurrencelist_v$ **do**
      $t \leftarrow z_{di}$
      `decrementCounts(`$n_{d,t}$`, `$n_{t,v}$`, `n_t`)`
      if $n_{d,t} = 0$ **then** $tlist_d$`.removeTopic(`t`)`
      `updateSBucket(`f`, `q`, `t`, `v`)`
      } decrement counts and therefore update $tlist_d$, q and f
      $r_{\text{sum}} \leftarrow 0$
      for index i, topic t' in $tlist_d$ **do**
        $r_{\text{sum}} \leftarrow r_{\text{sum}} + n_{d,t'} \times q(t')$
        $r(i) \leftarrow r_{\text{sum}}$
      } build up r-bucket by using only the topics in $tlist_d$
      $u \leftarrow$`randomBetween(0, 1)`$\times(r_{\text{sum}} + f$`.getSum()`$)$
      if $u < p_{sum}$ **then**
        $t \leftarrow tlist_d$`(lowerBound(`r`, `$tlist_d$`.size(), `u`))`
      else
        $t \leftarrow f$`.sample(`$u - r_{\text{sum}}$`)`
      } decide in which bucket u falls in and sample from it
      $z_{di} \leftarrow t$
      `incrementCounts(`$n_{d,t}$`, `$n_{t,v}$`, `n_t`)`
      if $n_{d,t} = 1$ **then** $tlist_d$`.addTopic(`t`)`
      `updateSBucket(`f`, `q`, `t`, `v`)`
      } increment counts and therefore update $tlist_d$, q and f
$\Theta \leftarrow$ compute from $\mathbf{z}$ according to equation 2.12
$\Phi \leftarrow$ compute from $\mathbf{z}$ according to equation 2.13
return $(\Theta, \Phi, \mathbf{z})$

Function `updateSBucket(F+Tree` f`, Array` q`, Topic` t`, Term` v`):`
  if $t \neq T+1$ **then** $q(t) \leftarrow \frac{n_{t,v}+\beta_{Tv}}{n_t+|\beta_T|}$
  else $q(t) \leftarrow \frac{n_{t,v}+\beta_{Bv}}{n_t+|\beta_B|}$
  f`.update(`t`, `$\alpha_t \times q(t)$`)`

Algorithm 6 : The F+LDA sampling algorithm for the LDA-BT$_{\text{dynamic}}$ model.

To construct an F+Tree sampler for the LDA-BT$_{\text{dynamic}}$ model, it is just a matter of inserting the β_B prior whenever the current topic is $t = T + 1$. The full LDA-BT$_{\text{dynamic}}$ F+Tree sampler is shown in algorithm 6.

4.2.2 Sampling Complexity

The amortized complexity of F+Tree LDA for a single word token is $\mathcal{O}(T_d + \log T)$, where T_d represents the number of topics with $n_{d,t} \neq 0$ for a given document d. This is due to the fact that computing the r bucket takes $\mathcal{O}(T_d)$ multiplications. Updating the F+Tree and sampling from s or r is done in $\mathcal{O}(\log T)$ complexity. The construction of a new F+Tree only happens when the sampler arrives at a new term which is relatively fast ($\mathcal{O}(T+\log T)$).

4.3 Metropolis Hastings based Samplers

AliasLDA [25] uses equation 4.5 as the decomposition of the Gibbs sampler update rule, which is the same as F+LDA. LightLDA [47] is inspired by AliasLDA, however it uses yet another decomposition where the algorithm alternates between two different proposal functions. WrapLDA [9] uses the exact same sampling method as LightLDA but in a more optimized way, such that it greatly reduces random memory access.
Where in section 4.2 an F+Tree is used for the s-bucket, AliasLDA uses a so called *Alias Table*. This data structure can be sampled in $\mathcal{O}(1)$, however, the update complexity is way more costly ($\mathcal{O}(T)$) compared to an F+Tree. The idea therefore is to not update the alias table at all and instead just using it T times, before creating a new one. To handle the resulting error, a *Metropolis Hastings* algorithm is used that accepts or rejects the proposals created by the sampling algorithm. LightLDA (and therefore also WrapLDA) also relies on alias tables in the same way. In this thesis the BT is however not integrated into the before mentioned samplers since it is unclear how the acceptance of the Metropolis Hastings step need to be modified.

4.4 Sampler Comparison

The paper [48] gives a comparison between the sampling processes of AliasLDA, F+LDA, LightLDA, and WarpLDA. These are four of the currently best performing samplers, with emphasis on analyzing huge corpora with possibly millions of documents and thousands of topics. Each one has its own advantages and disadvantages. However, in practice it seems that F+LDA and WrapLDA perform the best. The Metropolis Hastings based WarpLDA incurs an $\mathcal{O}(1)$ sampling complexity per word. However since it relies on proposals with acceptance rate π, only $\frac{1}{\pi}$ of the proposals get accepted. Furthermore, constructing an alias table is more expensive than constructing an F+Tree. F+LDA on the other hand has an $\mathcal{O}(T_d + \log T)$ sampling complexity per word, but no rejection rate. Which of the both

samplers performs best is therefore mostly dependent on the average document length. On a dataset with long documents, the Metropolis Hastings based WarpLDA is faster. In most other scenarios, F+LDA will be faster.
When analyzing very large corpora that span over millions of documents, the discussed algorithms usually do not fit into main memory. They therefore have to rely on storage media to access corpora information and count matrices. To further accelerate the sampling process, the workload is usually spread across multiple machines. Modern implementations often utilize a so called *Parameter Server Framework* which allows them to distribute the workload to multiple machines while maintaining globally shared parameters. At a large scale, performance of a sampler is therefore also dependent on how data is accessed and how well it can be parallelized on multiple machines. To evaluate how the implemented models perform on really big datasets is out of scope for this thesis. However since the discussed samplers are just slightly modified, without adding much complexity, it is expected that all samplers will still behave like discussed in [48].

4.5 Implementation

Popular LDA implementations include *Gensim* [33] (implemented with a mixture of *C* and *Python*), *Mallet LDA* [27] (implemented with *Java*) and *R LDA* [6] (implemented with a mixture of *C* and *R*). Another popular choice for implementing LDA models is *C++*. All models, that are discussed in this thesis, are implemented using Java, which offers acceptable performance and runs on a wide variety of modern operating systems. All important data structures such as the topic assignments $\mathbf{z}$ and the count matrices $n_{t,v}$, $n_{d,v}$ are represented using primitive data types and fixed size two dimensional arrays. If possible, more complex data structure like *List<>* are avoided to minimize memory footprint and to reduce unnecessary memory allocations in the sampling process. The Gibbs sampling process runs on a single thread. However, it is of course possible to run several models simultaneously on separate threads. The implemented program runs completely in main memory, which ensures the best possible execution time. This approach is however impracticable for very big datasets where the machine that executes the program can quickly run out of main memory. With the dataset tested in this thesis however, a single instance does not use more than roughly 2 GB of main memory.
Additionally, a program called *JPype* [2] is used to provide a link between Python and Java. This way parameters can easily be shared between the two programming languages, which enables the user to utilize the computing power of Java while retaining the ease of use of Python.

Chapter 5

Experiments

This section describes the experiments carried out to test the, in section 3.2 discussed, modified LDA models. At first, the two used datasets are described in section 5.1. In section 5.2 results of the normal LDA model are shown as a baseline. Afterwards the results of the LDA-BT_{static} and LDA-$BT_{dynamic}$ models are discussed in section 5.3 and 5.4. Both modified LDA models are further compared in section 5.5, by exploring the learned BT probabilities of two well performing models.

5.1 Used Datasets

This section describes the 20Newsgroups dataset as well as the FAZ dataset in detail. Both datasets are split into two different versions, which yields a total of four datasets.

5.1.1 The 20Newsgroups Dataset

The 20Newsgroups dataset [1] is a collection of $\sim$ 11000 documents, which are evenly distributed across 20 different newsgroups. Each document is a news article published by one of the 20 newsgroups. The newsgroups spread across subjects like *computer hardware*, *sports*, *religion* and *politics*. Some newsgroups are closely related while other are unrelated. To use the dataset in this thesis some preprocessing is done. All words are converted to lowercase. All non alphanumeric characters and all numbers are removed. Lastly, terms that appear less then 5 times in the whole dataset are removed. Stop-words are identified using the openly available *Natural Language Toolkit (NLTK)* [3] stop-word corpus. To get better insights on how the different models handle stop-words two different version of the 20Newsgroups dataset are considered. One includes all words, which is called $20N_{all}$ and one that excludes stop-words called $20N_{nsw}$. Table 5.1 shows additional information about the total number of words and stop-words. Further, figure 5.1 presents the label frequency among all documents as well as the tf and df of the most common terms.

D	V	W_{all}	W_{sw}	W_{sw}/W_{all}
$\sim 11k$	$\sim 20k$	$\sim 2m$	$\sim 0.84m$	0.43

Table 5.1: Additional information about the 20Newsgroups dataset. D represents the number of documents, V the number of terms in the vocabulary, W_{all} the the total number of word tokens and W_{sw} the number of word tokes that are classified as stop-words. It follows that the average document contains 180 word tokens, where about 77 of those are stop-words.

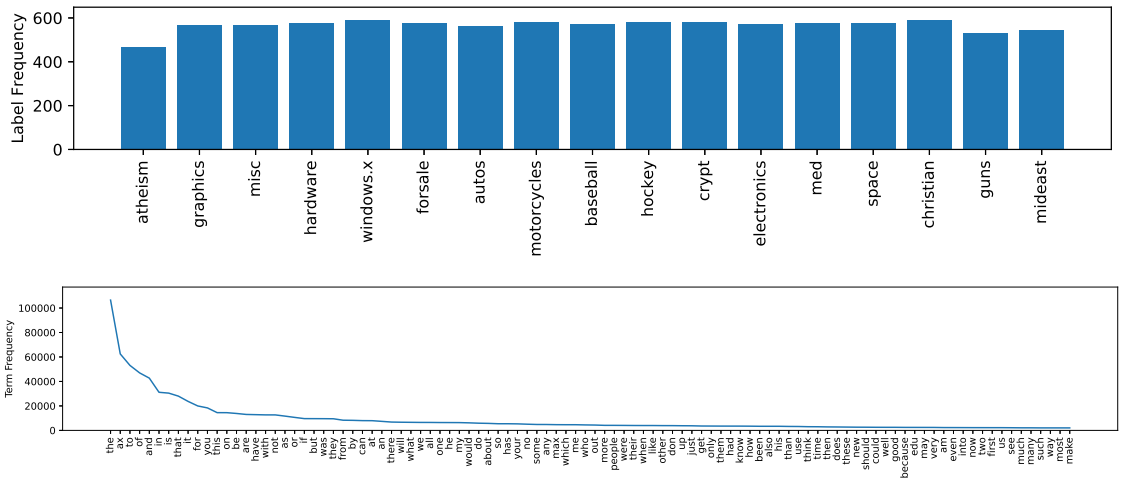


Figure 5.1: The bar chart shows the label frequency among all $\sim 11k$ documents. It reveals that the labels are distributed relatively evenly between the 20 newsgroups. The other graph shows the tf of the 100 most common terms. Term *the* alone accounts for about $\sim 5\%$ of all word tokens. As expected, both tf and df seem to follow a power law.

5.1.2 The FAZ Dataset

The FAZ dataset is a collection of new articles from the German newspaper Frankfurter Allgemeine Zeitung (FAZ). The used articles are all published between 01. September 2019 and 31. December 2019. Similar to 20Newsgroups all words are converted to lowercase and all non alphanumeric characters and all numbers are removed. Terms that appear less then 10 times in the whole dataset are removed too. Two versions of the dataset are created where FAZ_{nsw} contains no stop-words and FAZ_{all} contains all words. Stop-words are identified using the German NLTK stop-word corpus. Table 5.2 gives some details about the distribution of stop-words in the dataset. The original dataset also contains some additional information. Each document is annotated with one or multiple keywords that roughly describe its contents. To ovoid dealing with multi-label classification some further processing is done to convert the keyword assignments into a single-label clustering. First, each keyword that appears less than 10 times is filtered out. This leaves 43049 unique keywords where each document contains between 1 to 7 of them. Next, the keyword assignments of each document are converted into a 43049-dimensional vector that contains a 1 for each assigned keyword and a 0 otherwise. To find similarities between the resulting keyword-vectors a Jaccard distance matrix is created. Afterwards an agglomerative hierarchical clustering algorithm with complete linkage is executed. All keyword vectors with a maximum inter-cluster distance smaller than 0.99 are then assigned to the same cluster. This leaves 282 unique cluster which can be used to measure cluster performance. Figure 5.2 shows some most common terms.

D	V	W_{all}	W_{sw}	W_{sw}/W_{all}
$\sim 13k$	$\sim 33k$	$\sim 5.2m$	$\sim 2.5m$	0.48

Table 5.2: Additional information about the FAZ dataset. The average document contains 409 word tokens where about 196 of those are stop-words.

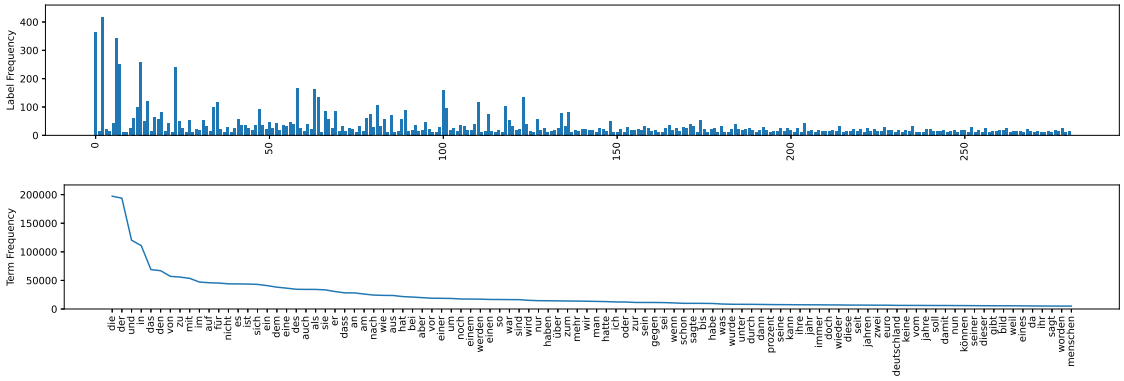


Figure 5.2: The bar chart shows the label frequency among all documents. The graph shows the tf of the 100 most common terms. As expected, both tf and df seem to follow a power law.

5.2 Baseline with Normal LDA

First the normal LDA model is tested to get a baseline score. The hyperparameter are chosen as symmetric priors $\alpha = \langle(\frac{50}{T})_{\times T}\rangle$ and $\beta = \langle 0.1_{\times V}\rangle$ (recall the discussion about choosing priors in section 2.4.1). Table 5.3 shows the average performance of the normal LDA model on all four datasets. Each model that is trained on $20N_{all}$ and $20N_{nsw}$ is executed 5 times for 500 iterations, while each model trained on FAZ_{all} and FAZ_{nsw} is executed 3 times for 500 iterations. This will be the same for all in this thesis discussed results.

As expected the model does score better on the datasets, where stop-words are removed. Figure 5.3 shows the total assignments of word tokens per topic on the $20N_{all}$ dataset. While a few topics contain very little stop-words, most of them contain a lot, which results in a worse clustering performance.

	ARI	*NMI*	C_v
$20N_{all}$	0.210 (±.004)	0.338 (±.006)	0.523 (±.003)
$20N_{nsw}$	0.262 (±.01)	0.393 (±.009)	0.732 (±.003)
FAZ_{all}	0.017 (±.001)	0.169 (±.003)	0.607 (±.003)
FAZ_{nsw}	0.026 (±.001)	0.272 (±.001)	0.71 (±.001)

Table 5.3: Average cluster performance (and standard deviation) of the normal LDA model on all datasets with symmetric priors $\alpha = \langle(\frac{50}{T})_{\times T}\rangle$, $\beta = \langle 0.1_{\times V}\rangle$.

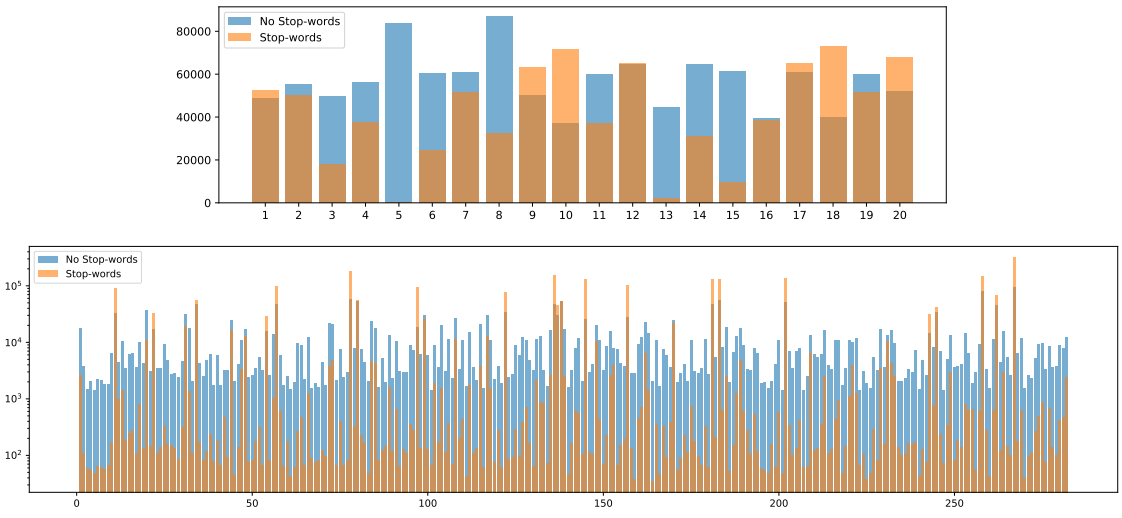


Figure 5.3: Total assignments per topic on the $20N_{all}$ and FAZ_{all} datasets with symmetric priors $\alpha = \langle(\frac{50}{T})_{\times T}\rangle$, $\beta = \langle 0.1_{\times V}\rangle$, using the normal LDA model. Notice that the second graph (FAZ_{all}) has a logarithmic scale on the y-axis.

5.3 Static Background Topic

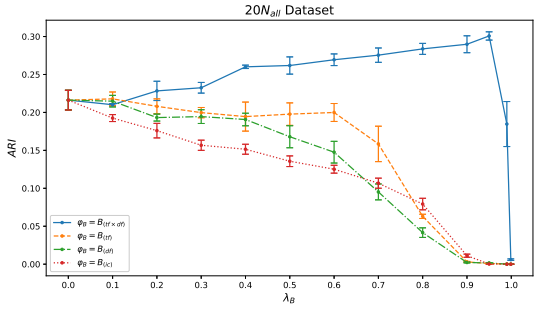
To test how the LDA-BT$_{\text{static}}$ model (defined in section 3.2.1) performs, it is run on all datasets with different parameters. The static distribution φ_B is initialized with either one of the four (in equations 3.3 to 3.6 described) methods, while λ_B is altered between 0 and 1.

Figure 5.4 shows the cluster performance for the different BT initializations with a varying λ_B value. The results indicate that a initialization with $\varphi_B = B_{(tf \times df)}$ improves the scores significantly, on both 20N$_{all}$ and FAZ$_{all}$, with an increasing λ_B value. The results constantly improve until a value of $\lambda_B = 0.95$ is reached. On FAZ$_{all}$ initializations with $\varphi_B = B_{(tf)}$ and $\varphi_B = B_{(tf)}$ do also improve results, although with a smaller λ_B value. The best observed scores for 20N$_{all}$ and FAZ$_{all}$ slightly outperform the best baseline results, established in section 5.2.

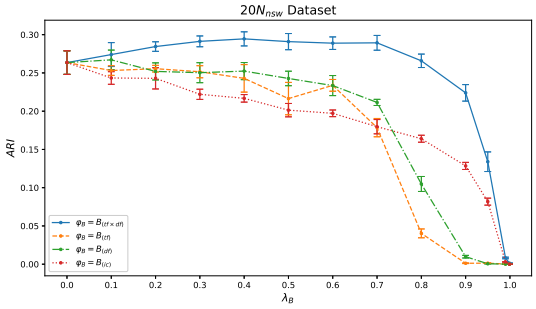
On the 20N$_{nsw}$ and FAZ$_{nsw}$ datasets performance gains are smaller since scores are already high, even with $\lambda_B = 0$. On FAZ$_{nsw}$ all initializations for the BT achieve an improvement, while on 20N$_{nsw}$ only $\varphi_B = B_{(tf \times df)}$ improves results. The best achieved scores on the filtered datasets are not significantly higher than the ones produced on the unfiltered datasets. It therefore seems that (as long as parameters are chosen right) removing stop-words is not a necessity in order to achieve good cluster performance with this model.

To further investigate the behavior of the LDA-BT$_{\text{static}}$ model, figure 5.5 shows the C_v coherence scores for all datasets. Similar to the cluster performance, the coherence score indicates that model performance improves on the datasets that include stop-words, with increasing λ_B. In this case $\varphi_B = B_{(tf \times df)}$, $\varphi_B = B_{(tf)}$ and $\varphi_B = B_{(df)}$ all seem to be viable options. However, coherence scores are constantly higher on the datasets that do not contain stop-words. Here the coherence score remains relatively stable until λ_B approaches 1. It therefore seems that in this case, removing stop-words does give an advantage and that the BT is not able to full compensate for the stop-words.

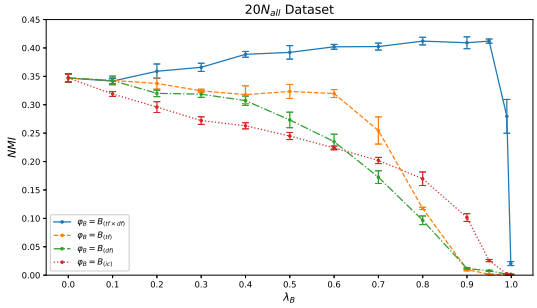
Altogether it seems that choosing $\varphi_B = B_{(tf \times df)}$ is the best option (out of the tested ones) for the LDA-BT$_{\text{static}}$ model. Figure 5.6 shows the distribution of word tokens over all topics on the 20N$_{all}$ and FAZ$_{all}$ datasets, produced by a LDA-BT$_{\text{static}}$ model with $\varphi_B = B_{(tf \times df)}$ and an appropriate λ_B value. The BT is by far the biggest topic in both cases. It also captures most of the stop-words. This way it forces the normal topics to include more specific words that have a relatively small $(tf \times df)$ value.



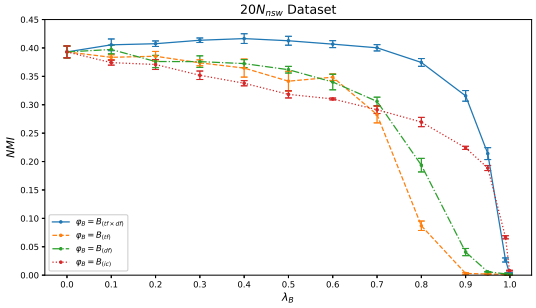
(a) *ARI* for the 20N$_{all}$ dataset



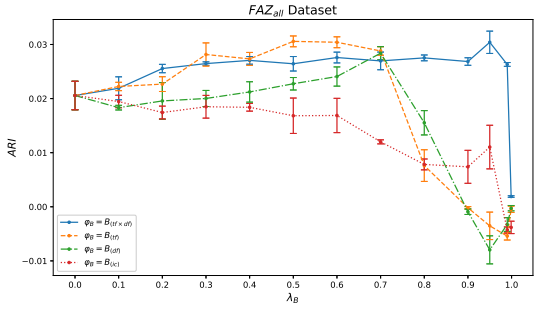
(b) *ARI* for the 20N$_{nsw}$ dataset



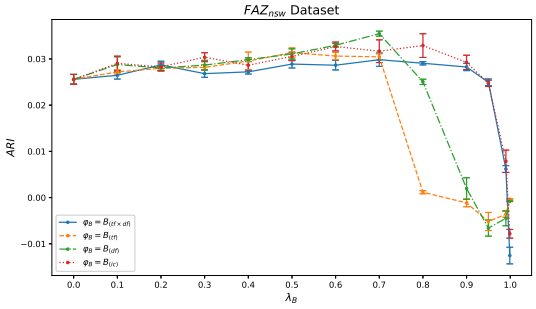
(c) *NMI* for the 20N$_{all}$ dataset



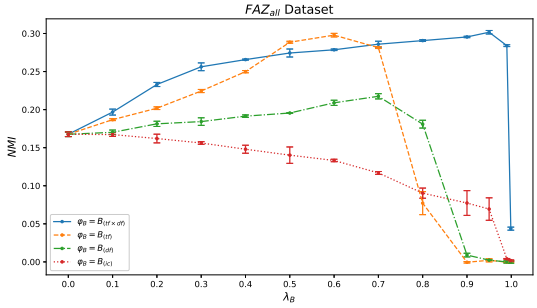
(d) *NMI* for the 20N$_{nsw}$ dataset



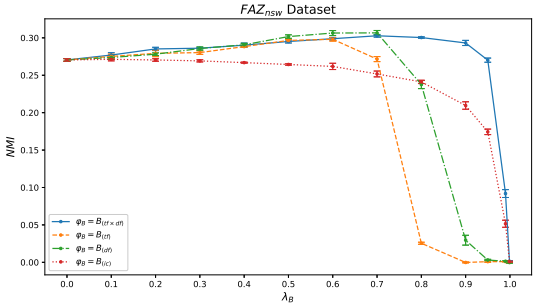
(e) *ARI* score, FAZ$_{all}$ dataset



(f) *ARI* score , FAZ$_{nsw}$ dataset

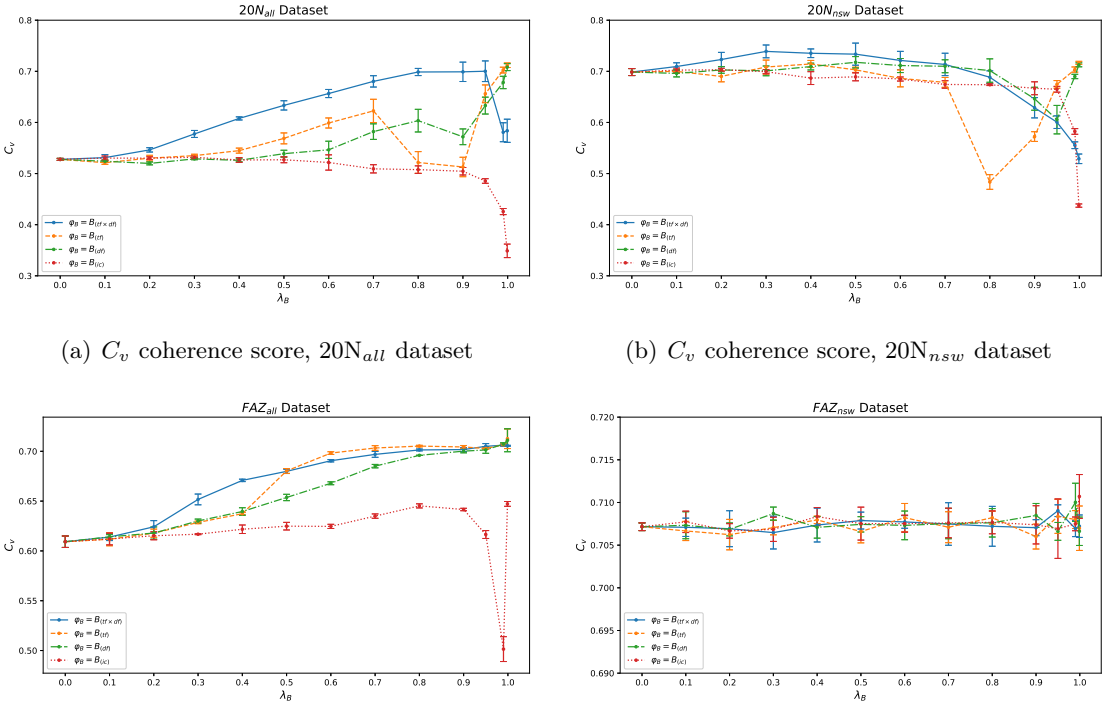


(g) *NMI* score, FAZ$_{all}$ dataset



(h) *NMI* score, FAZ$_{nsw}$ dataset

Figure 5.4: Cluster performance of the LDA-BT$_{\text{static}}$ model using different BT initializations and λ_B values on all datasets.



(a) C_v coherence score, 20N$_{all}$ dataset

(b) C_v coherence score, 20N$_{nsw}$ dataset

(c) C_v coherence score, FAZ$_{all}$ dataset

(d) C_v coherence score, FAZ$_{nsw}$ dataset

Figure 5.5: Coherence scores of the LDA-BT$_{\text{static}}$ model using different BT initializations and λ_B values on all datasets.

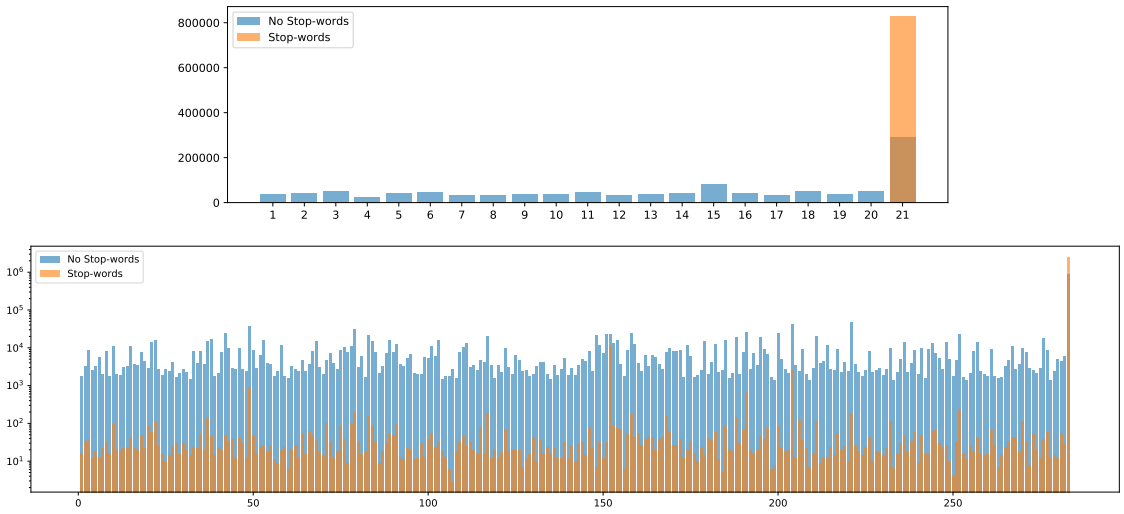


Figure 5.6: Total assignments per topic of a LDA-BT$_{\text{static}}$ model on the 20N$_{all}$ and FAZ$_{all}$ datasets. The models are initialized with $\varphi_B = B_{(tf \times df)}$ and $\lambda_B = 0.8$. The last topic on the right is the BT. Notice that the second graph (FAZ$_{all}$) has a logarithmic scale on the y-axis.

5.4 Dynamic Background Topic

It is difficult to look at the different priors of the LDA-BT$_{\text{dynamic}}$ model (defined in section 3.2.2) in a vacuum, as they can all influence each other. The following will therefore consider two different scenarios. First it is analyzed how an asymmetric α prior influences the model, while β_B and β_T are kept symmetric. Followed by an investigation on how β_B can influence the model when chosen asymmetric.

5.4.1 Symmetric Background Topic Prior

In this section the general probability of the BT is increased by altering the pseudo-count α_{t_B} of the α prior. It is worth mentioning that the here tested setting, where $\beta_B = \beta_T$, can be also achieved (in contrast to the case where $\beta_B \neq \beta_T$) with the normal LDA model. In this section the priors β_B and β_T are however kept symmetric. Therefore $\beta_B = \beta_T = \langle 0.1_{\times V} \rangle$ are chosen symmetric, while $\alpha = \langle (\frac{50}{T})_{\times T}, x \rangle$ is chosen asymmetric by altering x (note that the sum of pseudo-counts of all normal topics is $\sum_{t=1}^{T} \alpha_t = 50$). Figure 5.7 shows the size of each topic for two models trained on FAZ$_{all}$ and 20N$_{all}$ datasets. As expected, the BT is by far the biggest topic. Moreover it does capture most of the stop-words, even tough it has no prior knowledge about them (β_B is chosen symmetric).
The achieved cluster scores for a variety of different α_{t_B} values are shown in figure 5.8. These results indicate that model performance increases significantly on all datasets with an increasing α_{t_B} value. It seems that even tough FAZ$_{nsw}$ and 20N$_{nsw}$ do not contain stop-words, the BT is still beneficial. A likely explanation for this, is that both datasets still contain many common terms that hold little meaning, which were however not recognized as stop-words and therefore not removed beforehand. Such terms are captured by the BT, which thus improves performance. At a certain point however increasing α_{t_B} does not increase performance further and eventually even decreases performance again. On the datasets with no stop-words, peak performance is reached with a smaller α_{t_B} value. This behavior is expected since FAZ$_{all}$ and 20N$_{all}$ contain more words that need to be captured by the BT. Additional, with the right α_{t_B} the LDA-BT$_{\text{dynamic}}$ model is able to achieve similar cluster performance on datasets with and without stop-words.
Figure 5.9 shows the C_v coherence scores achieved in the same setting. On 20N$_{all}$ and 20N$_{nsw}$ coherence improves significantly with increasing α_{t_B} values. However the coherence on FAZ$_{nsw}$ seems to be unaffected, while on FAZ$_{all}$ only one drastic change can be observed at $\alpha_{t_B} = 0$.
Hence, for this particular case, it seems that removing stop-words from the dataset is not necessary as long as the BT is allowed to become big enough.

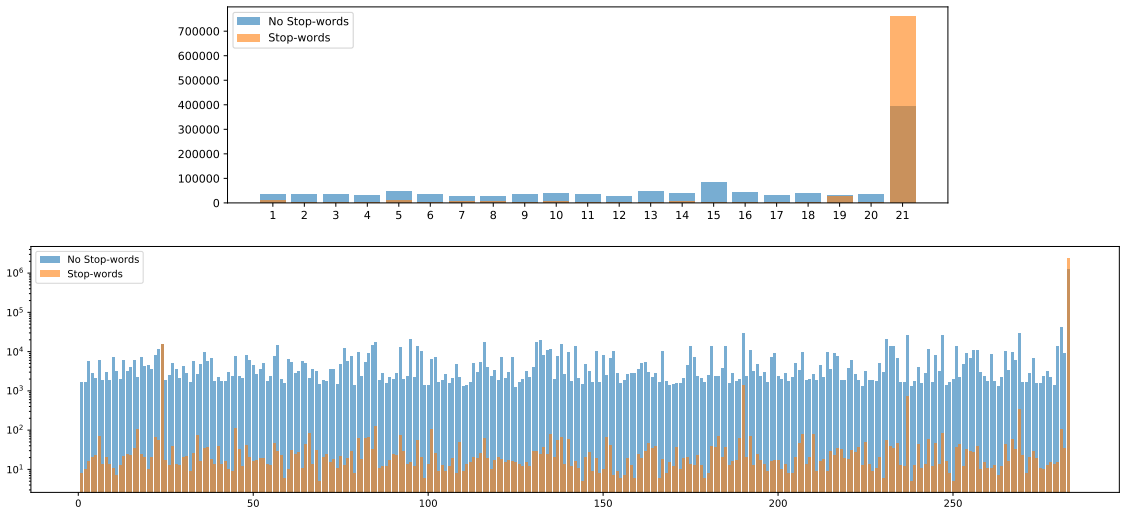
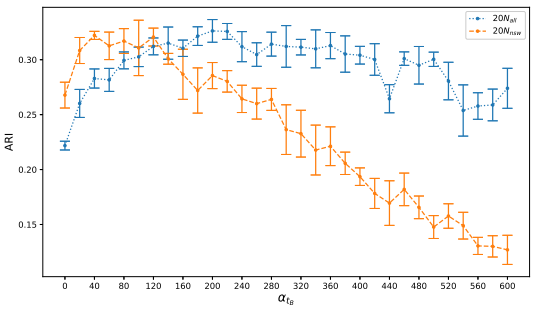
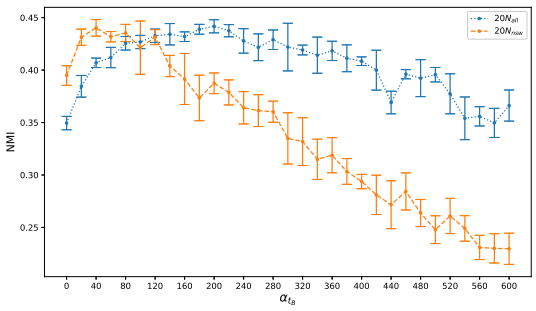


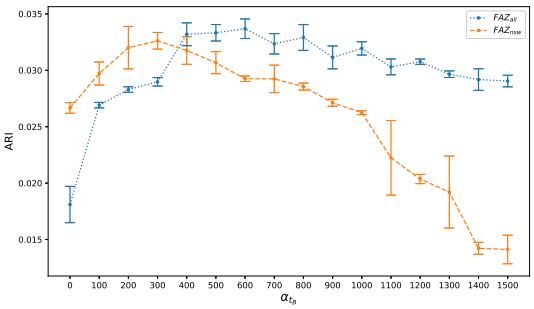
Figure 5.7: Total assignments per topic of a LDA-BT$_{\text{dynamic}}$ model on the 20N$_{all}$ and FAZ$_{all}$ datasets. The model was initialized with priors $\beta_B = \beta_T = \langle 0.1_{\times V} \rangle$. For the 20N$_{all}$ dataset $\alpha = \langle (\frac{50}{T})_{\times T}, 150 \rangle$ and for the FAZ$_{all}$ dataset $\alpha = \langle (\frac{50}{T})_{\times T}, 600 \rangle$ is chosen. The last topic on the right is the BT. Notice that the second graph (FAZ$_{all}$) has a logarithmic scale on the y-axis.



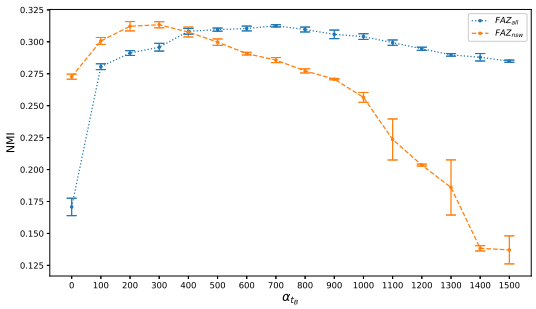
(a) *ARI* score, 20N$_{all}$ and 20N$_{nsw}$ dataset



(b) *NMI* score, 20N$_{all}$ and 20N$_{nsw}$ dataset

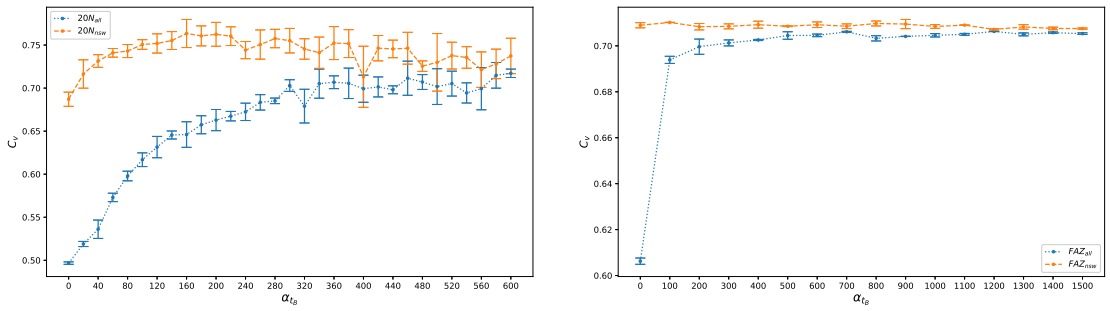


(c) *ARI* score, FAZ$_{all}$ and FAZ$_{nsw}$ dataset



(d) *NMI* score, FAZ$_{all}$ and FAZ$_{nsw}$ dataset

Figure 5.8: Cluster performance of the LDA-BT$_{\text{dynamic}}$ model on all datasets with varying α_{t_B} values. Hyperparameter $\beta_B = \beta_T = \langle 0.1_{\times V} \rangle$ are chosen symmetric, while $\alpha = \langle (\frac{50}{T})_{\times T}, x \rangle$ is chosen asymmetric by altering x.



(a) C_v coherence score, $20N_{all}$ and $20N_{nsw}$ dataset (b) C_v coherence score, FAZ_{all} and FAZ_{nsw} dataset

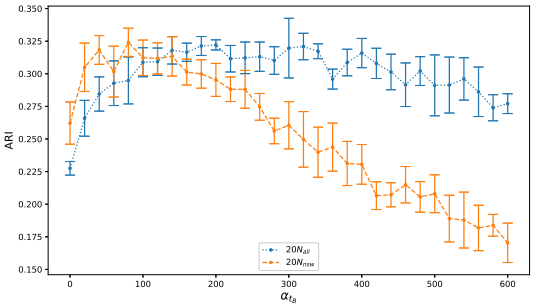
Figure 5.9: Coherence scores of the LDA-$BT_{dynamic}$ model on all datasets with varying α_{t_B} values. Hyperparameter $\beta_T = \beta_B = \langle 0.1_{\times V} \rangle$ are chosen symmetric. Prior $\alpha = \langle (\frac{50}{T})_{\times T}, x \rangle$ is chosen asymmetric by altering x.

5.4.2 Asymmetric Background Topic Prior

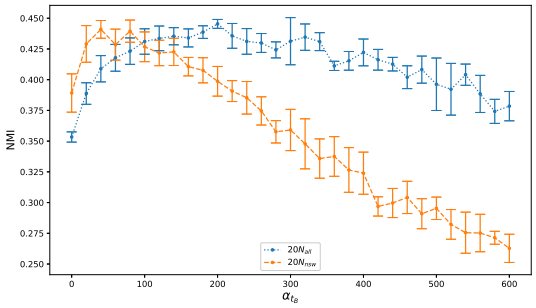
By altering the β_B prior it is possible to influence the shape of the BT. The BT is expected to capture terms that have a relatively high term and document frequency. A reasonable suggestion is therefore to initialize β_B depending on the frequency's of each term. One approach is to simply use the BT initializations defined in equations 3.3 to 3.6. A thing that needs to be considered however is the size of β_B and β_T. Dirichlet priors do not need to be normalized and can therefore be scaled to different sizes. In section 5.2 it is found that a symmetric prior of $\beta = \langle 0.1_{\times V} \rangle$ produces good results on both $20N_{all}$ and $20N_{nsw}$ datasets. The β_B prior is therefore chosen in such a way that its length is equal to $|\langle 0.1_{\times V} \rangle|$. Figure 5.10 shows the performance of a model with $\beta_B = \frac{0.1 \times B_{(tf \times df)}}{mean(B_{(tf \times df)})}$ and $\beta_T = \langle 0.1_{\times V} \rangle$ for different α_{t_B} values on $20N_{all}$ and $20N_{nsw}$ datasets.

The results for $20N_{all}$ and $20N_{nsw}$ look very similar to the ones produced with a symmetric β_B prior and it seems that altering the prior makes no real difference. The only observable difference compared to the symmetric β_B prior is that cluster performance drops slower in the range $\alpha_{t_B} \in [1000, 1500]$. Some further tests (not shown) however revealed that choosing $\beta_B = \langle 0.01_{\times V} \rangle$, as a smaller symmetric prior, has a similar effect. It is therefore questionable if the improvement is a result of the asymmetric β_B prior.

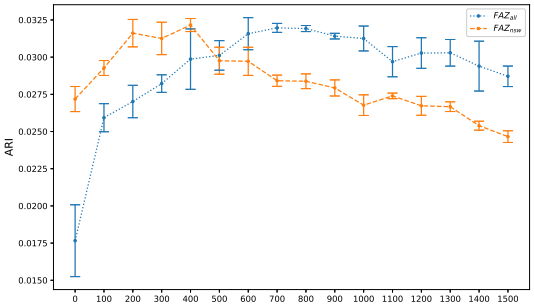
Figure 5.11 shows the coherence scores that are achieved with the same setting. Here the scores for FAZ_{all} and FAZ_{nsw} are very similar to the ones produced by a symmetric prior in figure 5.9. On the $20N_{nsw}$ dataset coherence scores drop slightly for higher α_{t_B} values, while scores for $20N_{all}$ seem to be unaffected by the asymmetric prior.



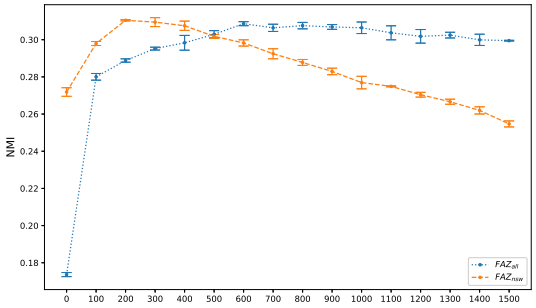
(a) *ARI* score, $20N_{all}$ and $20N_{nsw}$ dataset



(b) *NMI* score, $20N_{all}$ and $20N_{nsw}$ dataset

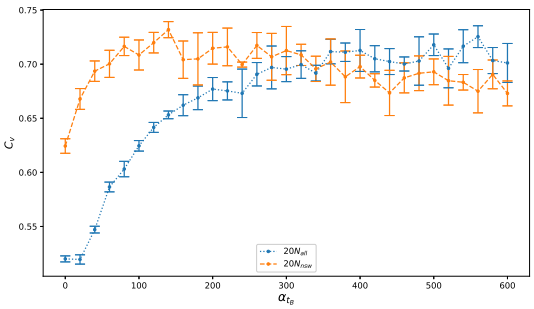


(c) *ARI* score, FAZ_{all} and FAZ_{nsw} dataset

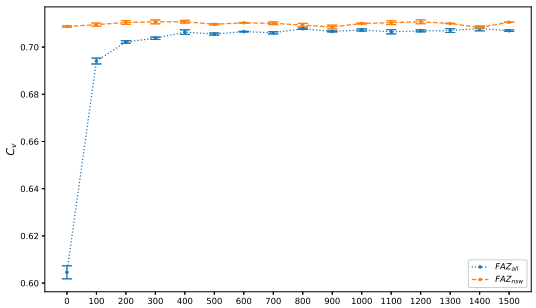


(d) *NMI* score, FAZ_{all} and FAZ_{nsw} dataset

Figure 5.10: Cluster performance of the LDA-$BT_{dynamic}$ model on all datasets with varying α_{t_B} values. Hyperparameter $\beta_T = \langle 0.1_{\times V} \rangle$ is chosen symmetric, while $\beta_B = \frac{0.1 \times B_{(tf \times df)}}{mean(B_{(tf \times df)})}$ is asymmetric. Prior $\alpha = \langle (\frac{50}{T})_{\times T}, x \rangle$ is chosen asymmetric by altering x.



(a) C_v coherence score, $20N_{all}$ and $20N_{nsw}$ dataset



(b) C_v coherence score, FAZ_{all} and FAZ_{nsw} dataset

Figure 5.11: Coherence score of the LDA-$BT_{dynamic}$ model on all datasets with varying α_{t_B} values. Hyperparameter $\beta_T = \langle 0.1_{\times V} \rangle$ is chosen symmetric, while $\beta_B = \frac{0.1 \times B_{(tf \times df)}}{mean(B_{(tf \times df)})}$ is asymmetric. Prior $\alpha = \langle (\frac{50}{T})_{\times T}, x \rangle$ is chosen asymmetric by altering x.

5.5 Static and Dynamic Background Topic Comparison

Both the LDA-BT$_{\text{static}}$ model (using $\varphi_B = B_{(tf \times df)}$) and the LDA-BT$_{\text{dynamic}}$ model were able to outperform the normal LDA algorithm on the FAZ$_{all}$ and 20N$_{all}$ dataset. Yet, the LDA-BT$_{\text{static}}$ model achieves only minor improvements on the FAZ$_{nsw}$ and 20N$_{nsw}$ datasets. It is furthermore consistently outperformed by the LDA-BT$_{\text{dynamic}}$ model.
Additionally, choosing asymmetric β_B prior proportional to $B_{(tf \times df)}$ seems to have no positive effect on model performance on 20N$_{all}$, 20N$_{nsw}$ and FAZ$_{all}$. However, on 20N$_{nsw}$ this does improve cluster performance when α_{t_B} is big. Hence it is therefore questionable if word weighting measures like $(tf \times df)$ are even appropriate to predict if terms belong into the BT or not. Figure 5.12 shows the learned φ_B distribution of an LDA-BT$_{\text{dynamic}}$ model trained on the 20N$_{nsw}$ dataset. Since this model achieved better *NMI* and *ARI* scores then the LDA-BT$_{\text{static}}$ model (initialized with $\varphi_B = B_{(tf \times df)}$) suppose that the learned φ_B distribution is a better representation of the BT than $B_{(tf \times df)}$. This can (subjectively) be confirmed by looking at the terms that are marked in red. Such terms are, according to the learned φ_B distribution, less likely to belong to the BT although they have a relatively high probability in $B_{(tf \times df)}$. Terms with a low probability like *god, program, file, data, drive, space, law, software, email* are (subjectively) more relevant for the 20Newsgroups dataset (every term can clearly be assigned to one of the in figure 5.1 shown labels), then terms with high probability like *would, like, know, get, also, even, may, much.* The learned φ_B and $B_{(tf \times df)}$ are relatively dissimilar which leads to the conclusion that $B_{(tf \times df)}$ is a sub optimal choice for the BT on this particular dataset.
Figure 5.13 shows a comparison between the learned φ_B distribution of an LDA-BT$_{\text{dynamic}}$ model and $B_{(tf \times df)}$ for the FAZ$_{nsw}$ dataset. Compared to the results for the 20N$_{nsw}$ dataset, φ_B and $B_{(tf \times df)}$ seem to be more similar on the FAZ$_{nsw}$ dataset. However φ_B still contains some spikes, where the probability is much lower compared to $B_{(tf \times df)}$.
These results, together with the constantly higher cluster performance of the LDA-BT$_{\text{dynamic}}$ model, seem to suggest that letting the model learn φ_B is a better option for the tested datasets than to initialize it with fixed probabilities. Choosing an asymmetric prior β_B proportional to $B_{(tf \times df)}$ (as conducted in section 5.4.2) seems to have, if at all, just a minor impact. Hence it's especially important to choose a appropriate α_{t_B} value depending on the dataset.

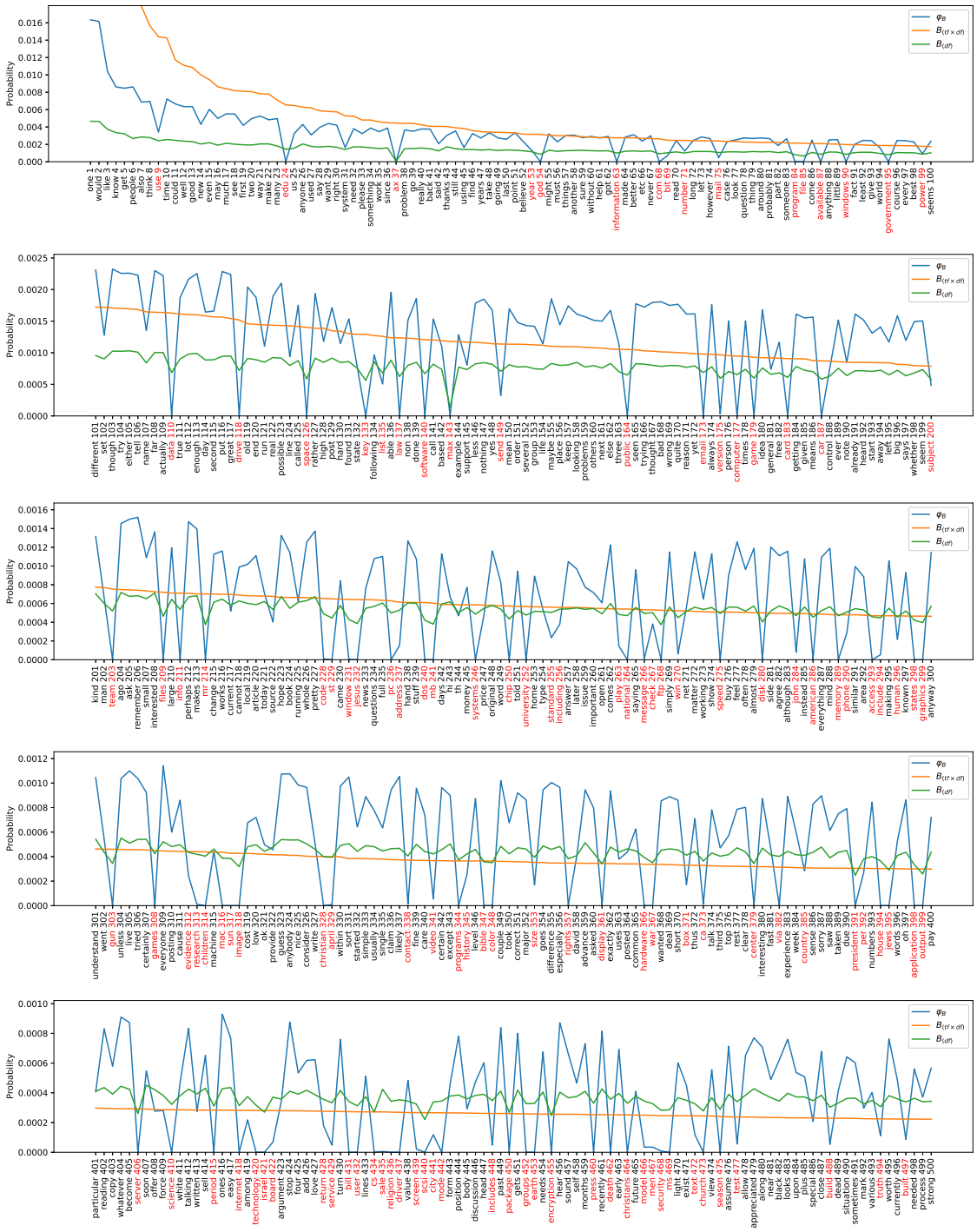


Figure 5.12: The learned φ_B distribution of an LDA-BT$_{\text{dynamic}}$ model compared to $B_{(df)}$ and $B_{(tf \times df)}$ on the 20N$_{nsw}$ dataset. Shown are the first 500 terms sorted by $B_{(tf \times df)}$. The model was trained with hyperparameters $\alpha = \langle (\frac{50}{T})_{\times T}, 50 \rangle$, $\beta_B = \beta_T = \langle 0.1_{\times V} \rangle$. Terms where φ_B has a low probability compared to $B_{(tf \times df)}$ are marked in red.

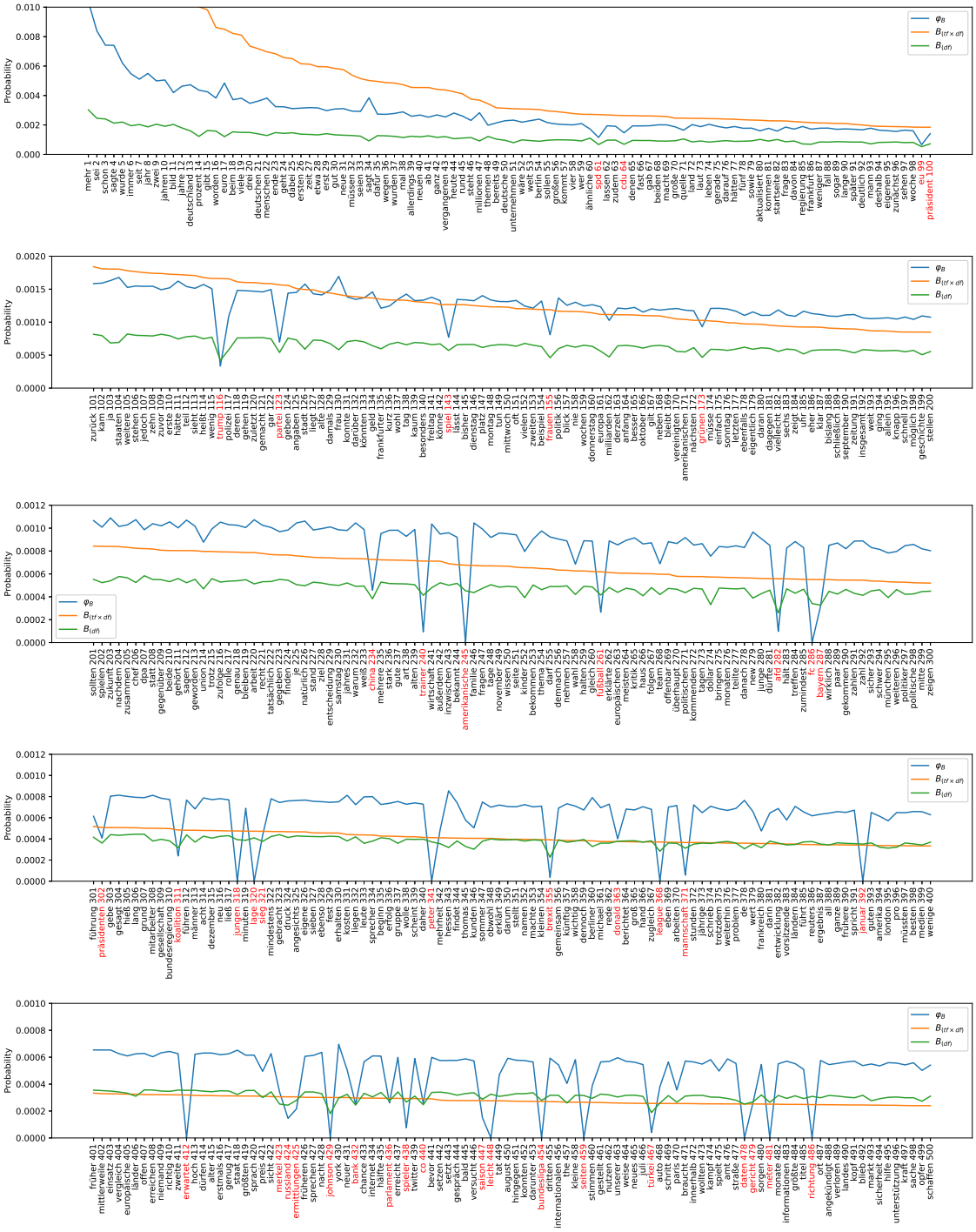
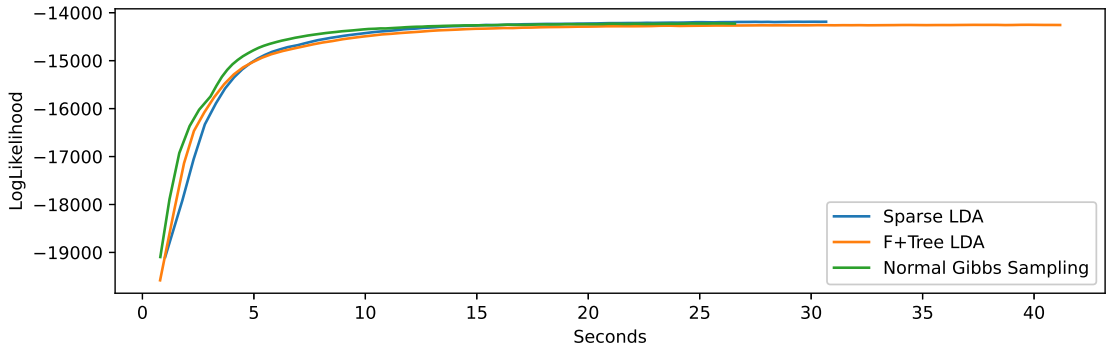


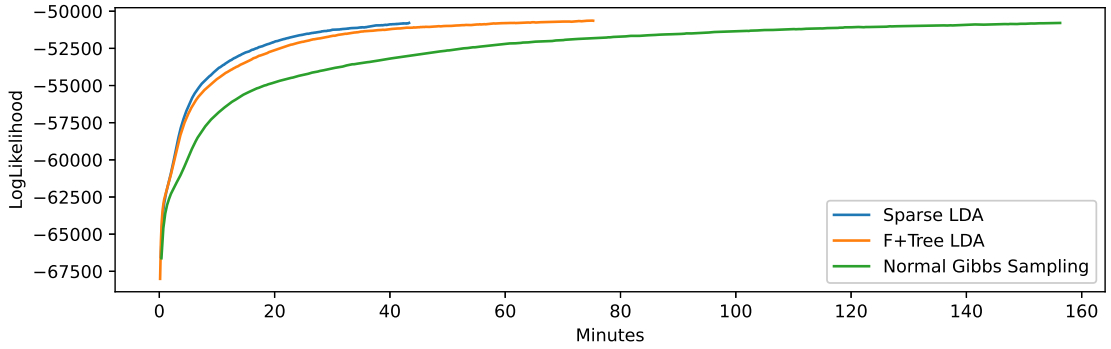
Figure 5.13: The learned φ_B distribution of an LDA-BT$_{\text{dynamic}}$ model compared to $B_{(df)}$ and $B_{(tf \times df)}$ on the FAZ$_{nsw}$ dataset. Shown are the first 500 terms sorted by $B_{(tf \times df)}$. The model was trained with hyperparameters $\alpha = \langle (\frac{50}{T})_{\times T}, 300 \rangle$, $\beta_B = \beta_T = \langle 0.1_{\times V} \rangle$. Terms where φ_B has a low probability compared to $B_{(tf \times df)}$ are marked in red.

5.6 Run Time Comparison

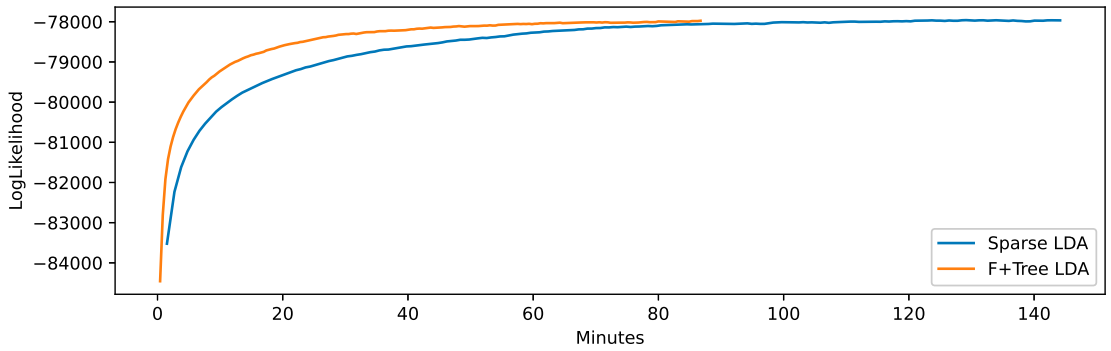
This section evaluates the run time of the, in section 3.2 discussed, sampling approaches implemented in Java (as discussed in section 4.5) and run on an Intel® Core™ i7-6850K processor. Figure 5.14 shows the run times of the normal Gibbs sampler, the SparseLDA sampler and the F+Tree LDA sampler. The shown run times are produced with the LDA-$\text{BT}_{\text{dynamic}}$ model, however the LDA-$\text{BT}_{\text{static}}$ model produces very similar run times. The results indicate that, as already discussed in section 4.4, the samplers all offer advantages and disadvantages depending of the size of the used dataset and the number of topics T. For the smaller 20N_{all} dataset with a small $T = 20$, the normal Gibbs sampler is the best choice (figure 5.14(a)) since it has no overhead for maintaining extra data structures like the other two samplers. However with increasing T the normal Gibbs sampler quickly becomes extremely slow. On the bigger FAZ_{all} dataset with a medium amount of topics $T = 282$ the SparseLDA sampler performs best (figure 5.14(b)) since topics are distributed sparsely (the sampling complexity is $\mathcal{O}(T_d + T_v)$) and maintaining it's data structures is less expensive than maintaining an F+Tree. However for $T = 3000$ (figure 5.14(c)) this advantage distinguishes and F+Tree LDA becomes the fastest sampler because of it's $\mathcal{O}(T_d + \log T)$ sampling complexity.



(a) $20N_{all}$ dataset with $T = 20$



(b) FAZ_{all} dataset with $T = 282$



(c) FAZ_{all} dataset with $T = 3000$

Figure 5.14: Run time comparison of different sampling methods using the LDA-BT_{dynamic} model on the FAZ_{all} dataset.

Chapter 6

Conclusion and Outlook

The aim of this thesis is to investigate how LDA results can be improved by utilizing a BT. The BT is particularly important in cases where stop-words are not removed as a preprocessing step. While a BT is already used in several works, it is usually just mentioned as a side note and not studied in detail. In this thesis two new models are developed, that incorporate a BT into LDA. The models are designed such that prior knowledge about the distribution of frequent terms can be incorporated by using term weighting schemes. The tested datasets are split into versions with and without stop-words to see the effect of the BT more clearly.

Both models are able to outperform the normal LDA model on datasets with stop-words. The LDA-$\text{BT}_{\text{dynamic}}$ model also improves results on the datasets without stop-words. From the tested term weighting schemes, $tf \times df$ seems to be the best option for a static BT probability distribution. At least for the tested datasets, letting the model learn the term probabilities on its own seems to be an even better solution, provided that the BT is allowed to become big enough. As already found in [41], choosing the right asymmetric α prior (which means to incorporate prior knowledge about the size of topics) seems to be more important than choosing the right β prior. When the BT is allowed to become big enough it is successful in capturing most of the stop-words even without any prior knowledge about their distribution. Yet, it remains unclear how big exactly the BT should be for a given dataset, since performance starts do degrade again if the BT becomes to large.

Utilizing a BT therefore does not necessarily make using LDA easier. The newly developed models offer even more options to tweak generated results. All parameters have to be understood by the user and need to be chosen carefully. Even then it is often a matter of trial and error until generated topics are appropriate. Another problem with all topic modeling is evaluation itself. Evaluating generated topics is not trivial and still an open research area. As of today, there is no measure that consistently predicts the quality of generated topics, in different use cases. The used evaluation measure depends a lot on what

is expected from generated topics and in which context the topic model is used. Cluster performance is an appropriate measure when LDA is used for text classification but can potentially be useless in other scenarios. Furthermore it can only be evaluated for labeled data. Topic coherence measures like C_v aim to correlate with human judgment on the meaningfulness of generated topics. However they give no real guarantee if LDA results are actually meaningful.
The experiments conducted suggest that a BT is a viable option for improving LDA results. Although this does not necessarily make the use of stop-word lists obsolete. No model was able to achieve a higher score on a unfiltered dataset compared to a filtered one. It therefore seems, that removing the most obvious stop-words in preprocessing and then letting the BT handle the remaining frequently used terms is the most advisable option.
In addition, the BT can successfully be integrated into state of the art sampling approaches, in form of SparseLDA and F+Tree LDA. Since just one topic is handled separately, the samplers retain their original complexity and therefore accelerate computation as intended. Since normal Gibbs sampling, SparseLDA and F+Tree LDA all have their advantages and disadvantages for certain dataset sizes, one can choose the sampler depending on the given task.

6.1 Further Work

This thesis focused on the four relatively simple word weighting schemes tf, df, $tf \times df$ and information content to incorporate prior knowledge into the LDA-$\text{BT}_{\text{dynamic}}$ and LDA-$\text{BT}_{\text{static}}$ models. There are of course other measures that could be evaluated that may improve results further. For example, paper [13] discusses several modifications of the *tf-idf* measure.
Another consideration is, to check if the developed models are more stable when run multiple times compared to normal LDA, which should make generated results more reliable. This could, for example be investigated using the *Similarity of multiple sets by Clustering with LOcal Pruning (S-CLOP)* measure, proposed by [34].
One further question that remains unanswered is how exactly α_{t_B} should be chosen for given datasets, when using the LDA-$\text{BT}_{\text{dynamic}}$ model. In [41] the authors rewrite α as the product αm, where α is a (scalar) concentration parameter and m a normalized vector (which is also known as the *base measure*). They then take a fully Bayesian approach to infer m by giving it its own hyperparameter α', which in turn is sampled from a gamma distribution. Since such an approach is computationally expensive they further implement the optimization techniques discussed in [40]. Some of those techniques are also investigated by [4] and [20]. Such an approach could be adopted to infer an α prior for the LDA-$\text{BT}_{\text{dynamic}}$ model by choosing an appropriate α' prior.

Bibliography

[1] 20 Newsgroups dataset. `http://qwone.com/~jason/20Newsgroups/`. Accessed: 10.08.2020.

[2] JPype. `https://jpype.readthedocs.io/en/latest/index.html`. Accessed: 10.08.2020.

[3] Natural Language Toolkit (NLTK). `https://www.nltk.org/`. Accessed: 10.08.2020.

[4] Arthur Asuncion, Max Welling, Padhraic Smyth, and Yee Whye Teh. On smoothing and inference for topic models. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, page 27–34, 2009.

[5] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[6] Jonathan Chang. *lda: Collapsed Gibbs Sampling Methods for Topic Models*, 2015. R package version 1.4.2.

[7] Jonathan Chang, Sean Gerrish, Chong Wang, Jordan L. Boyd-graber, and David M. Blei. Reading tea leaves: How humans interpret topic models. In *Advances in Neural Information Processing Systems*, volume 22, pages 288–296. 2009.

[8] Chaitanya Chemudugunta, Padhraic Smyth, and Mark Steyvers. Modeling general and specific aspects of documents with a probabilistic topic model. In *Advances in Neural Information Processing Systems*, volume 19, pages 241–248. 2007.

[9] Jianfei Chen, Kaiwei Li, Jun Zhu, and Wenguang Chen. Warplda: A cache efficient o(1) algorithm for latent dirichlet allocation. *Proceedings of the VLDB Endowment*, 9(1):744–755, 2016.

[10] William M. Darling. A theoretical and practical implementation tutorial on topic modeling and gibbs sampling. 2011.

[11] Hal Daumé III and Daniel Marcu. Bayesian query-focused summarization. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 305–312, 2006.

[12] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. L, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.

[13] Giacomo Domeniconi, Gianluca Moro, Roberto Pasolini, and Claudio Sartori. A comparison of term weighting schemes for text classification and sentiment analysis with a supervised variant of tf.idf. In *International Conference on Data Management Technologies and Applications*, volume 584, page 39, 2016.

[14] S. Dumais. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments and Computers*, 23(2):229–236, 1991.

[15] Georg K. Gerber, Robin D. Dowell, Tommi S. Jaakkola, and David K. Gifford. Automated discovery of functional generality of human gene expression programs. *PLOS Computational Biology*, 3(8), 2007.

[16] Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National academy of Sciences*, 101(1):5228–5235, 2004.

[17] Thomas L. Griffiths, Mark Steyvers, David M. Blei, and Joshua B. Tenenbaum. Integrating topics and syntax. In *Advances in Neural Information Processing Systems*, volume 17, pages 537–544. 2005.

[18] Tom Griffiths. Gibbs sampling in the generative model of latent dirichlet allocation. *Technical reports Stanford University*, 2002.

[19] Aria Haghighi and Lucy Vanderwende. Exploring content models for multi-document summarization. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 362–370, 2009.

[20] Gregor Heinrich. Parameter estimation for text analysis. 2005.

[21] Mikael Henaff, Kevin Jarrett, Koray Kavukcuoglu, and Yann LeCun. Unsupervised learning of sparse features for scalable audio classification. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, pages 681–686, 2011.

[22] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 50–57, 1999.

[23] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[24] Jagadeesh Jagarlamudi, Hal Daumé III, and Raghavendra Udupa. Incorporating lexical priors into topic models. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 204–213, 2012.

[25] Aaron Q. Li, Amr Ahmed, Sujith Ravi, and Alexander J. Smola. Reducing the sampling complexity of topic models. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 891–900, 2014.

[26] Xuezheng Liu, Lei Zhang, Mingjing Li, Hongjiang Zhang, and Dingxing Wang. Boosting image classification with lda-based feature combination for digital photograph management. *Pattern Recognition*, 38(6):887 – 901, 2005.

[27] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. `http://mallet.cs.umass.edu`, 2002.

[28] David Mimno, Wei Li, and Andrew McCallum. Mixtures of hierarchical topics with pachinko allocation. In *Proceedings of the 24th International Conference on Machine Learning*, page 633–640, 2007.

[29] David Mimno, Hanna Wallach, Edmund Talley, Miriam Leenders, and Andrew McCallum. Optimizing semantic coherence in topic models. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 262–272, 2011.

[30] Thomas P. Minka and Rosalind Picard. *A Family of Algorithms for Approximate Bayesian Inference*. PhD thesis, 2001.

[31] T.P. Minka and John Lafferty. Expectation-Propagation for the Generative Aspect Model. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, pages 352–359, 2002.

[32] W.M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[33] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, 2010.

[34] J. Rieger, Lars Koppers, C. Jentsch, and J. Rahnenführer. Improving reliability of latent dirichlet allocation by assessing its stability using clustering techniques on replicated runs. *ArXiv*, abs/2003.04980, 2020.

[35] Michael Röder, Andreas Both, and Alexander Hinneburg. Exploring the space of topic coherence measures. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 399–408, 2015.

[36] Michal Rosen-Zvi, Thomas Griffiths, Mark Steyvers, and Padhraic Smyth. The author-topic model for authors and documents. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, page 487–494, 2004.

[37] Claude E. Shannon. A mathematical theory of communication. *Bell Labs Technical Journal*, 27(3):379–423, 1948.

[38] Yee Whye Teh, David Newman, and Max Welling. A collapsed variational bayesian inference algorithm for latent dirichlet allocation. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, page 1353–1360, 2006.

[39] Hanna M. Wallach. Topic modeling: beyond bag-of-words. In *Proceedings of the 23rd international conference on Machine learning*, volume 148 of *ACM International Conference Proceeding Series*, pages 977–984, 2006.

[40] Hanna M. Wallach. *Structured Topic Models for Language*. PhD thesis, University of Cambridge, 2008.

[41] Hanna M. Wallach, David M. Mimno, and Andrew McCallum. Rethinking lda: Why priors matter. In *Advances in Neural Information Processing Systems*, volume 22, pages 1973–1981. 2009.

[42] Hanna M. Wallach, Iain Murray, Ruslan Salakhutdinov, and David Mimno. *Evaluation Methods for Topic Models*, page 1105–1112. 2009.

[43] Andrew T. Wilson and Peter A. Chew. Term weighting schemes for latent Dirichlet allocation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 465–473, 2010.

[44] Zhao Yang, René Algesheimer, and Claudio Tessone. A comparative analysis of community detection algorithms on artificial networks. *Scientific Reports*, 6, 2016.

[45] Limin Yao, David Mimno, and Andrew McCallum. Efficient methods for topic model inference on streaming document collections. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 937–946, 2009.

[46] Hsiang-Fu Yu, Cho-Jui Hsieh, Hyokun Yun, S.V.N. Vishwanathan, and Inderjit S. Dhillon. A scalable asynchronous distributed algorithm for topic modeling. In *Proceedings of the 24th International Conference on World Wide Web*, page 1340–1350, 2015.

[47] Jinhui Yuan, Fei Gao, Qirong Ho, Wei Dai, Jinliang Wei, Xun Zheng, Eric Po Xing, Tie-Yan Liu, and Wei-Ying Ma. Lightlda: Big topic models on modest computer clusters. In *Proceedings of the 24th International Conference on World Wide Web*, page 1351–1361, 2015.

[48] Lele Yut, Ce Zhang, Yingxia Shao, and Bin Cui. Lda*: A robust and large-scale topic modeling system. *Proceedings of the VLDB Endowment*, 10(11):1406–1417, 2017.

[49] ChengXiang Zhai and Sean Massung. *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining*. Association for Computing Machinery and Morgan and Claypool, 2016.