The Author-Topic Model

Ólavur Mortensen

Kongens Lyngby 2017

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

The goal of this thesis is to develop a scalable and user-friendly implementation of the author-topic model for the Gensim framework. To this end, a variational Bayes (VB) algorithm is developed to train the model.

In order to allow online training, stochastic variational inference is applied. This removes the need to store all documents in memory, and allows us to keep learning on new data.

Maximum Likelihood Estimation is applied to automatically learn the optimal hyperparameters of the priors over words and over topics.

A blocking VB method is applied, inspired by blocking Gibbs sampling, that relaxes the assumptions we make about the form of the posterior in the variational approximation. The resulting algorithm lends itself to optimizations that decrease the memory complexity of the algorithm, and speed up training by vectorizing the VB updates. Blocking VB also increases the variational lower bound more per iteration than standard VB.

In order to illustrate useful examples of the model, as well as demonstrate usage of the software, a tutorial is written and is accessible online. This tutorial uses data exploration and similarity queries to gain insight about authors in a dataset consisting of scientific papers from the NIPS conference.

Preface

In the past few years, the machine learning community has seen an explosion of high quality open source software, particularly in the Python programming language. In natural language processing we have libraries such as Gensim and SpaCy, in deep learning have Theano and TensorFlow, for general machine learning we have SciKit Learn, and in probabilistic programming languages we have MC Stan, and many others.

It is of great importance that machine learning tools are accessible. Too often academic research projects end in low quality code and/or code that never sees the light of day. When we develop machine learning software that is user-friendly and fast, we enable everyone from first year university students to tenured professors to industry professionals to tinker with the latest in machine learning research.

The advent of faster personal computers and tools such as automatic differentiation has enabled deep learning software like Theano. Software like this leads to wide adoption of deep learning, which in turn leads to an explosion in interest and new discoveries in the field.

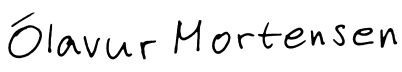
In a similar fashion, fast inference in probabilistic models with high quality and user-friendly implementations can lead to more interest and hence more research in Baysian data analysis. This is one of the goals of this project.

The other primary goal with this project is scalability. Having fast and scalable meachine learning tools lets us process much larger datasets at a much faster rate, and is imperative to the usefulness of machine learning in industry.

This project is conducted in order to obtain an Msc in engineering, specifically in "Mathematical Modelling and Computation", from the Technical University of Denmark.

A reader with a university level of understanding of statistics and programming should be able to understand the basics of this report. For an in depth understanding, knowledge of Bayesian data analysis, machine learning and natural language processing is required. Section 3 about the implementation discusses Python code, so Python literacy is required to fully understand that section.

Lyngby, 22-January-2017



Ólavur Mortensen

Acknowledgements

The multidisciplinary nature of this project was very well complimented with two great supervisors. I would like to thank Ole Winther from Cognitive Systems at DTU for his academic and theoretical support, and Lev Konstantinovskiy from RaRe Technologies for software development and open source support.

Notation and abbreviations

Note that these tables do not list all the notation and abbreviations used.

Mathematical notation:

$\log(x)$	The natural logarithm (base e) taken on x.
$\mathbb{I}(a = b)$	Identity function that yields 1 if $a = b$.
$\Gamma(x)$	The Gamma function taken on x.
$\Psi(x)$	The Digamma function taken on x.
D	Number of documents in corpus.
N_d	Number of words in document d.
A	Set of authors in corpus.
a	A single author identifier.
A_d	Set of authors in document d.
D_a	Set of documents for author a.
K	Number of topics.
k	A single topic identifier.
V	Number of unique words in vocabulary.
V_d	Number of unique words in document d.
v	A single vocabulary word.
β_k	Topic-word distribution for topic k.
θ_a	Author-topic distribution for author a.
α	θ's hyperparameter
η	β's hyperparameter
$w_{dn} \in 1, ..., V$	Word n in document d.
$z_{dn} \in 1, ..., K$	Topic assignment of word n in document d.
$x_{dn} \in 1, ..., A_d$	Author assignment of word n in document d.
y_{-i}	All elements of the vector y except y_i.
$x_{\cdot}$	All elements of vector x along the specified axis.
Dir	Dirichlet distribution
$Unif$	Uniform distribution
$Mult$	Multinomial distribution

Abbreviations:

BOW	Bag-of-words
CVB	Collapsed variational Bayes
LDA	Latent Dirichlet Allocation
NLP	Natural language processing
VB	Variational Bayes

Figures, tables and algorithms

List of Figures

List of Tables

List of Algorithms

Contents

Chapter 1

Introduction

Author-topic models promise to give data scientists a tool to simultaneously gain insight about authorship and content in terms of topics. The authors can represent many kinds of metadata attached to documents, for example, tags on posts on the web. The model can be used for data exploration, as features in machine learning pipelines, for author (or tag) prediction, or to simply leverage your topic model with existing metadata.

The author-topic model is very closely related to Latent Dirichlet Allocation (LDA) (Blei et al. 2003 [BNJ03]), which is a very popular model in the field of natural language processing.

There is a hands-on tutorial of the software that accompanies this thesis. This tutorial can be viewed at:

- Tutorial link: http://nbviewer.jupyter.org/github/rare-technologies/gensim/blob/develop/docs/notebooks/atmodel_tutorial.ipynb.

This tutorial gives examples as well as demonstrating the usage of the software, and is an essential part of this thesis.

Section 2 covers the theory behind the model and the method we apply to train

it. The goal of this section is to give some context, describe the model, as well as the algorithm used to train the model.

Section 3 goes through some technical details with the implementation of the model, to supplement the tutorial. We also introduce the open source package which this algorithm was developed for, namely Gensim.

In section 4, we present various tests of the implementation. This section is meant to supplement the tutorial.

It is recommended that the reader proceeds by reading the remainder of this section and the theory section (2), then reads the tutorial before proceeding to the implementation section (3).

In the remainder of this section, we give a gentle introduction to the author-topic model by example, discuss some application areas, and motivate the need for a new implementation.

1.1 Example

The author-topic model produces a set of latent topics as probability distributions over words, and in turn represents authors as probability distributions over the set of topics. A simple example is provided below.

An author-topic model with 10 topics is trained on a dataset. In table 1.1, we see the top 10 most important words in one of the topics, along with their probabilities. Notice that the model seems to have captured that one of the topics in the dataset has something to do with electronics and signal processing.

Similarly, table 1.2 shows the most important topics for a particular author, excluding the topics with probability below some threshold. Since the ID of the topic in table 1.1 is 0, we conclude that this author does not have anything to do with electronics and signal processing.

1.2 Applications

Naturally, topic modelling can be applied to automatic tagging and summarization of documents in large datasets. Topic modelling can also be used for

Word	Probability
chip	0.0146
circuit	0.01196
analog	0.0114
control	0.0100
implementation	0.00780
design	0.00726
implement	0.00636
signal	0.00633
vlsi	0.00594
processor	0.00565

Table 1.1: Example of a topic.

Topic ID	Probability
9	0.423
3	0.421
5	0.111
4	0.0432

Table 1.2: Example of an author in the author-topic model.

classification, information retrieval, and to build recommender systems, for example.

Topic models are not limited to text data. The "documents" can represent observations in many different kinds of data, and "words" then represent features of these observations. Topics then represent some type of components that describe the latent structure of the data.

Ngo et al. 2016 [NEFY16] apply the author-topic model to fMRI (functional magnetic resonance imaging) data to learn something about how various behavioural tasks relate to different areas of the brain. In this context, the authors represent the behavioural tasks, words represent areas in the brain (specifically, the magnitude of the activation in a voxel of the brain), and topics represent "cognitive components" (i.e. some latent structure).

The author-topic model, like LDA, is applicable in analysis of genomic data. The model can be used to learn commonalities between diseases or phenotypes (physical characteristics of organisms) based on different genes, by learning the author-topic representation with authors as diseases and phenotypes and biomarkers as features (words). Such models can be used to help in the process of designing medication, or in personalized healthcare.

The author-topic model could be used to build a tag prediction system, or as a component in a pipeline for such a system. If we have tagged documents, for example posts on a website, we can learn an author-topic representation of these tags. This representation would allow us to make similarity queries, i.e. ask which tags are most similar to a particular tag. Furthermore, we could ask which tags are most similar to the topic representation of a single document, and tag that document with the tags that have similarity above some value.

Building complex machine learning systems such as the ones discussed above is outside of the scope of this thesis, as the emphasis was on developing the algorithm and implementation.

1.3 Why do we need a new implementation?

There are a few existing implementations of the author-topic model, but all of these are either slow, not user-friendly, and/or have poor documentation.

There is one implementation that is reasonably fast, which is included in the Matlab Topic Modeling Toolbox [1]. This implementation suffers from a couple of problems, however, which is that (1) it is not particularly user friendly, (2) it is developed in a proprietary environment, and (3) it is not well documented. Furthermore, it applies Gibbs sampling to train the model, whereas we would rather have an implementation that uses variational Bayes (VB).

Reasons that VB is preferred over Gibbs sampling include:

- It is fast.
- It is easy to develop an online (streamable) algorithm.
- It is easy to develop a parallel algorithm.

The goal of this thesis is to develop a scalable and user-friendly implementation of the author-topic model for Gensim. For this to become a reality, a VB algorithm had to be developed, which will be described later in section 2.

[1] http://psiexp.ss.uci.edu/research/programs_data/toolbox.htm

Chapter 2

Theory

The goal of this section is to give some context in terms of Bayesian data analysis, natural language processing, and topic modelling, and to describe the author-topic model and derive the VB algorithm to train it.

In section 2.1, we provide a very brief conceptual introduction to Bayesian data analysis. Section 2.2 serves as introductory material for section 2.4, and contains concepts that will be relevant throughout the report. We briefly discuss topic models in general in section 2.3. In section 2.4, we describe the author-topic model, derive the variational Bayes updates for the model, and describe some algorithms using these updates, and more.

2.1 Bayesian data analysis

The author-topic model is what is often referred to as a Bayesian hierarchical model. What this means should become clear in the subsequent sections, although a thorough treatment of the subject is far beyond the scope of this report.

Intuitively, Bayesian data analysis is quite different from other types of data

modelling. Below, we discuss one interpretation of this distinction (as there are many).

In classical data analysis, we write down a cost function that describes the phenomena in the data that we want to capture. This cost function is minimized to obtain the best possible model of the data.

In Bayesian data analysis, we posit a statistical procedure that describes how the data was generated. In other words, we describe a *causal model* of the phenomena in the data that we are interested in. A cost function must then be *derived* from this model, such that we can optimize it.

Note at this point that these two paradigms do not cover all available data analysis methods, but is a useful conceptual distinction.

As we see, what separates Bayesian data analysis apart is not so much its goal, but rather its process. In the remainder of this section, and when we introduce the author-topic model, we shall see that the crux of Bayesian data analysis is that as we increase the complexity of the model, the difficulty of training the model grows very fast.

2.2 Probabilistic natural language processing

Some simple models that are related to the author-topic model will be introduced, and we discuss estimation in such models. The section introduces a lot of the notation, theory and concepts that we will be using when dealing with the author-topic model in section 2.4. We will cover graphical models and estimation of random variables and parameters, among other things.

Throughout, we have documents consisting of a set of words w_n for $n \in \{1, ..., N\}$ and $w_n \in \{1, ..., V\}$, i.e. a document of length N with a vocabulary of size V. At times, we also have documents $d \in \{1, ..., D\}$ and words labelled w_{dn}. A collection of documents is referred to as a *corpus*.

In the subsequent sections, we will discuss a simple multinomial model, then a Dirichlet-multinomial model, and finally a hierarchical version of the Dirichlet-multinomial. The solution to the inference problem in these models will underline the difficulties in training the author-topic model.

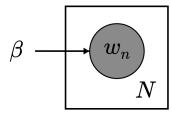


Figure 2.1: Graphical model of the multinomial. See section 2.2.1

2.2.1 The multinomial

Consider a simple statistical model such that $w_n \sim Mult(\beta)$ where $\beta \in \mathbb{R}^V$. This model posits a generative process such that each word in the document is drawn independently from the multinomial distribution, parametrized by β. In other words, the probability that word w_n takes on value v is $p(w_n = v|\beta) = Mult(w_n = v|\beta) = \beta_v$[1].

Figure 2.1 shows the multinomial in an illustration referred to as a *graphical model*. The grayed out circle represents an observed quantity (the words, w_n), the symbol with no circle represents a parameter (β), and the box tells us that this process repeats N times. As more complexity is added to these probabilistic models, more concepts in graphical models will be explained.

2.2.1.1 Maximum likelihood estimation in the multinomial

In our treatment of these probabilistic models, we will for the most part concern ourselves with estimation of random variables. β is not a random variable, but a parameter; however, we can still estimate β. We can compute a maximum likelihood estimate (MLE) of β by maximizing the log likelihood $\log p(w|\beta) = \log \prod_n p(w_n|\beta)$ w.r.t. β. The MLE turns out to be $\hat{\beta}_v = \frac{n_v}{V}$, where $n_v = \sum_n \mathbb{I}(w_n = v)$, i.e. the number of words that take on value v. We will also derive this estimate.

We write the log likelihood as

$$\log p(w|\beta) = \log \prod_n p(w_n|\beta) = \sum_n \log p(w_n|\beta) = \sum_n \log \beta_{w_n} = \sum_v n_v \log \beta_v,$$

and then we form the Lagrangian by adding the constraint $\sum_v \beta = 1$,

$$\ell(\beta, \lambda) = \sum_v n_v \log \beta_v - \lambda(\sum_v \beta - 1),$$

[1]Note that we implicitly assume that we draw *one* word from the multinomial. When n words are drawn from the multinomial, the equation is somewhat different.

where λ is the Lagrange multiplier. To obtain the estimate that maximizes this likelihood, we take the partial derivative of the Lagrangian w.r.t β and λ,

$$\frac{\partial \ell(\beta, \lambda)}{\partial \beta} = \frac{n_v}{\beta_v} - \lambda, \quad \frac{\partial \ell(\beta, \lambda)}{\partial \lambda} = -(\sum_v \beta_v - 1).$$

We let $\frac{\partial \ell(\beta,\lambda)}{\partial \beta} = \frac{\partial \ell(\beta,\lambda)}{\partial \lambda} = 0$, so

$$\frac{n_v}{\hat{\beta}_v} - \hat{\lambda} = -(\sum_v \hat{\beta}_v - 1) = 0.$$

We see that

$$n_v = \hat{\beta}_v \hat{\lambda},$$

so

$$\sum_v n_v = \sum_v \hat{\beta}_v \hat{\lambda} = V,$$
$$\hat{\lambda} \sum_v \hat{\beta}_v = V,$$
$$\hat{\lambda} = V,$$

and finally, since $\hat{\beta}_v = \frac{n_v}{\hat{\lambda}}$ and $\hat{\lambda} = V$, we have our MLE:

$$\hat{\beta}_v = \frac{n_v}{V}.$$

2.2.2 The Dirichlet-multinomial

We take the multinomial model from the previous section, and add a *Dirichlet prior* to the multinomial distribution over words. In figure 2.2, we can see that β now depends on a parameter $\eta \in \mathbb{R}_+^V$, and is thus a *random variable*, indicated by the circle around it. As η is the parameter of the prior distribution, we refer to it as a *hyperparameter*. We write this model as

$$\beta \sim Dir(\eta),$$
$$w_n \sim Mult(\beta).$$

We will discuss the significance of the Dirichlet prior in a later section, as it is a crucial element in the author-topic model. For now, we shall discuss estimation in the Dirichlet-multinomial model.

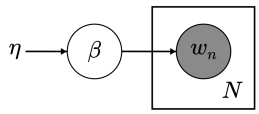


Figure 2.2: Graphical model of the Dirichlet-multinomial. See section 2.2.2

2.2.2.1 Estimating beta in the Dirichlet-multinomial

As mentioned, β is now a random variable. As the Dirichlet-multinomial is a very simple model, we are able to estimate β directly through manipulation of the joint probability distribution. We shall see later that it is not as easy to estimate the random variables of interest in the author-topic model, and we thus have to be more clever about it.

The joint probability distribution fully describes the model by assigning a probability to all combinations of all random variables, parameters and observed quantities. The Dirichlet-multinomial has the joint probability

$$p(w, \beta|\eta) = Dir(\beta|\eta) \prod_n Mult(w_n|\beta).$$

To estimate β, we would like to know the posterior distribution, which is the distribution over β conditioned on the data w. The posterior is equal to the joint distribution divided by the marginal likelihood,

$$p(\beta|w, \eta) = \frac{p(w, \beta|\eta)}{p(w|\eta)}.$$

The marginal likelihood is obtained by integrating the joint distribution over β,

$$p(w|\eta) = \int p(\beta, w|\eta) d\beta.$$

Unlike in the author-topic model, this integral can actually be solved analytically. However, we do not have to perform this calculation. Since we know that the Dirichlet and multinomial are *conjugate distributions*, we know that the posterior will be a re-parametrized Dirichlet. To find the parametrization

of the posterior, we continue by simplifying the joint distribution.

$$
\begin{aligned}
p(w, \beta|\eta) &= \frac{\Gamma(\sum_v \eta_v)}{\prod_v \Gamma(\eta_v)} \prod_v \beta_v^{\eta_v - 1} \prod_n \prod_v \beta_v^{\mathbb{I}(w_n = v)} \\
&= \frac{\Gamma(\sum_v \eta_v)}{\prod_v \Gamma(\eta_v)} \prod_v \beta_v^{\eta_v - 1} \prod_v \beta_v^{\sum_n \mathbb{I}(w_n = v)} \\
&= \frac{\Gamma(\sum_v \eta_v)}{\prod_v \Gamma(\eta_v)} \prod_v \beta_v^{\eta_v - 1 + \sum_n \mathbb{I}(w_n = v)} \\
&= \frac{\Gamma(\sum_v \eta_v)}{\prod_v \Gamma(\eta_v)} \prod_v \beta_v^{\eta_v - 1 + n_v}. \quad (2.1)
\end{aligned}
$$

We recognize the last line as an *unnormalized* Dirichlet distribution with parameters $\eta_v + n_v$. This means that the posterior is $p(\beta|w, \eta) = Dir(\eta_1 + n_1, ..., \eta_V + n_V)$. We can obtain a posterior estimate of β by using the mode, which is $mode(\beta_v|w, \eta) = \frac{\eta_v + n_v - 1}{\sum_{v'} \eta_{v'} - V}$.

2.2.2.2 Estimating eta in the Dirichlet-multinomial

We now have a new parameter in the model, namely η. As with the multinomial model, the fact that η is a parameter, and not a random variable, doesn't stop us from estimating it.

Minka et al. 2003 [Min03] show how to compute the MLE of η using a relatively simple optimization procedure similar to the one we used to estimate the β parameter in the multinomial model. This method has been applied in LDA to estimate the prior distributions, and we shall also use it for the author-topic model. We shall describe how this method is applied to the author-topic model in section 2.4.4, after we have described the inference algorithm.

Occasionally, one might put a prior on η, which is referred to as a *hyperprior*. This would make η a random variable which we could make inference on through Bayesian methods. It is, however, not usual to apply a hyperprior in LDA-like models.

2.2.2.3 The Dirichlet prior

Placing the Dirichlet prior on the multinomial allows us to include some "prior" information into the model. It is possible to make many different kinds of

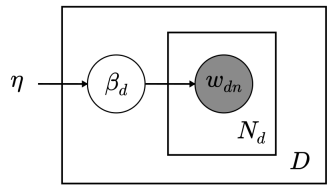


Figure 2.3: Graphical model of the hierarchical Dirichlet-multinomial. See section 2.2.3

constraints on the multinomial through the Dirichlet prior. For example, we can control how likely a word is a priori (before observing data).

We can control how sparse β is, in the sense that *most* of the density is allocated to one word, although all words still have non-zero density (in that sense it's not strictly speaking sparse). The Dirichlet encourages sparsity when $\eta_v < 1, \ \forall v$.

The Dirichlet prior can also be viewed as a way of *smoothing* the multinomial distribution. If we compare the multinomial MLE of β and the posterior mode of the Dirichlet-multinomial, we see that the latter is a smoothed version of the former, where every term gets a non-zero probability.

If $\eta_1 = \eta_2 = \cdots = \eta_V$, we say that the Dirichlet is symmetric. In this case all the words are equally likely a priori.

The Dirichlet and multinomial are conjugate distributions, which is very desirable in Bayesian models as it makes inference a great deal easier. We saw in section 2.2.2 that a multinomial with a Dirichlet prior resulted in a Dirichlet posterior with modified hyperparameters.

2.2.3 Hierarchical Dirichlet-multinomial

Adding an extra layer of complexity, we define a hierarchical model

$$\beta_d \sim Dir(\eta),$$

$$w_{dn} \sim Mult(\beta_d),$$

where each document has its private word distribution β_d, but they are all linked through the same prior. Figure 2.3 shows the model in graphical form.

Then the joint distribution becomes

$$p(w, \beta|\eta) = \prod_d Dir(\beta_d|\eta) \prod_n Mult(w_{dn}|\beta_d),$$

and by very similar derivation as with the Dirichlet-multinomial, we get that each document's word distribution follows a Dirichlet such that $p(\beta_d|w, \eta) = Dir(\eta_1 + n_{d1}, ..., \eta_V + n_{dV})$.

2.3 Topic models

We will now describe topic models in general in more detail. This discussion will be conducted in an intuitive manner to introduce the idea of topic modelling, as the author-topic model is essentially an extension of a standard topic model. We will also discuss some other extensions of the standard topic model.

An important concept in probabilistic topic models is the idea of a *latent variable*. Typically, a latent variable indicates something about the state of an observation in the dataset. In topic models, assigning a word in a document to a particular topic corresponds to estimating a latent variable. In the author-topic model we have an extra latent variable, corresponding to author assignment.

For each document a topic model assigns a weight to each latent topic that indicates to which degree the document expresses each topic. Similarly, for each topic a weight is assigned to each word in the vocabulary. When these weights represent probabilities, we are dealing with a probabilistic model. LDA is a probabilistic topic model, and so is the author-topic model.

Topic models are a type of *unsupervised* learning; we have no target that we are trying to predict, we are simply learning the structure of the data by making some assumptions about the data. As with any unsupervised method, estimating the quality of the model is a difficult task, as we cannot simply compute the error rate. In section 2.4.3, we discuss how to evaluate a trained topic model.

Intuitively, topic models are closely related to mixture models (i.e. clustering); they can be viewed as a type of "soft clustering" where each document exhibits a partial belonging to each ot the classes (topics). Models such as the topic model are thus referred to as *mixed membership models*.

It is common to use standard matrix decompositions of the *Term Frequency - Inverse Document Frequency* (TF-IDF) matrix of your corpus as topic mod-

elling techniques. Using Singular Value Decomposition (SVD) in this context is referred to as Latent Semantic Indexing (LSI), or Latent Semantic Analysis (LSA). It is also common to use Non-negative Matrix Factorization in this setting.

One of the reasons why Bayesian modelling is so compelling is that we can construct arbitrarily complex models relatively easily. It may not be clear how to extend an author-topic model from NMF or LSI, but with a Bayesian model this is quite easy. With that said, training such a model is often difficult.

There are many extensions of LDA, other than the author-topic model. There is LDA-HMM (Hsu et al. 2006 [HG06]) which relaxes the exchangeability of words (unigram model) in LDA and endows the model with a Markovian property; in other words, the order of the words is important in LDA-HMM. The dynamic topic model (DTM, Blei et al. 2006 [BL06]) gives a representation that illustrates how topics in the corpus have evolved over time. Supervised Latent Dirichlet Allocation (sLDA, Blei et al. 2008 [BMB08]) is a type of regression model that estimates a response (output) variable based on the topic assignments of the words in each document.

There is a topic model that is related to the author-topic model, but has little to do with Latent Dirichlet Allocation, called the structural topic model (STM, Roberts et al. 2013 [RSTA13]). STM draws the topic proportions, not from a Dirichlet, but from a logistic-normal generalized linear model based on document covariates. These covariates are first and foremost the word frequencies, and secondly metadata attached to the documents. So in other words, the STM generates topics based on arbitrary features in each document. If you add author labels as features, you get a type of author-topic model; if you add the date as a feature, you can get a type of dynamic topic model. The STM is thus related to the author-topic model in the sense that you can leverage your model by metadata, and learn something about the topical content in relation to that metadata.

In the next section, we describe the author-topic model and algorithms to train it. For a brief description of LDA, see appendix C.1 (it is recommended to read section 2.4 first).

2.4 The author-topic model

As in the previous section, we describe the model by a set of linked probability distributions,

$$\theta_a \sim Dir(\alpha),$$
$$\beta_k \sim Dir(\eta),$$
$$x_{dn} \sim Unif(\frac{1}{|A_d|}),$$
$$z_{dn} \sim Mult(\theta_a, x_{dn} = a),$$
$$w_{dn} \sim Mult(\beta_k, z_{dn} = k),$$

where $x_{dn} \sim Unif(\frac{1}{|A_d|})$ means that the author of a word w_{dn} is drawn uniformly with probability one over the number of authors in document d. $z_{dn} \sim Mult(\theta_a, x_{dn} = a)$ means that we draw z_{dn} from θ_a assuming that $x_{dn} = a$. We describe the intuition behind each of these parameters.

- θ_a is a probability vector such that θ_{ak} is the probability that author a writes about topic k.
- β_k is, likewise, a probability vector such that the probability that word v is used in topic k is equal to β_{kv}.
- x_{dn} is a latent variable that indicates which author is responsible for word n in document d.
- z_{dn} is also a latent variable and indicates which topic generated word n in document d.

As with the simpler models in section 2.2, we illustrate the author-topic model using a graphical model in figure 2.4.

We can interpret edges in the graph as dependence between two variables, e.g. z_{dn} depends on θ_a. Strictly speaking, the absence of an edge represents conditional independence, e.g. when conditioned on z_{dn}, w_{dn} is independent of θ_a, i.e. $p(w_{dn}|\beta_k, z_{dn} = k)$ does not depend on θ_a.

Intuitively, the author-topic model can be viewed as a process that generates words in a document based on the authors of that document. Note that this is more of a thought experiment that gives an intuitive understanding of the model, rather than a realistic view of the model.

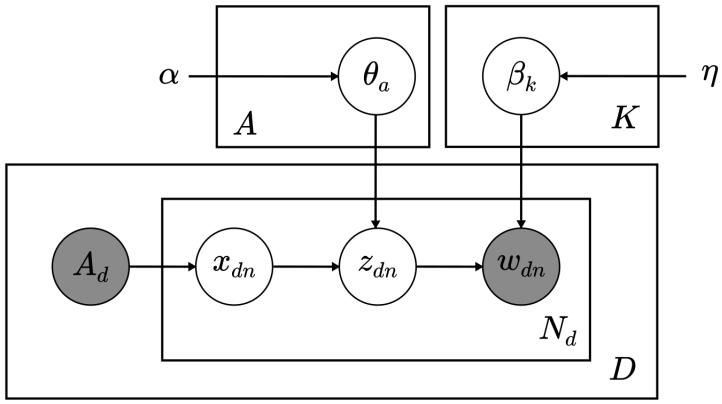


Figure 2.4: Graphical model of the author-topic model. See section 2.4 for details.

- For each author $a \in \{1, ..., A\}$ draw $\theta_a \sim Dir(\alpha)$.
- For each topic $k \in \{1, ..., K\}$ draw $\beta_k \sim Dir(\eta)$.
- For each document $d \in \{1, ..., D\}$:
  - Given document d's authors, A_d.
  - For each word in the document $n \in \{1, ..., N_d\}$.
    - Assign an author to the current word by drawing $x_{dn} \sim Unif(\frac{1}{|A_d|})$.
    - Conditioned on x_{dn}, assign a topic by drawing $z_{dn} \sim Mult(\theta_a)$.
    - Conditioned on z_{dn}, choose a word by drawing $w_{dn} \sim Mult(\beta_k)$.

A great deal of complexity is added, going from the hierarchical Dirichlet-multinomial in section 2.2.3 to the author-topic model. However, we see that all the basic building blocks are contained in the hierarchical Dirichlet-multinomial. Rather than drawing words from a multinomial that is conditioned on the document, we now condition it on a topic, which in turn is drawn from a Dirichlet-multinomial.

Note that when each document is attributed to exactly one author, and each author is only attributed to one document, then the author-topic model is equivalent to a standard topic model.

2.4.1 Inference

As in section 2.2, now that we have described the model we will now concern ourselves with estimating the random variables and parameters of the model. In Bayesian models such as the author-topic model, this is referred to as *inference*. In particular, we are interested in inferring θ and β.

The joint probability distribution of the author-topic model is

$$\begin{aligned} &p(\theta, \beta, z, x, w|\alpha, \eta, A) \\ &\qquad = p(\theta|\alpha)p(\beta|\eta)p(x|A)p(z|x,\theta)p(w|z,\beta) \\ &= \prod_{a=1}^{A} Dir(\theta_a|\alpha) \prod_{k=1}^{K} Dir(\beta_k|\eta) \prod_{d=1}^{D} \prod_{n=1}^{N_d} Unif(x_{dn}|A_d) \\ &\qquad Mult(z_{dn}|\theta_a, x_{dn} = a) Mult(w_{dn}|\beta_k, z_{dn} = k). \end{aligned} \tag{2.2}$$

The posterior distribution is

$$p(\theta, \beta, z, x|w, \alpha, \eta, A) = \frac{p(\theta, \beta, z, x, w|\alpha, \eta, A)}{p(w|\alpha, \eta, A)},$$

but as mentioned in section 2.2.2, we cannot compute the marginal likelihood in the author-topic model analytically. As is quite common with Bayesian models, the marginal likelihood $p(w|\alpha, \eta, A)$ contains an integral that is not solvable in closed form. This integral can approximated by numerical methods, but such approximations are exponential in complexity, which renders such solutions practically useless. In this situation, we say that the posterior is *intractable*.

The aforementioned problem is the crux of Bayesian inference. There exist many different methods of approximating the posterior (without evaluating the marginal likelihood). Choosing an inference method may be critical as both computational cost and accuracy may vary by a lot.

2.4.1.1 Exchangeability

A very important concept in Bayesian modelling is exchangeability. We mention it now because we require the joint distribution to discuss exchangeability.

A model is exchangeable if the joint distribution is invariant to permutation of the indices of the variates. For example, consider a model that has some

random variables θ_i, $i \in [1, N]$. If $p(\theta_1, ..., \theta_N) = p(\theta_{\pi(1)}, ..., \theta_{\pi(N)})$ where π is any random permutation of the indices from 1 to N, then this model is exchangeable.

If we look at the joint distribution of the author-topic model in equation 2.2, we can imagine permutating all the indices a, k, d and n, and it is clear that this would not change the value of the joint distribution. This means that the author-topic model is exchangeable. We will discuss one consequence of this later when we introduce an online algorithm.

2.4.1.2 Approximate inference in the author-topic model

We are not able to derive the solution to the model analytically, as we did in section 2.2, so we need to approximate the posterior somehow. There are two major candidates for this task for the author-topic model: Gibbs sampling and variational Bayes.

We can liken Gibbs sampling to the method we used to obtain a solution to the Dirichlet-multinomial model in section 2.2.2. The Dirichlet-multinomial is a conjugate model, so that the posterior is a re-parametrized Dirichlet. In the author-topic model, we have conditional conjugacy, such that $p(z_{dn}|z_{-dn}, x, \theta, \beta, \alpha, \eta, A)$ and $p(x_{dn}|x_{-dn}, z, \theta, \beta, \alpha, \eta, A)$ are both Dirichlet. This means that we can draw samples from these conditional posteriors; in the Dirichlet-multinomial we draw one sample and we are done, but in Gibbs sampling, we alternately draw samples of x_{dn} and z_{dn} until convergence.

Rosen-Zvi et al. 2004 [RZGSS04] 2010 [RZGSS05] originally introduced the author-topic model and applied Gibbs sampling. They used a method referred to as *blocking* Gibbs sampling, where one relaxes the independence assumption between x and z and instead draws samples from $p(z_{dn}, x_{dn}|z_{-dn}, x_{-dn}, \theta, \beta, \alpha, \eta, A)$. Blocking improves convergence as we make less assumptions about the form of the posterior. As we shall see later, we apply this blocking concept in a variational Bayes setting.

The second method, variational Bayes, we liken to our solution to the multinomial model in section 2.2.1. In the multinomial model, we derived a closed form solution by directly optimizing the log likelihood. As discussed above, we do not have access to the likelihood $p(w|\alpha, \eta, A)$. The idea in VB is thus to introduce an approximate distribution so that we are able to maximize a lower bound on the log likelihood. We describe this solution in detail in section 2.4.2.

VB and Gibbs sampling: advantages and disadvantages: A key difference between VB and Gibbs sampling is that in the former we optimize a lower bound, and therefore we never approach the true solution, while in the latter we converge towards the true posterior given enough iterations. VB converges faster than Gibbs sampling at first, but eventually is overtaken by Gibbs sampling. Therefore, if the accuracy attained with VB is not enough, and one is willing to wait for the Gibbs sampler to converge to the desired accuracy, then that may be the right choice. However, as we care a lot about computation time and scalability, and VB has shown to give satisfactory results for LDA-like models, it is fitting for our purposes.

Another advantage with VB is that it is easy to derive an online algorithm, as done in Hoffman et al. 2013 [HBWP13]. We derive an online algorithm in section 2.4.2.5.

Collapsed approximation: In collapsed Gibbs sampling, we only sample the latent variables by integrating out all other random variables. Collapsed Gibbs sampling tends to be more efficient than its standard version as we are sampling in a lower dimensional space (Murphy 2012 [Mur12], p. 841).

Teh et al. 2007 [TNW07] introduced collapsed variational Bayes (CVB) to train LDA, applying the same principle as in collapsed Gibbs sampling, and reported faster convergence than standard VB.

Ngo et al. 2016 [NEFY16] presented CVB for the author-topic model, and applied blocking in their model as well. CVB would be the natural next step in an effort to improve the performance of the author-topic model algorithm. See section B for a brief description of author-topic model training using CVB.

2.4.2 Variational Bayes algorithm

We pose the *variational distribution* $q(\theta, \beta, z, x)$ which approximates the posterior. The variational distribution is fully factorized,

$$\begin{aligned} q(\theta, \beta, z, x) &= q(\theta|\gamma)q(\beta|\lambda)q(x|\mu)q(z|\phi) \\ &= \prod_a q(\theta_a|\gamma_a) \prod_k q(\beta_k|\lambda_k) \prod_{d,n} q(x_{dn}|\mu_{dn}) \prod_{d,n} q(z_{dn}|\phi_{dn}) \\ = \prod_a Dir(\theta_a|\gamma_a) &\prod_k Dir(\beta_k|\lambda_k) \prod_{d,n} Mult(x_{dn}|\mu_{dn}) \prod_{d,n} Mult(z_{dn}|\phi_{dn}). \end{aligned} \quad (2.3)$$

By introducing the variational parameters γ, λ, μ and ϕ, we make the assumption that the model parameters are *independent* (note that z_{dn} does not depend on θ or x in q). This is not necessarily a realistic assumption, but one that lets us approximate the solution.

Defining a fully factorized (all variables are independent) approximate distribution is the standard method in variational Bayes. However, we shall use a slightly different variational distribution. We describe this approximate distribution now, and get into the details of why this particular distribution was chosen in section 2.4.2.7.

We define the variational distribution as

$$\begin{aligned} q(\theta, \beta, z, x) &= q(\theta|\gamma)q(\beta|\lambda)q(x, z|\phi) \\ &= \prod_a q(\theta_a|\gamma_a) \prod_k q(\beta_k|\lambda_k) \prod_{d,n} q(x_{dn}, z_{dn}|\phi_{dn}) \\ &= \prod_a Dir(\theta_a|\gamma_a) \prod_k Dir(\beta_k|\lambda_k) \prod_{d,n} q(x_{dn}, z_{dn}|\phi_{dn}), \end{aligned} \quad (2.4)$$

where $q(x_{dn} = a, z_{dn} = k|\phi_{dn}) = \phi_{dnak}$. In this approximation, we assume that x and z are dependent. This leads to an inference algorithm that has better computational properties than the standard formulation. This formulation is essentially *blocking variational Bayes*, as discussed in section 2.4.1.2.

Note that

$$\phi_{dnak} = \begin{cases} q(z_{dn} = k, x_{dn} = a), & \text{if } a \in A_d, \\ 0 & \text{otherwise,} \end{cases}$$

and $\sum_k \sum_a \phi_{dnak} = 1$.

Figure 2.5 shows a graphical model, as in figure 2.4, of the model $q(\theta, \beta, z, x)$.

Even though the standard VB algorithm was not used in the end, it was the logical first step towards variational inference. Therefore, the standard VB algorithm is described in appendix A, for completeness. In the next section, we continue with the blocking VB algorithm.

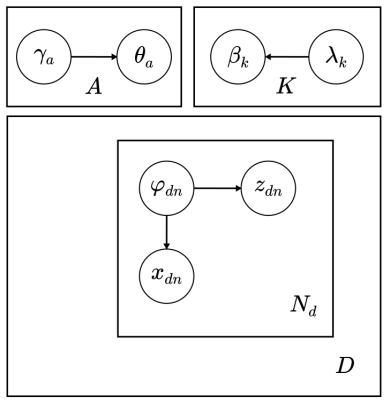


Figure 2.5: Graphical model of variational approximation. See section 2.4.2 for details.

2.4.2.1 Obtaining the lower bound

It can be shown that

$$\log p(w|\alpha, \eta, A) \geq E_q[\log p(\theta, \beta, x, z, w|\alpha, \eta, A)] - E_q[\log q(\theta, \beta, z, x)],$$

where E_q is the expectation over $q(\theta, \beta, z, x)$. The right-hand-side of the inequality above is denoted $\mathcal{L}$ and is referred to as the *lower bound* on the marginal likelihood. The goal in variational inference is to make this lower bound as tight as possible, thereby maximizing the marginal likelihood of the data. This is possible because the lower bound is defined by only things that we can easily compute, that is the joint probability and the variational distribution. See appendix C.2 for more information on the lower bound, including derivation of the inequality above.

As we shall see later, we employ the simplest possible optimization scheme to maximize the lower bound, that is, we take the derivative w.r.t. each parameter, equate it to zero, and isolate the parameter. We do this for each index in each variational parameter until the lower bound converges.

We obtain the lower bound to the marginal log likelihood.

$$\begin{aligned}\log p(w|\alpha,\eta,A) &\geq E_q[\log p(\theta,\beta,x,z,w|\alpha,\eta,A)] - E_q[\log q(\theta,\beta,z,x)] \\ &= \mathcal{L}(\gamma,\lambda,\phi) \\ = \sum_a E_q[\log Dir(\theta_a|\alpha)] &+ \sum_k E_q[\log Dir(\beta_k|\eta)] + \sum_{d,n} E_q[\log Unif(x_{dn}|A_d)] \\ &+ \sum_{d,n}\sum_{a\in A_d} E_q[\log Mult(z_{dn}|\theta_a)] + \sum_{d,n,k} E_q[\log Mult(w_{dn}|\beta_k)] \\ - \sum_a E_q[\log Dir(\theta_a|\gamma_a)] - \sum_k E_q[\log Dir(\beta_k|\lambda_k)] &- \sum_{d,n} E_q[\log q(x_{dn},z_{dn}|\phi_{dn})]\end{aligned} \tag{2.5}$$

We evaluate each of the expectations in the lower bound given above. To evaluate $E_q[\log Dir(\theta_a|\alpha)]$ we write the Dirichlet in the form

$$Dir(\theta_a|\alpha) = \exp\left\{\left(\sum_k(\alpha_k-1)\log\theta_{ak}\right) + \log\Gamma\left(\sum_k\alpha_k\right) - \sum_k\log\Gamma(\alpha_k)\right\},$$

which is in the form of an exponential family distribution. Note that we are not assuming symmetric priors here, i.e. we are not assuming that $\alpha_k = \alpha\ \forall k$. Blei et al. 2003 [BNJ03] show that $E_q[\log\theta_{ak}] = \Psi(\gamma_{ak}) - \Psi(\sum_{k'}\gamma_{ak'})$, where Ψ is the digamma function, which is the first derivative of $\log\Gamma$. So the first expectation is

$$E_q[\log Dir(\theta_a|\alpha)] = \sum_k((\alpha_k-1)E_q[\log\theta_{ak}]) + \log\Gamma\left(\sum_k\alpha_k\right) - \sum_k\log\Gamma(\alpha_k).$$

The rest of the Dirichlet expectations are derived in the same manner. We have the expectations of the multinomials,

$$\begin{aligned}E_q[\log Mult(z_{dn}|\theta)] &= \sum_k\sum_{a\in A_d}\int q(x_{dv}=a,z_{dv}=k)q(\theta_{ak})\log p(z_{dn}|\theta)d\gamma \\ &= \sum_k\sum_{a\in A_d}\phi_{dnak}E_q[\log\theta_{ak}],\end{aligned} \tag{2.6}$$

because $\int q(\theta_{ak}) \log p(z_{dn}|\theta) d\gamma = E_q[\log \theta_{ak}]$.

$$E_q[\log Mult(w_{dn}|\beta)] =$$
$$\sum_{a \in A_d} \sum_k \sum_v \int \mathbb{I}(w_{dn} = v) q(x_{dn} = a, z_{dv} = k) q(\lambda_{kv}) \log p(w_{dn} = v|\beta_{kv}) d\lambda$$
$$= \sum_{a \in A_d} \sum_k \sum_v \mathbb{I}(w_{dn} = v) \phi_{dnak} E_q[\log \beta_{kv}], \quad (2.7)$$

and similarly,

$$E_q[\log q(x_{dn}, z_{dn}|\phi_{dn})] = \sum_{a \in A_d} \sum_k \phi_{dnak} \log \phi_{dnak}.$$

Finally, we have

$$E_q[\log Unif(x_{dn}|A_d)] = \log \frac{1}{|A_d|}.$$

Recall that we compute the expectations of $\log \theta_a$ and $\log \beta_k$ using the digamma function, as described above.

Before we write out the lower bound using the computed expectations, we introduce a variable n_{dv}, which indicates how many times dictionary word v is observed in document d. The n_{dv} variable simplifies the equations a bit, first by eliminating the need of the identity function, second by letting us loop over v rather than n. We also collect some of the terms in the equation, to make it a bit shorter. Finally, we add constraints such that $\sum_{a \in A_d} \sum_k \phi_{dvak} = 1 \ \forall d, v$ with corresponding Lagrange multipliers ℓ_{dv}.

$$\mathcal{L}(\gamma, \lambda, \phi) =$$
$$\sum_a \left(\sum_k (\alpha_k - \gamma_{ak}) E_q[\log \theta_{ak}] - \log \Gamma(\sum_k \gamma_{ak}) + \sum_k \log \Gamma(\gamma_{ak}) + \log \Gamma(\sum_k \alpha_k) - \sum_k \log \Gamma(\alpha_k) \right)$$
$$+ \sum_k \left(\sum_v (\eta_v - \lambda_{kv}) E_q[\log \beta_{kv}] - \log \Gamma(\sum_v \lambda_{kv}) + \sum_v \log \Gamma(\lambda_{kv}) + \log \Gamma(\sum_v \eta_v) - \sum_v \log \Gamma(\eta_v) \right)$$
$$+ \sum_{d,v} n_{dv} \sum_{a \in A_d} \sum_k \phi_{dvak} (E_q[\log \theta_{ak}] + E_q[\log \beta_{kv}] - \log \phi_{dvak})$$
$$+ \sum_{d,v} n_{dv} \frac{1}{|A_d|} + \sum_{d,v} \ell_{dv} \left(\left(\sum_{a \in A_d} \sum_k \phi_{dvak} \right) - 1 \right) \quad (2.8)$$

2.4.2.2 Obtaining the update equations

To maximize the lower bound, we apply a simple coordinate ascent method: take the derivative of $\mathcal{L}(\gamma, \lambda, \phi)$ w.r.t. each of the variational parameters, equate it to zero, and isolate the corresponding parameter.

The lower bound depends on ϕ_{dvak} via the terms

$$\mathcal{L}_{[\phi_{dvak}]} = n_{dv}\phi_{dvak}(E_q[\log\theta_{ak}] + E_q[\log\beta_{kv}] - \log\phi_{dvak}) + \ell_{dv}\left(\left(\sum_{a'\in A_d}\sum_{k'}\phi_{dva'k'}\right) - 1\right), \quad (2.9)$$

and the partial derivative of the lower bound w.r.t. ϕ_{dvak} is

$$\frac{\partial\mathcal{L}}{\partial\phi_{dvk}} = n_{dv}E_q[\log\theta_{ak}] + n_{dv}E_q[\log\beta_{kv}] - n_{dv}\log\phi_{dvk} - 1 + \ell. \quad (2.10)$$

Setting the derivative equals to zero and isolating ϕ_{dvak} yields

$$\phi_{dvk} \propto \exp\{E_q[\log\theta_{ak}] + E_q[\log\beta_{kv}]\}.$$

The normalization constant (note the "proportional to" in the equation above) is $\exp(1-\ell)$, although we just normalize ϕ_{dvak} directly, as we shall see later.

The lower bound depends on γ via the terms

$$\mathcal{L}_{[\gamma_{ak}]} = (\alpha_k - \gamma_{ak})(\Psi(\gamma_{ak}) - \Psi(\sum_{k'}\gamma_{ak'})) - \log\Gamma(\sum_{k}\gamma_{ak}) + \log\Gamma(\gamma_{ak}) + \sum_{d\in D_a}\sum_{v} n_{dv}\phi_{dvak}(\Psi(\gamma_{ak}) - \Psi(\sum_{k'}\gamma_{ak'})). \quad (2.11)$$

where we have defined the set $D_a = \{d \mid a \in A_d\}$. The partial derivative is

$$\frac{\partial \mathcal{L}}{\partial \gamma_{ak}} = (\alpha_k - \gamma_{ak})(\Psi'(\gamma_{ak}) - \Psi'(\sum_{k'} \gamma_{ak'})) + \sum_{d \in D_a} \sum_{v} n_{dv} \phi_{dvak} \left((\Psi'(\gamma_{ak}) - \Psi'(\sum_{k'} \gamma_{ak'})) \right), \quad (2.12)$$

This yields the update equation

$$\gamma_{ak} = \alpha_k + \sum_{d \in D_a} \sum_{v} n_{dv} \phi_{dvak}.$$

The lower bound depends on λ via the terms

$$\mathcal{L}_{[\lambda_{kv}]} = (\eta_v - \lambda_{kv})(\Psi(\lambda_{kv}) - \Psi(\sum_{v'} \lambda_{kv'})) - \log \Gamma(\sum_{v} \lambda_{kv}) + \log \Gamma(\lambda_{kv}) + \sum_{d} n_{dv} \sum_{a \in A_d} \phi_{dvak} (\Psi(\lambda_{kv}) - \Psi(\sum_{v'} \lambda_{kv'})), \quad (2.13)$$

and the partial derivate is

$$\frac{\partial \mathcal{L}}{\partial \lambda_{kv}} = (\eta_v - \lambda_{kv})(\Psi'(\lambda_{kv}) - \Psi'(\sum_{v'} \lambda_{kv'})) + \sum_{d} n_{dv} \sum_{a \in A_d} \phi_{dvak} (\Psi'(\lambda_{kv}) - \Psi'(\sum_{v'} \lambda_{kv'})), \quad (2.14)$$

which yields the last update equation,

$$\lambda_{kv} = \eta_v + \sum_{d} n_{dv} \sum_{a \in A_d} \phi_{dvak}.$$

2.4.2.3 Computing the lower bound

To measure convergence of the algorithm, we compute the lower bound $\mathcal{L}(\gamma, \lambda, \phi)$. Using that

$$\phi_{dvak} = \frac{\exp\{E_q[\log \theta_{ak}] + E_q[\log \beta_{kv}]\}}{\sum_k \sum_{a \in A_d} \exp\{E_q[\log \theta_{ak}] + E_q[\log \beta_{kv}]\}}$$

we can simplify the computation, by reducing the third line in equation 2.8, as follows:

$$\begin{aligned}\sum_{d,v} n_{dv} \sum_{a \in A_d} \sum_{k} \phi_{dvak} (E_q[\log \theta_{ak}] + E_q[\log \beta_{kv}] - \log \phi_{dvak}) \\ = \sum_{d} \sum_{v} n_{dv} \log \sum_{k} \sum_{a \in A_d} \exp\{\theta_{ak} + \beta_{kv}\} \\ = \sum_{d} \sum_{v} n_{dv} \log \sum_{k} \exp\{\beta_{kv}\} \sum_{a \in A_d} \exp\{\theta_{ak}\}. \quad (2.15)\end{aligned}$$

Of course, we do not need to worry about the Lagrange multipliers, as their contribution to the lower bound can be assumed to be zero. Computing the lower bound thus becomes easy.

2.4.2.4 VB algorithm

Algorithm 1 shows pseudo-code for training the author-topic model using the VB updates we have derived in the previous sections. The outline of the algorithm can loosely be described as follows.

- Initialize parameters.
- Until lower bound converges:
  - For all combinations of d, v, a and k, update ϕ, γ and λ.

As indicated in the pseudo-code, updating the local variables (ϕ and γ) is often referred to as the M-step, while updating the global variables (λ) is referred to as the E-step.

The problem with algorithm 1 is that it requires us to store all variables in memory, and ϕ can get quite huge ($D \times V_d \times A_d \times K$, sparse matrix). Luckily, the online algorithm, presented in the next section, alleviates our memory troubles.

2.4.2.5 Online VB

Hoffman et al. 2013 [HBWP13] describe an online VB algorithm referred to as *stochastic variational inference*. We apply this method to our VB algorithm for

Algorithm 1 Training the author-topic model using VB.

function AT-VB(w_{dn}, A, K, α, η, τ_1, τ_2)
Initialize γ and λ randomly according to a gamma distribution, and compute $E_q[\log \theta_{ak}]$ and $E_q[\log \beta_{kv}]$.
Compute $\mathcal{L}$.
repeat
Set $\mathcal{L}_{prev} := \mathcal{L}$.
M-step.
for $d = 1$ to D **do**
for $v = 1 \in V_d$ **do**
for $k = 1$ to K **do**
for $a \in A_d$ **do**
$\phi_{dvak} \propto \exp\{E_q[\log \theta_{ak}] + E_q[\log \beta_{kv}]\}$.
end for
end for
Normalize ϕ_{dvak} to sum to 1 over k and a:
$\phi_{dvak} := \phi_{dvak} / \left(\sum_{a \in A_d} \sum_{k=1}^{K} \phi_{dvak}\right)$
end for
end for
for $a = 1$ to A **do**
for $k = 1$ to K **do**
$\gamma_{ak} := \alpha_k + \sum_{d \in D_a} \sum_v n_{dv} \phi_{dvak}$.
end for
end for
Compute $E_q[\log \theta_{ak}]$, as γ has been updated.
E-step.
for $k = 1$ to K **do**
for $v = 1$ to V **do**
$\lambda_{kv} := \eta_v + \sum_d n_{dv} \sum_{a \in A_d} \phi_{dvak}$.
end for
end for
Compute $E_q[\log \beta_{kv}]$, as λ has been updated.
Compute $\mathcal{L}$.
until $(\mathcal{L} - \mathcal{L}_{prev})/\mathcal{L}_{prev} < \tau_1$
end function

the author-topic model. The online algorithm allows us to let the documents come in a stream, so that when we have looked at one document we can discard it. It also allows us to keep learning if we obtain more data.

Simply put, we compute an estimate of λ for document t as $\tilde{\lambda}_{kv} := \eta_v + Dn_{tv} \sum_{a \in A_t} \phi_{tvak}$, as if this document represented the entire corpus. Next, we interpolate between this "local" variable $\tilde{\lambda}$ and the "global" variable λ, as $\lambda := (1 - \rho_t)\lambda + \rho_t \tilde{\lambda}$, where $\rho_t = (\tau_0 + t)^{-\kappa}$. $\tau_0 \geq 0$ is referred to as the *offset* and $\kappa \in (0.5, 1]$ as the *decay*.

We treat γ similarly as λ in the online algorithm. The reason these two parameters are treated differently than ϕ is that they both require a sum over documents, which of course is not possible with an online algorithm.

This online algorithm is based on the idea that the lower bound can be written in terms of each document (see appendix C.3),

$$\mathcal{L}(\gamma, \lambda, \phi) = \sum_d \mathcal{L}_d(\gamma, \lambda, \phi).$$

We can then find combinations of ϕ and γ that are locally optimal (i.e. optimal for $\mathcal{L}_d(\gamma, \lambda, \phi)$), and update λ accordingly.

Pseudo-code for the online algorithm can be seen in algorithm 2. As mentioned in the previous section, this algorithm has much more manageable memory requirements than the standard algorithm (also referred to as "batch VB"). We must store a ϕ matrix that is $V_d \times A \times K$, which will be discarded as soon as we move on to the next document. We have to store γ and λ, but their size is quite manageable ($A \times K$ and $K \times V$, respectively).

Updating the model w.r.t. all documents in the corpus is referred to as a *pass* over the corpus. The number of M-steps, i.e. how often the inner loop is repeated, is referred to as an *iteration* over a document. Conversely, in the offline algorithm, updating all the variational parameters once is called an iteration. We will be using these terms later.

It was mentioned previously that the author-topic model is exchangeable. If this were not the case, we might not be able to formulate an online algorithm because the state of a single document would depend on other factors that cannot be accessed in that document. Clearly, this algorithm suffers from this problem to some extent anyway, but in a way where it is possible to formulate an online algorithm.

Algorithm 2 Online training of the author-topic model using VB.

function ONLINE-AT-VB(w_{dn}, A, K, α, η, τ, τ_0, κ)
Initialize γ and λ randomly according to a gamma distribution, and compute $E_q[\log \theta_{ak}]$ and $E_q[\log \beta_{kv}]$.
for $t = 1$ to ∞ **do**
Set $\rho_t := (\tau_0 + t)^{-\kappa}$.
repeat
for $v = 1 \in V_t$ **do**
for $k = 1$ to K **do**
for $a \in A_t$ **do**
$\phi_{vak} \propto \exp\{E_q[\log \theta_{ak}] + E_q[\log \beta_{kv}]\}$.
end for
end for
Normalize ϕ_{vak} to sum to 1 over k and a:
$\phi_{vak} := \phi_{vak} / \left(\sum_{a \in A_t} \sum_{k=1}^{K} \phi_{vak} \right)$
end for
for $a \in A_t$ **do**
for $k = 1$ to K **do**
$\tilde{\gamma}_{ak} := \alpha_k + |D_a| \sum_{v \in V_t} n_{tv} \phi_{vak}$.
end for
end for
Compute $E_q[\log \theta_{ak}]$ based on a temporary estimate of γ.
$\hat{\gamma} := (1 - \rho_t)\gamma + \rho_t \tilde{\gamma}$.
until Average absolute change in $\tilde{\gamma}$ is less than τ.
for $k = 1$ to K **do**
for $v = 1 \in V_t$ **do**
$\tilde{\lambda}_{kv} := \eta_v + D n_{tv} \sum_{a \in A_t} \phi_{vak}$.
end for
end for
$\gamma := \hat{\gamma}$
$\lambda := (1 - \rho_t)\lambda + \rho_t \tilde{\lambda}$
Compute $E_q[\log \beta_{kv}]$ based on new estimate of λ.
end for
end function

We also train on *mini-batches*, as described in Hoffman et al. 2013 [HBWP13]. The optimal $\tilde{\lambda}_S$ are found based on a batch of size S, and then scaled by $1/S$ and interpolated as usual,

$$\lambda := (1 - \rho_t)\lambda + \frac{\rho_t}{S}\tilde{\lambda}_S.$$

2.4.2.6 Asymptotic complexity

In appendix C.4, we show that the asymptotic complexity of the algorithm and bound evaluation are $O(MV_dA_dK + V_dA_dK + KV)$ and $O(V_dKA_d + KV + AK)$, respectively. Note that M is the number of iterations per document. These results are useful both for comparing with other algorithms, and to use as a sanity check when testing the scalability of the algorithm empirically, which we will do in section 4.

We see that we need some data structure to store all the A_d and D_a sets in. The memory complexity of these are $O(\sum_d A_d)$ and $O(\sum_a D_a)$. That makes the memory complexity of the algorithm $O(V_dA_dK + AK + KV + \sum_d A_d + \sum_a D_a)$, where the first three terms are the size of ϕ_t, the size of γ, and the size of λ.

2.4.2.7 Advantages with blocking VB

As mentioned in section 2.4.2, we factorize the variational distribution using the blocking method, where we relax the assumption that the latent variables are independent. This lead to the algorithm we saw in the previous section, while a standard VB formulation leads to the algorithm in section A.

There are two reasons why blocking VB is preferred, which is that it

- avoids the need to store the variational analogue of the latent variables (ϕ) by computing them implicitly in the other updates,
- and allows vectorization of the variational variable updates.

Avoiding the need to store ϕ brings the memory complexity of the algorithm from $O(V_dA_dK + AK + KV + \sum_d A_d + \sum_a D_a)$ down to $O(AK + KV + \sum_d A_d + \sum_a D_a)$. The standard VB formulation of the author-topic model does not lend itself to this optimization, but luckily the blocking VB algorithm does.

The updates in blocking VB are somewhat simpler than in standard VB, and together with the implicit computation of ϕ, this enables vectorization of the γ and λ updates. Using standard libraries for linear algebra, we can greatly speed up the updates by vectorizing them.

In theory, blocking VB should improve convergence as it makes less assumptions about the form of the posterior. As we shall see in section 4, the blocking VB algorithm has a slight advantage.

See appendix C.5 for details on the vectorized updates.

2.4.3 Evaluating a trained model

In this section, we discuss some measures to evaluate the quality of a trained model.

2.4.3.1 Perplexity

The perplexity of a dataset $\mathcal{D}$ is given by

$$perplexity(\mathcal{D}) = \exp\left(-\frac{1}{\sum_{d\in\mathcal{D}} N_d}\sum_{d\in\mathcal{D}} \mathcal{L}(\gamma, \lambda, \phi)\right), \tag{2.16}$$

where $\mathcal{L}(\gamma, \lambda, \phi)$ is evaluated as discussed in section 2.4.2.3.

The lower bound is an estimate of the *predictive performance* of the model (the log likelihood is at least as high as the lower bound). Thereby, the perplexity is a measure of predictive performance as well.

2.4.3.2 Conditional likelihood

Occasionally, the conditional likelihood is used as a measure of model performance. The conditional likelihood can be computed as

$$\begin{aligned} p(w|\theta,\beta,A) &= \prod_d \prod_n p(w_{dn}|\theta,\beta,A_d) \\ &= \prod_d \prod_n \sum_k \sum_{a\in A_d} p(w_{dn}, z_{dn}=k, x_{dn}=a|\theta,\beta,A_d) \\ &= \prod_d \prod_n \sum_k \sum_{a\in A_d} p(w_{dn}|z_{dn}=k,\beta_k)p(z_{dn}=k|x_{dn}=a,\theta_a)p(x_{dn}=a|A_d) \\ &= \prod_d \frac{1}{|A_d|} \prod_n \sum_k \sum_{a\in A_d} \theta_{ak}\beta_{kw_{dn}}. \end{aligned} \quad (2.17)$$

As estimates of θ and β, we can use normalized versions of γ and λ to compute the conditional likelihood.

The conditional likelihood can be useful to compare to a Gibbs sampler, for example. In Teh et al. 2007 [TNW07], the conditional likelihood (which they refer to as "word probabilities") is used to compare CVB and collapsed Gibbs sampling performance when training LDA.

The conditional likelihood can also be used in the perplexity, in place of the lower bound.

2.4.3.3 Topic coherence

While the lower bound (and thereby the perplexity) and the conditional likelihood tell us how well the algorithm is converging, they generally don't tell us much about the quality of the topics (Chang et al. 2009 [CGWB09]).

Mimno et al. 2011 [MWT+11] had domain experts annotate the quality of topics from a trained LDA model. They then devise a measure that they find to be strongly correlated with the human annotator's judgement of topic quality. This measure is the *topic coherence*, computed as

$$C(t,V^{(t)}) = \sum_{m=2}^{M}\sum_{l=1}^{m-1} \log \frac{D(v_m^{(t)}, v_l^{(t)})+1}{D(v_l^{(t)})},$$

where $D(v, u)$ is the number of documents where words v and u occur together in (the *co-document frequency*), and $D(v) = D(v, v)$ is the frequency of word v.

How does model fit not imply topic quality? The notion that the predictive power (like the lower bound) does not correlate with the quality of the model may seem counter intuitive, so an explanation shall be provided.

Say we have trained several models with different random initializations. Naturally, the lower bound will convey whether we need to keep training, or if the model has indeed converged. Once all the models have been trained to convergence, comparing the predictive power does not correlate strongly with human judgement of the resulting topics.

These topic models essentially just do dimensionality reduction of the input space, like a type of probabilistic matrix factorization. In fact, LDA is closely related to mPCA (multinomial Principal Component Analysis), a particular flavour of a probabilistic formulation of PCA. The models happen to give a representation of the data that we perceive as "topics". As we do not directly maximize the quality of the topics w.r.t. some measure that we care about, we cannot expect the predictive performance of the model to convey this.

Author-topic coherence. The topic coherence measure was developed for standard topic models, not the author-topic model. However, topic coherence is a perfectly valid measure of topic quality, as the topics resulting from the two models, LDA and author-topic models, have the same intuition behind them. One might consider using *co-author frequency* instead, i.e. base the topic coherence on the number of authors' documents that word pairs occur together in. This would require experimental results to discern whether this measure indeed correlates with topic quality.

2.4.4 Hyperparameter optimization

As mentioned in section 2.2.2.2, we can apply an optimization procedure by Minka et al. 2003 [Min03] to estimate the parameter of the Dirichlet distribution. This method applies Newton iterations to increase the log likelihood of the data. Of course, in our case we do not know the likelihood, but what we can do is maximize the lower bound w.r.t. the Dirichlet parameter.

First, we will describe the method presented in Minka et al. 2003 [Min03] to

estimate the Dirichlet parameter, and then we will discuss how to apply it to the author-topic model.

2.4.4.1 Dirichlet MLE

Given $\mathcal{D} = (p_1, p_2, ..., p_N)$ such that $p_i \sim Dir(\alpha_1, ..., \alpha_K),\ \forall i$, the log likelihood of the data is defined as

$$\log p(\mathcal{D}|\alpha) = N \log \Gamma(\sum_k \alpha_k) - N \sum_k \log \Gamma(\alpha_k) + N \sum_k (\alpha_k - 1) \log \bar{p}_k,$$

where $\bar{p}_k = \frac{1}{N} \sum_i p_{ik}$. Basically, we want to maximize the log likelihood (equation above) w.r.t. the data (the p_i's) which are drawn from a Dirichlet distribution.

To maximize the log likelihood, we perform Newton iterations,

$$\alpha \leftarrow \alpha - H^{-1} g,$$

where g_i and H^{-1} are the gradient and the Hessian of the log likelihood function, respectively, defined as,

$$g_i = \frac{\partial}{\partial \alpha_i} \log p(\mathcal{D}|\alpha),$$
$$H_{ij}^{-1} = \frac{\partial^2}{\partial \alpha_i \partial \alpha_j} \log p(\mathcal{D}|\alpha).$$

Minka et al. 2003 [Min03] show that we can compute $H^{-1}g$ very efficiently by first defining

$$q_{jk} = -N \Psi'(\alpha_k) \delta_{jk},$$
$$z = N \Psi'(\sum_k \alpha_k),$$

where δ_{jk} is the Kronecker delta function. We define

$$b = \frac{\sum_j g_j / q_{jj}}{1/z + \sum_j 1/q_{jj}}.$$

The gradient can be simply computed as

$$g_k = N \Psi(\sum_j \alpha_j) + N \Psi(\alpha_k) + N \log \bar{p}_k.$$

Finally we can compute

$$(H^{-1}g)_k = \frac{g_k - b}{q_{kk}}.$$

From the expression above, we observe that each Newton iteration has linear complexity $O(K)$.

2.4.4.2 Hyperparameter optimization in the author-topic model

We describe how the method in the section above applies to the author-topic model. We shall describe it in terms of α, but using it to update η is equivalent.

As mentioned, we maximize the lower bound w.r.t. the hyperparameters. The bound depends on α through the following terms,

$$\mathcal{L}_{[\alpha]} = \sum_a \left(\sum_k (\alpha_k - 1) E_q[\log \theta_{ak}] + \log \Gamma \left(\sum_k \alpha_k \right) - \sum_k \log \Gamma(\alpha_k) \right),$$

which we can write as

$$\mathcal{L}_{[\alpha]} = A \sum_k (\alpha_k - 1) \sum_a E_q[\log \theta_{ak}] + A \log \Gamma \left(\sum_k \alpha_k \right) - A \sum_k \log \Gamma(\alpha_k).$$

We note that this expression is equivalent to the log likelihood expression in the previous section if we set $\log \bar{p}_k = \sum_a E_q[\log \theta_a]$. We can thus apply the Newton algorithm to update the Dirichlet parameters in the author-topic model.

Note that we can only perform a single update of α per iteration of the VB algorithm. That is, when we update α, we have to update $E_q[\log \theta_{ak}]$ before we can make another α update. Think of it this way: when we change α then the current θ is no longer the optimal one (given α).

In Hoffman et al. 2013 [HBWP13], the step-size ρ_t is used in hyperparameter estimation as well,

$$\alpha \leftarrow \alpha - \rho_t H^{-1} g.$$

As mentioned, a Newton iteration has complexity $O(K)$. Averaging over $E_q[\log \theta_{ak}]$ is $O(A)$, making this method $O(A + K)$. Similarly, updating η is $O(K + V)$.

Chapter 3

Implementation

The goal with this section is to motivate the design of the implementation, describe some aspects of it, and discuss caveats and possible improvements.

First, we introduce the Gensim package, which this author-topic model implementation was developed for. After that, we discuss some technical details with the implementation.

As this section is meant to supplement the tutorial ([Mor16]), it is recommended that the tutorial is read first.

3.1 Gensim

Gensim is a library written in Python that aims to bridge the gap between academic research in NLP and applications in the real world. While the gap is not very large today in 2016, when Gensim was released in 2010 this was a real issue. Gensim continues to be a popular NLP tool and to be at the forefront of bringing the newest NLP research to the public.

An article describing Gensim was released in 2010 by Rehurek and Sojka [RS10],

and Rehurek published his PhD dissertation in 2011 [Reh11] describing it in more detail.

Gensim's motto is "Topic modelling for humans", and in addition to topic modelling, Gensim also contains a very popular implementation of the word embedding algorithm "Word2Vec".

The primary goal of this project is to provide a quality implementation of the author-topic model in the Gensim library. The quality of the implementation is based on

- Speed
- Scalability
- User friendliness

In the next two sections, we introduce the guiding principles and usage requirements that guide the development of the author-topic model implementation. As we shall see later, the author-topic model has a number of caveats to these principles and requirements, but sticks to them as much as possible.

3.1.1 Design philosophy

We follow Gensim's design philosophy, which can be read about in Rehurek et al. 2010 [RS10].

The implementation must be **user friendly**. All models in Gensim have a particular interface, so that once you learn how to use Gensim, you know how to use all the models in Gensim.

The implementation should be **corpus size independent**. This is why so much focus was on developing an online algorithm for the author-topic model. With the online algorithm, the memory footprint of the model does not depend on the number of documents (which can be quite large), but only the vocabulary size, number of topics and number of authors.

3.1.2 Gensim crash-course

All Gensim models have a particular interface and data structure. We briefly introduce some of these concepts.

In Gensim, each document in a corpus is represented as a list of (*word_id*, *word_count*) tuples, where *word_id* is an integer ID of a word and *word_count* is the number of times that word has been observed in the document. This representation is a type of bag-of-words (BOW) model, also called a unigram model. A corpus may be a list of documents in this representation, or some sort of an iterable.

A core concept in the Gensim framework is *document streaming*. Models must (if possible) accept iterable corpora such that processing takes place on a per-document basis.

In the Gensim pipeline it is common to use two functionalities: transformations and similarity queries. Essentially, once a model is trained a user can transform the data to the vector space as `model[corpus]` and submit queries to a similarity object `index = MatrixSimilarity(model[corpus])` as `sims = index[query]`.

3.2 Author-Topic model implementation

The structure of the program is described very briefly, and we discuss documentation and unit tests. Some aspects of the implementation are discussed in more detail, and possible improvements are suggested.

As mentioned previously, the author-topic model is quite similar to LDA. Furthermore, the algorithm we have developed to train it is also similar to the algorithm used to train LDA in Gensim. This has allowed us to make the structure of the author-topic model class (`AuthorTopicModel`) very similar to the LDA class (`LdaModel`). The `AuthorTopicModel` class *inherits* the `LdaModel` class, and overwrites methods if necessary, and adds methods if necessary.

Because the author-topic model and LDA code are so similar, all the Gensim developers who are used to working on LDA can easily start working on the author-topic model.

The author-topic model is a part of the Gensim codebase, which can be found at https://github.com/RaRe-Technologies/gensim.

Some of the sections below are relevant for users, others are more for the developers who will work further on this implementation in the future.

3.2.1 Documentation

The documentation consists of the docstrings in the code that describe the classes and their methods. Webpages with the documentation are generated automatically using Sphinx[1]. When the next version of Gensim is released, the documentation will appear in the Gensim API reference[2].

The goal of the documentation is to describe the code in rough terms, and to check parameters and usage of individual classes and methods.

In contrast to the documentation, the role of the tutorial is to give users an introduction to the author-topic model in Gensim, assuming some pre-existing experience with Gensim, and to illustrate some useful examples.

3.2.2 Program flow

The goal of this section is to give a very basic understanding of how the program is designed; the program is somewhat complex and thus requires close study to be fully understood.

In figure 3.1, we see a flow chart that illustrates what happens in the class when the model is trained. This is a very simplified illustration, as not all methods and variables are shown.

The constructor (`__init__`) of the `AuthorTopicModel` class is called and initializes the class. As the constructor is called with a `corpus` and an `author2doc` dictionary, the `update` method is called once initialization is complete. If, for example, `corpus` was omitted when the constructor was called, the class would be initialized, and then wait for further instructions.

When the `update` method is called from the constructor with a `corpus` and an `author2doc` dictionary, the training starts. The `update` method alternately calls `log_perplexity` to evaluate the lower bound, and the methods performing the E-step and M-step, as in the pseudo-code in section 2.4.2, i.e. `do_estep` and `do_mstep` respectively.

[1] http://www.sphinx-doc.org
[2] https://radimrehurek.com/gensim/apiref.html.

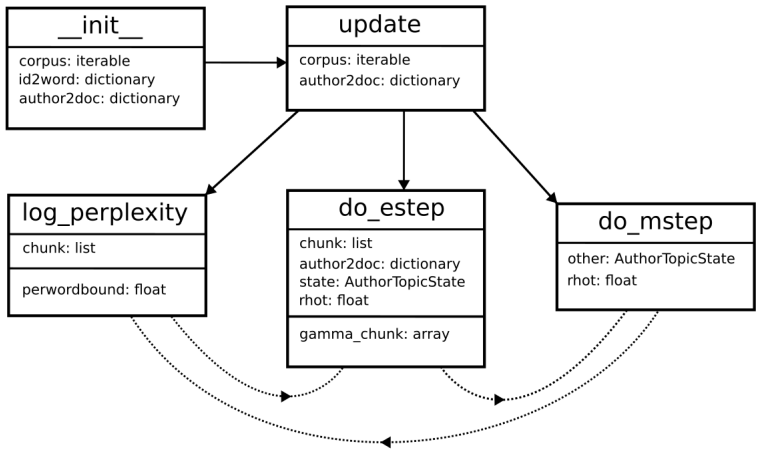


Figure 3.1: This figure illustrates how various methods in the AuthorTopicModel class are called during training. The solid arrows indicate that one method calls another, while the dashed lines indicate looping.

Mini-batches of size `chunksize` are sent to `do_estep`. Mini-batches are also referred to as "chunks".

If automatic `alpha` and `eta` tuning are enabled, they are updated subsequently to the E-step and M-step, respectively.

3.2.3 Unit tests

Unit tests of the author-topic model are designed to test as many aspects of the model as possible. When anything in the Gensim codebase is changed, these unit tests will run automatically and tell the developer whether (s)he has broken something in the author-topic model.

The unit tests are essentially a retrofit of the tests of the `LdaModel` class. Tests that do not make sense for the author-topic model are removed, some tests are changed slightly to fit the author-topic model, and some new tests are introduced as well.

3.2.4 Details

In this section, we go through some details and caveats with the implementation, in no particular order.

3.2.4.1 Multiprocessing and distributed computing

LDA in Gensim supports both multiprocessing and distributed computation, but the author-topic model implementation does not do so at the moment. Much of the infrastructure that allows both multiprocessing and distributed computation is already in place, as the model inherits it from LDA. Therefore, enabling these functionalities should be a relatively pain free task.

Multiprocessing will allow us to train the models much faster on a single machine.

Distributed computation spreads the computational load, as well as the storage requirements, across a cluster of machines, allowing us to run far larger experiments than we can on a single machine.

3.2.4.2 Corpus transformation

As mentioned earlier, Gensim models should have a `model[doc]` interface, where `doc` is a BOW representation of a document. The author-topic model class does this somewhat differently, by accepting the name of an author and simply returning the corresponding row of γ (which in the class is a state variable `model.state.gamma`). The interface is then `model[author_name]`.

3.2.4.3 "get_document_topics" not implemented

The class inherits the `get_document_topics` method from LDA, but it is not clear what this method should do in the author-topic model. Therefore, this method is overwritten such that it raises an error (with an appropriate message) if called.

3.2.4.4 Constructing missing author dictionaries

The author-topic model requires both a mapping from authors to documents (`author2doc`) and from documents to authors (`doc2author`). However, given one of these, the other one can be constructed automatically. There are therefore functions implemented for this, so the class only requires one of them, and the user also can construct the dictionary they are missing. `construct_author2doc` and `construct_doc2author` can be found as functions in the `atmodel.py` file, not as methods of the class.

3.2.4.5 Update with new documents and authors

The implementation has been designed to allow the user to train further on both previously seen and unseen data. If this was not the case, the model would be static; if new authors arrived in your data, you would have to train the entire model from scratch.

The `update` method is based on the idea that the model is updated w.r.t. all the authors in the input `author2doc`, rather than all the documents in the input `corpus`. Consider an author that has been trained on 100 documents to convergence, and now a single new document is added; in this case, if the author is only trained on the new document, he will converge to that document and "forget" all the 100 other documents.

When `update(corpus, author2doc)` is called, the following process takes place.

- Randomly initialize variational distribution of all new authors (i.e. add rows to `model.state.gamma`).
- Combine input data with all previously seen data.
- Train on all documents of all authors in input `author2doc`.

3.2.4.6 Serialized corpora

Gensim models accept serialized corpora, i.e. data that is stored on the hard-drive rather than in memory. This is possible in the `AuthorTopicModel` class, but there are some caveats.

Details on how to use the serialized functionality can be found in the tutorial ([Mor16]).

The MmCorpus (Matrix Market) format was chosen to use for serialization because it is an indexable format, i.e. it is possible to retrieve some document as `doc = corpus[d]`. Furthermore, Gensim has an API for this format.

The author-topic model requires that we know which author corresponds to which documents, and we therefore need to know the index of the current document when training. Under normal circumstances, this would be easy as the corpus is processed from the first to the last document in sequence, so we just need to keep track of the index. Updating the model on new unseen data (as explained above) makes this difficult, and requires that we are able to retrieve a document by its index.

Update and serialized corpora: When a serialized corpus is used, and update is called with new documents, these new documents have to be added to the corpus. Unfortunately, the MmCorpus format does not support concatenating the two corpora in an efficient way. Therefore, the entire corpus needs to be re-serialized (i.e. read from disk, and written again to disk). This will become more and more cumbersome as more data is added.

Adding the functionality to efficiently append documents to an MmCorpus in the Gensim API would therefore make this process much faster.

3.2.4.7 Inference on held-out data

At the moment, the model does not support inference on held-out data. In the context of the author-topic model, inference on held-out data would correspond to inferring the topic distribution of a new author without modifying the model, that is, updating γ for the new author without updating λ.

Implementing this functionality should be relatively simple. One could for example use the following procedure.

- Add a row to γ (`model.state.gamma`).
- Call the `inference` method with `collect_sstats=False` (so that λ is not updated).
- Remove the rows again from γ.

The lack of this functionality also means that it is not possible to evaluate the bound on held-out data. This is important in order to obtain a measure of how well the model generalizes.

With this functionality, it would be possible to define the corpus transformation operation `model[corpus]` to accept a set of documents as input, and infer the author-topic distribution assuming all the input documents belong to a single author.

3.2.4.8 Potential memory problems with author dictionaries

As discussed in section 2.4.2.6, the memory complexity of the `author2doc` and `doc2author` dictionaries are $O(\sum_d A_d)$ and $O(\sum_a D_a)$, respectively. The memory requirement of these data structures thus scales with the number of documents, which is very undesirable for a streamable algorithm. In practice, however, these dictionaries will not cause problems, for the most part. If problems do occur, it needs to be considered if some serialized data structure that has $O(1)$ look-up speed (as dictionaries do) can be used.

Chapter 4

Results

In this section, the implementation is tested for various purposes. The implementation is compared to a pre-existing one, we test the blocking VB algorithm, the online algorithm, vectorization speed up, and hyperparameter MLE. Lastly, we show that the implementation scales as we would expect it to in accordance with the theory.

Refer to the tutorial ([Mor16]) for examples and analysis of the author-topic representation.

When the variational lower bound is plotted, the first point is excluded, for the most part. The first bound is simply a function of the initialization of γ and λ and thus is not informative when, for example, comparing two algorithms. Furthermore, the first bound is usually much lower than the rest by orders of magnitude, and excluding the first point therefore makes it easier to compare. Assume that the initial bound is excluded unless otherwise stated.

4.1 Data

Three datasets were used in this project, the NIPS dataset (discussed in the tutorial [Mor16]), a data dump from StackExchange, and an artificially generated

dataset.

The StackExchange dataset is only used in one particular test below. It can be downloaded at https://archive.org/details/stackexchange. There are several categories in this dataset, and the "cooking" category was used. The data was pre-processed in a similar manner to the NIPS dataset (as discussed in the tutorial, [Mor16]).

For some of the experiments discussed in this section, artificial data is automatically generated. This allows us to control all aspects of the data; for example, we can control the number of authors per document and number of words per document, which is useful for testing the scalability of the implementation in section 4.4.

4.2 Comparison with an existing implementation

We compare the implementation we have developed with one that was available before the start of this project. This is the Python-Topic-Model (PTM) [1]. The PTM implementation of the author-topic model applies blocking Gibbs sampling to train the model.

We do not compare any type of model fit or topic coherence, but rather simple tests show that our algorithm is significantly superior to the PTM algorithm.

Both algorithms, PTM and Gensim, are run on a small subset of the NIPS dataset, pre-processed in exactly the same manner. Our algorithm is run for 100 passes over the data with 1 iteration per document, and the PTM algorithm is run for 10 iterations of Gibbs sampling. The top 10 words from the resulting topics from both algorithms are shown below.

Topics from Gensim author-topic model:

```
Topic 0: net, layer, memory, bit, character, classifier, node,
connection, code, machine

Topic 1: gaussian, bound, approximation, matrix, density, estimate,
generalization, class, log, sample
```

[1] https://github.com/arongdari/python-topic-model

Topic 2: neuron, circuit, spike, signal, chip, analog, voltage, cell, frequency, noise

Topic 3: rule, sequence, concept, fig, energy, instance, cell, class, positive, neuron

Topic 4: cell, neuron, activity, field, connection, response, visual, map, cortex, orientation

Topic 5: image, constraint, distance, solution, optimization, graph, threshold, object, surface, dimensional

Topic 6: image, object, visual, signal, motion, component, response, stimulus, filter, direction

Topic 7: hidden, recognition, speech, word, layer, trained, hidden_unit, context, net, architecture

Topic 8: prediction, recurrent, noise, series, nonlinear, sample, trained, better, table, signal

Topic 9: control, action, policy, reinforcement, optimal, trajectory, dynamic, controller, robot, reinforcement_learning

Topics from PTM author-topic model:

Topic 0: register, sharply, open, establishes, bottom, cm, quasi, th, blumer, involved

Topic 1: improvement_over, exploited, 5i, tic, arising, alarm, coincides_with, self_organizing, lnp, obey

Topic 2: miss, identi, human_face, distributed_representation, involved, workshop_on, earliest, perceive, infomax, mann

Topic 3: port, 5i, hint, identity_matrix, cross_entropy, hidden_markov, representing, thinking, self_organizing, la_jolla

Topic 4: pling, naturally, rein_forcement, experimental, gr, intend, open, chervonenkis, equiv, written_a

```
Topic 5: charging, tolerance, naturally, gr, exploited, preserve,
aligned, opponent, active, be_interpreted

Topic 6: salk_edu, must_satisfy, facilitates, binding, barron,
recognition, unity, equipment, erroneous, localize

Topic 7: binding, gap_between, vivo, false, often, kolmogorov,
exploited, increased, reading, electric

Topic 8: erroneous, unity, ao, environment, occurrence, cannot_be,
active, reward, excitation, formant

Topic 9: chervonenkis, power_spectrum, twelfth, active, intend,
charging, blumer, nine, mann, hyperplanes
```

Our algorithm takes 2 minutes and 44 seconds, and already we see the some topics emerging. These are some of the same topics as in the tutorial, such as "circuits" and "reinforcement learning".

The PTM algorithm has not started to converge to any meaningful topics, even after the 15 minutes and 47 seconds it took to train the model.

The difference between the two algorithms is evident.

4.3 Algorithmic development tests

Some algorithms and functionality that were developed are tested. The goal of this section is to show that some of the key developments work as expected. In addition, all tests show that the algorithm increases the bound.

4.3.1 Comparing blocking and non-blocking VB

In this section, we show that blocking VB not only increases the bound as fast as standard VB, but that it in fact increases it faster. As mentioned in section 2.4.2.7, blocking VB should be better in theory, because it assumes the topic and author assignments to be dependent variables.

Two programs were made for this test that are almost identical except for the updates.

It turns out that the gain from blocking VB depends on the data. When both algorithms are tested on a subset of the NIPS data, there is virtually no difference. We test both algorithms on a subset of the StackExchange data as well, and in figure 4.1 we see that the blocking VB algorithm converges faster.

Most likely, the reason for this dependence on data is the number of documents per author and/or the number of authors per document; the higher these numbers, the higher the amount of correlation between the two latent variables (author assignment and topic assignment). Imagine you have one author per document and one document per author (i.e. equivalent to standard LDA), then the correlation between x and z has no significance, and whether or not we apply blocking VB makes no difference.

The NIPS subset had 1.97 authors per document, while the StackExchange subset had 3.91 authors per document, where the number of authors per document is computed as $\sum_d A_d/D$.

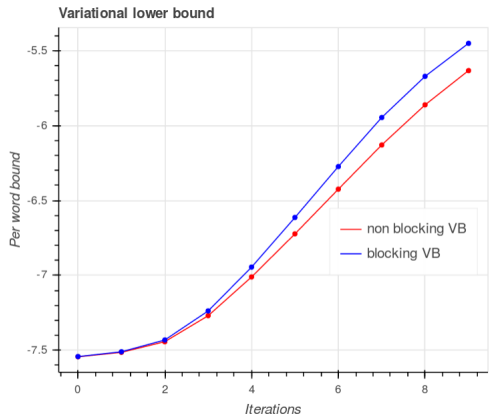


Figure 4.1: Per-word lower bound for blocking and non-blocking VB algorithms, see section 4.3.1 for details.

To test this idea that when the number of authors per document is high, the blocking VB algorithm does better, we generate some data. In figure 4.2, both algorithms are tested again, with two different generated datasets. The only difference between the two datasets is that the number of authors per document is different, and we see that the bound converges faster.

However, as we see in table 4.1, there is a trade-off between speed and accuracy with blocking VB, as blocking VB is slower than non-blocking VB. This is no surprise, as the blocking algorithm requires us to sum over $A_d \times K$ when normalizing ϕ.

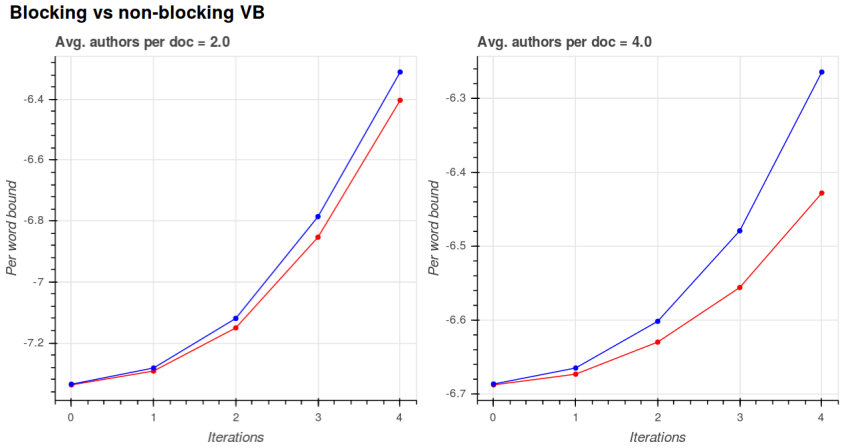


Figure 4.2: Testing blocking (blue line) and non-blocking VB (red line) algorithms on generated data. See section 4.3.1 for details.

Avg. authors per doc	Blocking (bound / sec)	Non-blocking (bound / sec)
2.0	-6.2 / 7.9	-6.4 / 5.9
4.0	-6.26 / 6.5	-6.44 / 10.5

Table 4.1: Testing blocking and non-blocking VB algorithms on generated data. The table shows the final bound and the time it took to train the model. See section 4.3.1 for details.

The goal of applying blocking was to avoid storing ϕ and to enable vectorization. However, these tests show that the blocking algorithm not only works, but that it potentially has an advantage over standard VB. This advantage is, however, a trade-off between model fit and training time.

4.3.2 Comparing offline and online algorithms

The algorithms are tested on a subset of the NIPS dataset. We test the offline algorithm for 100 iterations, and the online algorithm for 100 passes and either 1 or 10 iterations per document.

In figure 4.3, we see the lower bound for each of the tests plotted against the iteration number. The online algorithm outperforms the offline, and 10 iterations per document slightly outperforms 1 iterations per document. The online algorithm converges much faster in the early iterations, but the offline algo-

rithm catches up. However, we are not taking the time it takes to complete each iteration (or pass).

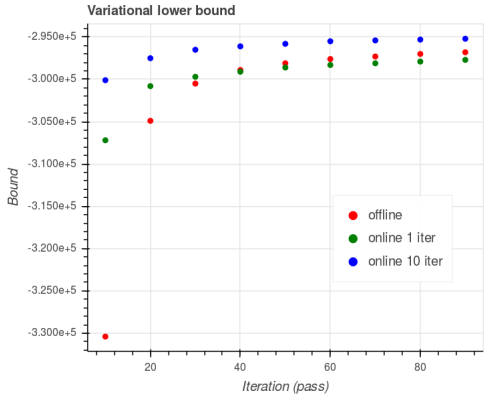


Figure 4.3: Comparison of online and offline algorithms in terms of bound convergence. See section 4.3.2 for details.

In figure 4.4, we again see the lower bound, this time plotted against the execution time. While the online algorithm has reached a higher bound in 100 passes than the offline did in 100 iterations, it seems possible that the offline algorithm will eventually overtake the online algorithm. However, it seems that it will take such a long time for the offline algorithm to catch up that it is simply not worth it (take note of the execution time listed in the x-axis).

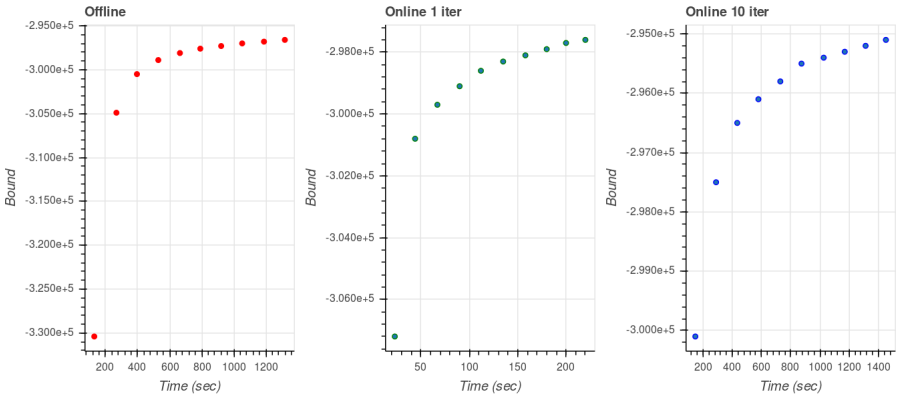


Figure 4.4: Comparison of online and offline algorithms in terms of bound convergence. See section 4.3.2 for details. The initial bound is included in these graphs.

Number of documents	Size of vocabulary	Number of authors	Speed-up
90	681	166	6.6
286	2245	536	5.1
1740	8640	2720	2.4

Table 4.2: Speed-up from vectorization. Number of topics $K = 10$. See section 4.3.3 for details.

Comparing figure 4.3 and figure 4.4, we see that while the online algorithm using 10 iterations per document outperformed the one using only 1 iteration per document, the former is more than 6 times slower than the latter. As a side note, we can conclude that increasing the number of iterations doesn't seem to be worth it.

4.3.3 Vectorization speed-up

The vectorized is compared to the non-vectorized code, to get an idea of the speed-up gained. In table 4.2, we see a table with the dimensionality of the problem and the speed-up. The speed-up is just t_1/t_2 where t_1 is the time taken by the vectorized code, and t_2 the time taken by the non-vectorized version.

While the speed-up is quite significant, we see that it scales quite badly with the size of the data. This is counter intuitive because the speed-up from vectorization increases as the vectors and matrices involved increase in size. It turns out that the reason for this is that the bottleneck of the algorithm when the dataset is large is something we cannot vectorize, namely the update of $E_q[\log \beta]$.

In table 4.3, we see the relative time it takes to update $E_q[\log \beta]$ and γ in the non-vectorized code. We see that as we increase the amount of data, the relative time it takes to update $E_q[\log \beta]$ increases. This means that as the problem size increases, the part we are optimizing by vectorizing the code becomes a smaller part of the total computation taking place, thus the speed-up diminishes.

Furthermore, as the complexity of the $E_q[\log \beta]$ update is $O(KV)$, while the complexity of the γ update is $O(A_d K V_d)$, the former will grow more quickly.

While it is unforunate that the speed-up does not scale well, we still benefit from vectorization at all scales.

The data in figure 4.3 were obtained by using a profiling tool, that runs a program and outputs statistics such as the % of total execution time for each

Number of documents	Update $E_q[\log \beta]$ (in %)	Update γ (in %)
90	1.6	20.4
286	2.8	20.8
1740	9.3	19.6

Table 4.3: Relative execution time (% of total execution time) taken by $E_q[\log \beta]$ and γ in the non-vectorized code.

line in the code.

4.3.4 Hyperparameter MLE

We check that the automatic learning of α and η using MLE, as described in section 2.4.4, speeds up convergence by increasing the bound. In figure 4.5, we have trained the model on the NIPS data, and the bound is shown in four different scenarios: without any optimization, with automatic learning of α, with automatic learning of η, and with automatic learning of both hyperparameters. As we see, the bound is increased as expected, and optimizing both hyperparameters gives even better results. We also note that optimizing the prior on word probabilities η seems to improve the result more than optimizing the prior on topic probabilities α.

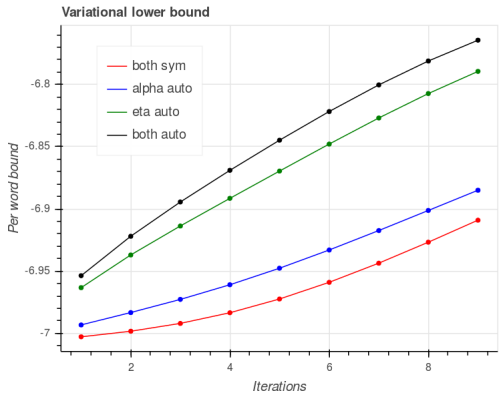


Figure 4.5: Test of hyperparameter MLE on NIPS data. See section 4.3.4 for details.

4.4 Scalability

In this section, we will show how the execution time of the program scales with various quantities; specifically, four variables of the data are tested, number of documents (D), number of authors per document (A_d), size of vocabulary (V), number of words per document (V_d), and one parameter of the model is tested, namely number of topics (K). We compare the scalability with the asymptotic complexity (i.e. "theoretical scalability") derived in Appendix C.4 and also discussed in section 2.4.2.5.

Three other factors that effect the execution time are the chunk size (size of mini-bathces), number of passes over data, and number of iterations per document. However, these were not tested empirically, but we shall discuss their effect later in this section.

4.4.1 Empirical scalability

All the tests in this section are done on the generated data. We time the training of the model as well as computing the bound on the corpus. All the results can be seen in figures 4.6 to 4.10. All execution times are averaged over several runs of the program.

The asymptotic complexity of the algorithm tells us that when we vary a single of these factors, independently, we should expect to see a linear trend in the execution time. All the plots in figure 4.6 to 4.10 show linear scalability w.r.t. the different factors. Most notably, figure 4.6 is important in the context of an online, corpus independent, algorithm.

As a side note, while figures 4.7 (right) and 4.10 (right) do not look 100% linear, we attribute this to some side effects, and still interpret them as linear. The most important thing is that these plots are defnitely note superlinear, for example quadratic.

4.4.2 Other factors in execution time

As mentioned previously in this section, three other factors that effect the execution time are the chunk size (size of mini-bathces), number of passes over data, and number of iterations per document.

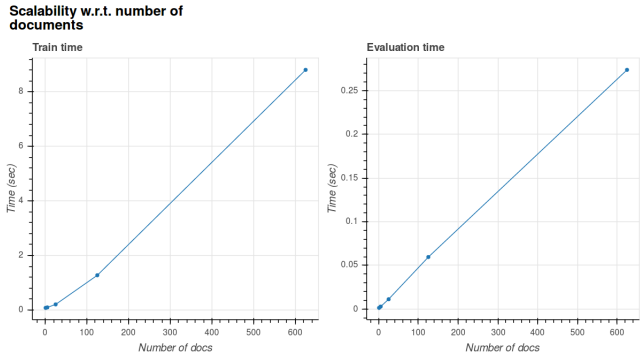


Figure 4.6: Execution time of training and bound evaluation as a function of the number of documents in the corpus. See section 4.4 for details.

The chunk size essentially dictates how many E-steps we do in a row before we do a single M-step. For the most part, the M-step is very computationally intensive to perform, which means that increasing the chunk size will lower the training time. As a result, it is a good idea to keep the chunk size as big as possible (but no larger than what can fit in memory). On the other hand, the chunk size also affects the resulting representations; one may decide that lowering the chunk size will improve the topics, and therefore will have to strike a balance between the quality of the topics and the execution time.

If the running time for a single pass over the corpus is t, then the time for running P passes is $P \cdot t$, it is that simple. How many passes will be necessary to converge will depend on the chunk size and the number of iterations per document, so it is again a matter of striking a balance between model fit and time.

The number of iterations dictates how many times we iterate over a single document in the M-step. Clearly, increasing the number of iterations will increase the execution time linearly. Once again, choosing the number of iterations is a matter of compromising between model fit and training time.

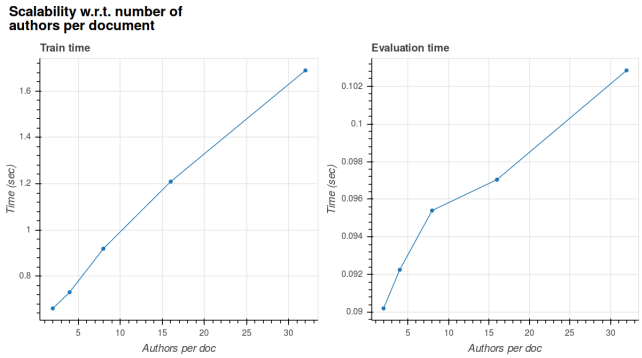


Figure 4.7: Execution time of training and bound evaluation as a function of the average number of authors per document. See section 4.4 for details.

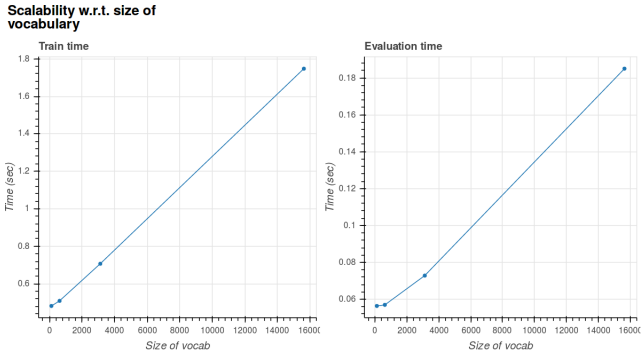


Figure 4.8: Execution time of training and bound evaluation as a function of the number of unique words in the corpus. See section 4.4 for details.

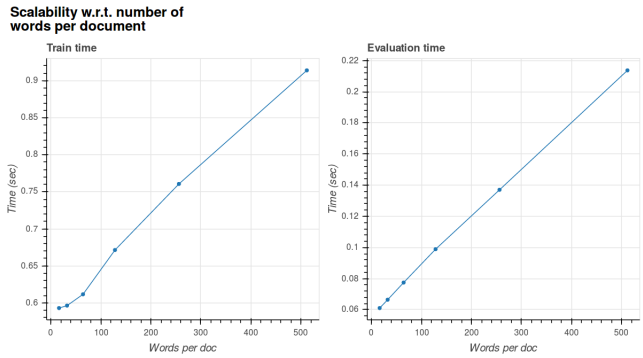


Figure 4.9: Execution time of training and bound evaluation as a function of the number of words per document. See section 4.4 for details.

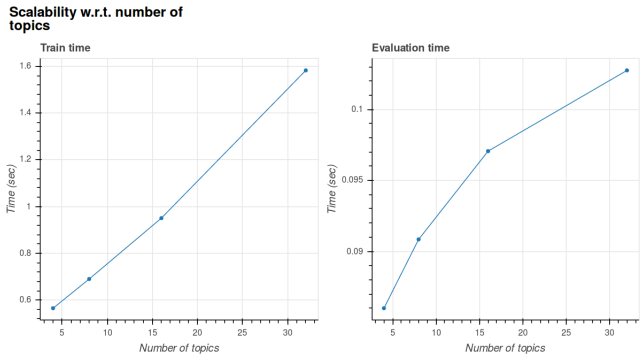


Figure 4.10: Execution time of training and bound evaluation as a function of the number of topics. See section 4.4 for details.

Chapter 5

Discussion

This projects attempts to develop the author-topic model in order to catch up with many years of development of LDA, both on the theory side and in terms of implementation in the Gensim community. It is therefore not surprising that there is still work to be done. In this section, we summarize some of the problematic aspects of the implementation and some general improvements that can be made.

The discussion proceeds in order of importance (loosely), the most important subjects appearing first.

As mentioned in section 1.3, the MatLab Topic Modeling Toolbox[1] contains an implementation of the author-topic model that is quite useful. It would be very interesting to see a comparison between it, and our implementation. The problem is that it does not compute any sort of evaluation metric which would facilitate an objective comparison, for example, an approximation to the likelihood (like in section 2.4.3.2). The MatLab Topic Modeling Toolbox is no longer being maintained, so this requires someone else to become familiarized with the implementation, and introduce some relevant evaluation metric.

While section 4.4 shows that the implementation scales well (as well as we expect it to), a truly large scale experiment with real data should be made to test the

[1] http://psiexp.ss.uci.edu/research/programs_data/toolbox.htm

scalability in a real world setting. Such a test would also showcase the usefulness of the implementation at scale.

It is important to be able to evaluate the model fit based on held-out data, but as discussed in section 3.2.4.7, this is not possible at the moment. Enabling this functionality has high priority in the further development of the implementation. The level of difficulty of this task is not clear as one would have to decide exactly what it means to make inference on held-out data (in the context of the author-topic model), and possibly have to make some structural changes to the program.

As discussed in section 3.2.4.6, the use of serialized corpora in the implementation is far from ideal. Improving this functionality has high priority in the further development of the implementation, but can prove to be a difficult task.

As discussed in section 3.2.4.1, enabling multiprocessing and distributed computation should be easy, relatively speaking, and would allow us to train models much faster and to run far larger experiments.

It would be interesting to see a high-quality and scalable implementation of the author-topic model that applies CVB in Gensim. However, the first natural step in this direction is to apply CVB to LDA in Gensim, and develop that implementation as far as we can. After that, doing the same for the author-topic model should be relatively easy.

As mentioned in section 2.4.3.3, it may be a good idea to develop a topic coherence measure specifically for the author-topic model. This requires human annotators to score topics in terms of quality from several trained models, so that a topic coherence measure that correlates with the human judgement can be devised empirically.

The potential memory problem with the author dictionaries, discussed in section 3.2.4.8, is not a high priority, as it may not pose a problem at all.

Chapter 6

Conclusion

A scalable algorithm was developed to train the author-topic model by applying stochastic variational inference as well as blocking. The algorithm was implemented in a fashion that facilitates seamless integration with the Gensim framework.

Useful examples of the topic model were demonstrated in a tutorial, and the open source community quickly embraced this new feature in Gensim.

There is still work to be done. There are a number of caveats to the implementation, outlined throughout this thesis, and many of them can be alleviated. Hopefully, the development will continue within the Gensim community.

Appendix A

Standard VB

The derivation of the algorithm given a standard VB approach is described. A lot of details are left out, but everything should be clear if section 2.4.2 has been read beforehand.

We pose the *variational distribution* $q(\theta, \beta, z, x)$ which approximates the posterior. The variational distribution is fully factorized,

$$\begin{aligned} q(\theta, \beta, z, x) &= q(\theta|\gamma)q(\beta|\lambda)q(x|\mu)q(z|\phi) \\ &= \prod_a q(\theta_a|\gamma_a) \prod_k q(\beta_k|\lambda_k) \prod_{d,n} q(x_{dn}|\mu_{dn}) \prod_{d,n} q(z_{dn}|\phi_{dn}) \\ &= \prod_a Dir(\theta_a|\gamma_a) \prod_k Dir(\beta_k|\lambda_k) \prod_{d,n} Mult(x_{dn}|\mu_{dn}) \prod_{d,n} Mult(z_{dn}|\phi_{dn}). \end{aligned} \quad \text{(A.1)}$$

Figure A.1 shows a graphical model, as in figure 2.4, of the model $q(\theta, \beta, z, x)$.

A.0.1 Obtaining the lower bound

We obtain the lower bound to the marginal log likelihood.

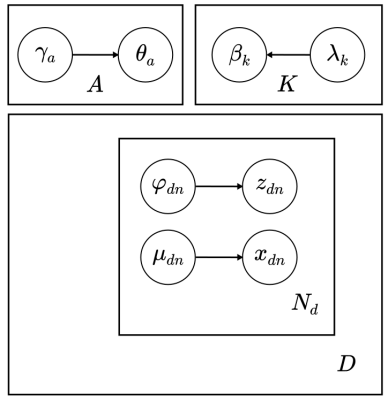


Figure A.1: Graphical model of standard VB approximation. See section A for details.

$$\begin{aligned}
\log p(w|\alpha,\eta,A) &\geq E_q[\log p(\theta,\beta,x,z,w|\alpha,\eta,A)] - E_q[\log q(\theta,\beta,z,x)] \\
&= \mathcal{L}(\gamma,\lambda,\phi,\mu) \\
&= \sum_a E_q[\log Dir(\theta_a|\alpha)] + \sum_k E_q[\log Dir(\beta_k|\eta)] + \sum_{d,n} E_q[\log Unif(x_{dn}|A_d)] \\
&\quad + \sum_{d,n} \sum_{a\in A_d} E_q[\log Mult(z_{dn}|\theta_a)] + \sum_{d,n,k} E_q[\log Mult(w_{dn}|\beta_k)] \\
&\quad - \sum_a E_q[\log Dir(\theta_a|\gamma_a)] - \sum_k E_q[\log Dir(\beta_k|\lambda_k)] - \sum_{d,n} E_q[\log Mult(x_{dn}|\mu_{dn})] \\
&\quad - \sum_{d,n} E_q[\log Mult(z_{dn}|\phi_{dn})] \qquad \text{(A.2)}
\end{aligned}$$

We evaluate each of the expectations in the equation above. To evaluate $E_q[\log Dir(\theta_a|\alpha)]$ we write the Dirichlet in the form

$$Dir(\theta_a|\alpha) = \exp\left\{\left(\sum_k (\alpha_k - 1)\log\theta_{ak}\right) + \log\Gamma\left(\sum_k \alpha_k\right) - \sum_k \log\Gamma(\alpha_k)\right\},$$

The first expectation is

$$E_q[\log Dir(\theta_a|\alpha)] = \sum_k ((\alpha_k - 1)E_q[\log\theta_{ak}]) + \log\Gamma\left(\sum_k \alpha_k\right) - \sum_k \log\Gamma(\alpha_k)$$

where $E_q[\log \theta_{ak}] = \Psi(\gamma_{ak}) - \Psi(\sum_{k'} \gamma_{ak'})$.

The rest of the Dirichlet expectations are derived in the same manner. We have the expectations of the multinomials,

$$E_q[\log Mult(z_{dn}|\theta)] = \sum_k \sum_{a \in A_d} \int q(x_{dv} = a) q(z_{dv} = k) q(\theta_{ak}) \log p(z_{dn}|\theta) d\gamma$$
$$= \sum_k \sum_{a \in A_d} \mu_{dna} \phi_{dnk} E_q[\log \theta_{ak}], \quad \text{(A.3)}$$

$$E_q[\log Mult(w_{dn}|\beta)] = \sum_k \sum_v \int \mathbb{I}(w_{dn} = v) q(z_{dv} = k) q(\lambda_{kv}) \log p(w_{dn} = v|\beta_{kv}) d\lambda$$
$$= \sum_k \sum_v \mathbb{I}(w_{dn} = v) \phi_{dnk} E_q[\log \beta_{kv}], \quad \text{(A.4)}$$

and similarly,

$$E_q[\log Mult(z_{dn}|\phi_{dn})] = \sum_k \phi_{dnk} \log \phi_{dnk}.$$

$$E_q[\log Mult(x_{dn}|\mu_{dn})] = \sum_{a \in A_d} \mu_{dna} \log \mu_{dna}.$$

Finally, we have

$$E_q[\log Unif(x_{dn}|A_d)] = \frac{1}{|A_d|}.$$

We simplify the bound by introducing the n_{dv} variable. We also collect some of the terms in the equation, to make it a bit shorter. Finally, we add constraints such that $\sum_k \phi_{dvk} = 1 \ \forall d, v$ with corresponding Lagrange multipliers ℓ^{ϕ}_{dv}, and

treat μ similarly.

$$\begin{aligned}
&\mathcal{L}(\gamma,\lambda,\mu,\phi) = \\
&\sum_a \left(\sum_k (\alpha_k - \gamma_{ak}) E_q[\log\theta_{ak}] - \log\Gamma(\sum_k \gamma_{ak}) + \sum_k \log\Gamma(\gamma_{ak}) + \log\Gamma(\sum_k \alpha_k) - \sum_k \log\Gamma(\alpha_k) \right) \\
&+ \sum_k \left(\sum_v (\eta_v - \lambda_{kv}) E_q[\log\beta_{kv}] - \log\Gamma(\sum_v \lambda_{kv}) + \sum_v \log\Gamma(\lambda_{kv}) + \log\Gamma(\sum_v \eta_v) - \sum_v \log\Gamma(\eta_v) \right) \\
&+ \sum_{d,v} n_{dv} \left(\sum_k \phi_{dvk} \left(\left(\sum_{a\in A_d} \mu_{dva} E_q[\log\theta_{ak}] \right) + E_q[\log\beta_{kv}] - \log\phi_{dvk} \right) - \sum_{a\in A_d} \mu_{dva} \log\mu_{dva} \right) \\
&\quad + \sum_{d,v} \frac{1}{|A_d|} + \sum_{d,v} \ell^{\phi}_{dv} \left(\left(\sum_k \phi_{dvk} \right) - 1 \right) + \sum_{d,v} \ell^{\mu}_{dv} \left(\left(\sum_a \mu_{dva} \right) - 1 \right) \qquad \text{(A.5)}
\end{aligned}$$

A.0.2 Obtaining the update equations

To maximize the lower bound, we apply a simple coordinate ascent method: take the derivative of $\mathcal{L}(\gamma,\lambda,\mu,\phi)$ w.r.t. each of the variational parameters, equate it to zero, and isolate the corresponding parameter.

The lower bound depends on ϕ_{dvk} via the terms

$$\begin{aligned}
\mathcal{L}_{[\phi_{dvk}]} = n_{dv}\phi_{dvk} \left(\left(\sum_{a\in A_d} \mu_{dvk} E_q[\log\theta_{ak}] \right) + E_q[\log\beta_{kv}] - \log\phi_{dvk} \right) \\
+ \ell^{\phi}_{dv} \left(\left(\sum_{k'} \phi_{dvk'} \right) - 1 \right), \qquad \text{(A.6)}
\end{aligned}$$

and the partial derivative of the lower bound w.r.t. ϕ_{dvk} is

$$\frac{\partial\mathcal{L}}{\partial\phi_{dvk}} = n_{dv} \sum_{a\in A_d} \mu_{dvk} E_q[\log\theta_{ak}] + n_{dv} E_q[\log\beta_{kv}] - n_{dv}\log\phi_{dvk} - 1 + \ell^{\phi}. \qquad \text{(A.7)}$$

Setting the derivative equals to zero and isolating ϕ_{dvk} yields

$$\phi_{dvk} \propto \exp\left\{\sum_{a \in A_d} \mu_{dvk} E_q[\log \theta_{ak}] + E_q[\log \beta_{kv}]\right\}.$$

The normalization constant (note the "proportional to" in the equation above) is $\exp(1 - \ell^{\phi})$, although we just normalize ϕ_{dvk} directly, as we shall see later.

We move on to μ, the lower bounds depends on it via the terms

$$\mathcal{L}_{[\mu_{dva}]} = n_{dv}\left(\sum_k \phi_{dvk}\mu_{dva}E_q[\log\theta_{ak}] - \mu_{dva}\log\mu_{dva}\right) + \sum_{d,v}\ell^{\mu}_{dv}\left(\left(\sum_{a'}\mu_{dva'}\right) - 1\right),$$

the partial derivative is

$$\frac{\partial \mathcal{L}}{\partial \mu_{dva}} = n_{dv}\left(\sum_k \phi_{dvk}E_q[\log\theta_{ak}] - 1 - \log\mu_{dva}\right) + \ell^{\mu},$$

which yields the update equation

$$\mu_{dva} \propto \exp\{\log\frac{1}{|A_d|} + \sum_k \phi_{dvk}E_q[\log\theta_{ak}]\},$$

where we ignore constant terms as in the ϕ update. We can of course simplify this to

$$\mu_{dva} \propto \frac{1}{|A_d|}\exp\{\sum_k \phi_{dvk}E_q[\log\theta_{ak}]\}.$$

The lower bound depends on γ via the terms

$$\begin{aligned}\mathcal{L}_{[\gamma_{ak}]} = \\ &(\alpha_k - \gamma_{ak})(\Psi(\gamma_{ak}) - \Psi(\sum_{k'}\gamma_{ak'})) - \log\Gamma(\sum_k \gamma_{ak}) + \log\Gamma(\gamma_{ak}) \\ &\quad + \sum_{d \in D_a}\sum_v n_{dv}\phi_{dvk}\mu_{dvk}(\Psi(\gamma_{ak}) - \Psi(\sum_{k'}\gamma_{ak'})). \quad \text{(A.8)}\end{aligned}$$

where we have defined the set $D_a = \{d \mid a \in A_d\}$. The partial derivative is

$$\frac{\partial \mathcal{L}}{\partial \gamma_{ak}} = (\alpha_k - \gamma_{ak})(\Psi'(\gamma_{ak}) - \Psi'(\sum_{k'} \gamma_{ak'})) + \sum_{d \in D_a} \sum_{v} n_{dv} \phi_{dvk} \mu_{dva} \left((\Psi'(\gamma_{ak}) - \Psi'(\sum_{k'} \gamma_{ak'})) \right), \quad \text{(A.9)}$$

This yields the update equation

$$\gamma_{ak} = \alpha_k + \sum_{d \in D_a} \sum_{v} n_{dv} \mu_{dvk} \phi_{dvk}.$$

The lower bound depends on λ via the terms

$$\mathcal{L}_{[\lambda_{kv}]} = (\eta_v - \lambda_{kv})(\Psi(\lambda_{kv}) - \Psi(\sum_{v'} \lambda_{kv'})) - \log \Gamma(\sum_{v} \lambda_{kv}) + \log \Gamma(\lambda_{kv}) + \sum_{d} n_{dv} \phi_{dvk} (\Psi(\lambda_{kv}) - \Psi(\sum_{v'} \lambda_{kv'})), \quad \text{(A.10)}$$

and the partial derivate is

$$\frac{\partial \mathcal{L}}{\partial \lambda_{kv}} = (\eta_v - \lambda_{kv})(\Psi'(\lambda_{kv}) - \Psi'(\sum_{v'} \lambda_{kv'})) + \sum_{d} n_{dv} \phi_{dvk} (\Psi'(\lambda_{kv}) - \Psi'(\sum_{v'} \lambda_{kv'})), \quad \text{(A.11)}$$

which yields the last update equation,

$$\lambda_{kv} = \eta_v + \sum_{d} n_{dv} \phi_{dvk}.$$

A.0.3 VB algorithm

Algorithm 3 show pseudo-code using the VB updates derived in the previous section.

Notice that in the μ update, we have removed the dependency on $|A_d|$, the number of authors in the document, because it disappears in the normalization term anyway.

Algorithm 3 Training the author-topic model using standard VB.

function AT-VB(w_{dn}, A, K, α, η, τ_1, τ_2)
  Initialize γ and λ randomly according to a gamma distribution.
  Compute $E_q[\log \theta_{ak}]$ based on γ.
  Compute $E_q[\log \beta_{kv}]$ based on λ.
  Compute $\mathcal{L}$.
  repeat
    Set $\mathcal{L}_{prev} := \mathcal{L}$.
    for $d = 1$ to D **do**
      for $v \in V_d$ **do**
        for $k = 1$ to K **do**
          $\phi_{dvk} \propto \exp\left\{\sum_{a \in A_d} \mu_{dva} E_q[\log \theta_{ak}] + E_q[\log \beta_{kv}]\right\}$.
        end for
        Normalize ϕ_{dvk} to sum to 1 over k:
        $\phi_{dvk} := \phi_{dvk} / \left(\sum_{k=1}^{K} \phi_{dvk}\right)$
      end for
    end for
    for $d = 1$ to D **do**
      for $v \in V_d$ **do**
        for $a \in A_d$ **do**
          $\mu_{dva} \propto \exp\{\sum_k \phi_{dvk} E_q[\log \theta_{ak}]\}$.
        end for
        Normalize μ_{dva} to sum to 1 over a:
        $\mu_{dva} := \mu_{dva} / \left(\sum_{a \in A_d} \mu_{dva}\right)$
      end for
    end for
    for $a = 1$ to A **do**
      for $k = 1$ to K **do**
        $\gamma_{ak} := \alpha_k + \sum_{d \in D_a} \sum_v n_{dv} \mu_{dva} \phi_{dvk}$.
      end for
    end for
    Update $E_q[\log \theta_{ak}]$ based on γ.
    for $k = 1$ to K **do**
      for $v = 1$ to V **do**
        $\lambda_{kv} := \eta_v + \sum_d n_{dv} \phi_{dvk}$.
      end for
    end for
    Update $E_q[\log \beta_{kv}]$ based on λ.
    Compute $\mathcal{L}$.
  until $(\mathcal{L} - \mathcal{L}_{prev})/\mathcal{L}_{prev} < \tau_1$
end function

A.0.4 Online VB

Algorithm 4 show pseudo-code for the online VB algorithm. The online algorithm is derived similarly to the one in section 2.4.2.5.

Algorithm 4 Online training the author-topic model using standard VB.

function ONLINE-AT-VB(w_{dn}, A, K, α, η, τ, κ)
  Initialize γ and λ randomly according to a gamma distribution.
  Compute $E_q[\log \theta_{ak}]$ based on γ.
  Compute $E_q[\log \beta_{kv}]$ based on λ.
  for $t = 1$ to ∞ **do**
    Set $\rho_t := (\tau_0 + t)^{-\kappa}$.
    repeat
      for $v \in V_t$ **do**
        for $k = 1$ to K **do**
          $\phi_{tvk} \propto \exp\left\{\sum_{a \in A_t} \mu_{tva} E_q[\log \theta_{ak}] + E_q[\log \beta_{kv}]\right\}$.
        end for
        Normalize ϕ_{tvk} to sum to 1 over k:
        $\phi_{tvk} := \phi_{tvk} / \left(\sum_{k=1}^{K} \phi_{tvk}\right)$
      end for
      for $v \in V_t$ **do**
        for $a \in A_t$ **do**
          $\mu_{tva} \propto \exp\{\sum_k \phi_{tvk} E_q[\log \theta_{ak}]\}$.
        end for
        Normalize μ_{tva} to sum to 1 over a:
        $\mu_{tva} := \mu_{tva} / \left(\sum_{a \in A_t} \mu_{tva}\right)$
      end for
      for $a = 1$ to A **do**
        for $k = 1$ to K **do**
          $\tilde{\gamma}_{ak} := \alpha_k + |D_a| \sum_v n_{tv} \mu_{tva} \phi_{tvk}$.
        end for
      end for
      $\hat{\gamma} := (1 - \rho_t)\gamma + \rho_t \tilde{\gamma}$
      Update $E_q[\log \theta_{ak}]$ based on a temporary estimate $\hat{\gamma}$.
    until Average absolute change in $\tilde{\gamma}$ is less than τ.
    $\gamma := \hat{\gamma}$
    for $k = 1$ to K **do**
      for $v \in V_t$ **do**
        $\tilde{\lambda}_{kv} := \eta_v + D n_{tv} \phi_{tvk}$.
      end for
    end for
    $\lambda := (1 - \rho_t)\lambda + \rho_t \tilde{\lambda}$
    Update $E_q[\log \beta_{kv}]$ based on λ.
  end for
end function

Appendix B

Collapsed variational Bayes

We give a brief overview of CVB before introducing the algorithm. We refer to Ngo et al. 2016 [NY16] for more details. See section 2.4.1.2 for context.

In CVB, the variational distribution is given by

$$q(\theta, \beta, x, z) = q(\theta, \beta | x, z) \prod_{d=1}^{D} \prod_{n=1}^{N_d} q(x, z | \phi_{dn}), \tag{B.1}$$

where ϕ_{dn} are the *variational parameters*, such that

$$\phi_{dnka} = \begin{cases} q(z_{dn} = k, x_{dn} = a), & \text{if } a \in A_d, \\ 0 & \text{otherwise.} \end{cases}$$

We note that the random variables θ and β are not decoupled in this variational distribution. We also see that θ and β depend on the latent variables x and z, rather than some variational parameter as in standard VB. The basic idea of CVB is that the global variables θ and β are marginalized out of the variational distribution, and only the latent variables $q(x, y|\phi)$ are estimated.

As in standard VB, we have the inequality that defines the lower bound,

$$p(w|\alpha,\eta,A) \quad \geq \quad E_q[\log p(\theta,\beta,x,z,w|\alpha,\eta,A)] \; - \; E_q[\log q(\theta,\beta,x,z)]. \quad \text{(B.2)}$$

By allowing $q(\theta,\beta|x,z) = p(\theta,\beta|x,z,w,\alpha,\eta,A)$, we can marginalize $q(\theta,\beta|x,y)$ out of the lower bound. Now the lower bound becomes

$$\begin{aligned}\mathcal{L}(\phi) &= E_q[\log p(\theta,\beta,x,z,w|\alpha,\eta,A)] - E_q[\log q(\theta,\beta,x,z)] \\ &= E_q[\log p(\theta,\beta,x,z,w|\alpha,\eta,A)] - E_q[\log p(\theta,\beta|x,z,w,\alpha,\eta,A)] \\ &\qquad - E_q[\log q(x,z)] \\ &= E_q[\log p(x,z,w|\alpha,\eta,A)] - E_q[\log q(x,z)]. \quad \text{(B.3)}\end{aligned}$$

This is our objective in CVB. We see that the objective only depends on the latent variables x and z. As in standard VB, we add the Lagrangian, constraining ϕ_{dnak} to sum to one.

$$\mathcal{L}_\lambda(\phi) = E_q[\log p(x,z,w|\alpha,\eta,A)] - E_q[\log q(x,z)] + \sum_{d=1}^{D}\sum_{n=1}^{N_d}\left(\lambda_{dn}\sum_{k=1}^{K}\sum_{a=1}^{A}(\phi_{dnka}-1)\right).$$

We maximize the lower bound by taking its derivative w.r.t. the variational parameters ϕ_{dnka}, equating it to 0, and isolating ϕ_{dnka}.

We skip the derivations required to obtain the update equations for ϕ_{dnka}, they can be found in Ngo et al. 2016 [NY16]. The update equation turns out to be the following.

$$\phi_{dnka} = \frac{\exp(E_{q(z_{-dn},x_{-dn})}(\log(\eta+N^{-dn}_{\cdot\cdot kw_{dn}})-\log(V\eta+N^{-dn}_{\cdot\cdot k\cdot})-\log(K\alpha+N^{-dn}_{\cdot a\cdot\cdot})+\log(\alpha+N^{-dn}_{\cdot ak\cdot})))}{\sum_{a'}\sum_{k'}\exp(E_{q(z_{-dn},x_{-dn})}(\log(\eta+N^{-dn}_{\cdot\cdot k'w_{dn}})-\log(V\eta+N^{-dn}_{\cdot\cdot k'\cdot})-\log(K\alpha+N^{-dn}_{\cdot a'\cdot\cdot})+\log(\alpha+N^{-dn}_{\cdot a'k'\cdot})))}, \quad \text{(B.4)}$$

where $N_{dakw_{dn}}$ is the number of times the dictionary word w_{dn} has been observed in document d where it was assigned to author a and topic k. A dot implies that the corresponding indices are summed out. $N^{-dn}_{dakw_{dn}}$ indicates that w_{dn} is excluded.

B.1 Gaussian approximation

It is important to note one other step involved in CVB, which is Gaussian approximation. Essentially, when we obtain the update equations, they require

the computation of some expectations of a particular form, one of them being $E_{q(z_{-dn},x_{-dn})}[\log(V\eta+N_{..k.}^{-dn})]$. As $N_{..k.}^{-dn}$ is a sum of independent Bernoulli random variables, we can approximate its distribution with a Gaussian. Assuming this, it turns out that

$$E_q(N_{..k.}^{-dn}) = \sum_{(d',n')\neq(d,n)} \sum_{a\in A_{d'}} \phi_{d'n'ka},$$

$$Var_q(N_{..k.}^{-dn}) = \sum_{(d',n')\neq(d,n)} \left(\sum_{a\in A_{d'}} \phi_{d'n'ka}\right)\left(1-\sum_{a\in A_{d'}} \phi_{d'n'ka}\right).$$

Then a second order Taylor series approximation of the expectation we want to evaluate becomes

$$E_{q(z_{-dn},x_{-dn})}[\log(V\eta+N_{..k.}^{-dn})] \approx \log(V\eta+E_q[N_{..k.}^{-dn}]) + \frac{Var_q(N_{..k.}^{-dn})}{2(V\eta+E_q(N_{..k.}^{-dn}))^2}$$

Deriving these Gaussian approximations for all the expectations in equation B.4, we obtain the following update equation:

$$\begin{aligned}\phi_{dnka} \propto\ & (\eta+E_q(N_{..kw_{dn}}^{-dn}))(V\eta+E_q(N_{..k.}^{-dn}))^{-1}(K\alpha+E_q(N_{.a.}^{-dn}))^{-1}(\alpha+E_q(N_{.ak.}^{-dn})) \\ & \times exp\Bigg(-\frac{Var_q(N_{..kw_{dn}}^{-dn})}{2(\eta+E_q(N_{..kw_{dn}}^{-dn}))^2} + \frac{Var_q(N_{..k.}^{-dn})}{2(V\eta+E_q(N_{..k.}^{-dn}))^2} \\ & + \frac{Var_q(N_{.a.}^{-dn})}{2(K\alpha+E_q(N_{.a.}^{-dn}))^2} - \frac{Var_q(N_{.ak.}^{-dn})}{2(\alpha+E_q(N_{.ak.}^{-dn}))^2}\Bigg) \qquad \text{(B.5)}\end{aligned}$$

As the expectations and variances in equation B.5 only depend on ϕ_{dnka}, we see that we do actually not need to keep track of $N_{dakw_{dn}}$.

The expectations and variances in equation B.5 are given below. The first term:

$$E_q(N_{..kw_{dn}}^{-dn}) = \sum_{(d',n')\neq(d,n)} \left(\sum_{a\in A_{d'}} \phi_{d'n'ka}\right) \mathbb{I}(w_{d'n'}=w_{dn})$$

$$Var_q(N_{..kw_{dn}}^{-dn}) = \sum_{(d',n')\neq(d,n)} \left(\sum_{a\in A_{d'}} \phi_{d'n'ka}\right)\left(1-\sum_{a\in A_{d'}} \phi_{d'n'ka}\right) \mathbb{I}(w_{d'n'}=w_{dn})$$

The expectation and variance in the second term in the equation:

$$E_q(N_{..k.}^{-dn}) = \sum_{(d',n')\neq(d,n)} \sum_{a\in A_{d'}} \phi_{d'n'ka}$$

$$Var_q(N_{\cdot\cdot k\cdot}^{-dn}) = \sum_{(d',n')\neq(d,n)} \left(\sum_{a\in A_{d'}} \phi_{d'n'ka}\right)\left(1 - \sum_{a\in A_{d'}} \phi_{d'n'ka}\right)$$

The expectation and variance in the third term in the equation:

$$E_q(N_{\cdot a\cdot\cdot}^{-dn}) = \sum_{(d',n')\neq(d,n)} \sum_k \phi_{d'n'ka}$$

$$Var_q(N_{\cdot a\cdot\cdot}^{-dn}) = \sum_{(d',n')\neq(d,n)} \left(\sum_k \phi_{d'n'ka}\right)\left(1 - \sum_k \phi_{d'n'ka}\right)$$

The expectation and variance in the fourth term in the equation:

$$E_q(N_{\cdot ak\cdot}^{-dn}) = \sum_{(d',n')\neq(d,n)} \phi_{d'n'ka}$$

$$Var_q(N_{\cdot ak\cdot}^{-dn}) = \sum_{(d',n')\neq(d,n)} (\phi_{d'n'ka})\,(1 - \phi_{d'n'ka})$$

B.2 Parameter estimation

Once ϕ_{dnka} has been computed, θ_{ak} and β_{kv} can be estimated as

$$\theta_{ak} = \frac{\alpha + E_q(N_{\cdot ak\cdot})}{K\alpha + E_q(N_{\cdot a\cdot\cdot})} = \frac{\alpha + \sum_{d,n} \phi_{dnka}\mathbb{I}(a \in A_d)}{K\alpha + \sum_{d,n}\sum_{k'} \phi_{dnk'a}\mathbb{I}(a \in A_d)}, \tag{B.6}$$

and

$$\beta_{kv} = \frac{\eta + E_q(N_{\cdot\cdot kv})}{V\eta + E_q(N_{\cdot\cdot k\cdot})} = \frac{\eta + \sum_{d,n,a} \phi_{dnka}\mathbb{I}(w_{dn} = v)\mathbb{I}(a \in A_d)}{V\eta + \sum_{d,n,a} \phi_{dnka}\mathbb{I}(a \in A_d)}. \tag{B.7}$$

B.3 Algorithm

Algorithm 5 shows pseudo-code that ties all the equations described in the sections above together. We make passes over the entire data until we reach our convergence criterion based on the lower bound. We also keep iterating over the same document until it has converged, based on whether the variational parameters change significantly.

Algorithm 5 Training the author-topic model using CVB.

function AT-CVB(w_{dn}, A, K, α, η, τ_1, τ_2)
Initialize $\phi_{dnka} := \frac{1}{A \cdot K}$ for all d, n, k and a.
Evaluate lower bound $\mathcal{L}$, and compute θ_{kv} and β_{ak} using equation B.6 and B.7.
repeat
Set $\mathcal{L}_{prev} := \mathcal{L}$.
for $d = 1$ to D **do**
repeat
for $n = 1$ to N_d **do**
for $k = 1$ to K **do**
for $a = 1 \in A_d$ **do**
Update ϕ_{dnka} according to B.5.
end for
end for
Normalize ϕ_{dnka} (for d, n only):
$\phi_{dnka} := \phi_{dnka} / \left(\sum_{a \in A} \sum_{k=1}^{K} \phi_{dnka} \right)$
end for
until $\max_{n \in N_d} \left\{ \frac{1}{A \cdot K} \sum_{a \in A_d} \sum_{k=1}^{K} |\phi_{dnka} - \phi'_{dnka}| \right\} < \tau_1$.
end for
Compute $\mathcal{L}$.
until $(\mathcal{L} - \mathcal{L}_{prev})/\mathcal{L}_{prev} < \tau_2$
end function

Appendix C

Miscellaneous

C.1 Latent Dirichlet Allocation

The LDA model is described very briefly, and an algorithm for training it is outlined. It is recommended to read section 2.4 before reading this section.

The posterior distribution that describes LDA is

$$\begin{aligned} &p(\theta, \beta, z, w|\alpha, \eta) = p(\beta|\eta)p(\theta|\alpha)p(z|\theta)p(w|z, \beta) \\ &= \prod_{k=1}^{K} Dir(\beta_k|\eta) \prod_{d=1}^{D} Dir(\theta_d|\alpha) \prod_{n=1}^{N_d} Mult(z_{dn}|\theta_d)Mult(w_{dn}|\beta_k, z_{dn} = k). \end{aligned} \quad \text{(C.1)}$$

LDA is represented as a graphical model in figure C.1.

The VB algorithm is derived in a similar fashion as described for the author-topic model in section 2.4.2. In algorithm 6, the procedure for training LDA using offline VB is outlined. This algorithm is equivalent to algorithm 1.

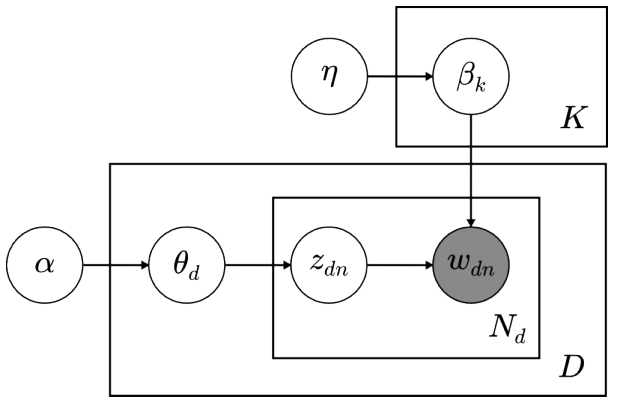


Figure C.1: Graphical model of LDA. See section C.1

Algorithm 6 Training LDA using VB.

function LDA(w_{dn}, K, α, η, τ_1, τ_2)
  Initialize γ and λ randomly according to a gamma distribution.
  Compute $\mathcal{L}$.
  repeat
    Set $\mathcal{L}_{prev} := \mathcal{L}$.
    M-step.
    for $d = 1$ to D **do**
      for $v = 1$ to V **do**
        for $k = 1$ to K **do**
          $\phi_{dvk} \propto \exp\{E_q[\log \theta_{dk}] + E_q[\log \beta_{kv}]\}$.
        end for
        Normalize ϕ_{dvk} to sum to 1 over k:
        $\phi_{dvk} := \phi_{dvk} / \left(\sum_{k=1}^{K} \phi_{dvk}\right)$
      end for
    end for
    for $k = 1$ to K **do**
      $\gamma_{dk} := \alpha_k + \sum_v n_{dv}\phi_{dvk}$.
    end for
    E-step.
    for $k = 1$ to K **do**
      for $v = 1$ to V **do**
        $\lambda_{kv} := \eta_v + \sum_d n_{dv}\phi_{dvk}$.
      end for
    end for
    Compute $\mathcal{L}$.
  until $(\mathcal{L} - \mathcal{L}_{prev})/\mathcal{L}_{prev} < \tau_1$
end function

C.2 Variational lower bound

We show that the marginal likelihood has a variational lower bound defined as

$$\log p(w|\alpha,\eta,A) \geq E_q[\log p(\theta,\beta,x,z,w|\alpha,\eta,A)] - E_q[\log q(\theta,\beta,z,x)],$$

where $q(\theta,\beta,x,z)$ is the variational distribution.

Below, we obtain the marginal likelihood by integrating all the random variables out of the joint distribution. Then we introduce the variational distribution $q(\theta,\beta,x,z)$, formulate the marginal likelihood as an expectation, and finally apply Jensen's inequality to obtain the lower bound.

$$\begin{aligned}
\log p(w|\alpha,\eta,A) &= \log \int\int \sum_x \sum_z \sum_w p(\theta,\beta,x,z,w|\alpha,\eta,A) d\theta d\beta \\
&= \log \int\int \sum_x \sum_z \sum_w \frac{p(\theta,\beta,x,z,w|\alpha,\eta,A) q(\theta,\beta,z,x)}{q(\theta,\beta,z,x)} d\theta d\beta \\
&= \log E_q \left[\frac{p(\theta,\beta,x,z,w|\alpha,\eta,A)}{q(\theta,\beta,z,x)} \right] \\
&\geq E_q \left[\log p(\theta,\beta,x,z,w|\alpha,\eta,A)\right] - E_q \left[\log q(\theta,\beta,z,x)\right] \\
&= \mathcal{L}(q). \quad \text{(C.2)}
\end{aligned}$$

The Jensen's inequality states that if ψ is a concave function, then $\psi(E[X]) \geq E[\psi(X)]$.

It can also be shown that

$$\log p(w|\alpha,\eta,A) = \mathcal{L}(q) + \mathbb{KL}(q||p),$$

where $\mathbb{KL}(q||p)$ is the Kullback-Leibler divergence taken on q and the posterior, defined as

$$\mathbb{KL}(q||p) = -E_q \left[\frac{p(\theta,\beta,x,z|w,\alpha,\eta,A)}{q(\theta,\beta,z,x)} \right].$$

We see that if q is allowed to take any form, it can come arbitrarily close to the posterior; thus, the lower bound is reaches its maximum when the KL-divergence vanishes. The more we constrain the form of q, the less tight the lower bound will be.

C.3 Per-document lower bound

The lower bound is given by

$$
\begin{aligned}
&\mathcal{L}(\gamma, \lambda, \phi) = \\
&\sum_a \left(\sum_k (\alpha_k - \gamma_{ak}) E_q[\log \theta_{ak}] - \log \Gamma(\sum_k \gamma_{ak}) + \sum_k \log \Gamma(\gamma_{ak}) + \log \Gamma(\sum_k \alpha_k) - \sum_k \log \Gamma(\alpha_k) \right) \\
&+ \sum_k \left(\sum_v (\eta_v - \lambda_{kv}) E_q[\log \beta_{kv}] - \log \Gamma(\sum_v \lambda_{kv}) + \sum_v \log \Gamma(\lambda_{kv}) + \log \Gamma(\sum_v \eta_v) - \sum_v \log \Gamma(\eta_v) \right) \\
&\quad + \sum_{d,v} n_{dv} \sum_{a \in A_d} \sum_k \phi_{dvak} (E_q[\log \theta_{ak}] + E_q[\log \beta_{kv}] - \log \phi_{dvak}) \\
&\quad + \sum_{d,v} \frac{1}{|A_d|} + \sum_{d,v} \ell_{dv} \left(\left(\sum_{a \in A_d} \sum_k \phi_{dvak} \right) - 1 \right) \qquad \text{(C.3)}
\end{aligned}
$$

which can be represented as a per-document lower bound

$$
\begin{aligned}
&\mathcal{L}(\gamma, \lambda, \phi) = \sum_d \mathcal{L}_d(\gamma, \lambda, \phi) = \\
&\quad \sum_d \frac{1}{D} \sum_a (\sum_k (\alpha_k - \gamma_{ak}) E_q[\log \theta_{ak}] - \log \Gamma(\sum_k \gamma_{ak}) + \sum_k \log \Gamma(\gamma_{ak}) \\
&\qquad + \log \Gamma(\sum_k \alpha_k) - \sum_k \log \Gamma(\alpha_k)) \\
&\quad + \frac{1}{D} \sum_k (\sum_v (\eta_v - \lambda_{kv}) E_q[\log \beta_{kv}] - \log \Gamma(\sum_v \lambda_{kv}) + \sum_v \log \Gamma(\lambda_{kv}) \\
&\qquad + \log \Gamma(\sum_v \eta_v) - \sum_v \log \Gamma(\eta_v)) \\
&\quad + \sum_v n_{dv} \sum_{a \in A_d} \sum_k \phi_{dvak} (E_q[\log \theta_{ak}] + E_q[\log \beta_{kv}] - \log \phi_{dvak}) \\
&\qquad + \sum_v \frac{1}{|A_d|} + \sum_v \ell_{dv} \left(\left(\sum_{a \in A_d} \sum_k \phi_{dvak} \right) - 1 \right) \qquad \text{(C.4)}
\end{aligned}
$$

C.4 Asymptotic complexity of online VB algorithm

We derive the asymptotic complexity of the online algorithm. These results are useful both for comparing with other algorithms, and to use as a sanity check when testing the scalability of the algorithm empirically, which we will do in section 4. We also derive the asymptotic complexity of computing the lower bound.

The asymptotic complexity of an algorithm is given by "big O" notation, for example as $O(n)$. This essentially means that the algorithm's running time is on the order of n, although in reality it is somewhat more technical.

The complexity is given on a per-document basis; to obtain the complexity of training on a corpus consisting of D documents, simply sum over the complexity of each document.

First, we will obtain the complexity of each update (ϕ, γ and λ). For each document d, there are $V_d A_d K$ updates of ϕ, each costing $O(1)$. Normalizing each ϕ_{dv} costs $O(A_d K)$, so it costs $O(V_d A_d K)$ per document. This makes the complexity of the entire ϕ update $O(V_d A_d K)$.

There are $A_d K$ updates of γ, each costing $O(V_d)$. Updating $E_q[\log \theta_{ak}]$ costs $O(A_d K)$, taking the Ψ function (in the Dirichlet expectation) to be constant $O(1)$. Interpolating between the local and global γ is $O(A_d K)$. This gives $O(A_d K V_d + A_d K + A_d K) = O(A_d K V_d)$.

In addition, if we say that the inner loop runs for a maximum of M iterations, the complexity of the ϕ and γ updates are to be multiplied by M.

There are $O(KV_d)$ updates of λ per document, each costing $O(A_d)$. Interpolating costs $O(KV_d)$, and updating $E_q[\log \beta_{kv}]$ costs $O(KV)$. Updating λ thus costs $O(KV_d A_d + KV_d + KV) = O(KV_d A_d + KV)$.

The complexity of the algorithm is then $O(M(V_d A_d K + A_d K V_d) + KV_d A_d + KV)$, which can be simplified to $O(MV_d A_d K + V_d A_d K + KV)$.

The asymptotic complexity of computing the lower bound is the sum of the per-document bound $O(V_d K A_d)$, the β bound $O(KV)$ and the θ bound $O(AK)$, giving $O(V_d K A_d + KV + AK)$.

Table C.1 summarizes all the important complexities derived in this section.

	Complexity
Full algorithm	$O(MV_dA_dK + V_dA_dK + KV)$
ϕ	$O(V_dA_dK)$
γ	$O(A_dKV_d)$
λ	$O(KV_dA_d + KV)$
Bound	$O(V_dKA_d + KV + AK)$

Table C.1: Complexity of algorithm. See section C.4 for details.

C.5 Vectorized updates details

The vectorized updates are derived.

The γ update is

$$\tilde{\gamma}_{ak} := \alpha_k + |D_a| \sum_v n_{tv} \phi_{tvak},$$

which we can write as

$$\tilde{\gamma}_{ak} := \alpha_k + |D_a| \sum_v n_{tv} \frac{\exp\{E_q[\log \theta_{ak}]\} \exp\{E_q[\log \beta_{kv}]\}}{\sum_{a' \in A_t} \sum_{k'} \exp\{E_q[\log \theta_{a'k'}]\} \exp\{E_q[\log \beta_{k'v}]\}}.$$

Similarly, the λ update is

$$\tilde{\lambda}_{kv} := \eta_v + Dn_{tv} \sum_{a \in A_t} \phi_{tvak},$$

which we can write as

$$\tilde{\lambda}_{kv} := \eta_v + Dn_{tv} \sum_{a \in A_t} \frac{\exp\{E_q[\log \theta_{ak}]\} \exp\{E_q[\log \beta_{kv}]\}}{\sum_{a' \in A_t} \sum_{k'} \exp\{E_q[\log \theta_{a'k'}]\} \exp\{E_q[\log \beta_{k'v}]\}}.$$

Right away, we can see that the denominator in ϕ can be pre-computed before updating γ and λ. This quantity is the normalization factor of ϕ,

$$\phi_{t,norm} := \sum_{a \in A_t} \sum_k \exp\{E_q[\log \theta_{ak}]\} \exp\{E_q[\log \beta_{kv}]\}$$

We can write this as

$$\phi_{t,norm} := \sum_k \exp\{E_q[\log \beta_{kv}]\} \sum_{a \in A_t} \exp\{E_q[\log \theta_{ak}]\},$$

and vectorize the operation as

$$\phi_{t,norm} := \exp\{E_q[\log \beta_{.,v}]\} \sum_{a \in A_t} \exp\{E_q[\log \theta_{a,.}]\},$$

greatly speeding up the computation of the normalization factor in ϕ. $\phi_{t,norm}$ is thus a V_t dimensional vector.

By rearranging γ's update equation, we can compute a row of γ efficiently as

$$\tilde{\gamma}_{a,.} := \alpha + |D_a| \exp\{E_q[\log \theta_{a,.}]\} \frac{n_{t,.}}{\phi_{t,norm}} \exp\{E_q[\log \beta_{.,.}]\}^T.$$

Treating the λ update similarly, we obtain the vectorized update, first defining the *sufficient statistic*,

$$sstats_t = \left(\sum_{a \in A_t} \exp\{E_q[\log \theta_{a,.}]\} \right) \otimes \frac{n_{t,.}}{\phi_{t,norm}}$$

$$\tilde{\lambda}_{k,.} := \eta + (sstats_t)_k$$

In computing the sufficient statistics, we have applied the outer product $\otimes$ (or more generally, the Kronecker product). $sstats_t$ is thus a $K \times V$ matrix.

We have now defined vectorized ways of updating λ and γ and of computing the normalization factor in ϕ.

C.5.1 A note about the sufficient statistics

In practice, we interpolate between the sufficient statistics in the online algorithm, that is

$$sstats := (1 - \rho_t) sstats + \rho_t sstats_t.$$

While the way we described the online algorithm in section 2.4.2.5 η is added every time we interpolate, in practice we only add η when we compute λ.

This choice is a matter of practicality in the implementation. It does change how the algorithm converges, but whether it is for the better or for the worse is not clear.

C.5.2 Bound computation

As we can see in section 2.4.2.3, computing the lower bound requires computing $\phi_{\cdot,norm}$, essentially. So to speed up bound evaluation, we use the same vectorized method as described above to compute ϕ.

Bibliography

[BL06] David M Blei and John D Lafferty. Dynamic Topic Models. *International Conference on Machine Learning*, pages 113–120, 2006.

[BMB08] David M. Blei, Jon D. McAuliffe, and David M. Blei. Supervised Topic Models. *Advances in Neural Information Processing Systems 20*, 21(1):121–128, 2008.

[BNJ03] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[CGWB09] Jonathan Chang, Sean Gerrish, Chong Wang, and David M Blei. Reading Tea Leaves: How Humans Interpret Topic Models. *Advances in Neural Information Processing Systems 22*, pages 288—-296, 2009.

[HBWP13] MD Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic Variational Inference. 2013.

[HG06] Bo-June Hsu and James Glass. Style & topic language model adaptation using HMM-LDA. *Emnlp-2006*, (July):373–381, 2006.

[Min03] Thomas P. Minka. Estimating a Dirichlet distribution. *Annals of Physics*, 2000(8):1–13, 2003.

[Mor16] Olavur Mortensen. The author-topic model: LDA with metadata, http://nbviewer.jupyter.org/github/rare-technologies/gensim/blob/develop/docs/notebooks/atmodel_tutorial.ipynb. 2016.

[Mur12] Kevin P. Murphy. Machine Learning: A Probabilistic Perspective. *MIT Press*, page 25, 2012.

[MWT+11] David Mimno, Hanna M Wallach, Edmund Talley, Miriam Leenders, and Andrew McCallum. Optimizing Semantic Coherence in Topic Models. *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, (2):262–272, 2011.

[NEFY16] Gia H. Ngo, Simon B. Eickhoff, Peter T. Fox, and B.T. Thomas Yeo. Collapsed Variational Bayesian Inference of the Author-Topic Model: Application to Large-Scale Coordinate-Based Meta-Analysis. *Proceedings of the International Conference on Pattern Recognition in Neuroimaging (PRNI)*, 2016.

[NY16] Gia H. Ngo and B.T. Thomas Yeo. Paper that Ngo et al. hopefully will release soon. 2016.

[Reh11] Radim Rehurek. Scalability of Semantic Analysis in Natural Language Processing. *PhD Thesis*, page 147, 2011.

[RS10] Radim Rehurek and Petr Sojka. Software framework for topic modelling with large corpora. In *Lrec*, pages 45–50, 2010.

[RSTA13] Margaret E. Roberts, Brandon M. Stewart, Dustin Tingley, and Eduardo M. Airoldi. The structural topic model and applied social science. *NIPS 2013 Workshop on Topic Models*, pages 2–5, 2013.

[RZGSS04] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth. The author-topic model for authors and documents. *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 487–494, 2004.

[RZGSS05] Michael Rosen-Zvi, Thomas Griffiths, Padhraic Smyth, and Mark Steyvers. Learning author topic models from text corpora. *Journal of Machine Learning Research*, V(October):1–38, 2005.

[TNW07] Yw Teh, D Newman, and M Welling. A Collapsed Variational Bayesian Inference Algorithm for Latent Dirichlet Allocation. *Nips*, 19:1353–1360, 2007.