Understanding Latent Dirichlet Allocation

Park, Sihyung
naturale0@snu.ac.kr

Last Update: February 2021

Latent Dirichlet allocation (LDA) is a three-level bayesian hierarchical model that is frequently used for topic modelling and document classification. First proposed to infer population structure from genotype data, LDA not only allows to represent words as mixtures of topics, but to represent documents as a mixture of words, which makes it a powerful generative probabilistic model.

In this article, I would like to explain backgrounds and model structure, and to implement it from scratch with numpy.

Contents

Backgrounds

Before we get into the detail, I would like to mention and define terms that will be used frequently hereafter.

Tokens (or **words**) are grammatical and/or semantic unit of language, usually separated with each other in a sentence by a space[1]. **Documents** (or **sentence**) are collection of words in specific orders. A **Corpus** is a collection of documents. **Vocabulary** is the set of all words in a corpus.

Topics are latent structure of documents that cannot be explicitly observed, but can be represented as sets of similar words. **Bayesian hierarchical model** is a statistical model that sequentially conditioning random variables and their parameters with higher-order hyperparameters. **Topic modelling** is a process to identify such structure with statistically modeling the documents. **Generative models**, unlike discriminative models, learn the underlying distributional structure so that it can *generate* the random data.

While the terms might not be familiar to some, it will become clear with real model description in the next post.

TF-IDF matrix

In the beginning of embedding documents to numeric vectors, TF-IDF matrix was frequently used. Term frequency-inverse document frequency (TF-IDF) is a simple yet useful index to quickly convert documents into vectors. TF-IDF is

[1] While tokens are actually processed words by stemming/lemmatizing processes, I will not separate the two here.

calculated as follows. First, term frequency (TF) is defined for each words in vocabulary (v_i) and documents($\mathbf{w}_d$).

$$\mathrm{TF}(v_i, \mathbf{w}_d) = (\text{ of occurence of } v_i \text{ in } \mathbf{w}_d)$$

This is sometimes refered to as *raw* term frequency.

Inverse document frequency (IDF) is defined as an adjusted inverse of document frequency (DF).

$$\mathrm{DF}(v_i) = (\text{ of documents contaning } v_i)\mathrm{IDF}(v_i, D) = \log\left(\frac{\text{of documents in } D}{1 + \mathrm{DF}(v_i)}\right)$$

Finally, TF-IDF is a product of TF and IDF.

$$\text{TF-IDF}(v_i, \mathbf{w}_d, D) = \mathrm{TF}(v_i, \mathbf{w}_d) \cdot \mathrm{IDF}(v_i, D)$$

Computing TF-IDF for all documents and words in a corpus yields a document-term matrix. Row vectors of the matrix can be viewed as document embeddings.

Latent Semantic Analysis

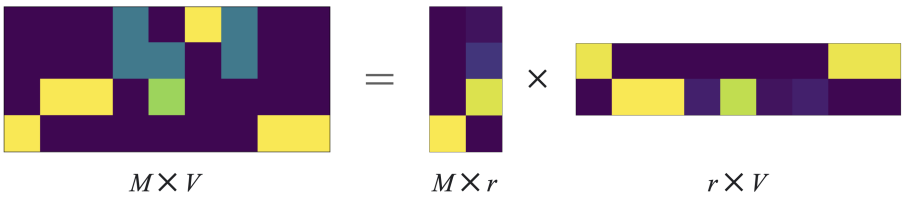

Figure 1: The scheme of latent semantic analysis.

TF-IDF LSA is just a singular value decomposition of TF-IDF matrix. The rationale is to capture lower-dimensional latent structure behind terms and documents. While TF-IDF reveals provides information, the matrix is sparse and the dimension of each word vector is as large as the size of vocabulary (usually in the order of millions). By applying r-truncated singular value decomposition to TF-IDF matrix of size $M \times V$ where M is the size of corpus (total number of documents) and V is the size of vocabulary, we get two matrices of size $M \times r$ and $r \times V$. The first matrix can be used as a document embedding, and the second as a word embedding.

Probabilistic LSA

pLSA is an attempt to model the latent topics as mixture distributions. While LSA is an exact decomposition of TF-IDF matrix, pLSA poses distributional assumptions to word and document generation and finds parameters that best explain it. In this model, **words are realized random variables that follow mixtures of multinomial distributions** and such **mixtures can be veiwed as topics**.

It was a paradigm shift to use hierarchical model for topic modelling, but pLSA has weakness: While pLSA is a generative model for words, it is not generative in terms of (unseen) documents. Hence as the number of documents in the training set grows, the number of parameters of the model linearly grows as well which makes it non-scalable and vulnerable to overfitting.

Latent Dirichlet Allocation

LDA solves the exact problem that pLSA had: it additionally models documents as mixture distribution of topics starting from the meaning behind bag-of-words assumption.

Under BoW assumption, order of words in a document, and in extension, order of documents in a corpus is disregarded. To put it in a statistical statement, BoW assumes that words and documents are *exchangeable*. de Finetti's theorem states that the distribution of a sequence of exchangeable random variables can be represented as a mixture (weighted mean) of distributions of IID random variables. **Thus a desired generative model has not only mixture representations of words, but also of documents.**

The result of consideration is LDA, Details of which will be covered in the next post.

The Model

In the previous section, topic models frequently used at the time of development of LDA was covered. At the end of the post, I briefly introduced the rationale behind LDA. In this post, I would like to elaborate on details of the model architecture.

Generative process

Suppose we have a corpus with vocabulary of size V and every words w and every topics z are one-hot encoded (i.e. $w^i = 1$ and $w^j = 0, j \neq i$ if w is the i-th term in the vocabulary). Define a document $\mathbf{w}$ as a finite sequence of N words

$$\mathbf{w} = (w_1, \cdots, w_N)$$

and denote a corpus D of M documents

$$D = \{\mathbf{w}_1, \cdots, \mathbf{w}_M\}.$$

For k prespecified number of topics, LDA assumes that a d-th document $\mathbf{w}_d \in D$ with length N_d is "generated" in the following steps[2]:

1. $\theta_d \sim \mathcal{D}_k(\alpha)$. The multinomial parameter $\theta_d \in \mathbb{R}^k$ is drawn from k-dimensional Dirichlet distribution.
2. For each words $w_{dn}, n = 1, \cdots, N_d$ in the document:
    1. $z_{dn} \sim \mathcal{M}\_k(1, \theta_d)$. A topic z_{dn} for a word w_{dn} is drawn from k-dim multinomial distribution.
    2. $w_{dn} \| z_{dn}, \beta \sim \mathcal{M}\_V(1, \beta_{i:z^i_{dn}=1})$. A word w_{dn} given the topic z_{dn} is drawn from V-dim multinomial distribution.

$\alpha \in \mathbb{R}^k, \beta \in \mathbb{R}^{k \times V}$ are hyperparameters that is common to all of the documents in a corpus. β is an unknown constant that satisfies $\beta_{ij} = P(w^j = 1 \| z^i = 1)$. That is, given a topic z such that $z^i = 1$, probability of w_n given β follows multinomial distribution with parameter β_i.

$\theta = (\theta_1, \cdots, \theta_M) \in \mathbb{R}^{M \times k}$ can be seen as the document-specific random topic matrix as θ_d generates topic for each words in d-th document as in the process.

Blei et al. presented a graphical representation of the model. This helps understanding the structure as it illustrates the "scope" of each parameters. Only the words w_{dn}'s, that are filled with grey, are observable. Note that each word w is generated by z which is generated by θ which is again generated by α; This makes LDA a *three-level hierarchical model*.

[2]The number of words in a document is determined by Poisson distribution ($N \sim \mathcal{P}(\xi)$). I omitted this part because N is an ancillary variable and it does not make the model description different even if it is considered as a constant.

Mixture representations

In the previous section, I mentioned that under LDA model both words and documents can be represented as mixture distributions by de Finetti's theorem. This property takes full advantage of bag-of-words assumption to reduce the number of parameters in large corpus and makes LDA a proper document model. Here, I would like to elaborate on it.

Words

By marginalizing over topics z, the word distribution is given by

$$P(w_{dn}|\theta_d, \beta) = \sum_z P(w_{dn}|z, \beta)P(z|\theta_d),$$

where it is a mixture distribution of mixture weights $P(z\|\theta_d)$ and components $P(w_{dn}\|z, \beta)$.

Documents

We assumed in LDA that words are generated by topics and topics are exchangable within a document. Thus the joint density of $\theta_d, \mathbf{z}_d, \mathbf{w}_d$ is

$$p(\theta_d, \mathbf{z}_d, \mathbf{w}_d|\alpha, \beta) = P(\theta_d|\alpha) \prod_{n=1}^{N_d} P(z_{dn}|\theta_d)p(w_{dn}|z_{dn}, \beta)$$

Marginalize it over z and use the mixture representation of word distribution to get

$$p(\theta_d, \mathbf{w}_d|\alpha, \beta) = P(\theta_d|\alpha) \prod_{n=1}^{N_d} P(w_{dn}|\theta_d, \beta).$$

Marginalize again with respect to θ_d and we get the continuous mixture representation of the document $\mathbf{w}_d$

$$p(\mathbf{w}_d|\alpha, \beta) = \int P(\theta_d|\alpha) \prod_{n=1}^{N_d} P(w_{dn}|\theta_d, \beta)\ d\theta_d,$$

where $P(\theta_d\|\alpha)$ are weights and $\prod_{n=1}^{N_d} P(w_{dn}\|\theta_d, \beta)$ are components.

Variational EM

Now that we know the structure of the model, it is time to fit the model parameters with real data. Among the possible inference methods, in this section I would like to explain the *variational expectation-maximization* algorithm.

Variational inference

Variational inference (VI) is a method to approximate complicated distributions with a family of simpler surrogate distributions. In order to compute posterior distribution of latent variables given a document $\mathbf{w}_d$

$$p(\theta_d, \mathbf{z}_d|\mathbf{w_d}, \alpha, \beta) = \frac{p(\theta_d, \mathbf{z}_d, \mathbf{w_d}|\alpha, \beta)}{p(\mathbf{w_d}|\alpha, \beta)},$$

it is necessary to compute the denominator

$$p(\mathbf{w_d}|\alpha, \beta) = \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \int \left(\prod_{i=1}^{k} \theta_{di}^{\alpha_i - 1} \right) \left(\prod_{n=1}^{N} \prod_{i=1}^{k} \prod_{j=1}^{V} (\theta_{di}\beta_{ij})^{w_{dn}^j} \right) d\theta.$$

However, it is intractable due to the coupling of θ and β at the inner most parenthesis. Because of this, we utilize VI and Jensen's inequality to achieve lower bound of the log likelihood $\log p(\mathbf{w}\|\alpha, \beta)$ for parameter estimation of LDA.

To be specific, we let the variational distribution q to be parametrized by the variational parameters $\gamma = \gamma(\mathbf{w})$ and $\phi = \phi(\mathbf{w})$, each works similarly to α and β of the true distribution, respectively. We set the variational distribution

$$q(\theta_d, \mathbf{z}_d|\gamma(\mathbf{w}_d), \phi(\mathbf{w}_d)) = q(\theta_d|\gamma(\mathbf{w}_d)) \prod_{n=1}^{N_d} q(z_{dn}|\phi_n(\mathbf{w}_d))$$

where

$$\theta_d \sim \mathcal{D}_k(\gamma(\mathbf{w}_d)), z_{dn} \sim \mathcal{M}_k(\phi(\mathbf{w}_d)).$$

Graphical representation of the surrogate is depicted in the figure 5 of Blei et al. (2003).

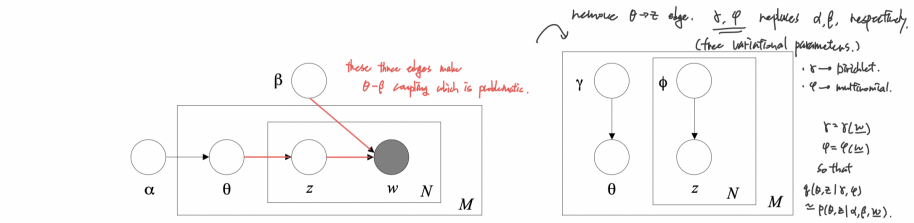

Figure 5: (Left) Graphical model representation of LDA. (Right) Graphical model representation of the variational distribution used to approximate the posterior in LDA.

Figure 2: The latent Dirichlet allocation by variational expectation-maximization algorithm.

Variational EM

Expectation maximization is a special case of minorization-maximization (MM) algorithm. I would like to use terminology from MM since it is more intuitive to explain the variational EM. To maximize a target likelihood function $f(x)$, EM algorithm works in the following way:

1. Set a family of simpler surrogate functions $\mathcal{G}$.
2. Repeat until convergence:
    1. (E-step) Minorize f at $x^{(t)}$ with $g^{(t+1)} \in \mathcal{G}$.
    2. (M-step) Update $x^{(t+1)} = \arg\max g^{(t+1)}(x)$.

Variational EM follows the framework of expectation-maximization, while uses variational inference to minorize the target function at the E-step. By Jensen's inequality, we get the lower bound on the log likelihood with respect to the variational distribution defined above:

$$\log p(\mathbf{w}|\alpha, \beta) \geq \mathrm{E}_q \log p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta) - \mathrm{E}_q \log q(\theta, \mathbf{z}) =: L(\gamma, \phi|\alpha, \beta).$$

Then variational EM algorithm for solving LDA is as follows:

1. Repeat until convergence:
    1. (E-step) Update $(\gamma^{(t+1)}, \phi^{(t+1)}) = \arg\max_{(\gamma,\phi)} L(\gamma, \phi \| \alpha^{(t)}, \beta^{(t)})$.
    2. (M-step) Update $(\alpha^{(t+1)}, \beta^{(t+1)}) = \arg\max_{(\alpha,\beta)} L(\gamma^{(t+1)}, \phi^{(t+1)} \| \alpha, \beta)$.

For β, ϕ and γ, closed form update formula can be easily derived by differentiating L, forming the Lagrangian and setting it to zero. I will leave this part for reading

since it is a trivial, exhausting calculation well-described in appendix A.3 (Blei et al., 2003).

For α, since L has Hessian of the form $\mathrm{diag}(h) + \mathbf{1}z\mathbf{1}'$, we use linear-time Newton-Raphson algorithm to update it.

To summarize, LDA solving variational EM algorithm repeats the following until the parameters converge.

- In the E-step, for $d = 1, \cdots, M$,
  1. For $n = 1, \cdots, N_d$ and $i = 1, \cdots, k$,
     1. $\phi_{dni}^{(t+1)} = \beta_{iw_{dn}} \exp\left(\Psi(\gamma_{di}^{(t)}) - \Psi\left(\sum_{i=1}^{k} \gamma_{di}^{(t)}\right)\right)$.
  2. Normalize $\phi_{dn}^{(t+1)}$ to sum to 1.
  3. $\gamma_{d}^{(t+1)} = \alpha^{(t)} + \sum_{n=1}^{N_d} \phi_{dn}^{(t+1)}$.
- In the M-step,
  1. $\beta_{ij}^{(t+1)} = \sum_{d=1}^{M} \sum_{n=1}^{N_d} \phi_{dni}^{(t+1)} \mathbf{w}_{dn}^{j}$.
  2. Update $\alpha^{(t+1)}$ with linear-time Newton method.

Here, Ψ is the digamma function. I have yet clarified the update rule of linear-time Newton-Raphson algorithm. This is in appendix A.4.2 of Blei et al. (2003), which in its core is a mere block matrix inversion formula

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}.$$

So I will replace the explanation to Python implementation (`_update()`) below.

Python implementation from scratch

E-step

```
def E_step(docs, phi, gamma, alpha, beta):
    """
    Minorize the joint likelihood function via variational inference.
    This is the E-step of variational EM algorithm for LDA.
    """
    # optimize phi
    for m in range(M):
        phi[m, :N[m], :] = (beta[:, docs[m]] * np.exp(
            psi(gamma[m, :]) - psi(gamma[m, :].sum())
        ).reshape(-1, 1)).T
```

```
        # Normalize phi
        phi[m, :N[m]] /= phi[m, :N[m]].sum(axis=1).reshape(-1, 1)
        if np.any(np.isnan(phi)):
            raise ValueError("phi nan")

    # optimize gamma
    gamma = alpha + phi.sum(axis=1)

    return phi, gamma
```

It is the exact translation of the update equation at the above.

M-step

```
def M_step(docs, phi, gamma, alpha, beta, M):
    """
    maximize the lower bound of the likelihood.
    This is the M-step of variational EM algorithm for (smoothed) LDA.

    update of alpha follows from appendix A.2 of Blei et al., 2003.
    """
    # update alpha
    alpha = _update(alpha, gamma, M)

    # update beta
    for j in range(V):
        beta[:, j] = np.array(
            [_phi_dot_w(docs, phi, m, j) for m in range(M)]
        ).sum(axis=0)
    beta /= beta.sum(axis=1).reshape(-1, 1)

    return alpha, beta
```

This is also the exact replication of the equations, but with some abstraction for readability.

`_update()` is the implementation of linear-time Newton-Raphson algorithm.

```
import warnings

def _update(var, vi_var, const, max_iter=10000, tol=1e-6):
    """
    From appendix A.2 of Blei et al., 2003.
    For hessian with shape `H = diag(h) + 1z1'`
```

```
    To update alpha, input var=alpha and vi_var=gamma, const=M.
    To update eta, input var=eta and vi_var=lambda, const=k.
    """
    for _ in range(max_iter):
        # store old value
        var0 = var.copy()

        # g: gradient
        psi_sum = psi(vi_var.sum(axis=1)).reshape(-1, 1)
        g = const * (psi(var.sum()) - psi(var)) \
            + (psi(vi_var) - psi_sum).sum(axis=0)

        # H = diag(h) + 1z1'
        ## z: Hessian constant component
        ## h: Hessian diagonal component
        z = const * polygamma(1, var.sum())
        h = -const * polygamma(1, var)
        c = (g / h).sum() / (1./z + (1./h).sum())

        # update var
        var -= (g - c) / h

        # check convergence
        err = np.sqrt(np.mean((var - var0) ** 2))
        crit = err < tol
        if crit:
            break
    else:
        warnings.warn(f"max_iter={max_iter} reached: values might not be optimal.")

    return var
```

_phi_dot_w() computes $\sum_{n=1}^{N_d} {}_{dni} w_{dn}^{j}$.

```
def _phi_dot_w(docs, phi, d, j):
    """
    \sum_{n=1}^{N_d} _{dni} w_{dn}^j
    """
    return (docs[d] == j) @ phi[d, :N[d], :]
```

Results

I ran LDA inference on $M = 2000$ documents of Reuters News title data with $k = 10$ topics. Top 9 important words in each topic (a mixture of word distribution) extracted from fitted LDA is as follows:

```
TOPIC 00: ['ec' 'csr' 'loss' 'bank' 'icco' 'unit' 'cocoa' 'fob' 'petroleum']
TOPIC 01: ['raises' 'acquisition' 'prices' 'prime' 'rate' 'w' 'completes' 'mar', 'imports']
TOPIC 02: ['year' 'sets' 'sees' 'net' 'stock' 'dividend' 'l' 'industries' 'corp']
TOPIC 03: ['pct' 'cts' 'gdp' 'opec' 'shr' 'february' 'plc' 'ups' 'rose']
TOPIC 04: ['u' 'japan' 'fed' 'trade' 'gaf' 'ems' 'says' 'dlr' 'gnp']
TOPIC 05: ['usda' 'f' 'ag' 'international' 'sell' 'report' 'ge' 'corn' 'wheat']
TOPIC 06: ['k' 'market' 'money' 'rate' 'eep' 'treasury' 'prime' 'says' 'mln']
TOPIC 07: ['qtr' 'note' '4th' 'net' 'loss' 'ico' 'corp' '1st' 'group']
TOPIC 08: ['dlrs' 'mln' 'bp' 'march' 'corp' 'week' 'canada' 'e' 'bank']
TOPIC 09: ['unit' 'buy' 'says' 'sale' 'c' 'sells' 'completes' 'american' 'grain']
```

Topic-word distribution (β) and document-topic distribution (θ) recovered from LDA is as follows. i-th column from the figure left represents probabilities of each words to be generated given the topic z_i so it sums to 1. Similarly, d-th row from the figure right represents the d-th document($\mathbf{w}_d$)'s mixture weights on topics, so it also sums to 1.

Full code and result are available here (GitHub).

Gibbs Sampling

In the last section, I explained LDA parameter inference using variational EM algorithm and implemented it from scratch. In this section, let's take a look at another algorithm proposed in the original paper that introduced LDA to derive approximate posterior distribution: *Gibbs sampling*. In addition, I would like to introduce and implement from scratch a *collapsed Gibbs sampling method* that can efficiently fit topic model to the data.

Problem setting in the original paper

Pritchard and Stephens (2000) originally proposed the idea of solving population genetics problem with three-level hierarchical model. The problem they wanted to address was "inference of population struture using multilocus genotype data." For those who are not familiar with population genetics, this is basically a clustering problem that aims to cluster individuals into clusters (population)

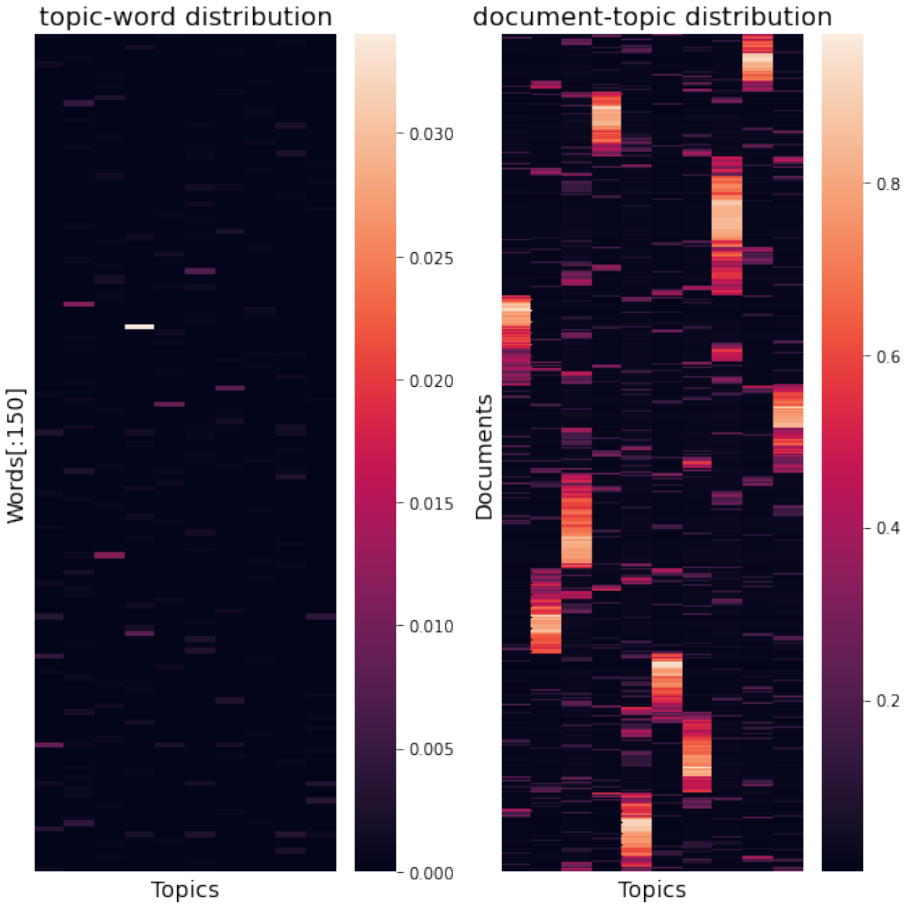

Figure 3: Result of the variational EM

based on similarity of genes (genotype) of multiple prespecified locations in DNA (multilocus).

The researchers proposed two models: one that only assigns one population to each individuals ("model without admixture"), and another that assigns mixture of populations ("model with admixture"). The latter is the model that later termed as LDA. Before we get to the inference step, I would like to briefly cover the original model with the terms in population genetics, but with notations I used in the previous sections.

"Model with admixture"

In population genetics setup, our notations are as follows:

- V is the total number of possible alleles in every loci.
- w_n: genotype of the n-th locus. One-hot encoded so that $w_n^i = 1$ and $w_n^j = 0, \forall j \neq i$ for one $i \in V$.
- z_n: population of origin of w_n.
- $\mathbf{w}\_d = (w_{d1}, \cdots, w_{dN})$: genotype of d-th individual at N loci.
- $D = (\mathbf{w}_1, \cdots, \mathbf{w}_M)$: whole genotype data with M individuals.

Generative process of genotype of d-th individual $\mathbf{w}_d$ with k predefined populations described on the paper is a little different than that of Blei et al. (2003).

1. $\beta_k \sim \mathcal{D}_V(\eta)$.
2. $\theta_d \sim \mathcal{D}_k(\alpha)$. θ_di is the probability that d-th individual's genome is originated from population i.
3. for $n = 1, \cdots, N$:
    1. z_{dn} is chosen with probability $P(z_{dn}^i = 1 \| \theta_d, \beta) = \theta_{di}$.
    2. w_{dn} is chosen with probability $P(w_{dn}^i = 1 \| z_{dn}, \theta_d, \beta) = \beta_{ij}$.

Since β is independent to θ_d and affects the choice of w_{dn} only through z_{dn}, I think it is okay to write $P(z_{dn}^i = 1 \| \theta_d) = \theta_{di}$ instead of formula at 2.1 and $P(w_{dn}^i = 1 \| z_{dn}, \beta) = \beta_{ij}$ instead of 2.2.

The only difference between this and (vanilla) LDA that I covered so far is that β is considered a Dirichlet random variable here. In fact, this is exactly the same as **smoothed LDA** described in Blei et al. (2003) which will be described in the next section.

Gibbs sampling

To estimate the intracktable posterior distribution, Pritchard and Stephens (2000) suggested using **Gibbs sampling**. Gibbs sampling is a method of Markov chain Monte Carlo (MCMC) that approximates intractable joint distribution by consecutively sampling from conditional distributions. Suppose we want to sample from joint distribution $p(x_1, \cdots, x_n)$. Assume that even if directly sampling from it is impossible, sampling from conditional distributions $p(x_i \| x_1 \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$ is possible. Then repeatedly sampling from conditional distributions as follows

1. Repeat:
    1. Sample $x_1^{(t+1)}$ from $p(x_1 \| x_2^{(t)}, \cdots, x_n^{(t)})$.
    2. Sample $x_2^{(t+1)}$ from $p(x_2 \| x_1^{(t+1)}, x_3^{(t)}, \cdots, x_n^{(t)})$.
    3. $\vdots$
    4. Sample $x_n^{(t+1)}$ from $p(x_n \| x_1^{(t+1)}, \cdots, x_{n-1}^{(t+1)})$.

gives us an approximate sample $(x_1^{(m)}, \cdots, x_n^{(m)})$ that can be considered as sampled from the joint distribution for large enough m's.

Below is a paraphrase, in terms of familiar notation, of the detail of the Gibbs sampler that samples from posterior of LDA. Let

$$\mathbf{n}_i = (n_{i1}, \cdots, n_{iV}) \mathbf{m}_d = (m_{d1}, \cdots, m_{dk})$$

where n_{ij} the number of occurrence of word j under topic i, m_{di} is the number of loci in d-th individual that originated from population i.

1. Update $\beta^{(t+1)}$ with a sample from $\beta_i \| \mathbf{w}, \mathbf{z}^{(t)} \sim \mathcal{D}_V(\eta + \mathbf{n}_i)$.
2. Update $\theta^{(t+1)}$ with a sample from $\theta_d \| \mathbf{w}, \mathbf{z}^{(t)} \sim \mathcal{D}_k(\alpha^{(t)} + \mathbf{m}_d)$.
3. Update $\mathbf{z}_d^{(t+1)}$ with a sample by probability

$$P(z_{dn}^i = 1 | \mathbf{w}, \beta^{(t+1)}) = \frac{\theta_{di} \beta_{iw_{dn}}^{(t+1)}}{\sum_{d=1}^{M} \theta_{di} \beta_{iw_{dn}}^{(t+1)}}.$$

4. Update $\alpha^{(t+1)}$ by the following process:
    1. Sample α' from $\mathcal{N}(\alpha^{(t)}, \sigma^2_{\alpha^{(t)}})$ for some $\sigma^2_{\alpha^{(t)}}$.
    2. Let $a = \frac{p(\alpha' \| \theta^{(t)}, \mathbf{w}, \mathbf{z}^{(t)})}{p(\alpha^{(t)} \| \theta^{(t)}, \mathbf{w}, \mathbf{z}^{(t)})} \cdot \frac{\phi_{\alpha'}(\alpha^{(t)})}{\phi_{\alpha^{(t)}}(\alpha')}$.
    3. Do not update $\alpha^{(t+1)}$ if $\alpha' \leq 0$. Update $\alpha^{(t+1)} = \alpha'$ if $a \geq 1$, otherwise update it to α' with probability a.

The update rule in step 4 is called **Metropolis-Hastings algorithm**.

Collapsed Gibbs sampling

While the proposed sampler works, in topic modelling we only need to estimate document-topic distribution θ and topic-word distribution β. Griffiths and Steyvers (2002) boiled the process down to evaluating the posterior $P(\mathbf{z}\|\mathbf{w}) \propto P(\mathbf{w}\|\mathbf{z})P(\mathbf{z})$ which was intractable. Notice that we marginalized the target posterior over β and θ. This makes it a **collapsed Gibbs sampler**; the posterior is *collapsed* with respect to β, θ.

Marginalizing the Dirichlet-multinomial distribution $P(\mathbf{w}, \beta\|\mathbf{z})$ over β from *smoothed* LDA, we get the posterior topic-word assignment probability

$$P(\mathbf{w}|\mathbf{z}) = \left(\frac{\Gamma(V\eta)}{\Gamma(\eta)^V}\right)^k \prod_{i=1}^{k} \frac{\prod_{j=1}^{V}\Gamma(n_{ij}+\eta)}{\Gamma(n_{i\cdot}+V\eta)}$$

where n_{ij} is the number of times word j has been assigned to topic i, just as in the vanilla Gibbs sampler. Marginalizing another Dirichlet-multinomial $P(\mathbf{z}, \theta)$ over θ yields

$$P(\mathbf{z}) = \left(\frac{\Gamma(k\alpha)}{\Gamma(\alpha)^k}\right)^M \prod_{d=1}^{M} \frac{\prod_{i=1}^{k}\Gamma(n_{di}+\alpha)}{\Gamma(n_{d\cdot}+k\alpha)}$$

where n_{di} is the number of times a word from document d has been assigned to topic i. Multiplying these two equations, we get

$$P(z_{dn}^i = 1|\mathbf{z}_{(-dn)}, \mathbf{w}) \propto \frac{n_{(-dn),iw_{dn}}+\eta}{n_{(-dn),i\cdot}+V\eta}\frac{n_{(-dn),di}+\alpha}{n_{(-dn),d\cdot}+k\alpha},$$

where $\mathbf{z}\_(-dn)$ is the word-topic assignment for all but n-th word in d-th document, $n_{(-dn)}$ is the count that does not include current assignment of z_{dn}.

The first term can be viewed as a (posterior) probability of $w_{dn}\|z_i$ (i.e. β_{dni}), and the second can be viewed as a probability of z_i given document d (i.e. θ_{di}). We run sampling by sequentially sample $z_{dn}^{(t+1)}$ given $\mathbf{z}_{(-dn)}^{(t)}, \mathbf{w}$ after one another.

After sampling $\mathbf{z}\|\mathbf{w}$ with Gibbs sampling, we recover θ and β with

$$\hat{\beta}_{iw_n} = \frac{n_{iw_n}+\eta}{n_{i\cdot}+V\eta}, \hat{\theta}_{di} = \frac{n_{di}+\alpha}{n_{d\cdot}+k\alpha},$$

which are marginalized versions of the first and second term of the last equation, respectively.

Python implementation from scratch

Here, I would like to implement the collapsed Gibbs sampler only, which is more memory-efficient and easy to code.

The sampler

This is the entire process of gibbs sampling, with some abstraction for readability.

```
def run_gibbs(docs, vocab, n_topic, n_gibbs=2000, verbose=True):
    """
    Run collapsed Gibbs sampling
    """
    # initialize required variables
    _init_gibbs(docs, vocab, n_topic, n_gibbs)

    if verbose:
        print("\n", "="*10, "START SAMPLER", "="*10)

    # run the sampler
    for t in range(n_gibbs):
        for d in range(M):
            for n in range(N[d]):
                w_dn = docs[d][n]

                # decrement counter
                i_t = assign[d, n, t]  # previous assignment
                n_iw[i_t, w_dn] -= 1
                n_di[d, i_t] -= 1

                # assign new topics
                prob = _conditional_prob(w_dn, d)
                i_tp1 = np.argmax(np.random.multinomial(1, prob))

                # increment counter with new assignment
                n_iw[i_tp1, w_dn] += 1
                n_di[d, i_tp1] += 1
                assign[d, n, t+1] = i_tp1
```

```
        # print out status
        if verbose & ((t+1) % 50 == 0):
            print(f"Sampled {t+1}/{n_gibbs}")
```

In `_init_gibbs()`, instantiate variables (numbers `V`, `M`, `N`, `k` and hyperparameters `alpha`, `eta` and counters and assignment table `n_iw`, `n_di`, `assign`).

```
def _init_gibbs(docs, vocab, n_topic, n_gibbs=2000):
    """
    Initialize t=0 state for Gibbs sampling.
    Replace initial word-topic assignment
    ndarray (M, N, N_GIBBS) in-place.
    """
    # initialize variables
    init_lda(docs, vocab, n_topic=n_topic, gibbs=True)

    # word-topic assignment
    global assign
    N_max = max(N)
    assign = np.zeros((M, N_max, n_gibbs+1), dtype=int)
    print(f"assign: dim {assign.shape}")

    # initial assignment
    for d in range(M):
        for n in range(N[d]):
            # randomly assign topic to word w_{dn}
            w_dn = docs[d][n]
            assign[d, n, 0] = np.random.randint(k)

            # increment counters
            i = assign[d, n, 0]
            n_iw[i, w_dn] += 1
            n_di[d, i] += 1
```

`_conditional_prob()` is the function that calculates $P(z^i_{dn} = 1 \| \mathbf{z}_{(-dn)}, \mathbf{w})$ using the multiplicative equation above.

```
def _conditional_prob(w_dn, d):
    """
    P(z_{dn}^i=1 | z_{(-dn)}, w)
    """
    prob = np.empty(k)

    for i in range(k):
        # P(w_dn | z_i)
        _1 = (n_iw[i, w_dn] + eta) / (n_iw[i, :].sum() + V*eta)
        # P(z_i | d)
        _2 = (n_di[d, i] + alpha) / (n_di[d, :].sum() + k*alpha)
```

```
        prob[i] = _1 * _2

    return prob / prob.sum()
```

After running `run_gibbs()` with appropriately large `n_gibbs`, we get the counter variables `n_iw`, `n_di` from posterior, along with the assignment history `assign` where `[:, :, t]` values of it are word-topic assignment at sampling t-th iteration.

Recover $\hat{\beta}$ and $\hat{\theta}$

Now we need to recover topic-word and document-topic distribution from the sample.

```
 = np.empty((k, V))
 = np.empty((M, k))

for j in range(V):
    for i in range(k):
        [i, j] = (n_iw[i, j] + eta) / (n_iw[i, :].sum() + V*eta)

for d in range(M):
    for i in range(k):
        [d, i] = (n_di[d, i] + alpha) / (n_di[d, :].sum() + k*alpha)
```

Finally we can plot it as a heatmap.

Full code and result are available here (GitHub).

Smoothed LDA

From background to two inference processes, I covered all the important details of LDA so far. One thing left over is a difference between (basic) LDA and *smooth LDA*. Consider this last section as a cherry on top.

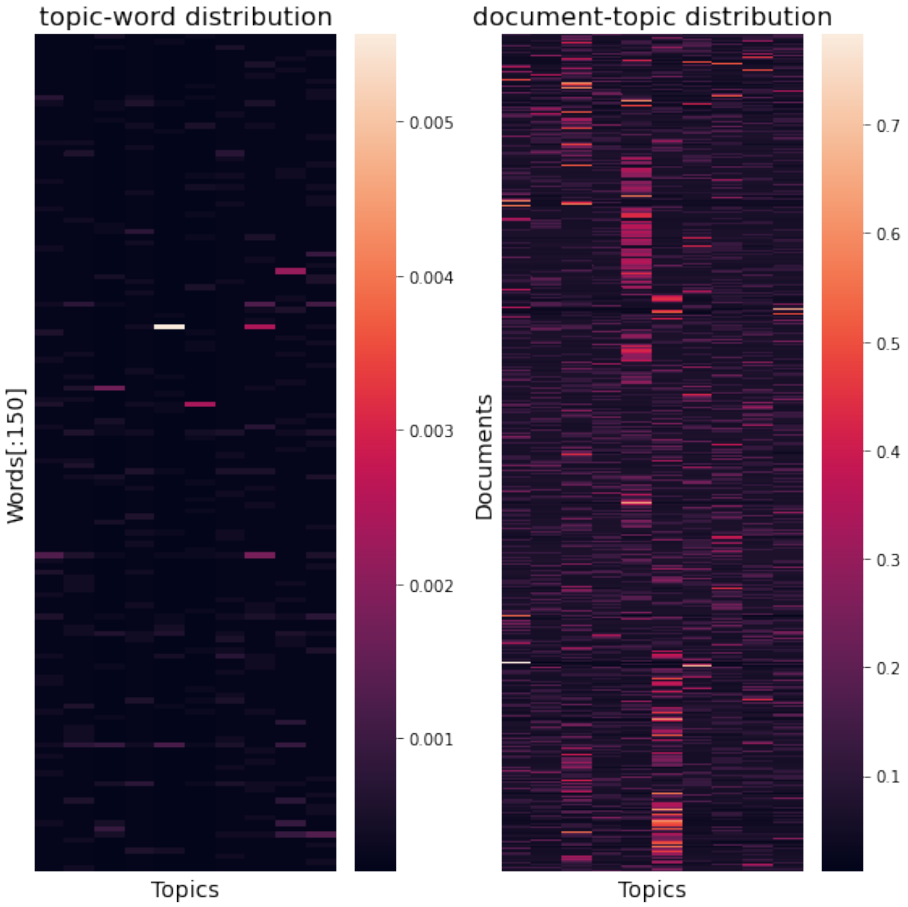

Figure 4: Result of the Gibbs sampling

Empirical vs. fuller Bayes

Not every Bayesian approches are the same. One of the most important aspect when applying Bayesian model to real-world data is decision of hyperparameters. For the task, empirical Bayes method "fit" the model to the data to derive, in many times, *point estimate* of hyperparameter, while standard (fuller) Bayes predefine the *prior distribution* before any data and update it later using the data as an evidence.

Recall the basic LDA that I explained from the first to the third sections. Pay attention to the part that hyperparameter β does not have a distribution; it is assumed to be an unknown but *fixed* value. This makes the basic LDA an empirical Bayes model. Can we extend this to make a fuller Bayes model?

Smooth LDA

In order to extend the model to be fully Bayesian, all we need to do is to regard β as another hidden parameter by endowing β a Dirichlet prior $\beta_i \sim \mathcal{D}_V(\vec{\eta})$ for all $i = 1, \cdots, k$ where $\vec{\eta} = (\eta, \cdots, \eta)$ is a V-length vectors of **the same elements** η by exchangeability. Blei et al. (2003) provides a graphical representation of this extended, or **smoothed** version of LDA. I added a modified variational distribution for it next to the figure.

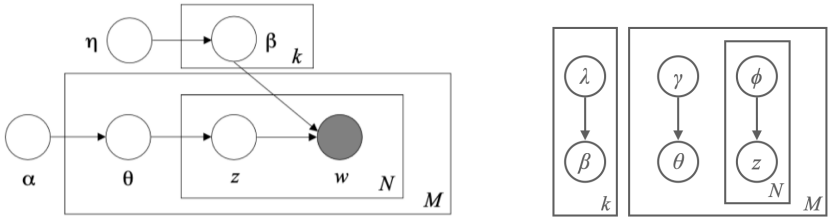

Figure 7: Graphical model representation of the smoothed LDA model. Variational distribution

Figure 5: The smoothed LDA

So why would we want to make a model fully Bayesian? This is because of generability. If we use fixed point estimate for β, we get word-topic assignment that assigns probability zero to words that were not in training data. By giving β a distribution, we can smooth it out to assign positive probability to all known and unknown words. In this perspective, fuller Bayes method can be seen as a smoothing method for hierarchical models.

Variational EM for smooth LDA

I already explained inference methods for smooth LDA: Gibbs sampling with Metropolis-Hastings rule proposed by Pritchard et al. (2000), and Collapsed gibbs sampling that Griffiths and Steyvers (2002) proposed are the ones. Here I would like to continue the discussion and be more specific on variational EM method that Blei et al. (2003) presented. If you are not familiar with variational EM algorithm, please take a look at the previous sections before moving on.

E-step

As in the figure that I put next to the Figure 7, we add additional variational parameter λ that acts as a surrogate of η and consider $\beta_i \sim \mathcal{D}_V(\lambda_i)$ for $i = 1, \cdots, k$. Then the variational distribution becomes

$$q(\theta, \mathbf{z}, \beta | \gamma, \phi, \lambda) = \prod_{i=1}^{k} \mathcal{D}_V(\beta_i | \lambda_i) \prod_{d=1}^{M} q\big(\theta_d, \mathbf{z}_d | \gamma(\mathbf{w}_d), \phi(\mathbf{w}_d)\big)$$

Since β part is multiplicative to the others, update rule for ϕ and γ remains the same. (Actually, update rule of ϕ changes a little as it contains a β term which is now considered a random variable.) Update rule for λ is easy to derive by (again) differentiating the variational lower bound L and setting it to zero.

There were no derivation depicted in the Blei et al. (2003), so I did one by myself. The variational lower bound L is

$$
\begin{aligned}
L(\gamma,\phi,\lambda|\alpha,\eta) = &\sum_{i=1}^{k}\left(\log\Gamma(V\eta) - V\log\Gamma(\eta) + (\eta-1)\sum_{j=1}^{V}\left(\Psi(\lambda_{ij}) - \Psi\left(\sum_{j=1}^{V}\lambda_{ij}\right)\right)\right)\\
&+\sum_{d=1}^{M}\left(\log\Gamma\left(\sum_{i=1}^{k}\alpha_i\right) - \sum_{i=1}^{k}\log\Gamma(\alpha_i) + \sum_{i=1}^{k}(\alpha_i-1)\left(\Psi(\gamma_{di}) - \Psi\left(\sum_{i=1}^{k}\gamma_{di}\right)\right)\right)\\
&+\sum_{d=1}^{M}\sum_{n=1}^{N_d}\sum_{i=1}^{k}\phi_{dni}\left(\Psi(\gamma_{di}) - \Psi\left(\sum_{i=1}^{k}\gamma_{di}\right)\right)\\
&+\sum_{d=1}^{M}\sum_{n=1}^{N_d}\sum_{i=1}^{k}\sum_{j=1}^{V}\phi_{dni}w_{dn}^{j}\left(\Psi(\lambda_{ij}) - \Psi\left(\sum_{j=1}^{V}\lambda_{ij}\right)\right)\\
&-\sum_{i=1}^{k}\left(\log\Gamma\left(\sum_{j=1}^{V}\lambda_{ij}\right) - \sum_{j=1}^{V}\log\Gamma(\lambda_{ij}) + \sum_{j=1}^{V}(\lambda_{ij}-1)\left(\Psi(\lambda_{ij}) - \Psi\left(\sum_{j=1}^{V}\lambda_{ij}\right)\right)\right)\\
&-\sum_{d=1}^{M}\left(\log\Gamma\left(\sum_{i=1}^{k}\gamma_{di}\right) - \sum_{i=1}^{k}\log\Gamma(\gamma_{di}) + \sum_{i=1}^{k}(\gamma_{di}-1)\left(\Psi(\gamma_{di}) - \Psi\left(\sum_{i=1}^{k}\gamma_{di}\right)\right)\right)\\
&-\sum_{d=1}^{M}\sum_{n=1}^{N_d}\sum_{i=1}^{k}\phi_{dni}\log\phi_{dni}.
\end{aligned}
$$

Organize only the terms with λ_{ij} and differentiate it then we get

$$
\begin{aligned}
\frac{\partial L}{\partial\lambda_{ij}} = &\eta\left(\Psi'(\lambda_{ij}) - \Psi'\left(\sum_{j=1}^{V}\lambda_{ij}\right)\right)\\
&+\sum_{d=1}^{M}\sum_{n=1}^{N_d}\phi_{dni}w_{dn}^{j}\left(\Psi'(\lambda_{ij}) - \Psi'\left(\sum_{j=1}^{V}\lambda_{ij}\right)\right)\\
&-M\left(\Psi\left(\sum_{j=1}^{V}\lambda_{ij}\right) - \Psi\left(\lambda_{ij}\right)\right)\\
&-M\left(\Psi(\lambda_{ij}) - \Psi\left(\sum_{j=1}^{V}\lambda_{ij}\right)\right)\\
&-\lambda_{ij}\left(\Psi'(\lambda_{ij}) - \Psi'\left(\sum_{j=1}^{V}\lambda_{ij}\right)\right)
\end{aligned}
$$

Thus letting it zero yields the update rule for λ:

$$
\lambda_{ij}^{(t+1)} = \eta^{(t)} + \sum_{d=1}^{M}\sum_{n=1}^{N_d}\phi_{dni}^{(t)}w_{dn}^{j}.
$$

M-step

Update rule for α is the same: we update it using linear-time Newton-Raphson. What is important is that update rule for η is also the same as α. It can be confirmed by organizing only the terms with η.

$$L_{[\eta]} = \sum_{i=1}^{k} \left(\log \Gamma(V\eta) - V \log \Gamma(\eta) + (\eta - 1) \sum_{j=1}^{V} \left(\Psi(\lambda_{ij}) - \Psi\left(\sum_{j=1}^{V} \lambda_{ij} \right) \right) \right).$$

This is exactly the same as $L_{[\alpha]}$ if replacing η with α, k with M, and η with γ. (Recall that $\vec{\eta}$ is a vector of all the same elements) Thus by exactly the same algorithm (implemented as `_update()`), we can update η.

To summarize, the **whole process of variational EM is as follows**: First, initialize $\phi_{dni}^{(0)} := 1/k$ for all $i = 1, \cdots, k$, $n = 1, \cdots, N_d$ and $d = 1, \cdots, M$ along with $\gamma_{di}^{(0)} := \alpha_i^{(0)} + N/k$ for all $i = 1, \cdots, k$, $d = 1, \cdots, M$. Then repeat the following until convergence.

1. In the E-step, for $d = 1, \cdots, M$,
    1. For $n = 1$ to N and $i = 1$ to k,
        1. $\phi_{dni}^{(t+1)} = \exp\Big(\Psi(\lambda_{ij}^{(t)}) - \Psi\big(\sum_{j=1}^{V} \lambda_{ij}^{(t)} \big) + \Psi(\gamma_{di}^{(t)} - \Psi\big(\sum_{i=1}^{k} \gamma_{di}^{(t)} \big) \Big)$.
        2. For $j = 1, \cdots, V$,
            1. $\lambda_{ij}^{(t+1)} = \eta^{(t)} + \sum_{d=1}^{M} \sum_{n=1}^{N_d} \phi_{dni}^{(t+1)} w_{dn}^{j}$.
    2. Normalize $\phi_{dn}^{(t+1)}$ to sum to 1.
    3. $\gamma_{d}^{(t+1)} = \alpha + \sum_{n=1}^{N_d} \phi_{dni}^{(t+1)}$.
2. In the M-step,
    1. Update $\alpha^{(t+1)}$ with linear-time Newton-Raphson.
    2. Update $\vec{\eta}^{(t+1)}$ with linear-time Newton-Raphson.

Python code for the algorithm is in the last part of this notebook (GitHub).

References

- Griffiths, Steyvers. 2004. **Finding scientific topics**. Proceedings of the National Academy of Sciences of the United States of America. 101: 5228-5235.
- Blei, Ng, Jordan. 2003. **Latent Dirichlet Allocation**. Journal of Machine Learning Research. 3 (4–5): 993–1022.
- Pritchard, Stephens, Donnelly. 2000. **Inference of Population Structure Using Multilocus Genotype Data**. Genetics. 155: 945–959.