"Infinite LDA" – Implementing the HDP with minimum code complexity

Technical note TN2011/1

Gregor Heinrich

arbylon.net
Darmstadt, Germany
gregor@arbylon.net

Abstract. Shows how the hierarchical Dirichlet process (HDP) may be implemented in a simple way, following the idea that the HDP is an extension to its parametric counterpart, latent Dirichlet allocation (LDA).
Document version: draft, version 0.92, 20 Feb. 2011 (version 0.1: May 2008).

1 Introduction

The hierarchical Dirichlet process [TJB+06] has been frequently used to implement non-parametric admixture models, i.e., such models that estimate the number of mixture components, which parametric models like latent Dirichlet allocation [BNJ03] fall short of.

Unfortunately, the structure of the HDP seems to impose a level of complexity in existing implementations like that of Yee Whye Teh ("np-bayes", Matlab / C) or that of Chong Wang (C++, with split-merge operations) that makes them highly demanding to understand for non-experts without an extensive amount of time for analysing them, despite their stability and excellent quality. One may argue that this time invested will facilitate understanding of the underlying principles, and it indeed does, but to "flatten the learning curve", a simpler basis for experimentation may be a desirable alternative.

This is the reason for "yet another HDP implementation", and in particular, the goal may be to come (subjectively) as close as possible to an understanding of the HDP as an extension of the finite LDA model, adding an estimator for the number of topics and the associated parameters. Because of this goal, the implementation considered here is simply called "infinite LDA".

This note is targeted at students of LDA that want to walk some extra steps to understand what's beyond the finite model. From my experience, there seems to be such a high interest in making LDA independent of choosing the number of topics K that it isn't a stark exaggeration to say that a large portion of the people studying LDA have at least thought about studying the HDP as its natural extension. This note is therefore hopefully filling a gap of "simpler" texts about the subject. It will give some simple introduction to the HDP in Section 2 and will explain the open-source implementation from a technical and implementational background in Section 3.

2 Some background on the HDP

We can view the latent Dirichlet allocation (LDA) model as a Bayesian version of the admixture model. An admixture is a mixture of mixtures, and in LDA, documents are modeled as mixtures of topics, and topics as mixtures of terms (unique words). A more detailed explanation of LDA is found in the original literature [BNJ03] and [GrSt04], or in tutorials like [Hein09b]. The intuition behind this type of mixing is illustrative for text: A document is composed of some thematic units (topics), which are obscured by word-choice noise. By being able to match documents via topics, documents become similar that have synonymous but literally different words, which would be missed for literal matching. Bayesian analysis of this in LDA is supposed to find the topic distributions of the documents and the term-distributions of the topics given the words observed in documents.

The question that arises in models like LDA is how many topics a given set of texts has, or equivalently how many its mixtures are composed of, which is a dimension that should be estimated from the data. This is where the Dirichlet process enters the scene, and in the following I try to explain the Dirichlet process based on an understanding of the Dirichlet distribution and using some derivation steps from [Teh10]. However, in this document, a minimum of concepts will be introduced necessary to understand the "mechanism" behind the HDP. For more technically thorough explanations, please refer to [TJB+06,TeJo09,Neal98,Teh10]. In fact, these articles are explicitly recommended after understanding the concepts described here.

Dirichlet process. The Dirichlet process (DP) is a stochastic process that generates probability distributions, in a way generalising the Dirichlet distribution (DD) that generates multinomial parameter vectors and thus distributions over discrete categories.

Let's have a look at the DD, which is described in more detail in [Hein09b]. The DD $\mathrm{Dir}(\vec{\alpha})$ has a positive parameter vector $\vec{\alpha}$, which may actually be considered a discrete distribution G_0 over a set of categories $k \in [1, K]$ scaled by α:

$$\mathrm{Dir}(\vec{\alpha}) \equiv \mathrm{Dir}(\alpha G_0)\,, \quad G_0 = \textstyle\sum_{k=1}^{K} \pi_{0k}\delta(x-k)\,, \quad \textstyle\sum_k \pi_{0k} = 1 \tag{1}$$

where $\delta(x)$ is the Dirac function that has a unit integral and is non-zero only at $x = 0$. We call G_0 a base distribution of the DD. In effect, G_0 is a distribution that has K spikes with height π_{0k} at locations 1 to K, and the DD produces from this samples that are distributions of the same form:

$$G \sim \mathrm{Dir}(\alpha G_0) \quad \Rightarrow \quad G = \textstyle\sum_{k=1}^{K} \pi_k \delta(x-k)\,, \quad \textstyle\sum_k \pi_k = 1\,, \tag{2}$$

that is, samples G are discrete distributions with parameters $\vec{\pi}_k$. In fact, the samples G have an expectation of G_0, from which they deviate according to the precision (inverse variance) parameter α.

Informally speaking, the DP extends the DD by generalising the base distribution from the discrete G_0 to "almost any" distribution H_0 defined on some support set $S(H_0)$. For instance, the Gaussian has a support set of $S(\mathcal{N}(\cdot)) = \mathbb{R}$. Interestingly, samples from the DP then are still discrete but instead of the category labels k of the DD (or their

locations on the x-axis, $\delta(x-k)$), the point masses of the DP samples are somewhere on the support set $S(H_0)$, and (2) generalises to:

$$H \sim \mathrm{DP}(\alpha H_0) \quad \Rightarrow \quad H = \textstyle\sum_{k=1}^{\infty} \pi_k \delta(x-\theta_k)\,, \quad \sum_k \pi_k = 1\,, \quad \theta_k \in S(H_0)\,, \tag{3}$$

that is, samples H are discrete distributions over an infinite number of points θ_k in the support space of the base distribution that have weights π_k. The defining property of the DP is that its samples have weights π_k and locations θ_k distributed in such a way that when partitioning $S(H)$ into finitely many arbitrary disjoint subsets $S_1, \ldots, S_J$, $J < \infty$, the sums of the weights π_k in each of these J subsets are distributed according to a Dirichlet distribution that is parametrised by α and a discrete base distribution (like G_0) whose weights are equal to the integrals of the base distribution H_0 over the subsets S_n. Denoting the subset weight by $H(S_n)$, i.e., the sum of the π_k whose θ_k lie in S_n, and the subset integral over the base distribution by $H_0(S_n)$, we may express the defining property of the DP as:

$$H(S_1), \ldots, H(S_J) \sim \mathrm{Dir}(\alpha H_0(S_1), \ldots, \alpha H_0(S_J))\,. \tag{4}$$

Taking the example of a Gaussian base distribution $\mathcal{N}(0,1)$ and three intervals $S_1 = (-\infty, -1]$, $S_2 = (-1, 1]$ and $S_3 = (1, \infty)$, we get:

$$H(S_1), H(S_2), H(S_3) \sim \mathrm{Dir}(\alpha\,\mathrm{erf}(-1), \alpha(\mathrm{erf}(1) - \mathrm{erf}(-1)), \alpha(1 - \mathrm{erf}(1)))\,. \tag{5}$$

Analogous to the DD that generates imperfect copies of the base distribution G_0 (the expectation of its samples) with the precision α, the DP generates samples that may be considered imperfect, discretised copies of the base distribution H_0 with precision α. The higher the precision α, the better the sums of weights in the subsets, $H(S_j)$ match the probability mass in the respective regions of the base distribution, $H_0(S_j)$.

Posterior. When sampling from a DD given prior samples $\{x_i\}$, the conjugacy of Dirichlet and multinomial distributions leads to a posterior that takes into account the prior observations as pseudo-counts added to the scaled base distribution:

$$\mathrm{Dir}(\vec{\pi}\,|\,\alpha G_0)\;|\;\{x_i\} = \mathrm{Dir}(\vec{\pi}\,|\,\alpha G_0 + \vec{n})\,, \quad \vec{n} = \{n_k\}\,,\; n_k = \textstyle\sum_{i=1}^{n} \delta(k - x_i)\,, \tag{6}$$

which may be easily justified using Bayes rule and the definitions of the multinomial and DD:

$$p(\vec{\pi}\,|\,\{x_i\}, \alpha G_0) \propto \textstyle\prod_i \mathrm{Mult}(x_i\,|\,\vec{\pi})\mathrm{Dir}(\vec{\pi}\,|\,\alpha G_0) \propto \prod_k \pi_k^{\alpha G_0(k) + n_k - 1} \propto \mathrm{Dir}(\vec{\pi}\,|\,\alpha G_0 + \vec{n})\,. \tag{7}$$

In the DP, this generalises to adding pseudo-counts to the scaled base distribution αH_0:

$$H(S_1), \ldots, H(S_N)\;|\;\{\theta_i\} \sim \mathrm{Dir}(\alpha H_0(S_1) + n_1, \ldots, \alpha H_0(S_N) + n_N)\,,\; n_j = \textstyle\sum_{i=1}^{n} \delta(S_j - \theta_i)\,. \tag{8}$$

where $\delta(S_j - x)$ is 1 for every $x{=}\theta_k \in S_j$. Considering that the DP sample has infinitely many "spikes" at locations θ_k, we may use this to rewrite the posterior given observations:

$$H\;|\;\{\theta_i\} \sim \mathrm{DP}(\alpha H_0 + \textstyle\sum_{i=1}^{n} \delta(x - \theta_i))\,, \tag{9}$$

which may be separated into a modified precision $\alpha' = \alpha + n$ and base distribution $H_0' = \alpha H_0/(\alpha+n) + 1/(\alpha+n)\cdot\sum_{i=1}^{n}\delta(x-\theta_i)$, i.e., a mixture distribution with components H_0 and $1/n\cdot\sum_i \delta(x-\theta_i)$, the empirical distribution of the observations.

Predictive distribution. When considering the predictive distribution of the DP, i.e., the probability of a new data point θ_i drawn from some $H \sim \mathrm{DP}(\alpha H)$ given known observations $\{\theta_i\}_{i=1}^n$, the DP reveals one of its most important properties: it's clustering behaviour.

Again starting with DD, consider to draw a distribution $G \sim \mathrm{Dir}(\alpha G_0)$, and to draw n data points $\{x_i\}_{i=1}^n$ from G. Knowing both αG_0 and $\{x_i\}$, the predictive distribution of a new value x becomes:

$$x \sim G \mid \{x_i\}, \alpha G_0 \sim \mathrm{Dir}(\vec{\pi} \mid \alpha G_0 + \vec{n}) \quad \Rightarrow \quad p(x{=}k \mid \{x_i\}, \alpha G_0) = \frac{n_k + \alpha G_0(k)}{n+\alpha}\,, \tag{10}$$

which again is due to the conjugacy between the DD and the multinomial distribution, integrating out G:

$$\begin{aligned} p(x{=}k \mid \{x_i\}, \alpha G_0) &= \int \mathrm{Mult}(x{=}k \mid \vec{\pi})\,\mathrm{Dir}(\vec{\pi} \mid \alpha G_0 + \vec{n})\; dG\,, \\ &= \frac{\mathrm{B}(\alpha G_0 + \vec{n} + \delta(x-k))}{\mathrm{B}(\alpha G_0 + \vec{n})} = \frac{\alpha G_0(k) + n_k}{\alpha + n}\,. \\ x \mid \{x_i\}, \alpha G_0 &\sim \frac{\alpha G_0 + \sum_{k=1}^{K} n_k \delta(x-k)}{\alpha+n}\,. \end{aligned} \tag{11}$$

This illustrates the well-known "rich-get-richer" behaviour of the DD, which amplifies dimensions k that have observations associated with them.

Generalising this to the DP, we consider a draw $H \sim \mathrm{DP}(\alpha H_0)$ as defined above and draw $\{\theta_i\}_{i=1}^n \sim H$. As H is discrete and therefore conjugate to the DD that governs the partition weights of H, we may as well integrate it out. We want the probability of choosing a subset S_j out of $H \mid \{\theta_i\}$, which we know from (6) follows a DD. Using (9) and (4), we get the subset probability masses as expectations over the posterior DP sample:

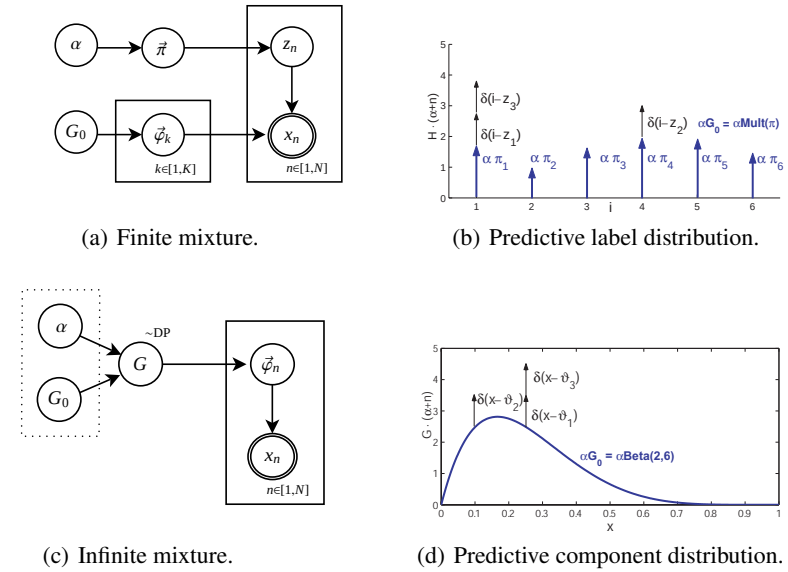
$$p(\theta \in S_j \mid \{\theta_i\}, \alpha H_0) = \Big\langle H(S_j) \mid \{\theta_i\}, \alpha H_0 \Big\rangle = \frac{\alpha H_0(S_j)}{n+\alpha} + \frac{\sum_i \delta(S_j - \theta_i)}{n+\alpha}\,. \tag{12}$$

Integrating out H here means summing over $H(S_j)$ over all partitions, which yields:

$$\theta \mid \{\theta_i\}, \alpha H_0 \sim \frac{\alpha H_0 + \sum_{i=1}^{n}\delta(\theta-\theta_i)}{n+\alpha} = \frac{\alpha H_0 + \sum_{k=1}^{K^+} n_k \delta(\theta-\theta_k)}{n+\alpha} \tag{13}$$

where K^+ is now a value that may increase as new data are observed: Either a new sample $\theta \sim H$ has been already seen, $\theta = \theta_k = \theta_i, i < n$, or a new θ_k is observed, incrementing K^+. Note that (11) and (13) are structurally identical but differ in effect due to the different properties of G_0 and H_0.

Similar to the DD, the DP enforces behaviour of the samples that tend to repeat existing observations, only that the arbitrary base distribution H_0 now replaces the discrete distribution G_0 (that adds weight to *existing* θ_k) and adds weight to *unknown* components out of a set of the infinitely many point masses in H. This modifies the "rich-get-richer" behaviour in the sense that the precision parameter α does not "pull" values



(a) Finite mixture. (b) Predictive label distribution.

(c) Infinite mixture. (d) Predictive component distribution.

Fig. 1. Mixture models: finite with Dirichlet prior in component weights, infinite with DP prior on the components (weights and locations), Beta(2, 6) as base distribution.

towards the discrete weights of the base distribution but rather to offer additional locations in the parameter space $S(H_0)$ that may fit observations better.

Dirichlet process mixtures. The clustering behaviour of the DP is exploited in the class of DP mixture (DPM) models that draw components $\theta_i \sim H$ from a DP sample $H \sim \mathrm{DP}(\alpha H_0)$, where H_0 is the component base distribution. According to (13), samples θ_i are either known components $\theta_i = \theta_k, k \leq K^+$ (or, equivalently, $\theta_i = \theta_j, j < n$) or new components drawn from $\theta_i \sim H_0$, and observations are drawn thus from known or unknown components $x_i \sim \theta_i$. The advantage of this scheme is that the number of components K^+ is not required as an input parameter of the model but grows with the data, leading to a fully *non-parametric* mixture model.

In Fig. 1, the principle of the DPM model is illustrated (bottom), comparing it with a finite mixture (top). Starting with the finite case, we have a model that uses Dirichlet-distributed mixture proportions, $\vec{\pi} \sim \mathrm{Dir}(\alpha)$ and a base distribution G_0. Note that the effect of the base distribution is not shown here; it can be any distribution parametrised by $\vec{\varphi}_k$ which themselves are drawn from their base distribution, $\vec{\varphi}_k \sim G_0$. The parameters α in the finite mixture influences the weighting between the K components, as is seen in (b).

The infinite case shown in (c) uses a DP sample as a prior on the components $\vec{\varphi}_n$, which takes the role of both the distribution over components $\vec{\pi} = \{\pi_k\}$ and over component locations $\vec{\varphi}_k$ in (a). As can be seen from the chart of the predictive component

distribution, there is a clustered probability weight at locations in the support set of G_0 where parameters have already been observed. The parameter α in the infinite mixture influences the weighting between the existing components and new components, as is seen in (d).

Sampling schemes. For both the DD and DP, illustrative sampling scenarios may be formulated based on the predictive distributions. For the DD, the Pólya urn scheme expresses the clustering behaviour: An urn is filled with of balls of unit size, K colours and in quantities that match the weights in the scaled discrete distribution αG_0 (for fractional $\alpha G_0(k)$, we allow ball segments, assuming that they are chosen with likelihoods that match their size). After drawing a ball (or segment) from the urn and observing its colour, it is returned to the urn together with an additional ball of the same colour, and the next sample can be taken. This is also called "sampling with over-replacement".

For the DP, the Pólya urn sampling scheme may be generalised: Instead of an initial set of balls (and segments) with K colours and counts according to αG_0, the urn is now filled with a liquid that has infinitely many different colour mixtures, distributed according to H_0. The absolute amount of liquid is equal to α. The first sample in this "generalised sampling with over-replacement" scheme now draws a volume of liquid and returns the liquid and an additional ball of the sampled liquid colour. The second sample either draws the ball or from the liquid and adds the sampled colour and so on.

Alternatively, in the DP and DPM literature, the metaphor of a "Chinese restaurant process" (CRP) has been used to illustrate the sampling behaviour: A Chinese restaurant has infinitely many tables, and customers enter the room, deciding whether to share a table k with other customers with probability n_k or choosing a new table with probability α. Sharing a table k corresponds to drawing values from a component $\vec{\theta}_i = \vec{\theta}_j$ that has been seen before, and sitting at a new table to a sample from the base distribution. The Chinese restaurant process is typically described by the label likelihoods of predictive samples, marginalising component locations θ_k from (13):

$$p(x{=}k|\{x_i\},\alpha) = \begin{cases} \dfrac{n_k}{n+\alpha} & n_k > 0 \\ \dfrac{\alpha}{n+\alpha} & \text{new table/colour/component.} \end{cases} \tag{14}$$

Hierarchical Dirichlet process. Although modelling mixtures with DP priors turns out simple, when considering admixture as in the LDA model, things become more complicated because in effect multiple parameters $\vec{\varphi}_k$ are supposed to be shared among documents (which are mixtures of $\vec{\varphi}_k$ with proportions defined by $\vec{\vartheta}_m$; please refer to [Hein09b] for explanations on LDA and the notation used). A naïve approach would be to sample weights $\vec{\vartheta}_m \sim G \sim \mathrm{DP}(\alpha G_0)$ and then from these weights the indices $z_{m,n}{=}k$ for the $\vec{\varphi}_k$ in the document. This works but does not implement admixture, as it clusters $\vec{\vartheta}_m$ to either new parameters or reusing existing ones, thus rather creating a hard clustering of documents, as has been shown in [YYT05].

Another approach that may be considered is to sample $\vec{\varphi}_k$ for every word (m,n), thus creating new components or topics as needed, i.e., sampling $\vec{\varphi}_{m,n} \sim H_0$. The question is how to make the $\vec{\varphi}_{m,n}$ shared between documents. If they are created from a single base distribution (following LDA, this should be $\mathrm{Dir}(\beta)$), i.e., $\vec{\varphi}_{m,n} \sim G_m \sim \mathrm{DP}(\alpha\mathrm{Dir}(\beta))$,

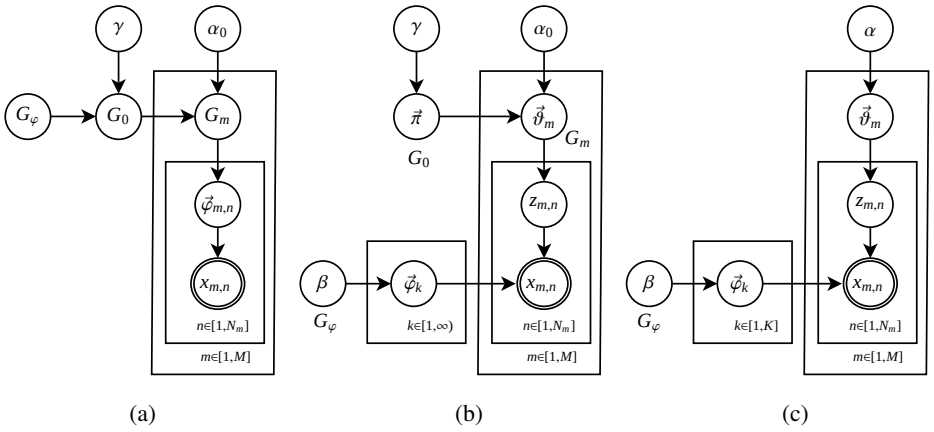


Fig. 2. Hierarical Dirichlet process: (a) Bayesian network of full model, (b) its representation with stick-breaking prior, and (c) LDA as the finite equivalent to the HDP.

each G_m will create its own set of spikes in the parameter space from the continuous Dirichlet base distribution. Therefore almost surely documents will have disjoint $\vec{\varphi}_{m,n}$, which prevents sharing of topics.

The solution is to share a *discrete* base distribution between documents whose spikes may be shared. The idea of the hierarchical DP is to create such a discrete base distribution for the document DPs by sampling from a DP itself. In such a hierarchy, the root DP gets the Dirichlet of the topics from LDA as a base distribution, $H \sim \mathrm{DP}(\gamma\mathrm{Dir}(\beta))$, and each document samples from H, so it "filters" its set of discrete masses, generating a distribution over a subset of them, $G_m \sim \mathrm{DP}(\alpha H)$. The masses $\vec{\varphi}_k$ shared between the different G_m have associated the same meaning as the $\vec{\varphi}_k \sim \mathrm{Dir}(\beta)$ in LDA.

The actual Bayesian network of the full model is given in Fig. 2(a), but according to the discrete nature of the sample G_0, a simplified representation may be given in Fig. 2(b) that uses an infinite-dimensional multinomial $\vec{\pi}$ to represent the root DP sample G_0 as the prior over document multinomials $\vec{\vartheta}_m$ that represent the document DP samples G_m and can be used to explicitly sample topic indicator variables $z_{m,n}$.

This representation has been often termed the stick-breaking prior (SBP [Teh10]) representation because a priori the point masses of $\vec{\pi}$ are distributed like the lengths of pieces broken off a stick infinitely many times with break points distributed $\mathrm{Beta}(1, \alpha)$ on the stick remaining at each step [Seth94]. This stick-breaking process is often referred to as GEM distribution (after Griffiths–Engen–McCloskey), $\vec{\pi} \sim \mathrm{GEM}(\alpha)$.

Interestingly, both $\vec{\vartheta}_m$ and $z_{m,n}$ are equivalent to those in LDA, only that $\vec{\vartheta}$ are infinite-dimensional and distributed according to $\vec{\vartheta}_m \sim \mathrm{Dir}(\alpha_0\vec{\pi})$. In this representation, the base distribution $G_\varphi = \mathrm{Dir}(\beta)$ can be explicitly indexed by the topic indicators $z_{m,n}$=k. For reference, the Bayesian network of LDA is given in Fig. 2(c).

In HDP, the DP clustering scheme still applies, but now an additional layer needs to be added to accommodate for the clustering between DPs, yielding the Chinese restaurant franchise (CRF) process [TJB+06] as a samling scheme: There are now M Chinese restaurants, and each one has infinitely many tables. On each table the restaurant serves one of infinitely many dishes that other restaurants may serve as well. When entering the restaurant, customers not only choose a table (topic; sample from G_φ appearing in G_0), but also whether they may have a dish popular among several restaurants (topic shared among documents; samples from G_φ appearing in several G_m).

3 Infinite LDA as simple HDP implementation

The published inference algorithms relevant to the HDP belong to Gibbs sampling [TJB+06] and variational methods (including collapsed ones) [KWT07]. Here, we focus on a Gibbs sampling method that empirically proved effective while staying close to the parametric case. Gibbs sampling for the HDP is based on one of the DP representations described above, SBP or CRF. Alternatively, a "direct assignment" technique that has been shown to be flexible enough to work with extensions of the basic HDP model [TJB+06,PNI+08] and will be adopted here. There are various other approaches to implement HDP inference, so in the seminal [TJB+04,TJB+06] several algorithms exhibit high implementation complexity and memory footprint, which led [GGJ06] to develop a simplified approach based on the CRF representation (i.e., integrating out the DPs), and in [BGC+09] further improvements and correction to this algorithm have been presented. Another approach is to use slice sampling from the SBP prior directly, similar to [TGG07] for the related Indian buffet process.

Directly sampling in the stick-breaking representation does not integrate out the Dirichlet process and modularises the inference equations into finite and infinite parts. This follows the original approach of [TJB+06] but uses a variant proposed by [Port10].

Consider an LDA model where the Dirichlet hyperparameter $\vec{\alpha}$ is split into a scalar precision α and a base distribution $\vec{\tau}$ with $\vec{\tau} \sim \mathrm{Dir}(\gamma/K)$. Taking this to the limit $K \to \infty$, the HDP is represented by a root distribution $\vec{\tau} \sim \mathrm{Dir}(\gamma/K)$, i.e., the weak limit approximation of the weights of a stick-breaking representation of the Dirichlet process [IsZa02], and its descendant distributions $\vec{\vartheta}_m \sim \mathrm{Dir}(\alpha\vec{\tau})$ [TJB+06,PBW08]. As in collapsed Gibbs sampling, the root level parameters $\vec{\tau}$ may in principle be integrated out. However, as the Dirichlet is not conjugate to itself, integration becomes difficult and things may be easier when the parameters are kept in the model directly.

Another advantage of the SBP-based approach is that the infinite extension of the coupled mixture (LDA) retains the structure of the finite case for the Gibbs update of z:

$$p(z_i{=}k|\cdot) \propto (n_{m,k}^{\neg i} + \alpha\tau_k) \cdot \frac{n_{k,t}^{\neg i} + \beta}{n_k^{\neg i} + V\beta} \tag{15}$$

with α as a precision parameter now. Note that the sampling space has $K{+}1$ dimensions, with the last point mass $\alpha\tau_{K+1}/V$, which represents all unseen components. Whenever this mass is sampled, a new topic is created.

The prior proportions $\vec{\tau}$ can be sampled by simulating how new components are created for $n_{m,k}$ draws from the DP with precision $\alpha\tau_k$ (dishes in the CRF), which is a

sequence of Bernoulli trials for each m and k:

$$p(m_{m,k,r}{=}1) = \frac{\alpha\tau_k}{\alpha\tau_k + r - 1} \; \forall r \in [1, n_{m,k}], m \in [1, M], k \in [1, K] \tag{16}$$

A posterior sample of the root distribution then is obtained via:

$$\vec{\tau} \sim \mathrm{Dir}(\{m_k\}_k, \gamma) \quad \text{with } m_k = \textstyle\sum_m \sum_r m_{m,k,r} \tag{17}$$

which updates the probability of the root distribution and consequently the means of the hyperparameters on the second HDP level. Note that $\vec{\tau}$ has dimension $K + 1$ because the mass belonging to γ in the Dirichlet sampler is associated with generating a component out of an infinite set of empty components. If a component has lost all its data points, it is merged with the unknown components in the mass associated with γ.

The simulation of new tables to find τ is due to [Port10] and simplifies the direct assignment scheme in [TJB+06] that showed that in general under CRP the number of tables u for n customers is $p(m_{m,k}|\alpha, n_{m,k}) \propto s(n_{m,k}, m_{m,k})(\alpha\tau_k)^m$ where $s(n, m)$ is an unsigned Stirling number of the first kind[1] [AbSt64].

For completeness, we note that sampling has also been shown effective with the stick-breaking distribution sampled from directly by slice sampling [TGG07] and there are different other published methods [IsJa03].

Hyperparameter sampling. Following the approach presented with the HDP [TJB+06], for the precision parameter of the DPs gamma priors may be assumed, introducing hyper-prior parameters a_γ, b_γ and a_α, b_α. There are two ways to be pursued for sampling them: [EsWe95] propose an auxiliary variable method that allows sampling using standard distributions, and [Rasm00,NGS+06] proposes an adaptive rejection sampler for a non-standard log-concave distribution [GiWi92].

Using the simpler first approach, auxiliary variables u, v are introduced for the DP hyperparameter γ of the root process that are governed by Bernoulli and beta distributions. The actual parameter is sampled then using the gamma distribution [EsWe95]:

$$u \sim \mathrm{Bern}\left(\frac{T}{T + \gamma}\right), \quad v \sim \mathrm{Beta}(\gamma + 1, T) \tag{18}$$

$$\gamma \sim \mathrm{Gam}(a_\gamma + K - 1 + u, b_\gamma - \log v) \tag{19}$$

where $T = \sum_k m_k$ is the augmented variable of the τ sampler described in the previous section.[2] Similar to this but with a small modification from [TJB+06], the second-level hyperparameter can be sampled using u_m, v_m as auxiliary parameters and the "dimen-

[1] Unsigned Stirling numbers of the first kind describe the number of permutation cycles of size m in a set of n elements. An iterative rule is $s(n + 1, m + 1) = s(n, m) + ns(n, m + 1)$ with $s(n, n) = 1$ and $s(n, m) = 0 \forall u > n$.

[2] It is the total number of tables in the CRF that serves as "count" of the items to be drawn from the root CRP

Algorithm `LdaGibbsHdp(`$\{\vec{w}\}$`);`
Input: word vectors $\{\vec{w}\}$
Optional input: initial values for hyperparameters α, β, γ, topic number K_0 and parameters for hyperparameter priors a_α, b_α etc. to speed up convergence
Global data: count statistics $\{n_{m,k}\}$, $\{n_{k,t}\}$ and their sums $\{n_m\}$, $\{n_k\}$, Dirichlet parameter $\vec{\tau}$, memory for full conditional array $p(z_i|\cdot)$, lists of used and unused topic indices $U_{1,0}$.
Output: topic associations $\{\vec{z}\}$, topic dimension K, multinomial parameters $\underline{\Theta}$ and $\underline{\Phi}$, hyperparameter estimates $\alpha\vec{\tau}, \beta, \gamma$

`// initialisation`
$K = K_0$ (default $K = 1$);
zero all count variables, $n_{m,k}, n_m, n_{k,t}, n_k$;
$U_1 = [1, K]$, $U_0 = [K + 1, K_{\max}]$;
for all documents $m \in [1, M]$ **do**
  for all words $n \in [1, N_m]$ in document m **do**
    sample topic index $z_{m,n}{=}k \sim \text{Mult}(1/K)$;
    increment counts and sum: $n_{m,k}$ += 1, $n_{k,t}$ += 1, n_k += 1;

sample $\vec{\tau}$ using (17);
`// Gibbs sampling over burn-in period and sampling period`
while not finished **do**
  for all documents $m \in [1, M]$ **do**
    for all words $n \in [1, N_m]$ in document m **do**
      `// for the current assignment of` k `to a term` t `for word` $w_{m,n}$`:`
      decrement counts and sums: $n_{m,k}$ −= 1; $n_{k,t}$ −= 1; n_k −= 1;
      `// multinomial sampling using (15) with range` $[1, K + 1]$`:`
      sample topic index $\tilde{k} \sim p(z_i|\vec{z}_{\neg i}, \vec{w}, \alpha\vec{\tau}, \beta, \gamma, K)$;
      if $\tilde{k} \in [1, K]$ **then**
        `// for the new assignment of` $z_{m,n}$ `to the term` t `for word` $w_{m,n}$`:`
        $k^* = U_1(\tilde{k})$; increment counts and sums: n_{m,k^*} += 1; $n_{k^*,t}$ += 1; n_{k^*} += 1;
      else
        `// create new topic from term` t `in document` m `as first assignment:`
        $k^* = \text{pop}(U_0)$, $\text{push}(U_1, k^*)$, K += 1;
        $n_{m,k^*} = 1, n_{k^*} = 1, n_{k^*,t} = 1$;
        sample $\vec{\tau}$ using (17);
      $z_{m,n} = k^*$;
  `// check convergence and read out parameters`
  if converged and L sampling iterations since last read out **then**
    `// the different parameters read outs are averaged.`
    read out parameter sets $\underline{\Phi}$ and $\underline{\Theta}$;
  else if not converged **then**
    for $k \in [1, K]$ **do**
      if $n_k = 0$ **then**
        `// prune empty topic` k
        $\text{remove}(U_1, k)$, $\text{push}(U_0, k)$, K −= 1, empty counts $n_{m,k}, n_{k,t}, n_k$;
    sample the number of tables used using and from this $\vec{\tau}$ using (17) or the "Antoniak-type" sampler;
    sample α using (21);
    sample γ using (19);
    estimate β;

Fig. 3. Gibbs sampling algorithm for the hierarchical Dirichlet process.

sion" T^3:

$$u_m \sim \text{Bern}\left(\frac{n_m}{n_m + \alpha}\right), \quad v_m \sim \text{Beta}(\alpha + 1, n_m) \tag{20}$$

$$\alpha \sim \text{Gam}(a_\alpha + T - \textstyle\sum_m u_m, b_\alpha - \textstyle\sum_m \log v_m) \tag{21}$$

Finally, the Dirichlet hyperparameter β can be estimated using one of the approaches in [Mink00,Hein09a], by using adaptive rejection sampling [GiWi92] or slice sampling [Wall08]. Another approach using auxiliary variables similar to the above approach is equally possible and has been derived in a similar setting to Dirichlet hyperparameters in [NAS+09].

Sampling algorithm. The complete sampling algorithm performs the three different sampler types for the parametric model, the non-parametric prior and the hyperparameters in turn, as shown in Fig. 3. A simple Java implementation can be reviewed in the Appendix.

Note that sampling $\vec{\tau}$ is triggered by the occurrence of new components in the inner loop, but pruning of empty components may be done still on a corpus-wise schedule for performance reasons. In terms of keeping track of data for a changing number of topics, a simple method is to maintain lists or stacks of active and inactive topic indices that new components are popped from and empty ones pushed onto for reuse. In a straightforward implementation, the overhead of dynamic memory handling should be reduced, using a reasonably high upper limit of (reusable) component labels $K_{\max}$ and working with static array structures of that size, possibly re-instantiating the potentially large rows in $n_{k,t}$ on demand.

4 Conclusion

In this note, we have introduced the properties of the Dirichlet process by comparing it to the Dirichlet distribution and derived the hierarchical Dirichlet process. Based on this, we have presented a simple implementation of the HDP that makes use of Teh's "direct assignment scheme" and may be seen as an infinite extension of the finite LDA model.

In the future, some investigation may go into the implementational aspects of parallelisms and serial fast sampling, analogous to the finite case of fast sampling LDA [PNI+08] or [YMM09].

To continue studies on the DP and HDP, the articles [TJB+06,TeJo09,Neal98,Teh10] are recommended and will introduce additional concepts that have been left out here for simplicity.

References

AbSt64. M. Abramowitz & I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, 1964. ISBN 0-486-61272-4. URL `http://www.math.sfu.ca/~cbm/aands/`.

[3] Note how the table count T changes its meaning from an actual count in the root DP to a number of components in the document DP.

BGC+09. P. Blunsom, S. Goldwater, T. Cohn & M. Johnson. A note on the implementation of hierarchical Dirichlet processes. In *Proceedings of the ACL-IJCNLP Conference, Singapore*, pp. 337–340. 2009.

BNJ03. D. Blei, A. Ng & M. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, Jan. 2003. URL `http://www.cs.berkeley.edu/~blei/papers/blei03a.ps.gz`.

EsWe95. M. D. Escobar & M. West. Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, 90:577–588, 1995.

GGJ06. S. Goldwater, T. Grifths & M. Johnson. Contexttual dependencies in unsupervised word segmentation. In *Proc. of the 44th Annual Meeting of the ACL and 21st International Conference on Computational Linguistics (COLING/ACL-2006), Sydney*. 2006.

GiWi92. W. R. Gilks & P. Wild. Adaptive rejection sampling for gibbs sampling. *Applied Statistics*, 41(2):337–348, 1992.

GrSt04. T. L. Griffiths & M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(Suppl. 1):5228–5235, April 2004.

Hein09a. G. Heinrich. A generic approach to topic models. In *Proc. European Conf. on Mach. Learn. / Principles and Pract. of Know. Discov. in Databases (ECML/PKDD)*. 2009.

Hein09b. ———. Parameter estimation for text analysis. Technical Report No. 09RP008-FIGD, Fraunhofer Institute for Computer Graphics (IGD), http://www.arbylon.net/publications/text-est2.pdf, Sept 2009. Version 2.9.

IsJa03. H. Ishwaran & L. F. James. Generalized weighted Chinese restaurant processes for species sampling. *Statistica Sinica*, 13:1211–1235, 2003.

IsZa02. H. Ishwaran & M. Zarepour. Exact and approximate sum-representations for the Dirichlet process. *Can. J. Statist.*, 30:269–283, 2002.

KWT07. K. Kurihara, M. Welling & Y. W. Teh. Collapsed variational Dirichlet process mixture models. In *Proceedings of the International Joint Conference on Artificial Intelligence*, vol. 20. 2007.

Mink00. T. Minka. Estimating a Dirichlet distribution. Web, 2000. URL `http://www.stat.cmu.edu/~minka/papers/dirichlet/minka-dirichlet.pdf`.

NAS+09. D. Newman, A. Asuncion, P. Smyth & M. Welling. Distributed algorithms for topic models. *JMLR*, 10:1801–1828, August 2009.

Neal98. R. M. Neal. Markov chain sampling methods for Dirichlet process mixture models. Tech. rep., Department of Statistics, University of Toronto, 1998.

NGS+06. D. Navarro, T. Griffiths, M. Steyvers & D. Lee. Modeling individual differences using Dirichlet processes. *Journal of Mathematical Psychology*, 50:101–122, 2006.

PBW08. I. Porteous, E. Bart & M. Welling. Multi-HDP: A non-parametric Bayesian model for tensor factorization. In *Proc. AAAI*. 2008.

PNI+08. I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth & M. Welling. Fast collapsed Gibbs sampling for latent Dirichlet allocation. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 569–577. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-193-4.

Port10. I. Porteous. *Mixture Block Methods for Non Parametric Bayesian Models with Applications*. Ph.D. thesis, University of California at Irvine, 2010.

Rasm00. C. E. Rasmussen. The infinite gaussian mixture model. *Advances in Neural Information Processing Systems 12*, pp. 554–560, 2000.

Seth94. J. Sethuraman. A constructive definition of Dirichlet priors. *Statistica Sinica*, 4:639–650, 1994.

Teh10. Y. Teh. *Encyclopedia of Machine Learning*, chap. Dirichlet Processes. Springer, 2010.

TeJo09. Y. W. Teh & M. I. Jordan. Hierarchical Bayesian nonparametric models with applications. In N. Hjort, C. Holmes, P. Müller & S. Walker (eds.), *To appear in Bayesian Nonparametrics: Principles and Practice*. Cambridge University Press, 2009.

TGG07. Y. W. Teh, D. Görür & Z. Ghahramani. Stick-breaking construction for the indian buffet process. In *Proc. AISTATS*. 2007.

TJB+04. Y. Teh, M. Jordan, M. Beal & D. Blei. Hierarchical Dirichlet processes. Tech. Rep. 653, Department of Statistics, University of California at Berkeley, 2004.

TJB+06. ———. Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101:1566–1581, 2006.

Wall08. H. M. Wallach. *Structured Topic Models for Language*. Ph.D. thesis, University of Cambridge, 2008.

YMM09. L. Yao, D. Mimno & A. McCallum. Efficient methods for topic model inference on streaming document collections. In *Proc. KDD09*. 2009.

YYT05. K. Yu, S. Yu & V. Tresp. Dirichlet enhanced latent semantic analysis. In *AI & Statistics (AISTATS-05)*. 2005.

A Source code of IldaGibbs.java

The dependencies in packages `org.knowceans.*` can be found in the `knowceans-tools` project on SourceForge. Check out from source control for latest versions. The NIPS corpus is included in svmlight format.

Listing 1.1. IldaGibbs.java

```
package org.knowceans.topics.simple;

import static java.lang.Math.log;
import static org.knowceans.util.Samplers.randBernoulli;
import static org.knowceans.util.Samplers.randBeta;
import static org.knowceans.util.Samplers.randGamma;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import org.knowceans.corpus.NumCorpus;
import org.knowceans.util.ArrayUtils;
import org.knowceans.util.CokusRandom;
import org.knowceans.util.IndexQuickSort;
import org.knowceans.util.Samplers;
import org.knowceans.util.StopWatch;
import org.knowceans.util.Vectors;

/**
 * LDA Gibbs sampler with nonparametric prior (HDP):
 * <p>
 * (m,k | alpha * tau | gamma), k->inf, (k,t | beta)
 * <p>
 * using Teh et al. (2006) approach for the direct assignment sampler, with
 * modular LDA parametric sampler first published by Griffiths (2002) and
 * explained in Heinrich (2005). For the original LDA paper, see Blei et al.
 * (2002).
 * <p>
 * The general idea is to retain as much as possible of the standard LDA Gibbs
 * sampler, which is possible by alternatingly sampling the finite case with K +
 * 1 topics and resampling the topic weights taking into account the current
 * assignments of data items to topics and pruning or expanding the topic set
 * accordingly.
 * <p>
 * I tried to find the (subjectively) best tradeoff between simplicity and the
 * JASA paper (Teh et al. 2006). Therefore I have only used the direct
 * assignment method.
 * <p>
 * The implementation uses lists instead of primitive arrays, but for
 * performance reasons, this may be changed to have a bound Kmax to allocate
 * fixed-size arrays, similar to a truncated DP.
 * <p>
 * Caveats: (1) Performance is not a core criterion, and OOP encapsulation is
 * ignored for compactness' sake. (2) Code still uses the likelihood function of
 * LDA, and without the hyperparameter terms.
```

```
 * <p>
 * LICENSE: GPL3, see: http://www.gnu.org/licenses/gpl-3.0.html
 * <p>
 * References:
 * <p>
 * D.M. Blei, A. Ng, M.I. Jordan. Latent Dirichlet Allocation. NIPS, 2002
 * <p>
 * T. Griffiths. Gibbs sampling in the generative model of Latent Dirichlet
 * Allocation. TR, 2002, www-psych.stanford.edu/~gruffydd/cogsci02/lda.ps
 * <p>
 * G. Heinrich. Parameter estimation for text analysis. TR, 2009,
 * www.arbylon.net/publications/textest2.pdf
 * <p>
 * G. Heinrich. "Infinite LDA" -- implementing the HDP with minimum code
 * complexity. TN2011/1, www.arbylon.net/publications/ilda.pdf
 * <p>
 * Y.W. Teh, M.I. Jordan, M.J. Beal, D.M. Blei. Hierarchical Dirichlet
 * Processes. JASA, 101:1566-1581, 2006
 *
 * @author (c) 2008-2011 Gregor Heinrich, gregor :: arbylon : net
 * @version 0.9
 */
public class IldaGibbs {

    /**
     * @param args
     */
    public static void main(String[] args) {

        int niter = 1000, niterq = 10;
        // load the NIPS0-12 corpus
        String filebase = "nips/nips";
        NumCorpus corpus = new NumCorpus(filebase + ".corpus");
        Random rand = new CokusRandom(56567651);
        // corpus.reduce(100, rand);
        corpus.split(10, 2, rand);
        NumCorpus train = (NumCorpus) corpus.getTrainCorpus();
        NumCorpus test = (NumCorpus) corpus.getTestCorpus();
        int[][] w = train.getDocWords(rand);
        int[][] wq = test.getDocWords(rand);
        int K0 = 0;
        int V = corpus.getNumTerms();
        double alpha = 1.;
        // beta = 1 --> K = 12,.5-->16, .1-->26@200, 75@500, 115@645 (beta should be larger),
        //
        double beta = .1;
        double gamma = 1.5;

        // run sampler
        IldaGibbs gs = new IldaGibbs(w, wq, K0, V, alpha, beta, gamma, rand);
        gs.init();
        System.out.println("initialised");
        System.out.println(gs);

        // initial test
        gs.initq();
        gs.runq(niterq);
        System.out.println("perplexity = " + gs.ppx());

        StopWatch.start();
        System.out.println("starting Gibbs sampler with " + niter
                + " iterations");

        gs.run(niter);
        System.out.println(StopWatch.format(StopWatch.stop()));

        // test
        gs.initq();
        gs.runq(niterq);
        System.out.println("perplexity = " + gs.ppx());
        System.out.println(gs);

        gs.packTopics();

        System.out.println("finished");
        System.out.println(gs);

    } // main

    private int[][] w;
    private int[][] wq;
    /**
     * inactive components: index = index in count arrays, element = value in z.
     * Filled on component removal.
     */
    private List<Integer> kgaps;
    /**
     * active components: value = value in z and count arrays, which should
     * always match. This array is never removed elements from but inactive
     * elements are set to -1. This allows to reuse components (gaps) and to
```

```
     * keep the indices of z and counts identical.
     */
    private List<Integer> kactive;
    private List<Integer>[] nmk;
    private int[][] nmkq;
    private List<int[]> nkt;
    private List<Integer> nk;
    private double[][] phi;
    private int[][] z;
    private int[][] zq;

    private double[] pp;
    /**
     * step to increase the sampling array
     */
    public final int ppstep = 10;
    /**
     * precision of the 2nd-level DP
     */
    private double alpha;
    /**
     * mean of the 2nd-level DP = sample from 1st-level DP
     */
    private ArrayList<Double> tau;
    /**
     * parameter of root base measure (= component Dirichlet)
     */
    private double beta;
    /**
     * precision of root DP
     */
    private double gamma;

    // hyperparameters for DP and Dirichlet samplers
    // Teh+06: Docs: (1, 1), M1-3: (0.1, 0.1); HMM: (1, 1)
    double aalpha = 5;
    double balpha = 0.1;
    double abeta = 0.1;
    double bbeta = 0.1;
    // Teh+06: Docs: (1, 0.1), M1-3: (5, 0.1), HMM: (1, 1)
    double agamma = 5;
    double bgamma = 0.1;
    // number of samples for parameter samplers
    int R = 10;

    /**
     * total number of tables
     */
    private double T;

    private Random rand;
    private int iter;
    /**
     * current number of non-empty components
     */
    private int K;
    private int M;
    private int Mq;
    private int Wq;
    private int V;
    private boolean inited = false;
    private boolean fixedK = false;
    private boolean fixedHyper = false;

    /**
     * parametrise gibbs sampler
     *
     * @param w
     *            word tokens
     * @param wq
     *            word tokens (testing)
     * @param K
     *            initial number of topics: may be 0 if gamma > 0.
     * @param V
     *            number of terms
     * @param alpha
     *            node A precision (document DP)
     * @param gamma
     *            node A precision (root DP), 0 for fixed K: plain LDA.
     * @param beta
     *            node B hyperparam
     * @param rand
     *            random number generator
     */
    public IldaGibbs(int[][] w, int[][] wq, int K, int V, double alpha,
            double beta, double gamma, Random rand) {
        // assign
        this.w = w;
        this.wq = wq;
        // start with 0 or more topics
```

```
        this.K = K;
        this.alpha = alpha;
        this.beta = beta;
        this.gamma = gamma;
        if (gamma == 0) {
            this.fixedK = true;
        }
        this.M = w.length;
        this.Mq = wq.length;
        this.V = V;
        this.rand = rand;
    }

    /**
     * initialise Markov chain
     */
    @SuppressWarnings("unchecked")
    public void init() {
        // allocate
        nmk = new ArrayList[M];
        nkt = new ArrayList<int[]>();
        nk = new ArrayList<Integer>();
        z = new int[M][];
        for (int m = 0; m < M; m++) {
            nmk[m] = new ArrayList<Integer>();
            for (int k = 0; k < K; k++) {
                nmk[m].add(0);
            }
            z[m] = new int[w[m].length];
        }
        // indexing lists
        kactive = new ArrayList<Integer>();
        kgaps = new ArrayList<Integer>();
        // create mean weights
        tau = new ArrayList<Double>();
        for (int k = 0; k < K; k++) {
            kactive.add(k);
            nkt.add(new int[V]);
            nk.add(0);
            // set to value for fixed K
            tau.add(1. / K);
        }
        // tau has one dimension more
        tau.add(1. / K);
        pp = new double[K + ppstep];
        // initialise (run without decrements because z[*][*] = -1)
        run(1);
        if (!fixedK) {
            updateTau();
        }
        inited = true;
    }

    /**
     * initialise Markov chain for querying
     */
    public void initq() {
        // compute parameters
        int Kg = K + kgaps.size();
        phi = new double[Kg][V];
        for (int kk = 0; kk < K; kk++) {
            int k = kactive.get(kk);
            for (int t = 0; t < V; t++) {
                phi[k][t] = (nkt.get(k)[t] + beta) / (nk.get(k) + V * beta);
            }
        }
        // allocate
        nmkq = new int[Mq][Kg];
        zq = new int[Mq][];
        Wq = 0;
        // initialise
        for (int m = 0; m < Mq; m++) {
            zq[m] = new int[wq[m].length];
            for (int n = 0; n < wq[m].length; n++) {
                int k = rand.nextInt(K);
                zq[m][n] = k;
                nmkq[m][k]++;
                Wq++;
            }
        }
    }

    /**
     * run Gibbs sampler
     *
     * @param niter
     *            number of Gibbs iterations
     */
    public void run(int niter) {
```

```
        for (iter = 0; iter < niter; iter++) {
            System.out.println(iter);
            System.out.println(this);
            for (int m = 0; m < M; m++) {
                for (int n = 0; n < w[m].length; n++) {
                    // sampling z
                    int k, kold = -1;
                    int t = w[m][n];
                    if (inited) {
                        k = z[m][n];
                        // decrement
                        nmk[m].set(k, nmk[m].get(k) - 1);
                        nkt.get(k)[t]--;
                        nk.set(k, nk.get(k) - 1);
                        kold = k;
                    }
                    // compute weights
                    double psum = 0;
                    // (37)
                    for (int kk = 0; kk < K; kk++) {
                        k = kactive.get(kk);
                        pp[kk] = (nmk[m].get(k) + alpha * tau.get(k)) * //
                                (nkt.get(k)[t] + beta) / (nk.get(k) + V * beta);
                        psum += pp[kk];
                    }
                    // likelihood of new component
                    if (!fixedK) {
                        pp[K] = alpha * tau.get(K) / V;
                        psum += pp[K];
                    }
                    double u = rand.nextDouble();
                    u *= psum;
                    psum = 0;
                    int kk = 0;
                    for (; kk < K + 1; kk++) {
                        psum += pp[kk];
                        if (u <= psum) {
                            break;
                        }
                    }
                    // reassign and increment
                    if (kk < K) {
                        k = kactive.get(kk);
                        z[m][n] = k;
                        nmk[m].set(k, nmk[m].get(k) + 1);
                        nkt.get(k)[t]++;
                        nk.set(k, nk.get(k) + 1);
                    } else {
                        assert (!fixedK);
                        z[m][n] = spawnTopic(m, t);
                        updateTau();
                        System.out.println("K = " + K);
                    }
                    // empty topic?
                    if (inited && nk.get(kold) == 0) {
                        // remove the object not the index
                        kactive.remove((Integer) kold);
                        kgaps.add(kold);
                        assert (Vectors.sum(nkt.get(kold)) == 0
                                && nk.get(kold) == 0 && nmk[m].get(kold) == 0);
                        K--;
                        System.out.println("K = " + K);
                        updateTau();
                    }
                } // n
            } // m
            if (!fixedK) {
                updateTau();
            }
            if (iter > 10 && !fixedHyper) {
                updateHyper();
            }
        } // i
    }

    /**
     * query Gibbs sampler. This assumes the standard LDA model as we know the
     * dimensionality from the training set, therefore topics need to be pruned.
     *
     * @param niter
     *            number of Gibbs iterations
     */
    public void runq(int niter) {
        for (int qiter = 0; qiter < niter; qiter++) {
            for (int m = 0; m < nmkq.length; m++) {
                for (int n = 0; n < wq[m].length; n++) {
                    // decrement
                    int k = zq[m][n];
                    int t = wq[m][n];
                    nmkq[m][k]--;
```

```
                    // compute weights
                    double psum = 0;
                    for (int kk = 0; kk < K; kk++) {
                        pp[kk] = (nmkq[m][kk] + alpha) * phi[kk][t];
                        psum += pp[kk];
                    }
                    // sample
                    double u = rand.nextDouble() * psum;
                    psum = 0;
                    int kk = 0;
                    for (; kk < K; kk++) {
                        psum += pp[kk];
                        if (u <= psum) {
                            break;
                        }
                    }
                    // reassign and increment
                    zq[m][n] = kk;
                    nmkq[m][kk]++;
                } // n
            } // m
        } // i
    }

    /**
     * adds a topic to the list of active topics, either by reusing an existing
     * inactive index (gap) or increasing the count arrays. NB: Within this
     * method, the state is inconsistent.
     *
     * @param m
     *            current document
     * @param t
     *            current term
     * @return index of topic spawned
     */
    private int spawnTopic(int m, int t) {
        int k;
        if (kgaps.size() > 0) {
            // reuse gap
            k = kgaps.remove(kgaps.size() - 1);
            kactive.add(k);
            nmk[m].set(k, 1);
            nkt.get(k)[t] = 1;
            nk.set(k, 1);
        } else {
            // add element to count arrays
            k = K;
            for (int i = 0; i < M; i++) {
                nmk[i].add(0);
            }
            kactive.add(K);
            nmk[m].set(K, 1);
            nkt.add(new int[V]);
            nkt.get(K)[t] = 1;
            nk.add(1);
            tau.add(0.);
        }
        K++;
        if (pp.length <= K) {
            pp = new double[K + ppstep];
        }
        return k;
    }

    /**
     * reorders topics such that no gaps exist in the count arrays and topics
     * are ordered with their counts descending. Removes any gap dimensions.
     */
    public void packTopics() {
        // sort topics by size
        int[] knew2k = IndexQuickSort.sort(nk);
        IndexQuickSort.reverse(knew2k);
        // reorder and weed out empty count arrays
        IndexQuickSort.reorder(nk, knew2k);
        IndexQuickSort.reorder(nkt, knew2k);
        for (int i = 0; i < kgaps.size(); i++) {
            nk.remove(nk.size() - 1);
            nkt.remove(nkt.size() - 1);
        }
        for (int m = 0; m < M; m++) {
            IndexQuickSort.reorder(nmk[m], knew2k);
            for (int i = 0; i < kgaps.size(); i++) {
                nmk[m].remove(nmk[m].size() - 1);
            }
        }
        // any new topics will be appended
        kgaps.clear();
        int[] k2knew = IndexQuickSort.inverse(knew2k);
        // rewrite topic labels
        for (int i = 0; i < K; i++) {
```

```
            kactive.set(i, k2knew[kactive.get(i)]);
        }
        for (int m = 0; m < M; m++) {
            for (int n = 0; n < w[m].length; n++) {
                z[m][n] = k2knew[z[m][n]];
            }
        }
    }

    /**
     * prune topics and update tau, the root DP mixture weights.
     */
    private void updateTau() {
        // (40) sample mk
        double[] mk = new double[K + 1];
        // TODO: average multi-sample?
        for (int kk = 0; kk < K; kk++) {
            int k = kactive.get(kk);
            for (int m = 0; m < M; m++) {
                if (nmk[m].get(k) > 1) {
                    // number of tables a CRP(alpha tau) produces for nmk items
                    mk[kk] += Samplers.randAntoniak(alpha * tau.get(k), //
                        nmk[m].get(k));
                } else {
                    mk[kk] += nmk[m].get(k);
                }
            }
        }
        // number of tables
        T = Vectors.sum(mk);
        mk[K] = gamma;
        // (36) sample tau
        double[] tt = Samplers.randDir(mk);
        for (int kk = 0; kk < K; kk++) {
            int k = kactive.get(kk);
            tau.set(k, tt[kk]);
        }
        tau.set(K, tt[K]);
    }

    /**
     * update scalar DP hyperparameters alpha, gamma and Dirichlet
     * hyperparameter beta. Assumes that T is updated (by updateTau).
     */
    private void updateHyper() {
        for (int r = 0; r < R; r++) {
            // gamma: root level (Escobar+West95) with n = T
            // (14)
            double eta = randBeta(gamma + 1, T);
            double bloge = bgamma - log(eta);
            // (13')
            double pie = 1. / (1. - (T * bloge / (gamma + K - 1)));
            // (13)
            int u = randBernoulli(pie);
            gamma = randGamma(agamma + K - 1 + u, 1. / bloge);

            // alpha: document level (Teh+06)
            double qs = 0;
            double qw = 0;
            for (int m = 0; m < M; m++) {
                // (49) (corrected)
                qs += randBernoulli(w[m].length / (w[m].length + alpha));
                // (48)
                qw += log(randBeta(alpha + 1, w[m].length));
            }
            // (47)
            alpha = randGamma(aalpha + T - qs, 1. / (balpha - qw));
        }
        int[] ak = (int[]) ArrayUtils.asPrimitiveArray(nk);
        int[][] akt = new int[K][V];
        for (int k = 0; k < K; k++) {
            akt[k] = nkt.get(k);
        }
        // beta = DirichletEstimation.estimateAlphaMap(//
        // akt, ak, beta, abeta, bbeta);
    }

    /**
     * @return the perplexity of the last query sample.
     */
    public double ppx() {
        // TODO: this uses LDA's perplexity --> add hyperparameters and DP stuff
        double loglik = 0;
        // compute thetaq
        double[][] thetaq = new double[Mq][K];
        for (int m = 0; m < Mq; m++) {
            for (int k = 0; k < K; k++) {
                thetaq[m][k] = (nmkq[m][k] + alpha)
                        / (wq[m].length + K * alpha);
            }
```

```
        }
        // compute ppx
        for (int m = 0; m < Mq; m++) {
            for (int n = 0; n < wq[m].length; n++) {
                double sum = 0;
                for (int k = 0; k < K; k++) {
                    sum += thetaq[m][k] * phi[k][wq[m][n]];
                }
                loglik += Math.log(sum);
            }
        }
        return Math.exp(-loglik / Wq);
    }

    /**
     * assemble a string of overview information.
     */
    @Override
    public String toString() {
        return String.format("ILDA: M = %d, K = %d, V = %d, "
                + "alpha = %2.5f, beta = %2.5f, gamma = %2.5f", //
            M, K, V, alpha, beta, gamma);
    }
}
```