Bounding Sample Errors in Approximate Distributed Latent Dirichlet Allocation

Alexander Ihler[1] **and David Newman**[1,2]

1 Department of Computer Science, University of California, Irvine, Irvine CA 92697 USA
2 NICTA Victoria Research Laboratory, University of Melbourne, VIC 3010, Australia

Abstract

Latent Dirichlet allocation (LDA) is a popular algorithm for discovering structure in large collections of text or other data. Although its complexity is linear in the data size, its use on increasingly massive collections has created considerable interest in parallel implementations. "Approximate distributed" LDA, or AD-LDA, approximates the popular collapsed Gibbs sampling algorithm for LDA models while running on a distributed architecture. Although this algorithm often appears to perform well in practice, its quality is not well understood or easily assessed. In this work, we provide some theoretical justification of the algorithm, and modify AD-LDA to track an error bound on its performance. Specifically, we upper-bound the probability of making a sampling error at each step of the algorithm (compared to an exact, sequential Gibbs sampler), given the samples drawn thus far. We show empirically that our bound is sufficiently tight to give a meaningful and intuitive measure of approximation error in AD-LDA, allowing the user to understand the trade-off between accuracy and efficiency.

Citation: UCI ICS Technical Report # 09-06
Original Date: October 2009
Last Updated: October 27, 2009

1 Introduction

Latent Dirichlet allocation (LDA) models, sometimes called topic models, have received considerable attention over the last decade for their ability to extract semantic content from collections of text documents. The extracted semantic content is useful for a variety of applications such as search, categorization and prediction, as well as understanding the structure of a collection and its metadata. For example, McCallum et al. used a topic model to discover groups of US senators based on their voting records and the text from bills [1]. Topic modeling can be especially useful in understanding the organization of very large scale systems and sets of documents; for example, Blei and Lafferty show how topic models make a powerful tool for browsing, exploring and navigating through the more than 100 years of the journal *Science* [2].

The complexity of LDA is linear in the size of the corpus and the number of topics being learned; however, for large collections of documents even linear complexity becomes computationally challenging. For example, learning a 1000-topic model of MEDLINE, which contains over a billion words, would take months on a single 3GHz processor. Text collections of this size can easily be found from a variety of sources such as email, online news, blogs, and literature databases. Applications of LDA are not limited to text; variants of LDA have also been applied to problems on many other types of data, including understanding the content of images [3] and capturing user rankings and preferences [4].

With the widespread availability of multicore processors and the need to topic model increasingly large collections, researchers have been motivated to investigate ways of parallelizing or distributing LDA's computations. Nallapati et al. [5] developed a parallel algorithm for variational inference in LDA models. However, many researchers prefer to use a collapsed Gibbs sampling approach for learning LDA models [6,7]. Gibbs sampling is fundamentally sequential in nature and thus can be difficult to correctly parallelize, prompting Newman et al. [8] to develop AD-LDA, a distributed algorithm that approximates collapsed Gibbs sampling for LDA. They showed excellent parallel efficiencies for large data sets, and experimentally demonstrated that AD-LDA learned models with similar properties and accuracy to those learned using the exact algorithm on a single processor. Wang et al. [9] show how AD-LDA (along with several optimization tricks) can be represented in either the MPI or MapReduce programming models, demonstrating that they are capable of scaling to very large data sets and providing an open-source implementation of the MPI version. Asuncion et al. [10] generalized the AD-LDA approach to include asynchronous distributed learning, and to use nonparametric versions of the LDA model. However, a significant drawback of AD-LDA is that since the algorithm only *approximates* Gibbs sampling, AD-LDA has no accuracy guarantees, and no inherent way to assess the degree of approximation.

We present a modified parallel Gibbs sampler for LDA which enables performance guarantees. Our algorithm obtains similar speedups to AD-LDA, but provides an on-line measure of the incurred error. This measure allows the user to bound and track the sampling error at each step, to continually assess the quality of the learned model.

2 Latent Dirichlet Allocation

Latent Dirichlet allocation is a probabilistic model which explains word co-appearances in text as arising from a relatively small number of possible semantic topics. Specifically, each document is considered to consist of a small number of topics, each of which is dominated by only a fraction of all possible words. The topics define a simplified representation of the documents, where words which co-appear regularly in documents will tend to appear together in a topic.

The input to LDA is the standard bag-of-words representation of a collection of text documents, where D documents are each represented as a sparse vector of $|W|$ nonnegative counts, with W being the set of words in the vocabulary. LDA models each document d as a mixture θ_d over T latent topics, where each topic ϕ_t is a multinomial distribution over the $|W|$ word vocabulary. In the generative model ϕ_t is drawn from a symmetric Dirichlet with parameter β, and θ_d is drawn from a symmetric

Algorithm 1: Collapsed Gibbs sampling for LDA.

initialize $\mathbf{z}$ at random;
$\mathbf{a}_{dt} = \#\{i : z_{di} = t\} + \alpha$;
$\mathbf{b}_{wt} = \#\{(d,i) : x_{di} = w, z_{di} = t\} + \beta$;
$\mathbf{c}_t = \#\{(d,i) : z_{di} = t\} + |W|\beta$;
repeat
  forall $d \in D,\ i \in \{1 \dots N_d\}$ **do**
    $t \leftarrow z_{di}$; $w \leftarrow x_{di}$; $\mathbf{a}_{dt}$--; $\mathbf{b}_{wt}$--; $\mathbf{c}_t$--;
    $t \sim$ Multinomial($\mathbf{a}_d\, \mathbf{b}_w \,/\, \mathbf{c}$);
    $z_{di} \leftarrow t$; $\mathbf{a}_{dt}$++; $\mathbf{b}_{wt}$++; $\mathbf{c}_t$++;
until *convergence* ;

Dirichlet with parameter α.[1] The i^{th} token in document d is generated by first drawing a topic assignment z_{di} from θ_d, then generating the token x_{di} from $\phi_{z_{di}}$. This generative process is represented as the graphical model shown in Figure 1.

Given the observed data $\mathbf{x}$, the goal is to compute the posterior distribution over the latent variables $\mathbf{z}$, ϕ_t and θ_d. Since exact inference is intractable, one can use variational or sampling methods to perform approximate inference [13]. The collapsed Gibbs sampling algorithm is one of the most commonly adopted methods, and performs well in practice [6,7].

Collapsed Gibbs sampling proceeds by marginalizing over ϕ_t and θ_d and sampling just the topic assignments $\mathbf{z}$. A Gibbs sampling step draws a new value for each topic assignment conditioned on the current values for all the other topic assignments. Given the current state of all but one variable z_{di}, the conditional probability of z_{di} is given by

$$p(z_{di} = t|\mathbf{z}^{\neg di}, \alpha, \beta) \propto p(x_{di}|z_{di} = t, \mathbf{z}^{\neg di}, \beta)\; p(z_{di} = t|\mathbf{z}^{\neg di}, \alpha)$$

where $\mathbf{z}^{\neg di}$ denotes the other assignments. Defining the word label $w = x_{di}$ we have

$$p(z_{di} = t|\mathbf{z}^{\neg di}, \alpha) \propto N_{dt}^{\neg di} + \alpha$$

$$p(x_{di}|z_{di} = t, \mathbf{z}^{\neg di}, \beta) \propto \frac{N_{wt}^{\neg di} + \beta}{N_t^{\neg di} + |W|\beta}$$

Here we have defined the summation of the current assignments (indexed by word w, document d, and topic t) by $N_{wdt} = |\{i : x_{di} = w, z_{di} = t\}|$, and use the convention that missing indices are summed out, so that $N_{dt} = \sum_w N_{wdt}$ and $N_{wt} = \sum_d N_{wdt}$. The superscript "$\neg di$" again indicates that word i in document d has been excluded from the summation.

It is convenient to write the distribution of z_{di} in a vector form over the possible topics t, so that

$$p(z_{di}|\mathbf{z}^{\neg di}, \alpha, \beta) = \text{Multinomial}(\mathbf{p} \propto \mathbf{a}_d \mathbf{b}_w / \mathbf{c}) \qquad (1)$$

where

$$\mathbf{a}_{dt} = N_{dt}^{\neg di} + \alpha \quad \mathbf{b}_{wt} = N_{wt}^{\neg di} + \beta \quad \mathbf{c}_t = N_t^{\neg di} + |W|\beta,$$

the product $\mathbf{a}_w \mathbf{b}_d / \mathbf{c}$ is taken element-wise, and $\mathbf{p}$ is normalized to sum to one. Pseudocode for the collapsed Gibbs sampling algorithm for LDA is listed in Algorithm 1.

[1] For simplicity, we assume symmetric Dirichlet priors with scalar parameters α, β [11], but our error bounds are easily extended to asymmetric Dirichlet priors with vector-valued parameters, and the algorithm can be extended to include learning or optimizing over these parameters as well [12].

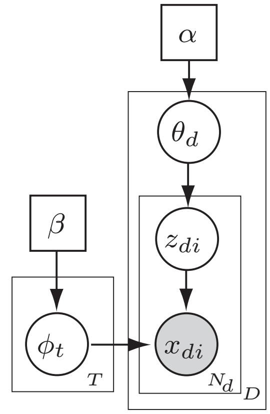


Figure 1. Graphical model for latent Dirichlet allocation. Each observed word x_{di} and its associated latent topic z_{di} is modeled as being generated by a combination of two factors, the topic distribution for that document, θ_d and the word distribution for that topic, ϕ_t.

Algorithm 2: AD-LDA: Perform collapsed Gibbs sampling updates on a distributed set of documents $D_1 \dots D_P$.

initialize $\mathbf{z}$ at random;
$\mathbf{a}_{dt} = \#\{i : z_{di} = t\} + \alpha$;
$\mathbf{b}_{wt} = \#\{(d,i) : x_{di} = w, z_{di} = t\} + \beta$;
$\mathbf{c}_t = \#\{(d,i) : z_{di} = t\} + |W|\beta$;
Partition $D \Rightarrow D_1 \dots D_P$;
repeat
  for $j = 1 \dots P$ *in parallel* **do**
    Copy $\mathbf{b}^j = \mathbf{b}$, $\mathbf{c}^j = \mathbf{c}$;
    forall $d \in D_j,\ i \in \{1 \dots N_d\}$ **do**
      $t \leftarrow z_{di}$; $w \leftarrow x_{di}$; $\mathbf{a}_{dt}$--; $\mathbf{b}^j_{wt}$--; $\mathbf{c}^j_t$--;
      $t \sim$ Multinomial($\mathbf{a}_d\, \mathbf{b}^j_w \,/\, \mathbf{c}^j$);
      $z_{di} \leftarrow t$; $\mathbf{a}_{dt}$++; $\mathbf{b}^j_{wt}$++; $\mathbf{c}^j_t$++;
  Update $\mathbf{b} = \mathbf{b} + \sum_j (\mathbf{b}^j - \mathbf{b})$, $\mathbf{c} = \mathbf{c} + \sum_j (\mathbf{c}^j - \mathbf{c})$;
until *convergence* ;

3 Parallel Gibbs Sampling for LDA

Each iteration of Gibbs sampling updates the topic assignment z_{di} for every word in every document in the collection. For sufficiently massive corpora of text, this can take a long time, and require days or even months of CPU time [8]. For example, learning $T = 1000$ topics for the $D = 19$ million citations in MEDLINE would take an estimated 4 hours to iterate once through the corpus, assuming a single 3GHz processor, and hundreds of these iterations would be required to reach stationarity. Distributed clusters of computers can be used to speed up this process. Even relatively small data sets may benefit from incorporating parallelism, by enabling desktops to take advantage of multicore computing architectures.

The obvious way to distribute the learning of an LDA model is to partition the data set by dividing the collection of documents into P sets, each of which is processed in parallel. This is the idea behind AD-LDA: after distributing the documents over P processors, Gibbs sampling is done on each set concurrently, and the results are combined after each processor has swept through its local data once. Note that we use P to represent both the number of partitions in the data and the number

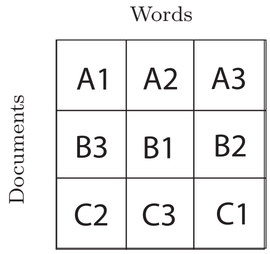


Figure 2. We partition the documents across processors, and divide the words into non-overlapping blocks. Processors sample blocks A1,B1,C1 asynchronously in parallel, re-synchronize, then continue to blocks A2,B2,C2, and so on.

of processors. Although these quantities need not be equal (we need as many partitions than we have processors to work in parallel, but could partition more finely if desired), for simplicity of presentation we assume that the two numbers are the same.

The AD-LDA algorithm is listed in Algorithm 2. Notice that since the data are split into groups along document lines, document-specific variables are accessed only by the process with ownership of that document. Although each processor appears to use a particular variable, in fact each processor is only accessing a processor-specific subsection of the vector, and no access conflicts can arise among the parallel processes. Thus, the partitioning of documents across processors enables us to also partition the variables $\mathbf{z}$ and $\mathbf{a}$ (representing N_{dt}) among the distributed computing elements so that the relevant parts of these variables are always local and up-to-date. For the other variables $(\mathbf{b}, \mathbf{c})$, AD-LDA makes a local copy $(\mathbf{b}^j, \mathbf{c}^j)$ which is updated using that processor's data, and the results are combined at the end of each iteration (line 2). Note that this Gibbs sampling process is *approximate*, in the sense that the topic values sampled are not from the same distributions as would be used in a sequential Gibbs sampling algorithm. In particular, the vectors $\mathbf{b}^j, \mathbf{c}^j$ are not changed to reflect the samples that have been already drawn by other processors, and are thus incorrect. In practice, this approximation appears to be minor, and AD-LDA provides good results. We shall see that, with a little more work, we can gauge this error "on the fly" and control it.

Our first algorithmic modification is to partition the data not only along shared documents but also along shared words. In addition to partitioning documents $D = D_1 \cup \ldots \cup D_P$, we also partition the observed words $W = W_1 \cup \ldots \cup W_P$. We create P^2 data partitions, each of which is defined by a subgroup of documents and words. We arrange the computation so that exactly P of these are done concurrently, and that these P subsets are orthogonal (no two share the same document or word); see Figure 2. The algorithm is outlined as Algorithm 3. The refined partitioning requires slightly more frequent synchronization among the processes (fewer data are processed during each parallel-for loop), but also involves fewer shared resources. The local Gibbs sampling steps can update $\mathbf{a}$ and $\mathbf{b}$ without potential overlap; only $\mathbf{c}$ uses a local copy at each process which is not kept up-to-date.

This data partitioning has recently been independently proposed by Yan et al. [14]. There, the additional partitioning is justified mainly from the perspective of reducing potential read/write conflicts to shared memory. However, we shall see that this modification is key to being able to bound the approximation error inherent in the algorithm.

In terms of approximation error, the smaller overlap among the resources used by each process that results from partitioning both document and word collections has two significant advantages. The first is that $\mathbf{c}$ (or N_t) represents the total number of words allocated to each topic, which is a "bulk" quantity and likely to be relatively stable from iteration to iteration. If $\mathbf{c}$ were constant, i.e., was not changed by each processor, our sampler would be exact. Intuitively, the more stable $\mathbf{c}$, the closer our approximation will be; we make this intuition precise in Section 4. The second important point is that $\mathbf{c}$ is relatively low-dimensional (T values); this small size makes it feasible to save and re-examine its values later, making it possible to retrospectively compare and evaluate its stability.

Algorithm 3: Word-block coordination in AD-LDA.

initialize $\mathbf{z}$ at random;
$\mathbf{a}_{dt} = \#\{i : z_{di} = t\} + \alpha$;
$\mathbf{b}_{wt} = \#\{(d,i) : x_{di} = w, z_{di} = t\} + \beta$;
$\mathbf{c}^0_t = \#\{(d,i) : z_{di} = t\} + |W|\beta$;
Partition $D \Rightarrow D_1 \ldots D_P$, $W \Rightarrow W_1 \ldots W_P$;
repeat
  for $k = 1 \ldots P$ *in sequence* **do**
    for $j = 1 \ldots P$ *in parallel* **do**
      $\bar{k} = k + j - 1 \mod P$;
      Copy $\mathbf{c}^j = \mathbf{c}^0$;
      forall $d \in D_j$, $i \in \{i : x_{di} \in W_{\bar{k}}\}$ **do**
        $t \leftarrow z_{di}$; $w \leftarrow x_{di}$; $\mathbf{a}_{dt}$--; $\mathbf{b}_{wt}$--; $\mathbf{c}^j_t$--;
        $t \sim \text{Multinomial}(\mathbf{a}_d\, \mathbf{b}_w \,/\, \mathbf{c}^j)$;
        $z_{di} \leftarrow t$; $\mathbf{a}_{dt}$++; $\mathbf{b}_{wt}$++; $\mathbf{c}^j_t$++;
    Update $\mathbf{c}^0 = \mathbf{c}^0 + \sum_j (\mathbf{c}^j - \mathbf{c}^0)$;
until *convergence* ;

It may be useful to design the partitioning of D and W so as to produce balanced computational loads across processors. The cost of any given block of data is the number of words which appear within it, i.e., $\sum_{D_i, W_j} N_{dw}$. For a balanced load, we should minimize the spread (maximum minus minimum) across any two blocks which are executed in parallel. However, in practice, a simple random partitioning into equal numbers of documents and words is very effective; we use this strategy for our experiments in Section 6. A more sophisticated optimization method is proposed by Yan et al. [14]. However, load balancing may become less important in more sophisticated scheduling and load allocation architectures such as MapReduce [9].

4 Analyzing the Sampling Error

Our version of AD-LDA has only one source of approximation error, specifically the differences in the topic counts N_t being updated locally at each processor. Using a particular, robust measure of distributional error, it turns out that we can efficiently track and bound the error experienced by the approximate algorithm at each step of the Gibbs sampler.

It is worth noting here that our bound will be on the probability of drawing an incorrect sample at *each step* of the algorithm, given the (possibly incorrect) samples drawn so far. While this does not capture the possible accumulation of errors, we argue that it remains a useful assessment of quality. In particular, AD-LDA as originally developed provides no feedback on quality, and can only be assessed anecdotally by also running a sequentially computed model [8]. A per-sample error bound allows us to consider the error due to distributed computation on the same footing as we might consider, for example, roundoff error in floating point calculations, or non-uniformity in a random number generator. Being a probability of error, it has an intuitive scale by which to gauge its magnitude, and thus provides a way to meaningfully assess the level of error experienced by the sampler.

4.1 Hilbert's Projective Metric

We make use of a measure of error between two vectors known as Hilbert's projective metric [15]. This metric has been successfully applied to analyze approximations and parallelism in a probabilistic inference algorithm called *belief propagation*; it was independently developed in [16] (there termed the *dynamic range*) for the purposes of analyzing the behavior of the belief propagation algorithm under small perturbations, and subsequently for designing efficient parallel partitioning and scheduling of the algorithm [17].

The projective metric $d(\mathbf{v}, \hat{\mathbf{v}})$ between two positive vectors $\mathbf{v}$ and $\hat{\mathbf{v}}$ is given by

$$d(\mathbf{v}, \hat{\mathbf{v}}) = \max_{t,t'} [\log(\mathbf{v}_t/\hat{\mathbf{v}}_t) - \log(\mathbf{v}_{t'}/\hat{\mathbf{v}}_{t'})] \tag{2}$$

This distance measure is closely related to the L_∞ or sup-norm applied to $\log \mathbf{v}$, and has a number of useful properties for analyzing the ways in which distributional errors behave during inference. The most important properties for our analysis are:

Theorem 1. *The projective metric is invariant to positive scaling on the vectors, so that if λ, λ' are positive scalars, we have*

$$d(\lambda\mathbf{v}, \lambda'\hat{\mathbf{v}}) = d(\mathbf{v}, \hat{\mathbf{v}}) \tag{3}$$

Proof. From the definition (2) we have

$$\begin{aligned} d(\lambda\mathbf{v}, \lambda'\hat{\mathbf{v}}) &= \max_{t,t'} \log \frac{\lambda\mathbf{v}_t \ \lambda'\hat{\mathbf{v}}_{t'}}{\lambda'\hat{\mathbf{v}}_t \ \lambda\mathbf{v}_{t'}} \\ &= \max_{t,t'} \log \frac{\mathbf{v}_t\hat{\mathbf{v}}_{t'}}{\hat{\mathbf{v}}_t\mathbf{v}_{t'}} \\ &= d(\mathbf{v}, \hat{\mathbf{v}}) \end{aligned}$$

□

Theorem 2. *Let $\mathbf{s}$ be any positive vector; then*

$$d(\mathbf{s}\mathbf{v}, \mathbf{s}\hat{\mathbf{v}}) = d(\mathbf{v}, \hat{\mathbf{v}}) \tag{4}$$

where the vector multiplication is taken elementwise, $(\mathbf{s}\mathbf{v})_t = \mathbf{s}_t\mathbf{v}_t$.

Proof. Again from (2) we have

$$\begin{aligned} d(\mathbf{s}\mathbf{v}, \mathbf{s}\hat{\mathbf{v}}) &= \max_{t,t'} \log \frac{\mathbf{s}_t\mathbf{v}_t \ \mathbf{s}_{t'}\hat{\mathbf{v}}_{t'}}{\mathbf{s}_t\hat{\mathbf{v}}_t \ \mathbf{s}_{t'}\mathbf{v}_{t'}} \\ &= \max_{t,t'} \log \frac{\mathbf{v}_t\hat{\mathbf{v}}_{t'}}{\hat{\mathbf{v}}_t\mathbf{v}_{t'}} \\ &= d(\mathbf{v}, \hat{\mathbf{v}}) \end{aligned}$$

□

Theorem 3. *Let $\sum_t \mathbf{v}_t = \sum_t \hat{\mathbf{v}}_t$. Then, adding any nonnegative vector $\mathbf{h}$ does not increase the distance between $\mathbf{v}$ and $\hat{\mathbf{v}}$:*

$$d(\mathbf{v}+\mathbf{h}, \hat{\mathbf{v}}+\mathbf{h}) \le d(\mathbf{v}, \hat{\mathbf{v}}) \tag{5}$$

Proof. Since $(\mathbf{s}\mathbf{v})_t = \mathbf{s}_t\mathbf{v}_t$, by the mean value theorem we have that

$$\min_{t'} \log \mathbf{v}_{t'}/\hat{\mathbf{v}}_{t'} \le 0 \le \max_t \log \mathbf{v}_t/\hat{\mathbf{v}}_t.$$

Furthermore, it is easy to show that for $a, b, c, d \ge 0$,

$$\frac{a}{c} \le \frac{b}{d} \quad \Rightarrow \quad \frac{a}{c} \le \frac{a+b}{c+d} \le \frac{b}{d}.$$

Since log is monotonic, and $\log \mathbf{h}_t/\mathbf{h}_t = \log 1 = 0$, we have

$$\log \frac{\mathbf{v}_{t'}}{\hat{\mathbf{v}}_{t'}} \le \log \frac{\mathbf{v}_{t'}+\mathbf{h}_{t'}}{\hat{\mathbf{v}}_{t'}+\mathbf{h}_{t'}} \le 0 \le \log \frac{\mathbf{v}_t+\mathbf{h}_t}{\hat{\mathbf{v}}_t+\mathbf{h}_t} \le \log \frac{\mathbf{v}_t}{\hat{\mathbf{v}}_t}.$$

□

Finally, we would like to relate $d(\mathbf{v}, \hat{\mathbf{v}})$ to more traditional norms, particularly when applied to normalized probability distributions.

Theorem 4. *Let $\mathbf{v}$, $\hat{\mathbf{v}}$ be two positive vectors with unit sum, so that $\sum_t \mathbf{v}_t = \sum_t \hat{\mathbf{v}}_t = 1$, and let $d(\mathbf{v}, \hat{\mathbf{v}}) = \epsilon$. Then the L_1 difference is bounded by*

$$\|\mathbf{v} - \hat{\mathbf{v}}\|_1 = \sum_t |\mathbf{v}_t - \hat{\mathbf{v}}_t| \le |1 - e^\epsilon| = \epsilon + O(\epsilon^2) \tag{6}$$

and therefore, the difference in probabilities assigned to any event E is also bounded:

$$\max_E \left| \sum_{t\in E} \mathbf{v}_t - \sum_{t\in E} \hat{\mathbf{v}}_t \right| \le \frac{1}{2}|1 - e^\epsilon| = \frac{1}{2}\epsilon + O(\epsilon^2) \tag{7}$$

Proof. Since $\sum(\mathbf{v} - \hat{\mathbf{v}}) = 0$, we know $\min_t \frac{\hat{\mathbf{v}}_t}{\mathbf{v}_t} \le 1 \le \max_t \frac{\hat{\mathbf{v}}_t}{\mathbf{v}_t}$. Thus, $\exp(-\epsilon) \le \frac{\hat{\mathbf{v}}_t}{\mathbf{v}_t} \le \exp(\epsilon)$. We then see

$$\sum_t |\mathbf{v}_t - \hat{\mathbf{v}}_t| = \sum_t \mathbf{v}_t \ |1 - \hat{\mathbf{v}}_t/\mathbf{v}_t| \le |1 - e^\epsilon|.$$

Inequality (7) then follows from Scheffé's identity [18]:

$$\begin{aligned} \max_E \left| \sum_{t\in E} \mathbf{v}_t - \sum_{t\in E} \hat{\mathbf{v}}_t \right| &= \sum_{t:\mathbf{v}_t > \hat{\mathbf{v}}_t} \mathbf{v}_t - \hat{\mathbf{v}}_t \\ &= \sum_{t:\mathbf{v}_t < \hat{\mathbf{v}}_t} \hat{\mathbf{v}}_t - \mathbf{v}_t = \frac{1}{2}\sum_t |\mathbf{v}_t - \hat{\mathbf{v}}_t|. \end{aligned}$$

□

4.2 Error Bounds in Parallel LDA

Let us now examine the distributions used while executing parallel collapsed Gibbs sampling for LDA, as in Algorithm 3. Consider a sequential version of this algorithm, in which process $j = 1$ executes first, then $j = 2$, and so on, and imagine that in addition to the local copy $\mathbf{c}^j$ of $\mathbf{c}^0$ used at each processor, the processor *also* updates the true global count $\mathbf{c}$. We will design our algorithm to assess and bound the error between the parallel and sequential distributions at each step of the algorithm, *given* the samples that have been drawn thus far. In other words, we compare the distributions

$$\mathbf{p} \propto \mathbf{a}_d\mathbf{b}_w/\mathbf{c} \qquad\qquad \hat{\mathbf{p}} \propto \mathbf{a}_d\mathbf{b}_w/\mathbf{c}^j$$

but draw our sample according to $\hat{\mathbf{p}}$, and update both $\mathbf{c}$ and $\mathbf{c}^j$ for the next step given this sample. Thus we assess the instantaneous difference in distributions between the sequential and parallel versions of the algorithm, but do not consider the

Algorithm 4: Bounding errors in AD-LDA.

Follow Algorithm 3, with modifications:

10a $\mathbf{h}^j_t = \#\{(d,i) : d \in D_j, x_{di} \in W_k, z_{di} = t\}$;
10b $\hat{\mathbf{v}}^j = \mathbf{c}^0 - \mathbf{h}^j$;
11 **forall** $d \in D_j,\ i \in \{i : x_{di} \in W_{\bar{k}}\}$ **do**
12 $\quad t \leftarrow z_{di};\ w \leftarrow x_{di};\ \mathbf{a}_{dt}\text{--};\ \mathbf{b}_{wt}\text{--};\ \mathbf{h}^j_t\text{--}$;
13 $\quad t \sim \text{Multinomial}(\mathbf{a}_d\,\mathbf{b}_w \,/\, (\hat{\mathbf{v}}^j + \mathbf{h}^j))$;
14 $\quad z_{di} \leftarrow t;\ \mathbf{a}_{dt}\text{++};\ \mathbf{b}_{wt}\text{++};\ \mathbf{h}^j_t\text{++}$;
15a Compute $\epsilon_j = d(\ \hat{\mathbf{v}}^j\ ,\ \hat{\mathbf{v}}^j + \sum_{k<j}(\mathbf{h}^k + \hat{\mathbf{v}}^k - \mathbf{c}^0)\ )$;
15b Update $\mathbf{c}^0 = \mathbf{c}^0 + \sum_j(\mathbf{h}^j + \hat{\mathbf{v}}^j - \mathbf{c}^0)$;

accumulation of errors in the distribution; see the discussion of this issue in Section 4.

The difficulty with this approach lies in the fact that the "true" $\mathbf{c}$ is not available at the time each processor evaluates its data. A comparison to $\mathbf{c}$ must be retrospective, i.e., it can take place only after the preceding processes have finished. However, the distributions evaluated change at each step, depending on $\mathbf{a}_d$ and $\mathbf{b}_w$, and moreover these vectors and $\mathbf{c}$, $\mathbf{c}^j$ all evolve with each step as topic assignments are changed. Thus it is not obvious that $\|\mathbf{p} - \hat{\mathbf{p}}\|_1$ can be evaluated without re-visiting each datum in sequence.

Luckily the bounds from Section 4.1 enable $\|\mathbf{p} - \hat{\mathbf{p}}\|_1$ to be bounded efficiently. To do so, we must slightly modify our algorithm and the quantities it keeps track of; these changes are given in Algorithm 4. Specifically, we first separate the topic counts associated with data in the current process, and denote this vector $\mathbf{h}^j$. We denote the remainder of the counts (those associated with other processes) as $\hat{\mathbf{v}}^j$. Each step of the Gibbs sampler in processor j affects only $\mathbf{h}^j$; $\hat{\mathbf{v}}^j$ remains constant. We can thus save $\hat{\mathbf{v}}^j$, and use it to retrospectively bound the error.

Once all processes have finished, we can compare the $\mathbf{c}^j$ used by each processor to the $\mathbf{c}$ which would have been computed sequentially. However, instead of comparing $\mathbf{c}^j$ and $\mathbf{c}$ (which evolve during the process), we compare $\hat{\mathbf{v}}^j$ and its sequentially obtained version, $\mathbf{v}$. As $\hat{\mathbf{v}}^j$ represents the topic counts of data assigned to other processes at the beginning of j's operations, $\mathbf{v}$ represents what these topic counts would have been had j waited until all prior processes had finished. The vector $\mathbf{v}$ can be easily computed given the process outputs, by summing up the changes in each preceding process' counts, and the error computed as:

$$\epsilon_j = d(\ \hat{\mathbf{v}}^j\ ,\ \mathbf{v}\ =\ \hat{\mathbf{v}}^j + \sum_{k<j}(\mathbf{h}^k + \hat{\mathbf{v}}^k - \mathbf{c}^0)\ )$$

We then have the following result:

Theorem 5. *The probability of drawing an incorrect sample at each step of the Gibbs sampler due to the parallel computation, given the values of all preceding samples, is bounded by* $(e^{\epsilon_j} - 1)/2 = \epsilon_j/2 + O(\epsilon_j^2)$.

Proof. Using the preceding theorems, we have that for an arbitrary nonnegative vector $\mathbf{h}^j$ and positive vectors $\mathbf{a}_d$, $\mathbf{b}_w$,

$$\begin{aligned} d(\hat{\mathbf{v}}^j\ ,\ \mathbf{v}) &\geq d(\hat{\mathbf{v}}^j + \mathbf{h}^j\ ,\ \mathbf{v} + \mathbf{h}^j\ ) \\ &= d(1/(\hat{\mathbf{v}}^j + \mathbf{h}^j)\ ,\ 1/(\mathbf{v} + \mathbf{h}^j)\ ) \\ &= d(\mathbf{a}_d\mathbf{b}_w\,/(\hat{\mathbf{v}}^j + \mathbf{h}^j)\ ,\ \mathbf{a}_d\mathbf{b}_w\,/(\mathbf{v} + \mathbf{h}^j)\ ) \\ &= d(\hat{\mathbf{p}}^j\ ,\ \mathbf{p}\ ) \end{aligned}$$

where the identities follow from equation (5), the definition (2), equation (4), and (3), respectively. Applying Theorem 4 completes the proof. □

Moreover, to track ϵ^j for each processor requires only $P \cdot T$ additional storage and sequential work after all processes have finished.

We separate $\mathbf{h}^j$ from $\mathbf{v}$ because $\mathbf{h}^j$ evolves during the algorithm, and we wish to bound the error using a single computation at the end. However, the fact that $\mathbf{h}^j$ only needs to be nonnegative suggests another small modification. If we also track the minimum value of $\mathbf{h}^j$ at any step of the Gibbs sampler, this count can be included in $\mathbf{v}$ as well, by modifying Algorithm 4:

10b $\hat{\mathbf{v}}^j = \mathbf{c}^0 - \mathbf{h}^j$; $\mathbf{h}^{j+} = \mathbf{h}^j$;
12b $\mathbf{h}^{j+}_t = \min(\mathbf{h}^{j+}_t, \mathbf{h}^j_t)$;
15a $\epsilon_j = d(\ \hat{\mathbf{v}}^j + \mathbf{h}^{j+}\ ,\ \hat{\mathbf{v}}^j + \mathbf{h}^{j+} + \sum_{k<j}(\mathbf{h}^k + \hat{\mathbf{v}}^k - \mathbf{c}^0)\ )$;

Again, this modification requires tracking only T additional values at each processor. In practice, however, it appears to provide only a very slight improvement over Algorithm 4.

5 Approximate Scaling Analysis

Before showing the empirical behavior of our error bounds, it is useful to describe a simple, approximate analysis of the error and its dependence on parameters such as the number of data, number of partitions, and number of topics. We will compare this expected scaling behavior with the empirical behavior we measure in Section 6.

Our error bound makes use of the idea that, if the counts allocated to the topic count vector $\mathbf{c}$ were unchanged by each block sampled in parallel, then the sampling would have proceeded correctly. Moreover, $\mathbf{c}$ is a bulk quantity (the sum of many individual topic assignments), making it likely to remain approximately constant. We are thus relying on the law of large numbers to make the allocation of counts approximately constant over the course of a set of parallel block operations. Here, we will use a simple approximation to this idea to gauge how we might expect the errors to scale with various parameters.

Let us suppose that the topics are approximately uniform, i.e., the actual allocation is centered around N/T counts per topic, and let K be the number of data being processed in parallel at any one time. Ideally, we have $K \approx N/P$, since there are P^2 blocks total, P of which are executed in parallel at any one time. If the data being sampled in parallel are are also approximately uniformly distributed across the topics and are assumed to be independent, the probability of each datum falling into a given topic is $1/T$, and thus the total number of these counts to fall in each topic is approximately normal with mean K/T and variance $\frac{K(T-1)}{T^2} \approx K/T$.

Suppose that we take the usual 95% confidence interval (plus or minus two standard deviations) to be an estimate of the range that we might observe in practice. Then, the total number of counts in each bin is

$$\frac{N}{T} \pm 2\sqrt{\frac{K}{T}},$$

and the projective measure of error between the endpoints of the 95% confidence interval and the nominal uniform (N/T)

Data Set	$\|D\|$	$\|W\|$	N
KOS	3,430	6,906	467,714
NIPS	1,500	12,419	1,932,365
Enron	39,861	28,102	6,412,172

Figure 3. Bag-of-words text data sets used in the experiments and their relative sizes, available from [19].

count vector is

$$
\begin{aligned}
d(\hat{p}\,,\,p) &\approx \log\left(\frac{N/T + 2\sqrt{K/T}}{N/T}\,\frac{N/T}{N/T - 2\sqrt{K/T}}\right)\\
&= \log\left(\frac{1 + 2\sqrt{KT}/N}{1 - 2\sqrt{KT}/N}\right)\\
&\approx \log\left(1 + 4\sqrt{KT}/N + O(KT/N^2)\right)\\
&\approx 4\frac{\sqrt{KT}}{N}\\
&= 4\sqrt{\frac{T}{NP}}
\end{aligned}
$$

Thus, under the previously described approximations, we would expect that the error we observe should increase with the square root of the number of topics T, and decrease with the square root of both the number of partitions P and the total number of data N. We shall see in Section 6 that these approximations are reasonably accurate in practice.

6 Experiments

We empirically validate our results using an OpenMP implementation of AD-LDA to show the speed-up of our modified algorithm, along with the scaling of our error bounds with problem size, number of data partitions, and topics, and their behavior over time (iteration number). We use several instances of bag-of-words text data from the UCI Machine Learning Repository [19], whose relative sizes (in terms of number of documents, unique words, and total number of words in the corpus) are listed in Table 3. Following the parameter choices of [8], we select $\alpha = .1$, $\beta = .01$, and $T = 200$ for all three data sets except where noted.

6.1 Timing and speed-up

Figure 4 shows the speed-up of our modified version of AD-LDA on an eight-core desktop machine, having subdivided the data into P partitions where P is the number of cores in use. As can be seen, speed-up is nearly linear in the number of cores, showing that (like the original formulation) our block-synchronized AD-LDA has excellent parallel efficiency. It should be noted however that our results are presented using a shared memory architecture. Using a cluster or other distributed architecture may see less efficiency due to slower communications, but we would expect its behavior again to be similar to the original AD-LDA algorithm.

6.2 Behavior over time

We can also examine the behavior of our error bound as the Gibbs sampler progresses. Figure 5(a) shows the error bound produced by Algorithm 4 on the largest of the data sets (Enron), as a function of the iteration number and for $P = 4$. For comparison, we also show the actual error experienced by the sampler (dashed line). The actual error is measured by running the sampler sequentially but simulating the distributed computations and comparing their respective sampling distributions. For a reasonable comparison, we report the maximum mis-allocated probability mass, $\frac{1}{2}\|\mathbf{p} - \hat{\mathbf{p}}\|_1$, observed during each iteration of the Gibbs sampler. This value represents the tightest bound possible when measuring error in the proposed sense.

The figure shows that both our algorithm's bound and the true error behave similarly, and in practice appear to be separated by an approximately constant factor over time (here, between about 2 and 3). The sampling error often appears to undergo an increase at the very beginning of the algorithm, as many topic assignments at each processor are changing rapidly. The error then begins to fall off, eventually arriving at some steady-state level of fluctuation. Similar shapes are observed for the other data sets as well.

The timing of this decrease in sampling error can be compared to the drop in perplexity (a measure of the current parameters' representation of the data) over time, shown in Figure 5(b). Note that, as in the empirical findings of [8], exact LDA and AD-LDA (at various numbers of partitions P) showed almost indistinguishable perplexity curves, and so we show only the exact (sequential) perplexity. In general, the increased error levels seen in Figure 5(a) appear to correspond roughly with the rapid decrease in perplexity, while the steady-state error levels are associated with the more gradual change in perplexity. Intuitively, significant improvements in perplexity correspond to highly probable (nearly non-random) topic assignments, which then cause significant drift across processors and results in the temporary increase in sampling error.

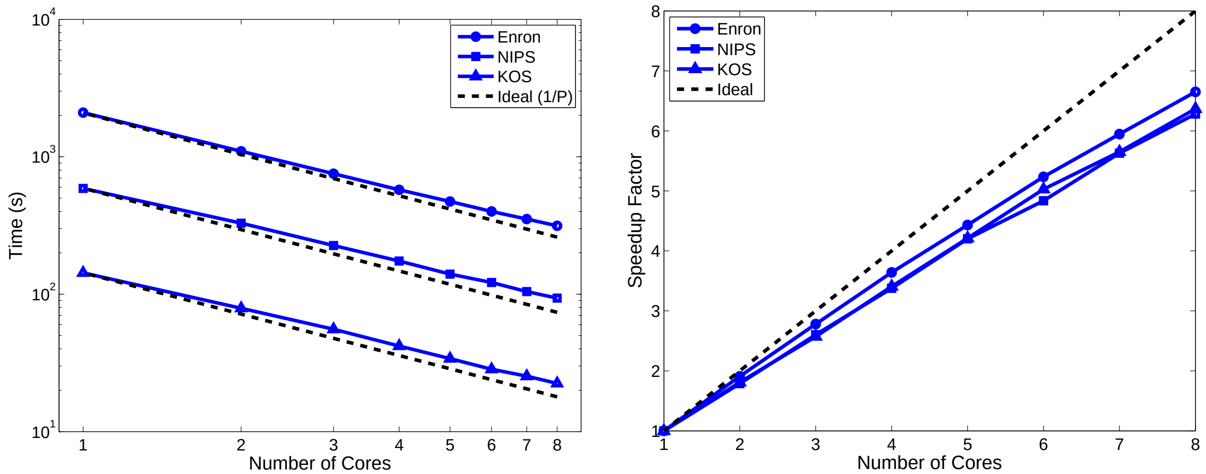
6.3 Scaling behavior

Next, we examine the scaling behavior of our error bound with the data size N (number of words in the corpus), the number of partitions P, and the number of topics T. The results of these experiments on all three data sets are shown in Figure 6. For comparison, we again show both the bound obtained by our algorithm (solid blue lines) and the actual sampling error experienced by the sampler (dashed lines). In both cases, we report the average values observed within the last 50 iterations (i.e., within the steady-state error regime depicted in Figure 5(a)), and averaged over 10 trials. All three data sets appear in each figure: KOS (circles), NIPS (squares), and Enron (triangles). Also shown in each figure is a reference line (red) to indicate the scaling behavior anticipated by the approximations in Section 5. All plots are logarithmic in both scales for clarity.

Figure 6(a) shows the error values as a function of data set size (one point per data set). As expected, the error decreases for larger data sets. Both the actual error and our bound decrease at approximately the same rate, which is roughly similar to the $1/\sqrt{N}$ behavior suggested by our analysis.

Figure 6(b) shows the error level as a function of the number of partitions P. Increasing the number of partitions also increases the number of times per iteration that the algorithm synchronizes and updates its counts. Thus as expected, we see that both the actual error and our bound decrease as the number of partitions grows. Again, across all sizes of partitionings, our error bound remains reasonably close to the actual maximum error observed during sampling, and again the trend is quite close to the anticipated $1/\sqrt{P}$ scaling behavior.

Finally, Figure 6(c) shows the error level as a function of the number of topics T. Increasing the number of topics increases both the actual error and our bound. In this case, the actual



(a) Computation time (b) Speedup factor

Figure 4. Parallel efficiency of the modified AD-LDA sampler. (a) Computation time for each data set, as a function of the number of cores used; (b) Speedup factor (sequential time divided by parallel time) as a function of the number of cores. Dashed lines show ideal (linear) speedup behavior. As can be seen, AD-LDA is quite close to optimal in all three data sets, indicating a high parallel efficiency.

error appears to scale in a similar way to our anticipated $\sqrt{T}$ behavior, but the error bound may be increasing at a slightly faster rate. Intuitively, our bound depends on the stability of the log-counts over each iteration; for a sufficiently large number of topics, some topic will have only a few counts, and will thus be unstable in our projective metric sense. The bound often becomes loose in these cases. This suggests that for very large values of T, such as nonparametric models in which T is effectively infinite [10], more research may be be required to provide a tight estimate of the error.

7 Conclusion

We have presented a variant of AD-LDA which has a number of advantages over the original. It reduces the number of sources of error by decreasing the shared resources between threads executing in parallel. Moreover, by using this modification and tracking slightly more information at each processor, the algorithm is able to retrospectively construct a bound on the probability of drawing an incorrect sample at each step once the processors resynchronize. Our empirical results show that the bounds closely track the actual maximum error experienced by AD-LDA, and quantitatively support the anecdotal evidence that AD-LDA provides accurate approximations.

This modification gives us the means to check the behavior of AD-LDA during execution, obtaining some assurance that our distributed implementation is not causing serious errors. Empirically, we see that larger data sets and smaller parallel blocks typically lead to better approximations, and that error typically increases during early mixing but falls off as the model stabilizes.

Although we have presented our bounds for LDA and text data, it should also be extensible to more general problems as well. In theory, our results are applicable to hierarchical and nonparametric variants of LDA as well [10, 20]. However, since we rely on the stability of the log-counts in distributed copies of shared data during sampling, in practice the resulting error bounds may become loose and may require additional research. Other potential extensions include analyzing asynchronous exchanges between processors [10], and more general mixture models such as those that arise for image features or other continuous data [3].

Acknowledgments

The authors would like to thank Arthur Asuncion, Max Welling, and Padhraic Smyth for many useful discussions on LDA and its distributed implementations.

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

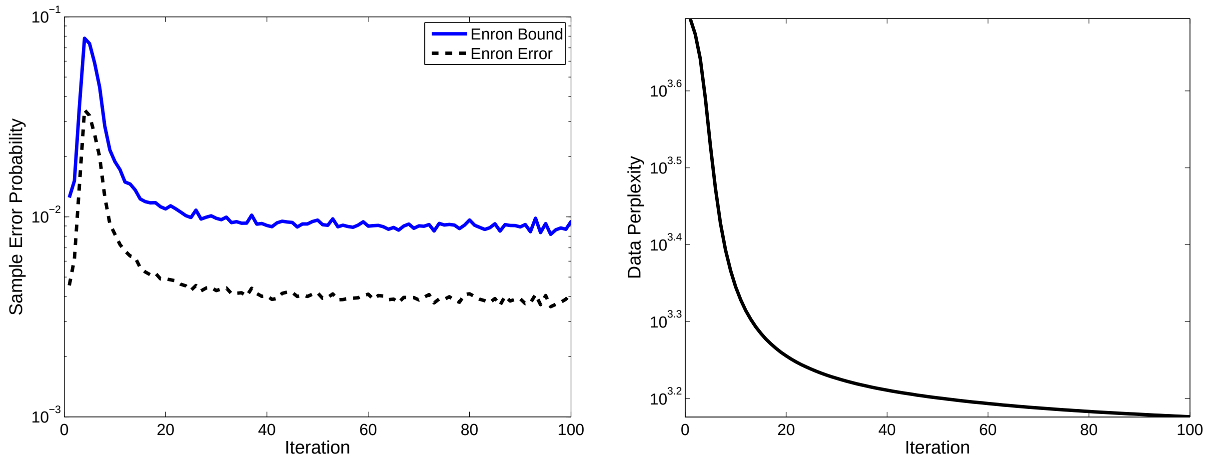
[1] A. McCallum, "Information extraction: Distilling structured data from unstructured text," *ACM Queue*, vol. 3, no. 9, pp. 48–57, Nov. 2005.

[2] D. Blei and J. Lafferty, "A correlated topic model of *Science*," *Annals of Applied Statistics*, vol. 1, no. 1, pp. 17–35, 2007.

[3] J. Sivic, B. Russell, A. Efros, A. Zisserman, and W. Freeman, "Discovering object categories in image collections," in *Proc. International Conference on Computer Vision*, 2005.

[4] T. Rubin and M. Steyvers, "A topic model for movie choices and ratings," in *Proc. Ninth International Conference on Cognitive Modeling*, Manchester, UK, 2009.

[5] R. Nallapati, W. Cohen, and J. Lafferty, "Parallelized variational em for latent Dirichlet allocation: An experimental evaluation of speed and scalability," in *ICDMW '07: Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 349–354.



(a) (b)

Figure 5. Algorithm behavior as a function of the iteration number on the Enron data set. (a) Sampling error, for $P = 4$ partitions. The solid curve shows the error bound computed using our method, while the dashed curve shows the maximum actual error experienced at each iteration (computed using sequential sampling). Errors are high early on, while many assignments are changing, then fall off to a lower steady-state value after the chain has mixed. (b) Perplexity of the Enron data set as a function of iteration. Perplexity falls off rapidly at first, then slows; the rapid improvement corresponds to approximately the same time that the sampling error is high. As observed in the empirical results in Newman et al. [8], sequential Gibbs sampling and AD-LDA for all tested numbers of partitions produced almost indistinguishable perplexity curves.

[6] M. Welling, Y. W. Teh, and B. Kappen, "Hybrid variational/Gibbs collapsed inference in topic models," in *Uncertainty in Artificial Intelligence (UAI)*, 2008, pp. 587–594.

[7] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh, "On smoothing and inference for topic models," in *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, June 2009.

[8] D. Newman, A. Asuncion, P. Smyth, and M. Welling, "Distributed inference for latent Dirichlet allocation," in *Proc. Neural Information Processing Systems (NIPS) 20*, Dec. 2007.

[9] Y. Wang, H. Bai, M. Stanton, W.-Y. Chen, and E. Y. Chang, "PLDA: Parallel latent Dirichlet allocation for large-scale applications," in *Proc. of 5th Int'l Conf. on Algorithmic Aspects in Information and Management (AAIM)*, June 2009.

[10] A. Asuncion, P. Smyth, and M. Welling, "Asynchronous distributed learning of topic models," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., 2009, pp. 81–88.

[11] T. L. Griffiths and M. Steyvers, "Finding scientific topics." *Proc Natl Acad Sci U S A*, vol. 101 Suppl 1, pp. 5228–5235, April 2004. [Online]. Available: http://dx.doi.org/10.1073/pnas.0307752101

[12] T. Minka, "Estimating a Dirichlet distribution," 2003. [Online]. Available: http://research.microsoft.com/en-us/um/people/minka/papers/dirichlet/

[13] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[14] F. Yan, N. Xu, and Y. Qi, "Parallel inference for latent Dirichlet allocation on graphics processing units," in To appear, *Advances in Neural Information Processing Systems*, Dec. 2009.

[15] P. J. Bushell, "Hilbert's metric and positive contraction mappings in a Banach space," *Archive for Rational Mechanics and Analysis*, 1973.

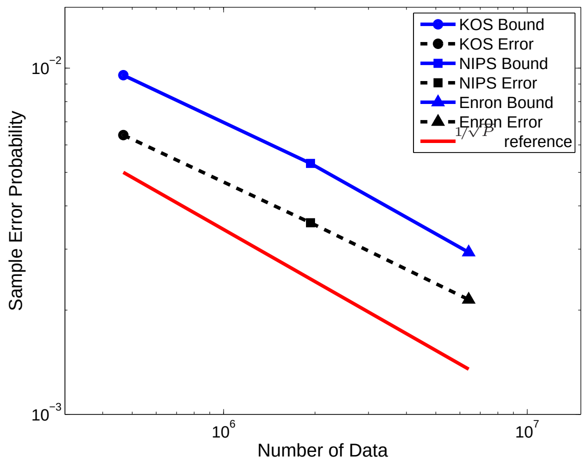
[16] A. T. Ihler, J. W. Fisher III, and A. S. Willsky, "Loopy belief propagation: Convergence and effects of message errors," *Journal of Machine Learning Research*, vol. 6, pp. 905–936, May 2005.

[17] J. Gonzalez, Y. Low, and C. Guestrin, "Residual splash for optimally parallelizing belief propagation," in *In Artificial Intelligence and Statistics (AISTATS)*, Clearwater Beach, Florida, April 2009.

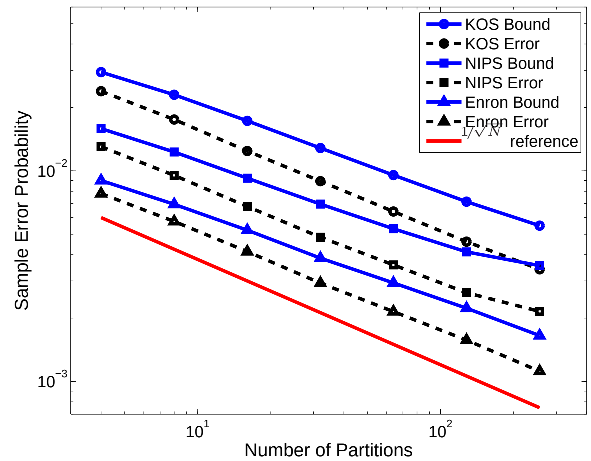
[18] L. Devroye and G. Lugosi, *Combinatorial Methods in Density Estimation*. New York: Springer, 2001.

[19] A. Asuncion and D. Newman, "UCI machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

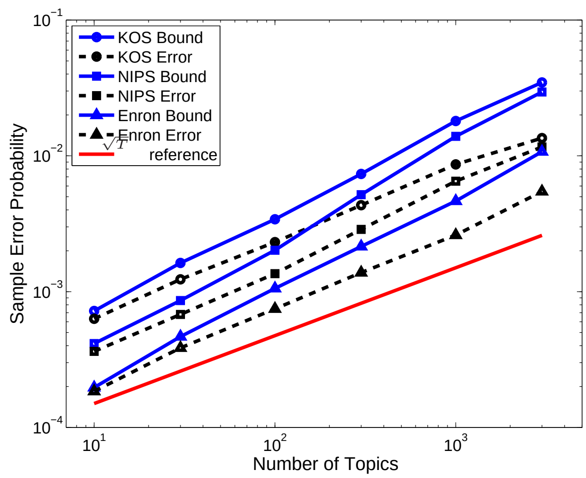
[20] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei, "Sharing clusters among related groups: Hierarchical Dirichlet processes," in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. Cambridge, MA: MIT Press, 2005, pp. 1385–1392.



(a)



(b)



(c)

Figure 6. Error scaling as a function of (a) the size of the data set, N; (b) the number of partitions, P; (c) the number of topics, T. Each plot shows both the error bound (solid, blue) and actual error probability (dashed, black) for the KOS (circles), NIPS (squares), and Enron (triangle) data sets. Also shown in each plot is a reference line showing the estimated scaling behavior described in Section 5. In all cases, the estimated behavior appears roughly similar to the observed curves for both the error bound and the true error probability.