DRAFT COPY; TO APPEAR IN: IEEE TRANS. KNOWLEDGE & DATA ENGINEERING

Understanding Errors in Approximate Distributed Latent Dirichlet Allocation

Alexander Ihler *Member, IEEE*, David Newman

Abstract—Latent Dirichlet allocation (LDA) is a popular algorithm for discovering semantic 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 theoretically or easily assessed on new data. In this work, we theoretically justify the approximation, and modify AD-LDA to track an error bound on 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 track the trade-off between accuracy and efficiency while executing in parallel.

Index Terms—data mining, topic model, parallel processing, error analysis

1 Introduction

Latent Dirichlet allocation (LDA) models, sometimes called topic models, have received considerable attention 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. 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* [1].

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 almost a billion words, would take weeks on a single 3GHz processor. Text collections of this size are not uncommon in email, news, blog, and literature databases. Variants of LDA have also been applied to other data types, including understanding the content of images [2] and user rankings [3].

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. [4] 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 [5], [6]. Gibbs sampling is fundamentally sequential in nature and thus can be difficult to correctly parallelize, prompting Newman et al. [7] 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 on several data sets that AD-LDA learned models with similar properties and accuracy to those learned using the exact algorithm on a single processor. Wang et al. [8] show how AD-LDA (along with several optimization tricks) can be represented in either the MPI or MapReduce programming models, demonstrated scaling to very large systems and provide an open-source implementation of the MPI version. Asuncion et al. [9] 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, and comes with no guarantees regarding the accuracy of this approximation, it does not inherit the same desirable theoretical properties that Gibbs sampling enjoys.

The term "approximation error" for an algorithm such as AD-LDA may be interpreted in a number of ways. First, any probabilistic model is by nature an approximation of the true system, and given the LDA model, algorithms such as collapsed Gibbs sampling perform approximate inference, as exact inference is computationally intractable even for small corpora of documents. These "approximations" are fundamental to the LDA approach; they can be justified anecdotally by examining the quality of topics learned on a few corpora, or by considering the modeling decisions inherent in the LDA description and theoretical properties of Gibbs sampling, such as its asymptotic consistency.

However, AD-LDA *intentionally* introduces an additional source of approximation in order to make the algorithm amenable to distributed computation, and it is this source of approximation error examined in this work. For example, as might be expected there is a fundamental trade-off between parallel efficiency, or how well we take advantage of our distributed computing platform, and the fidelity of our algorithm. However, AD-LDA as originally designed can only give feedback about one side of this trade-off: it is easy to measure parallel efficiency, but it provides no information about how much error we have introduced to obtain that efficiency. Moreover, although AD-LDA's anecdotal evidence is quite favorable [7], such data do not tell us what to expect in terms of new corpora, for example feedback about whether our algorithm continues to perform as expected, or how it might

A. Ihler is with the Department of Computer Science, University of California, Irvine, Irvine, CA 92697 USA. Tel: (949) 824-3645. Fax: (949) 824-4056. (e-mail: ihler@ics.uci.edu)

D. Newman is with the University of California, Irvine, Irvine, CA 92697 USA, and NICTA Victoria Research Laboratory, University of Melbourne, Melbourne, Vic 3010, Australia. (e-mail: newman@uci.edu)

alter as we change the model parameters.

We present a modified parallel Gibbs sampler for LDA which enables us to measure the fidelity aspect of this trade-off in terms of performance guarantees. Our algorithm obtains the same speedups as AD-LDA, but provides an on-line measure of the approximation quality compared to a sequential sampler. This measure allows the user to bound and track the sampling error at each step of the algorithm, and assess the quality of the approximate learning procedure.

2 Latent Dirichlet Allocation

Latent Dirichlet allocation is a probabilistic model that explains word co-appearances in text as arising from a relatively small number of possible semantic groups, or topics. 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 that 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 a set of documents D 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 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 $\phi_{z_{di}}$. We use N_d to indicate the number of tokens in document d. The complete generative process is represented as the graphical model shown in Fig. 1a.

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 [12]. The collapsed Gibbs sampling algorithm is one of the most commonly adopted methods, and performs well in practice [5], [6].

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} given the other assignments $\mathbf{z}^{\neg di}$ is

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

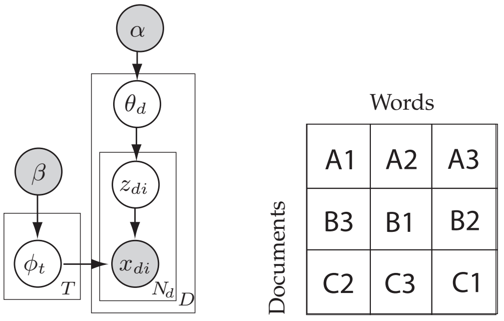
$$= \frac{N_{wt}^{\neg di}+\beta}{N_t^{\neg di}+|W|\beta} \quad \cdot \quad N_{dt}^{\neg di}+\alpha$$

where $w = x_{di}$, and we define summations of the current assignments (indexed by word w, document d, and topic t) by $N_{wdt} = |\{i : x_{di} = w, z_{di} = t\}|$, $N_{dt} = \sum_w N_{wdt}$, $N_{wt} = \sum_d N_{wdt}$, and $N_t = \sum_{d,w} N_{wdt}$. The superscript "$\neg di$" indicates excluding word i in document d from the sum. It is convenient to write this distribution in vector form:

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

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

where the product $\mathbf{a}_w\mathbf{b}_d/\mathbf{c}$ is taken element-wise, and $\mathbf{p}$ is normalized to sum to one. The initialization of $\mathbf{z}$ is arbitrary and typically chosen at random by, for example, uniform sampling. Pseudocode for the collapsed Gibbs sampling algorithm for LDA is listed in Algorithm 1.



(a) (b)

Fig. 1. (a) 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. (b) AD-LDA partitions the documents across processors; our modification also divides the words into non-overlapping blocks. Processors sample blocks A1,B1,C1 asynchronously in parallel, re-synchronize, then continue to blocks A2,B2,C2.

Algorithm 1: Collapsed Gibbs sampling for LDA.

```
initialize z at random, e.g., $z_{di} \sim \text{Multinomial}(1/T)$;
$\mathbf{a}_{dt} = |\{i : z_{di} = t\}| + \alpha$;
$\mathbf{b}_{wt} = |\{(d,i) : x_{di} = w, z_{di} = t\}| + \beta$;
$\mathbf{c}_t = |\{(d,i) : z_{di} = t\}| + |W|\beta$;
repeat
    forall $d \in D$, $i \in \{1 \ldots N_d\}$ do
        $t \leftarrow z_{di}$; $w \leftarrow x_{di}$; $\mathbf{a}_{dt}$--; $\mathbf{b}_{wt}$--; $\mathbf{c}_t$--;
        $t \sim \text{Multinomial}(\mathbf{a}_d\,\mathbf{b}_w\,/\,\mathbf{c})$;
        $z_{di} \leftarrow t$; $\mathbf{a}_{dt}$++; $\mathbf{b}_{wt}$++; $\mathbf{c}_t$++;
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 become extremely slow, and require days or even months of CPU time [7]. For example, learning $T = 1000$ topics for the $|D| = 8.2$ million abstracts in MEDLINE would take approximately one hour 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 speed up this process. While parallel processing can reduce topic modeling of huge collections from months to hours, it can also reduce modeling times for small collections from minutes to seconds, potentially enabling real-time use.

[1] For simplicity, we assume symmetric Dirichlet priors with scalar parameters α, β [10], 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 [11].

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

```
   Follow Algorithm 1 initialization (1-4)
5  Partition $D \Rightarrow D_1 \dots D_P$;
6  repeat
7      for $j = 1 \dots P$ in parallel do
8          Copy $\mathbf{b}^j = \mathbf{b}$, $\mathbf{c}^j = \mathbf{c}$;
9          forall $d \in D_j,\ i \in \{1 \dots N_d\}$ do
10             $t \leftarrow z_{di};\ w \leftarrow x_{di};\ \mathbf{a}_{dt}$--; $\mathbf{b}^j_{wt}$--; $\mathbf{c}^j_t$--;
11             $t \sim$ Multinomial($\mathbf{a}_d\, \mathbf{b}^j_w \,/\, \mathbf{c}^j$);
12             $z_{di} \leftarrow t;\ \mathbf{a}_{dt}$++; $\mathbf{b}^j_{wt}$++; $\mathbf{c}^j_t$++;
13     Update $\mathbf{b} = \mathbf{b} + \sum_j (\mathbf{b}^j - \mathbf{b})$, $\mathbf{c} = \mathbf{c} + \sum_j (\mathbf{c}^j - \mathbf{c})$;
14 until convergence ;
```

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

```
   Follow Algorithm 1 initialization (1-4);
5  Partition $D \Rightarrow D_1 \dots D_P$, $W \Rightarrow W_1 \dots W_P$;
6  repeat
7      for $k = 1 \dots P$ in sequence do
8          for $j = 1 \dots P$ in parallel do
9              $\bar{k} = k + j - 1 \mod P$;
10             Copy $\mathbf{c}^j = \mathbf{c}$;
11             forall $d \in D_j,\ i \in \{i : x_{di} \in W_{\bar{k}}\}$ do
12                 $t \leftarrow z_{di};\ w \leftarrow x_{di};\ \mathbf{a}_{dt}$--; $\mathbf{b}_{wt}$--; $\mathbf{c}^j_t$--;
13                 $t \sim$ Multinomial($\mathbf{a}_d\, \mathbf{b}_w \,/\, \mathbf{c}^j$);
14                 $z_{di} \leftarrow t;\ \mathbf{a}_{dt}$++; $\mathbf{b}_{wt}$++; $\mathbf{c}^j_t$++;
15         Update $\mathbf{c} = \mathbf{c} + \sum_j (\mathbf{c}^j - \mathbf{c})$;
16 until convergence ;
```

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 of processors, which for simplicity of presentation we assume are equal.

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 processor with ownership of that document. Thus, partitioning 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 13). 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, but has no guarantees. 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 $D = D_1 \cup \dots \cup D_P$, we also partition the observed words $W = W_1 \cup \dots \cup W_P$. We create P^2 data partitions, each 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 Fig. 1b. 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 on each processor which is not kept up-to-date. This data partitioning was also independently proposed by Yan et al. [13] to reduce potential read/write conflicts on a GPU. However, we show that this modification is key to being able to bound the approximation error inherent in AD-LDA.

In terms of approximation error, our partitioning's reduced number of shared resources has two significant advantages. The first is that $\mathbf{c}$ (or N_t) represents the total number of words allocated to each topic. This is a "bulk" quantity, and likely to be relatively stable within any given parallel for-loop. If $\mathbf{c}$ were constant, i.e., were not changed by each processor, our sampler would be exact. Intuitively, the more stable $\mathbf{c}$, the better our approximation; we make this precise in Section 4. The second point is that $\mathbf{c}$ is relatively low-dimensional (T values), making it feasible to save and re-examine it later for a retrospective evaluation of stability.

4 Analyzing the Sampling Error

Our version of AD-LDA has only a limited number of sources of approximation error, specifically the differences in the topic counts N_t being updated locally at each processor. Using a particular measure of distributional error, we can efficiently track and bound the error 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. Ideally, we might prefer to measure the difference between the distributions over joint topic assignments, $p(\mathbf{z})$, produced by the two algorithms, and in particular their stationary distributions at convergence. However, since $p(\mathbf{z})$ has size exponential in the number of tokens, reasoning directly about $p(\mathbf{z})$ is intractable (hence the use of Gibbs sampling for inference). Instead, as Gibbs sampling progresses towards the stationary distribution using small, tractable distributions over one variable at a time, we measure the differences incurred in each of these incremental steps. However, this per-sample error is not easy to directly relate to the joint distribution error for two reasons. First, it can potentially accumulate over samples, even super-linearly since errors may compound one another. Second and conversely, the joint error will be decreased at each step by the progress of Gibbs sampling towards the stationary distribution; however, the rate of this decrease (the *mixing rate* of the Gibbs sampler) is difficult to quantify accurately in practice. These two factors will balance one another to produce the total error between the joint distributions.

While our on-line measure does not capture the possible accumulation of errors, we argue that it remains a useful assessment of quality. Intuitively, it treats the distributed computation as additional "noise" being input into the Gibbs sampler, and allows us to measure the magnitude of the noise during a distributed execution. This view places the error of distributing the calculation on the same footing as we might consider, for example, a round-off error in floating point calculations or non-uniformity in a random number generator, either of which will have similar per-step effects on the Gibbs sampling process. Because our error values can be interpreted as a bound on a probability, it has a natural scale for interpreting its magnitude; a value of 10^{-4}, for example, means that on average we draw 99.99% of the samples correctly.

Providing some measure of error is important to understanding the trade-offs inherent in AD-LDA. AD-LDA as originally developed provides *no* feedback on quality, and can only be assessed anecdotally by also running a sequentially computed model [7]. As this is impractical in general, we have no way to know how well AD-LDA will do on any new data set. An error bound can be used to assure the user, for example, that the magnitude of error is similar to that observed previously.

4.1 Hilbert's Projective Metric

We use a measure of error between two vectors called Hilbert's projective metric [14]. This metric has been successfully applied to analyze approximations in a probabilistic inference algorithm called *belief propagation*; it was independently developed in [15] (there termed the *dynamic range*) for the purpose of analyzing the stability of belief propagation to small perturbations.

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

$$d(\mathbf{v}, \hat{\mathbf{v}}) = \max_{t,t'} \left[\log(\mathbf{v}_t/\hat{\mathbf{v}}_t) - \log(\mathbf{v}_{t'}/\hat{\mathbf{v}}_{t'})\right] \tag{1}$$

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

$$d(\lambda\mathbf{v}, \lambda'\hat{\mathbf{v}}) = d(\mathbf{v}, \hat{\mathbf{v}}) \tag{2}$$

Proof: Follows readily from the definition (1). □

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

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

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

Proof: Again follows directly from the definition (1). □

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 $\hat{\mathbf{v}}$:

$$d(\mathbf{v} + \mathbf{h}, \hat{\mathbf{v}} + \mathbf{h}) \le d(\mathbf{v}, \hat{\mathbf{v}}) \tag{4}$$

Proof: Since $\sum_t \mathbf{v}_t = \sum_t \hat{\mathbf{v}}_t$, by the mean value theorem,

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

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 relate $d(\mathbf{v}, \hat{\mathbf{v}})$ to more traditional norms, particularly 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

$$\|\mathbf{v} - \hat{\mathbf{v}}\|_1 = \sum_t |\mathbf{v}_t - \hat{\mathbf{v}}_t| \le |1 - e^\epsilon| = \epsilon + O(\epsilon^2) \tag{5}$$

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

$$\max_E \Big| \sum_{t \in E} \mathbf{v}_t - \sum_{t \in E} \hat{\mathbf{v}}_t \Big| \le \frac{1}{2}|1 - e^\epsilon| = \frac{1}{2}\epsilon + O(\epsilon^2) \tag{6}$$

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 (6) then follows from Scheffé's identity [16].

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

□

Although we use (6) to interpret the projective error, it is worth noting that the projective metric measures error on $\log \mathbf{v}$ and so is more sensitive to changes in small values than large ones. Therefore a given value of ϵ may correspond to an $\epsilon/2$ change between $\mathbf{v}_t \approx \mathbf{v}_{t'} \approx .5$, or a much smaller change in probability to an entry of $\mathbf{v}$ with less probability. This difference can make our interpretation of the bound through (6) somewhat pessimistic, particularly when some entries of $\mathbf{v}$ are very small, but empirically it often appears reasonably tight (see Sec. 5).

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 $\mathbf{c}$ used at each processor, the processor *also* updates the true global count $\mathbf{c}$. We design our algorithm to assess the error between the parallel and sequential distributions at each step of the algorithm, *given* the samples 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 accumulation of errors in the distribution; see the discussion at the beginning of 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. Comparison to $\mathbf{c}$ must be retrospective; it can take place only after the preceding processes have finished. However, the distributions 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

Algorithm 4: Bounding errors in AD-LDA.

```
   Follow Algorithm 1 initialization (1-4);
5  Partition $D \Rightarrow D_1 \dots D_P$, $W \Rightarrow W_1 \dots W_P$;
6  repeat
7      for $k = 1 \dots P$ in sequence do
8          for $j = 1 \dots P$ in parallel do
9              $\bar{k} = k + j - 1 \mod P$;
10a            $\mathbf{h}^j_t = |\{(d,i) : d \in D_j, x_{di} \in W_k, z_{di} = t\}|$;
10b            $\hat{\mathbf{v}}^j = \mathbf{c} - \mathbf{h}^j$;
11             forall $d \in D_j$, $i \in \{i : x_{di} \in W_{\bar{k}}\}$ do
12                 $t \leftarrow z_{di}$; $w \leftarrow x_{di}$; $\mathbf{a}_{dt}$--; $\mathbf{b}_{wt}$--; $\mathbf{h}^j_t$--;
13                 $t \sim \text{Multinomial}(\mathbf{a}_d\, \mathbf{b}_w \,/\, (\hat{\mathbf{v}}^j + \mathbf{h}^j))$;
14                 $z_{di} \leftarrow t$; $\mathbf{a}_{dt}$++; $\mathbf{b}_{wt}$++; $\mathbf{h}^j_t$++;

15a        Compute $\epsilon_j = d(\ \hat{\mathbf{v}}^j\ ,\ \hat{\mathbf{v}}^j + \sum_{k<j}(\mathbf{h}^k + \hat{\mathbf{v}}^k - \mathbf{c})\ )$;
15b        Update $\mathbf{c} = \mathbf{c} + \sum_j(\mathbf{h}^j + \hat{\mathbf{v}}^j - \mathbf{c})$;
   until convergence ;
```

$\|\mathbf{p} - \hat{\mathbf{p}}\|_1$ 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. 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 $\mathbf{c}$ that 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:

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

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 $\mathbf{a}_d$, $\mathbf{b}_w$,

$$\begin{aligned} d(\hat{\mathbf{v}}^j\ ,\ \mathbf{v}) &\geq d(\hat{\mathbf{v}}^j + \mathbf{h}^j\ ,\ \mathbf{v} + \mathbf{h}^j\ ) && \text{by (4)} \\ &= d(1/(\hat{\mathbf{v}}^j + \mathbf{h}^j)\ ,\ 1/(\mathbf{v} + \mathbf{h}^j)\ ) && \text{by (1)} \\ &= d(\mathbf{a}_d \mathbf{b}_w\,/(\hat{\mathbf{v}}^j + \mathbf{h}^j)\ ,\ \mathbf{a}_d \mathbf{b}_w\,/(\mathbf{v} + \mathbf{h}^j)\ ) && \text{by (3)} \\ &= d(\hat{\mathbf{p}}^j\ ,\ \mathbf{p}\ ) && \text{by (2)} \end{aligned}$$

Applying Theorem 4 completes the proof. □

Moreover, to track ϵ^j for each processor requires only $P \cdot T$ additional storage and sequential work at re-synchronization. In experiments (Sec. 5) we find values of ϵ ranging from 10^{-1} to 10^{-4}, depending on the data and parameter settings.

4.3 Approximate Scaling Analysis

In this section we sketch out 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 emphasize that while some of the assumptions of this scaling analysis may be unrealistic in practice, it will provide an idea of how the approximation error might depend on our choices, which we verify experimentally in Section 5.

Suppose that the random variation of our topic count vector $\mathbf{c}$ is centered around $N\omega$ for some normalized vector ω, and take $\mathbf{c}^j = N\omega$ for each j. Let us assume for the sake of approximation that the topic into which each sample falls is i.i.d. and distributed according to probability mass function ω, and note that for processor i, the sequential version of the algorithm will have processed approximately $K_i = N(i-1)/P^2$ data since the last resynchronization step. Then, the sequentially sampled value of $\mathbf{c}$ for any given parallel iteration is approximately normal, with mean $K_i\omega_t$ and covariance given by $\Sigma_{tt} = K_i\omega_t(1 - \omega_t)$ and $\Sigma_{tt'} = -K_i\omega_t\omega_{t'}$ for $t \neq t'$.

The projective metric bound consists of a series of maxima over the stochastic realization (samples) produced in each of the P partitions, and over a topic pair (t, t') within the projective metric definition itself. We will approximate these maxima using simple plug-in estimates taken from the samples' expected behavior. In particular we take $K = K_P \lesssim N/P$, and select the coordinate t with the largest variance, $\arg\max_t \sigma_t^2 = \omega_t(1 - \omega_t)$. Finally, we substitute the two-sigma distance in coordinate t as a reference "maximum" value which might be observed in practice. This gives a projective metric comparision between the vector $N\omega$ and $N\omega \pm 2\sqrt{K\sigma^2}$, or

$$\begin{aligned} d(\hat{p}\ ,\ p) &\approx \max_t \log\left(\frac{N\omega_t + 2\sqrt{K\sigma_t^2}}{N\omega_t}\,\frac{N\omega_t}{N\omega_t - 2\sqrt{K\sigma_t^2}}\right) \\ &\approx 4\max_t \sqrt{\frac{1}{NP\omega_t}}. \end{aligned}$$

If the topic counts are approximately uniform, $\omega_t \approx 1/T$, this expression simplifies to

$$\approx 4\sqrt{\frac{T}{NP}}.$$

Thus, this analysis suggests that the error 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. The general form of this approximation is intuitive, since we expect more potential sensitivity when the topic count vector is small (N small or T large), and increasing P increases the rate of synchronization among processes and reduces the number of data being processed in parallel at any one time. We shall see in Section 5 that these approximations are reasonably accurate in practice.

5 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 [17], whose relative sizes (in terms of number of documents, unique words, and total number of words in the

Data Set	$\lvert D\rvert$	$\lvert W\rvert$	N	$N/\lvert D\rvert$
KOS	3,430	6,906	467,714	136
NIPS	1,500	12,419	1,932,365	1288
Enron	39,861	28,102	6,412,172	160
MEDLINE	8,200,000	141,043	737,869,083	90

Fig. 2. Bag-of-words text data sets used in the experiments and their relative sizes, available from [17].

corpus) are listed in Fig. 2. The data sets consist of blog entries from dailykos.com (KOS), full papers from the NIPS conference (NIPS), publicly released emails from the Enron trial (Enron), and abstracts from the MEDLINE database (MEDLINE). Standard processing was used to create the bag-of-words data from the original text, with minor differences across the four data sets. For all data sets except MEDLINE, text was made lowercase and punctuation characters replaced by whitespace, then text was split on whitespace. We removed frequently occurring stopwords (such as ’the’), terms that occurred fewer than ten times in the corpus, and terms shorter than three characters. Stemming or lemmatization were considered unnecessary due to the large volume of data. MEDLINE was processed so as to preserve various multi-word-expressions such as names of diseases, conditions, chemicals, and therapies (e.g., “hepatitis-b” or “x-ray”).

On the largest of these (MEDLINE), for $T = 1000$ and $P = 16$, we achieve $\approx 85\%$ parallel efficiency on 8 cores (a speedup of ≈ 6.8), with error bound $\approx 4 \times 10^{-4}$. However, MEDLINE is prohibitively large for running multiple experiments at different settings, and for running single-threaded executions for comparison. For this reason, subsequent experiments focus on the three smaller data sets (KOS, NIPS, and Enron), on which sequential execution is more tractable. Moreover, by running a single-threaded version which simulates the parallel execution of each algorithm, we can track not only our own error bound (which can be computed in a distributed execution) but also the true value of the quantity it bounds, the error between each Gibbs sampling distribution given full versus only local information. The latter value *cannot* be computed by a parallel implementation, but will help give context to our bounds. For interpretability, we compute one-half the L_1 error between the two sampling distributions, $\frac{1}{2}\|\mathbf{p} - \hat{\mathbf{p}}\|_1$, which as discussed in Theorem 4 bounds the probability of an erroneous sample. Following the parameter choices of [7], we select $\alpha = .1$, $\beta = .01$, and $T = 200$ for all three data sets except where noted.

Timing and speed-up. Fig. 3a 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. Our modifications to AD-LDA do not significantly affect its parallel efficiency, since the algorithms differ only in the addition of error tracking steps. Like the original formulation we obtain nearly linear speed-up in the number of cores. Distributed memory implementations of AD-LDA may see less efficiency due to slower communications; see Wang et al. [8] for a study.

Fig. 3b shows timing information for all three data sets as P is increased on an 8-core machine. Up to $P = 8$, the time required decreases approximately linearly (as indicated in Fig. 3a). Efficiency then flattens (all 8 cores are in use), and as P continues to increase, efficiency begins to degrade. For very large P, each of the P^2 partitions contains relatively little data (for example, for the KOS data at $P = 256$, the $P^2 = 65536$ partitions contain on average only ≈ 7 tokens per partition). The synchronization time eventually outweighs the work per partition, reducing efficiency. For larger data sets such as Enron (≈ 100 tokens per partition at $P = 256$) this effect is reduced.

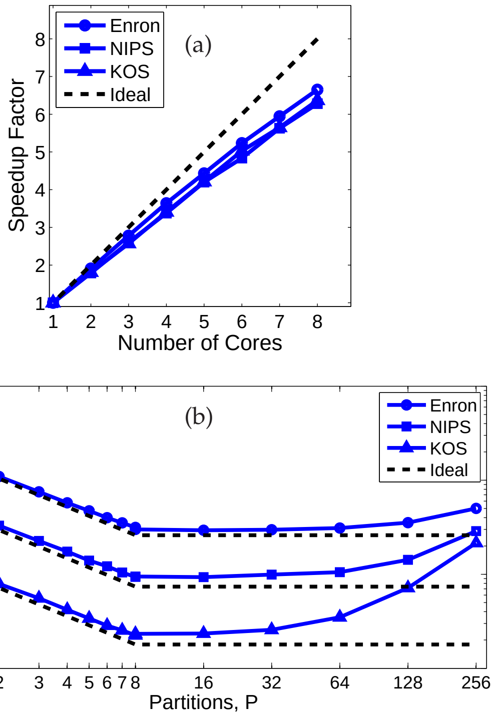


Fig. 3. Parallel efficiency of the modified AD-LDA sampler. (a) Speedup factor (sequential time divided by parallel time) for each data set as the number of cores used increases, and P equal to the number of cores. (b) Computation time using up to 8 cores, as P increases. Dashed lines show ideal (linear) speedup behavior. For moderate P, AD-LDA is quite close to optimal on all three data sets, indicating a high parallel efficiency. As P becomes large and P^2 approaches the number of data, there are too few data per partition and efficiency declines.

Behavior over time. We can also examine the behavior of our error bound as the Gibbs sampler progresses. Fig. 4 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).

Fig. 4a shows that our algorithm’s bound and the true error behave similarly, and in practice are separated by an approximately constant factor over time (here, between about 2 and 3). The sampling error increases initially during the algorithm, as many topic assignments at each processor are changing rapidly. The error then begins to decline, eventually reaching a steady-state level of fluctuation. Notably the increased error appears to correspond to the period during which perplexity drops most sharply, perhaps because during this period the assignments are changing rapidly as the algorithm moves toward a region of high probability. Similar shapes are also observed for the other data sets but are omitted for space. If we examine the shape of the bound as P increases in Fig. 4b, we see that the peak is attenuated for very high values of P, due to the more frequent re-synchronization.

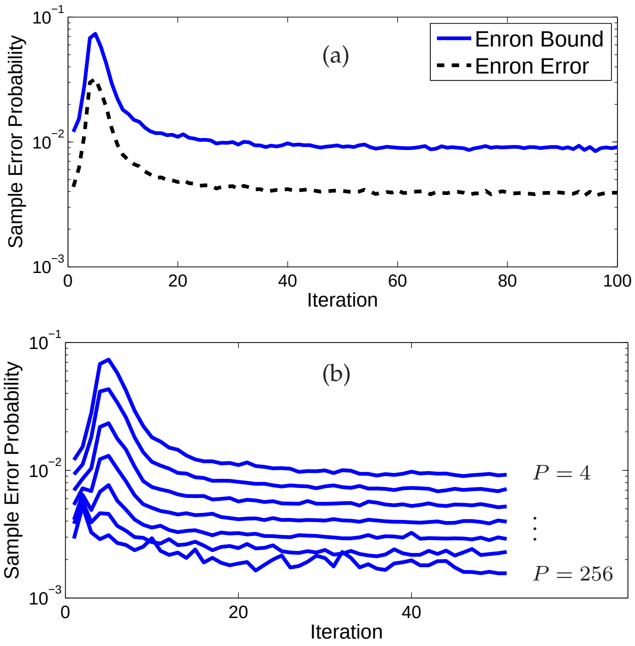


Fig. 4. (a) Sampling error as a function of iteration number on the Enron data set with $P = 4$ partitions. The solid curve shows the error bound computed using our method, while the dashed curve shows the maximum actual error experienced at each iteration (computed using sequential sampling). Errors are high early on, while many assignments are changing, then fall off to a lower steady-state value after the chain has mixed. (b) This effect is attenuated as the number of partitions P is increased, and fewer data are changing between re-synchronization steps.

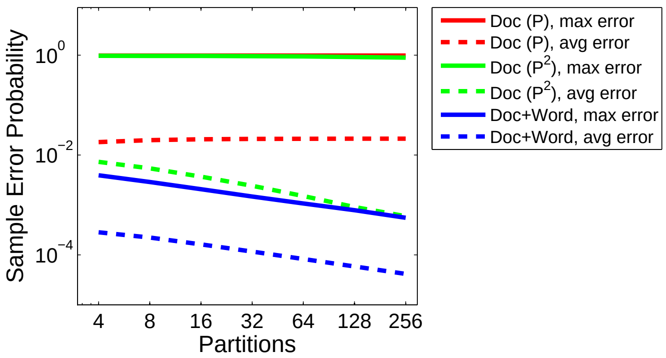


Fig. 5. Approximation quality as a function of partitions P for different versions of AD-LDA, showing maximum (solid) and average (dashed) steady-state error during the execution. The original AD-LDA (Algorithm 2, red) does not resynchronize more often with increasing P and so does not improve. Partitioning into P^2 groups and resychronizing each P (green) improves average error but not maximum error. Partitioning by word-block (Algorithm 3, blue) improves both worst-case and average error by orders of magnitude.

We should not be too concerned by this shape; although errors can be cumulative (see discussion in Section 4), since the algorithm initialization is arbitrary it is fair to imagine "initializing" both sequential and distributed LDA to the AD-LDA state after some number of iterations and then running the algorithm correspondingly longer. This would bypass the period of high error but throw away some iterations during which Gibbs sampling can be considered to have mixed. For this reason, in later comparisons we condense our summary of each run to a scalar, "steady-state" error by averaging the values over the second half of the run.

Effect of word-block coordination. We would also like to know how much the subdivision of data used in Algorithms 3 and 4 affects the accuracy of our sampling distributions. For this comparison, we evaluate the *true* sampling distribution error values observed in a simulated parallel execution, and compare three algorithms: the original AD-LDA (Algorithm 2), which subdivides into P partitions along documents; a version of AD-LDA which subdivides the data into P^2 partitions along documents; and our modified version (Algorithm 3), which subdivides into P^2 partitions along both documents and words. Fig. 5 shows the maximum and average steady-state error observed during a given iteration for each version.

The original AD-LDA (red) has a high maximum error ($\approx$ .97), since for example samples of rare words may be tightly coupled across partitions. Of course, such high-error sampling distributions are rare; the average error is much lower ($\approx 10^{-2}$), but neither improves with increasing P, since regardless of P resynchronization takes place only after all data have been sampled. The same partitioning method but using P^2 partitions and resynchronizing every P (green) has the same maximum error but improves its average error as P increases. Finally, the word-block coordinated version proposed here (blue) has worst-case error two to three orders of magnitude better ($\approx 10^{-3}$), even lower than the average behavior of the other versions; its average-case behavior is better still ($\approx 10^{-4}$).

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 Fig. 6. For comparison, we show both the projective metric bound obtained by our algorithm (solid blue lines) and the maximum L_1 error experienced at each iteration by the sampler (dashed lines). 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 4.3. All plots are logarithmic in both scales for clarity.

Fig. 6a 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 given by our analysis.

Fig. 6b 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, Fig. 6c 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 error appears to scale in a similar way to our anticipated $\sqrt{T}$

behavior, but the error bound increases 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 [9], more research may be be required to provide a tight estimate of the error.

6 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 processors 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 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. In theory, our results are applicable to hierarchical and nonparametric variants of LDA as well [9], [18]. 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. Another possible extension is to analyze asynchronous exchanges [9].

Acknowledgments

The authors thank Arthur Asuncion, Max Welling, and Padhraic Smyth for many useful discussions on LDA and its distributed implementations, as well as the anonymous reviwers for their suggestions.

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] D. Blei and J. Lafferty, “A correlated topic model of *Science*,” *Ann. Appl. Stat.*, vol. 1, no. 1, pp. 17–35, 2007.

[2] J. Sivic, B. Russell, A. Efros, A. Zisserman, and W. Freeman, “Discovering object categories in image collections,” in *Proc. ICCV*, 2005.

[3] T. Rubin and M. Steyvers, “A topic model for movie choices and ratings,” in *Proc. Intl. Conf. Cog. Model.*, Manchester, UK, 2009.

[4] R. Nallapati, W. Cohen, and J. Lafferty, “Parallelized variational em for latent Dirichlet allocation: An experimental evaluation of speed and scalability,” in *Proc. ICDMW*, 2007, pp. 349–354.

[5] M. Welling, Y. W. Teh, and B. Kappen, “Hybrid variational/Gibbs collapsed inference in topic models,” in *Proc. UAI*, 2008, pp. 587–594.

[6] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh, “On smoothing and inference for topic models,” in *Proc. UAI*, June 2009.

[7] D. Newman, A. Asuncion, P. Smyth, and M. Welling, “Distributed inference for latent Dirichlet allocation,” in *Proc. NIPS*, Dec. 2007.

[8] Y. Wang, H. Bai, M. Stanton, W.-Y. Chen, and E. Y. Chang, “PLDA: Parallel latent Dirichlet allocation for large-scale applications,” in *Proc. AAIM*, June 2009.

[9] A. Asuncion, P. Smyth, and M. Welling, “Asynchronous dis-

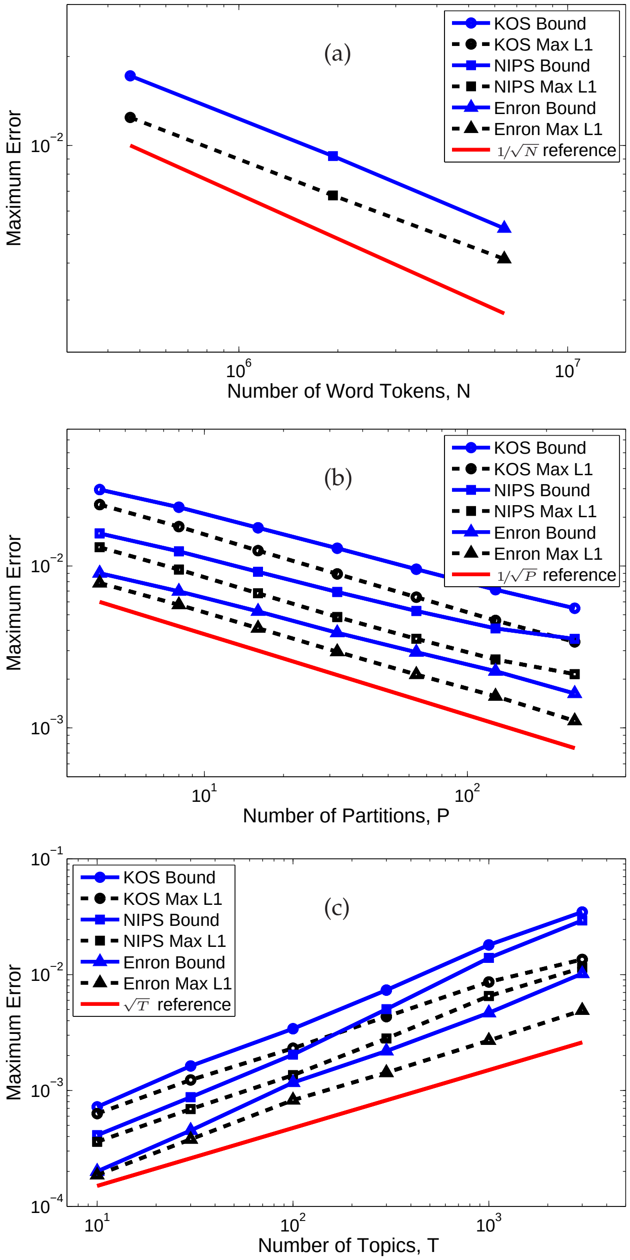


Fig. 6. Error scaling as a function of (a) the size of the data set, N; (b) the number of partitions, P; (c) the number of topics, T. Each plot shows both the projective metric error bound (solid, blue) and maximum L_1 error experienced (dashed, black) for the KOS (circles), NIPS (squares), and Enron (triangle) data sets. Also shown in each plot is a reference line showing the estimated scaling behavior of Section 4.3. In all cases, the estimated behavior appears roughly similar to the observed curves for both the error bound and the true error probability.

tributed learning of topic models," in *Proc. NIPS*, 2009, pp. 81–88.

[10] T. L. Griffiths and M. Steyvers, "Finding scientific topics." *Proc Natl Acad Sci*, vol. 101 Suppl 1, pp. 5228–5235, April 2004.

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

[12] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *JMLR*, vol. 3, pp. 993–1022, 2003.

[13] F. Yan, N. Xu, and Y. Qi, "Parallel inference for latent Dirichlet allocation on graphics processing units," in *NIPS*, Dec. 2009.

[14] P. J. Bushell, "Hilbert's metric and positive contraction mappings in a Banach space," *Arch. Rational Mech. Anal.*, vol. 52, no. 4, pp. 330–338, Dec. 1973.

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

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

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

[18] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei, "Sharing clusters among related groups: Hierarchical Dirichlet processes," in *Proc. NIPS*, 2005, pp. 1385–1392.

Alexander Ihler is an Assistant Professor in the Department of Computer Science at the University of California, Irvine. He received his Ph.D. in Electrical Engineering and Computer Science from MIT in 2005 and a B.S. with honors from Caltech in 1998. His research focuses on machine learning, graphical models, and algorithms for exact and approximate inference, with applications to areas including sensor networks, computer vision, data mining, and computational biology. His many research articles include two best paper awards, at Neural Information Processing Systems (NIPS) for his work on belief propagation and at Information Processing in Sensor Networks (IPSN) for his work on sensor localization.

David Newman is a Research Scientist in the Department of Computer Science at the University of California, Irvine. His research focuses on theory and application of topic models and related text mining and machine learning techniques. Newman's work combines theoretical advances with practical applications to improve the way people find and discover information. Newman received his Ph.D. from Princeton University.