Distributed LDA: Balancing Accuracy and Efficiency on Big Data

Greg Meyer

MSc Data Science, Department of Statistics, London School of Economics and Political Science, Columbia House, Houghton Street, London, WC2A 2AE, United Kingdom

Abstract

Topic models based on Latent Dirichlet Allocation (LDA) are used in a range of tasks including document navigation and trend analysis. The challenge of approximating the true posterior distribution that best models latent topics underlying a corpus remains an open research area. Moreover, with the recent explosion of data, the algorithm's ability to scale up to corpora with millions of documents is an integral requirement. The primary goal of this paper is to find an algorithm that achieves high model accuracy and runs within an acceptable timeframe where distributed computing will be leveraged to achieve the latter. In pursuit of the above goal, this paper explores a variety of algorithms used to build LDA models. Through a series of experiments weighing up computational efficiency against model accuracy, I conclude that while Gibbs sampling is not a viable option in a non-distributed setting, the efficiency gains from implementing the algorithm using multiple machines provides the best mix of model accuracy and computational efficiency.

1 Introduction

Topic modelling has fast developed into an indispensable tool used in scientific studies, trend analysis and document searches. Latent Dirichlet Allocation (LDA), a highly successful topic modelling technique, is a generative probabilistic framework first proposed by Blei et al. [2] to uncover topics representing documents within a corpus. LDA is premised on the assumption that documents have latent or unobserved topics in the form of a multinomial distribution over words. This is typically visualised as the top-N words that best emulate the contents of the topic.

Because LDA relies on a Bayesian framework, finding the true posterior of relevant parameters is the primary task. Unfortunately, calculating this posterior is not trivial - most modern LDA methods use variational inference algorithms [2] or Markov Chain Monte Carlo (MCMC) methods like collapsed Gibbs sampling [7] to approximate the true posterior distribution with as little error as possible. The classic trade-off faced by text analysts is the weigh-up between fast results and accuracy. Variational inference algorithms tend to give results in a fraction of the time of MCMC methods, but with lower accuracy.

The explosion of data in recent years has introduced another obstacle further complicating LDA analysis. With corpora sizes in the order of millions, LDA algorithms can no longer run on a single machine. This challenge is best described by the current coronavirus crisis where specialists restrict analysis to corpora of less than 10000 documents due to computational constraints. Evidently, research based on a subset of the coronavirus data is bound to produce less informative results - scaling up LDA analysis is of central importance. This paper explores implementing selected algorithms in Spark and Scala to leverage the power of parallel computing in the hope of achieving high accuracy coupled with fast results.

In particular, this paper addresses the challenge of applying LDA analysis to a dataset with more than 100 million words. First, a brief introduction to LDA is given followed by an explanation of algorithms used in the analysis. The paper proceeds to fit models using different algorithms with a single machine as well as a distributed architecture, paying close attention to the time taken to run analysis and model accuracy. Finally, I include a closing section testing how far I can scale up data volume while obtaining results within a reasonable timeframe.

2 Overview of LDA

LDA is a topic modelling algorithm used to cluster documents in a corpus into a fixed number of topics. The main idea behind LDA is that each document can be described by a distribution of topics (document-topic) and each topic can be described by a distribution of words (topic-word). For example, if we have three sentences:

1. I love traveling through South America
2. My favourite form of dance is salsa
3. I'd love a trip to Argentina to learn how to tango

LDA would classify sentence 1 as 100% topic 1, sentence 2 as 100% topic 2 and sentence 3 as 50% topic 1 and 50% topic 2. LDA would also give us a breakdown of which words represent the topics best:

Word	Topic 1	Topic 2
traveling	40%	3%
america	20%	5%
dance	5%	35%
tango	2%	25%

Table 1. Displays the distribution of topics over words. Only four words in the corpus have been included in the above table.

In essence, there are two stages to the LDA process. First, we select the number of topics/clusters. Second, we cycle through all the words in the corpus and randomly assign words to one topic. This allows us to calculate the document-topic and topic-word distributions. Because this initial ran-

dom allocation will not lead to meaningful topics, topics are iteratively updated. Specifics on how this is achieved are outlined in Sections 4 and 5. The problem at hand can be explained from a Bayesian context. Before proceeding, I define notation in Table 2 to be used throughout the paper.

Notation	Meaning
M	total number of documents in the corpus
k	number of pre-specified topics
θ_d	distribution of topics for document d
$z_{j,d}$	RV of topics for word j in document d
β_t	distribution of words for topic t
$w_{d,j}$	j^{th} word in the d^{th} document
w	represents any given word in the corpus
W	total number of words in the vocabulary
D	all the data in the corpus
α_d	parameter vector for each document d
η_t	parameter vector for each document t
d	document d where $d \in 1, ..., M$
t	topic t where $t \in 1, ..., k$

Table 2. Notation and corresponding meanings used throughout the paper

In the Bayesian framework, the problem is set up as follows:

$\theta_d \sim Dirichlet(\alpha)$

$\beta_t \sim Dirichlet(\eta)$

$z_{j,d} \sim Categorical(\theta_d)$

$w_{d,j} \sim Categorical(\beta_{z_{dj}})$

We aim to find the following posterior distribution:

$$P(\theta_{1:M}, z_{1:M}, \beta_{1:k}|D; \alpha_{1:M}, \eta_{1:k})$$

The above probability is simply the joint posterior of θ_d, $z_{w,d}$ and β_t $\forall d, w, t$ given all the data in the corpus (D) with parameters α_d and η_t. For experimental purposes, I assume $\alpha_d = \alpha$ and $\eta_t = \eta$ throughout the paper.

While the theory behind LDA is convincing, Bayesian statisticians face the problem that calculating the above posterior probability is not a trivial task.
Currently, most methods used in practice used to approximate the above posterior fall into two categories:

1. Variational Inference (VI)
2. Markov Chain Monte Carlo (MCMC)

MCMC methods have the added advantage of allowing the user to control approximation error thereby improving accuracy, however, this comes with the added cost of increased computational complexity.
Another challenge facing LDA methods is the rapidly increasing size of corpora. For this reason, distributed systems play a pivotal role in running LDA analysis.
The sections that follow address the above two problems by running selected LDA algorithms on the CORD-19 dataset [5] using distributed platforms. The trade-offs between accuracy and computational cost are compared.

3 Dataset

The dataset used for analysis is the COVID-19 Open Research Dataset (CORD-19) which consists of scholarly article about COVID-19 and the coronavirus family of viruses for use by the global research community [5]. The data was collected and cleaned on 08/04/2020. The corpus consisted of 47926 documents on this date. Due to duplicate documents and some faulty parses, the cleaned dataset resulted in 36236 documents with 100 414 583 total words and a vocabulary size of 424507.
The data cleaning process involved extracting relevant information from raw json files and subsequent conversion into Spark DataFrames. Next, the abstract and main text for each document is processed before performing LDA analysis. This includes tokenisation, removal of stopwords and lemmatisation. Finally, the corpus is converted to a series of sparse vectors for EM and online analysis, and to a text file for Gibbs analysis.

4 Spark Implementations

Spark's current implementation of LDA allows the choice between two forms of VI algorithms used to approximate the posterior distribution. These algorithms are Expectation-Maximisation (EM) and Online Variational Inference. I first give a brief outline of variational inference and then proceed to discuss the specifics of each algorithm.
VI can be considered an optimisation method. The procedure tries to identify the best approximation to the actual posterior by considering a parameterised family of distributions. It proceeds to optimise these parameters until this family of distributions best represents the posterior with respect to some error measure. One typically uses Kullback-Leibler (KL) divergence as the error measure due to its insensitivity to normalisation factors. This measure can be thought of as the distance between two distributions. Mathematically, VI tries to find w' in order to minimise the error E:

$$\omega' = argmin_{w \in \Omega} E(f_\omega, \pi) \tag{1}$$

$F_\Omega = f_\omega; \omega \in \Omega$ (approximating family of distributions)
Ω is the set of all possible parameters
π is the true posterior

Hence, if we tailor equation (1) to an LDA setting using KL divergence as the error we get:

$$\gamma', \phi', \lambda' = argmin_{\gamma,\phi,\lambda} KL(\mathrm{q}(\theta, z, \beta|\gamma, \phi, \lambda)||p(\theta, z, \beta|D; \alpha, \eta)) \tag{2}$$

q is the approximating family of distributions
p is the true posterior
γ, ϕ, λ are variational parameters

The algorithms I analyse in the upcoming sections focus on finding the variational parameters that minimise the KL di-

vergence (usually done by maximising the ELBO) in an attempt to obtain the closest fully defined approximation to the true posterior.
This formulation allows one to sidestep the problem of having to normalise the posterior, which would make the calculation intractable. It is important to note that VI methods are model-based and therefore result in bias. This can be controlled by the approximating family of distributions - complex families typically lead to lower bias and higher variance. The opposite is true for simple families [12]. In theory, it is this bias that makes VI models less accurate than methods like MCMC. The algorithms below are described in a way that follow their implementation in Spark. Details can be found in the Spark GitHub repository [11].

A. Expectation-Maximisation. The first option provided by Spark is the Map Estimation EM algorithm. Please refer to Asuncion et al. [1] for further mathematical detail. An outline is included below.

Notation:

N_{wtd}: how many times a word w which is assigned to topic t, appears in document d

N_* where $*$ is missing one or more subscripts (w, t or d): the count is summed over the missing subscript (eg. $N_{wt} = \Sigma_d N_{wtd}$)

$\gamma_{wdt} = P(z_{w,d} = t)$: probability of word w in document d having topic t

W: total number of words in the vocabulary

Algorithm:

1. Initialise hyperparameters α and η - if no initial values are given, α will default to a uniformly drawn value from $\frac{50}{K+1}$ and β defaults to 1.1

2. Randomly initialise N_{wt} and N_{td}

3. E Step: Compute $\gamma_{wdt}\ \forall w, d, t$

$$\gamma_{wdt} = \frac{(N_{wt}+\eta-1)(N_{td}+\alpha-1)}{N_t+W\eta-W} \tag{3}$$

4. M Step: Update variational parameters:

$N_{wt} = \Sigma_d N_{wd}\gamma_{wdt}$ (word update)
$N_{td} = \Sigma_w N_{wd}\gamma_{wdt}$ (document update)
$N_t = \Sigma_w N_{wt}$ (word update)
$N_d = \Sigma_t N_{td}$ (document update)

5. Repeat steps 3 and 4 until convergence is reached

6. Finally, using the converged variational parameters, Spark will return distribution values using the below estimates:

$\hat{\beta}_{wt} = \frac{N_{wt}+\eta-1}{N_t+W\eta-W}$ (probability topic t contains word w)

$\hat{\theta}_{td} = \frac{N_{td}+\alpha-1}{N_d+K\alpha-K}$ (probability document d contains topic t)

Relating this algorithm back to VI, the EM algorithm assumes q follows the mean field approximation which states q can be factorised as follows:

$$q(\theta, z, \beta, |\gamma, \phi, \lambda) = \prod_d q(\theta_{\cdot,d}|\gamma) \prod_{j,d} q(z_{j,d}|\phi) \prod_t q(\beta_{\cdot,t}|\lambda) \tag{4}$$

However, the important point to note, is that the EM algorithm assumes each variational function above is a point estimate (these are the N_s in the algorithm). Hence, the N_s can be viewed as variational parameters to be optimised in order to obtain the approximating distribution. This concept makes the EM algorithm a special case of VI.
Another observation regarding this algorithm is that variational parameters are only updated after each iteration, which requires a pass through the entire corpus.

B. Online Variational Inference. The second option provided by Spark is the online variational Bayes algorithm. Please refer to Hoffman et al. [8] for further mathematical detail. An outline is included below.

Algorithm (adopting notation from Table 2 and the variational paramters from equation (2)):

1. Initialise hyperparameters α and η - if no initial values are given, α and β default to a uniformly drawn value from $\frac{1}{K}$

2. Define $\rho = (\tau_0 + d)^{-\kappa}$ where $\tau_0 \geq 0$

3. Randomly initialise λ

4. E Step: Calculate ϕ_{dwt} and γ_{dt}

Set $\gamma_{dt} = 1$

$\phi_{dwt} \propto \exp[\mathbb{E}_q[\log\theta_{dt}] + \mathbb{E}_q[\log\beta_{tw}]]$

$\gamma_{dt} = \alpha + \Sigma_w n_{dw}\phi_{dwt} n_w$ where n_{dw} is the number of occurrences of word w in the document d

Repeat the calculations of ϕ_{dwt} and γ_{dt} until $\frac{1}{K}|\Delta\gamma_t| < \epsilon$ i.e. the change in γ_{dt} between iterations is very small

5. M Step: Compute λ (2 steps below)

Compute $\widetilde{\lambda}_{tw} = \eta + M n_{dw}\phi_{dwt}$

Set $\lambda = (1-\rho)\lambda + \rho\widetilde{\lambda}$

6. Repeat steps 4 and 5 over all documents in the corpus

Unlike the EM algorithm, the online optimiser processes a subset of the corpus on each iteration, and updates the variational parameters accordingly. Because the parameters are updated several times during an entire pass through the corpus, fewer full passes need to be made. This allows for increased computational efficiency and more efficient scaling to large datasets.
While VI algorithms provide accurate results in a relatively short space of time, theory suggests that better results can be obtained with MCMC methods [7]. The following section explores Gibbs sampling for LDA as a technique to improve model accuracy.

5 Gibbs Sampling

Griffiths & Steyvers propose that MCMC methods converge to a ground-truth faster than VI [7]. I explore these claims in a distributed setting. Before doing so I provide a brief description of Gibbs sampling and its link to LDA analysis.
In contrast to the VI methods implemented in Spark which revolve around optimisation, Gibbs sampling is a sampling method first described by Stuart and Donald Geman [6]. Intuitively, Gibbs sampling starts by assuming that θ and β are known and slowly updates these matrices in a way that maximises the likelihood function. This is done by iteratively drawing samples from each variable conditioned on the latest values of all other sampled variables. These sample distributions are Markov chains that are constructed in a way that if enough samples are drawn, we reach a stationary distribution that is the true posterior. While this convergence is theoretically guaranteed, there is no way of knowing how many iterations are required to reach the stationary distribution/true posterior. More formally, the Gibbs sampling process can be described as follows:

1. Initialise the vector $X^{(0)}$ where the superscript is the iteration and $X^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_n^{(i)})$

2. Sample $X^{(i+1)} = (x_1^{(i+1)}, x_2^{(i+1)}, ..., x_n^{(i+1)})$ where samples are taken for each x_j from its distribution conditioned on all the other x_s. The catch is that one must condition the x_j being sampled on all previous x_{j_s} from the same iteration. For example, to find $x_j^{(i+1)}$ we sample from:

$$P(x_j^{(i+1)} | x_1^{(i+1)}, ..., x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, ..., , x_n^{(i)})$$

3. Repeat step 2 for all j - samples obtained from the last iteration should approximate the joint distribution with low error. However, it can take a long time to reach this point.

Applying Gibbs sampling to an LDA setting, we draw from the three full conditional posteriors below:

$$P(z_{d,j} = t|\cdot) \propto \theta_{d,t}\beta_{t,w_{dj}} \tag{5}$$

$$P(\theta_{d,t}|\cdot) = Dir\left(\alpha + \Sigma_j \mathbb{I}(z_{d,j} = t)\right) \tag{6}$$

$$P(\beta_{t,w}|\cdot) = Dir\left(\eta + \Sigma_d \Sigma_j \mathbb{I}(w_{d,j} = w, z_{d,j} = t)\right) \tag{7}$$

The algorithm is implemented using these steps:

1. Sample $z_{d,j}$ $\forall d, j$ using (5)

2. Sample $\theta_{d,t}$ $\forall d, t$ using (6) and the z_s from step 1

3. Sample $\beta_{t,w}$ $\forall t, w$ using (7) and the z_s from step 1

4. Iterate over steps 1-3 conditioning on the updated sampled parameters at each iteration

The algorithm described above is a regular Gibbs sampler for LDA. Most research today involves the use of a collapsed Gibbs sampler (CGS) which displays better convergence properties in some cases. Instead of taking samples for each θ, β and z conditioned on all other variables, CGS integrates out/collapses θ and β and only samples z values. Whilst this requires fewer samples, it leads to dependencies among the z_s. In practice, this results in the need to update a counting matrix after each sample is taken. Contrastingly, the original Gibbs sampling algorithm described in this section allows us to sample z values in parallel thereby leveraging distributed computing. For this reason, I use the original Gibbs sampler for LDA analysis.

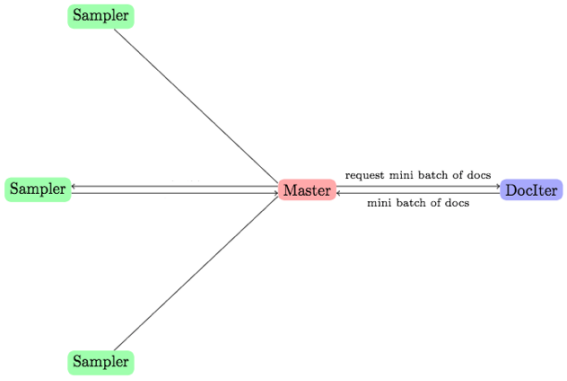

Fig. 1. System architecture of the distributed Gibbs sampler for LDA.

As mentioned in the Jupyter Notebook, I ran into the problem of attempting to run non-approximated CGS on a distributed platform. The individual updates per iteration are not suited to distributed computing. The subsequent Gibbs sampler I implement uses Scala map-reduce functions adapted to analyse the coronavirus corpus. The implementation is best described by the worker master architecture in Figure 1. DocIter sends batches of the corpus to the master node. This is done in case the entire corpus cannot be held in memory. The master then sends documents, θ and β values to the worker nodes which extract samples and return these to the master node. The master node performs parameter updates.
For small datasets, communications between workers and the master node is expensive relative to the sampling cost. However, it is expected that speed gains will become more significant with increasing dataset size where communication between nodes forms a smaller proportion of total runtime.

6 Methods and Evaluation Metrics

This section outlines experiments performed and metrics used to quantify model accuracy from these experiments. I run LDA analysis on the dataset [5] using 5 algorithms: the EM algorithm, distributed and single node versions of the online algorithm as well as distributed and single node variants of the Gibbs sampling algorithm (the collapsed and original variants). Ideally, one would perform experiments using hyperparameters that have been tuned with the aid of a grid search. However, due to limited GCP credits and the computational cost of hyperparameter tuning, all analysis is performed using fixed, pre-specified parameters. I set the num-

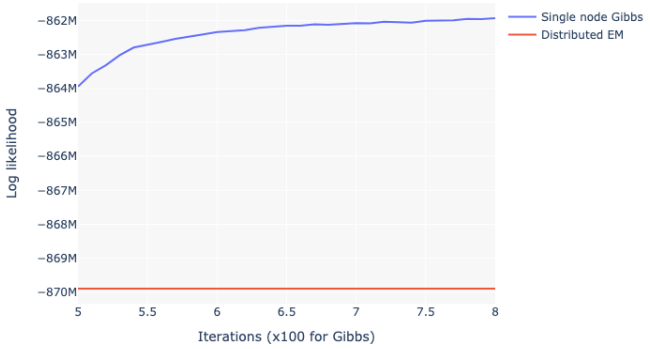

Fig. 2. Log likelihood vs iterations for VI and MCMC methods

ber of inferred topics to 10, a commonly used value in LDA literature. Preliminary experiments using selected iterations suggests VI methods' likelihoods do not increase after 5 iterations. With regard to MCMC algorithms, the likelihood continues to increase for more iterations than one can perform, however, these increases flatten out around 700 iterations for collapsed Gibbs and 50 iterations for original Gibbs. These observations can be seen in Figure 2. An important side point is that each iteration for CGS does not represent an entire pass through the data - it represents one sampling iteration. Convergence properties for distributed online and Gibbs algorithms have not been included in the figure for visualisation purposes. However, analysis in the Jupyter notebook suggests that the online algorithm also converges after 5 iterations and the distributed Gibbs after 50. The reason for differing convergences between single node Gibbs and the distributed version is that they use different algorithms as described in Section 5. A summary of hyperparameters used in experiments is provided in Table 3.

Algorithm	Iterations	Topics
Distributed EM	5	10
Single node online	5	10
Distributed online	5	10
Single node Gibbs (Collapsed GS)	700	10
Distributed Gibbs (Original GS)	50	10

Table 3. Hyperparameters used for LDA experiments

LDA is a form of clustering and hence classified as unsupervised learning. Unsupervised model evaluation is challenging, particularly in the case of topic models where subjective interpretation plays a role. There are, however, objective metrics that can be used to gain a better idea of model accuracy. This section proceeds to outline two classes of evaluation methods used to compare models from the abovementioned experiments. The results of these metrics applied to the data as well as computational efficiency findings are discussed in Section 7.

A. Perplexity. The first metric is perplexity. Perplexity measures how 'surprised' the model is when seeing documents that were not in the training corpus. A high perplexity suggests the model does not generalise well to the test set and hence we aim for models with low perplexities. I obtain an upper bound for model perplexities where perplexity is calculated as the negative log likelihood divided by the number of words in the test set.

Calculating perplexities on a test set is not a straightforward matter. The reason for this is the full likelihood for the test corpus is an intractable calculation. Hence, we first need to estimate this distribution before calculating the likelihood. Implementing this in Spark is easy - I just use the logPerplexity function. In order to implement this estimation for Gibbs sampling I use the first order version of the mean field approximation method described by Buntine in 3.3 [3]. Finally, I calculate the log perplexity with this estimated document-topic distribution.

B. Topic Coherence . Topic coherence is based on the notion that topics consisting of words with high semantic similarity tend to perform better. For example, the words 'exercise,' 'physical,' and 'athletes' scoring highly in a particular topic indicates that there is a common theme within the topic. Ultimately, all topic coherence measures attempt to assess human interpretability where high scores are considered accurate models. This section outlines two topic coherence measures used to quantify model accuracy in Section 7.

B.1. Pointwise Mutual Information (PMI). PMI is an example of a coherence measure which quantifies the discrepancy between the actual probability of co-occurrence versus the probability of individual occurrence under the assumption of independence. High co-occurrence is indicative of topic coherence. The formula is defined as follows [10]:

$$PMI(w_1, w_2) = \log \frac{p(w_1,w_2)}{p(w_1)p(w_2)}$$

In order to quantify each topic's coherence, I calculate the PMI for each pair of the top 10 words in a given topic. These values are averaged to get the topic coherence score for each topic. Finally, topic PMI scores are averaged to get the coherence score for a particular model. Higher values are indicative of better models.

B.2. Word Intrusion. Proposed by Chang et al. [4], word intrusion is an indirect approach where 'intruder words' are randomly injected into topics and human users are required to identify the intruder words. The idea behind this is that intruder words will be more identifiable in coherent topics and so the interpretability of each topic can be measured by the proportion of humans who correctly identify the intruder. A limitation of this method is that human annotations are required. Lau et al. solve this problem by proposing a method to automatically replicate human annotation [9]. I use this method to replicate the process of a human attempt to predict an intruder word. Model accuracy is evaluated based on the proportion of correct intruder word identifications. A higher accuracy is indicative of highly interpretable topics.

Implementation of all experiments and evaluations can be found in the Jupyter notebook included in this repository.

7 Results and Discussion

This section outlines the results obtained from running the algorithms in Sections 4 and 5, evaluates the models using metrics described in Section 6 and compares their computational speeds. I fit all models using the dataset from Section 3 and the hyperparameters in Table 3.

A. Topic-word distributions. The topic-word distributions for each algorithm can be found in the Appendix. The first metric I use in this section is an informal 'eyeballing' approach which is relevant in unsupervised learning, especially when one has an implicit knowledge of the dataset. It is immediately clear that the EM algorithm produces a non-informative result where all topics are the same.

The remaining topic-word distributions seem informative. Words that should be included as stopwords are still present, but I leave them in the results as the focus is on the distributed computing aspect, and rerunning the LDA algorithm multiple times is expensive. Both online and Gibbs methods separate the corpus into somewhat distinct topics. Topics produced by the Gibbs variants contain ideas that are not found when using the online variants. For example, topics 4 and 10 in Table 8 focus on the zoonotic nature of coronavirus and how to use its origin in finding a cure. Additional topics in Table 8 include the role of modelling in the fight against coronavirus (topic 5) and the risk to the public health system (topic 1). These ideas do not appear in Table 6 (distributed online algorithm). Results from Gibbs sampling seem to be more accurate and informative using the informal 'eyeballing' approach. The next section proceeds to analyse these claims more formally using the objective measures from Section 6.

Regarding the EM algorithm, the log perplexity tends to infinity further indicating the algorithm's poor performance on the large corpus. I believe the reason for this occurrence lies within the fundamental assumption of VI. VI assumes that we can approximate the true posterior with a family of distributions. The accuracy of this approximation relies on the how well the family of distributions is able to mimic the true posterior. As mentioned in Section 4, the approximating distribution for EM is simplified to the product of three point estimates (N_{wt}, N_{td} and N_t). With a corpus the size of the CORD-19 dataset, it is unlikely that the true posterior will be of such rudimentary nature. Hence, this poor approximation of the true posterior leads to poor results. The algorithm runs successfully on small subsets of the dataset which reinforces the idea that the algorithm is not suited to very large corpora. It is possible that this is the reason that Spark has switched to the online algorithm as the default. From this point onwards, I dismiss EM as a viable solution and do not evaluate the model further.

B. Perplexity. This section compares model perplexity. Perplexity is an expensive calculation that needs to run through the entire corpus. For this reason, I calculate the perplexity of one variant of the Gibbs and online algorithms. Different variants of the same algorithm perform similarly. Figure 3 confirms that Gibbs sampling performs better on the test set

Fig. 3. Model perplexity for Gibbs and online algorithms

than the online algorithm. The Gibbs sampling optimiser is less "perplexed" when seeing the test data.

There is, however, some evidence to show that perplexity is not the best metric for qualitative evaluation of topic models and in some cases, is inversely related to human judgement. For this reason, I proceed to evaluate models using topic coherence measures.

C. Topic coherence. This section restricts analysis to a comparison between the distributed variants of each algorithm. Their non-distributed counterparts display a similar behaviour. Results from the metrics described in Section 6B can be found in Figure 4. The PMI results observed are the averages taken over each topic's PMI score. The distributed Gibbs algorithm shows a higher value for PMI suggesting that words within topics frequently occur together. This implies that topics focus on one particular issue and display a high level of coherence.

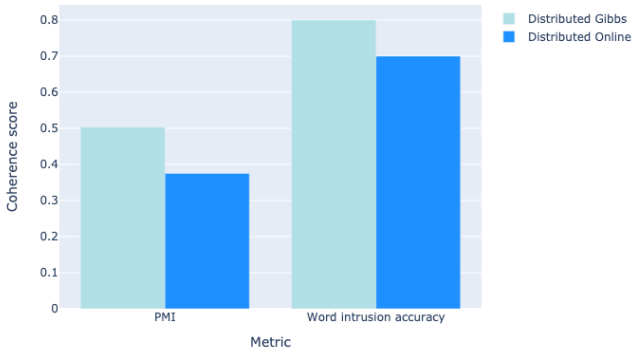

Fig. 4. Topic coherence measures for distributed online and Gibbs algorithms

Moreover, automatic methods identify intruder words with a higher accuracy in the Gibbs model when compared to the online model. In both cases, it is highly likely that one is able to identify when a word does not belong to a topic, however, Gibbs sampling leads to a slightly higher accuracy. This is a desirable feature of the model as it confirms that highly weighted words within topics support a common theme.

Both algorithms perform well on the large corpus (and can be improved with further data cleaning and hyperparameter tuning). While the online algorithm still produces favourable

results, Gibbs sampling seems to give more informative, accurate clusters. With this in mind, it is important to analyse the cost of this increased accuracy. This is explored in the section that follows.

D. Efficiency. I now run LDA on several subsets of the data to determine the computational efficiency of each algorithm using a master worker setup with four workers. The distributed algorithms were run on GCP using n1-standard-4 type machines for the master and workers. I use a single master machine with three worker machines for the distributed results. One must note that each of the algorithms is run for a different number of iterations. While this makes efficiency comparisons difficult, I make the claim that each algorithm runs until the marginal improvements are negligible. This allows me to compare algorithms based on how long they take to reach a model within a small range of the fully converged model. Results showing how the algorithms scale with increasing data volume are shown in Figure 5.

Fig. 5. LDA runtimes across different algorithms

It is clear that distributing computation over multiple workers leads to significant increases in efficiency. The first observation to note, as mentioned at the end of Section 5, is that the efficiency difference between single node Gibbs and distributed Gibbs is insignificant for small corpora sizes. This occurs because communication costs between master and worker nodes are not negligible when corpora are small. However, as the corpus size increases, computational gains from using a distributed framework are large.

Comparing the Gibbs and online algorithms, it is clear that Gibbs sampling is extremely time consuming when run on a single node. However, the gap between runtime of the online and Gibbs algorithms is significantly smaller when both are executed using multiple workers. While using the online algorithm is still more efficient, increased accuracy obtained from using Gibbs sampling would justify a marginally longer runtime. This make Gibbs sampling an attractive option when running distributed LDA analysis.

8 Scaling up

This section explores the computational efficiency of online and Gibbs LDA by fitting models on one of the largest publicly available reviews datasets. The Amazon review dataset (2018) is available at this link. The data is 128GB (34GB compressed) and consists of over 233 million Amazon reviews and a vocabulary size of 31 087 974. Due to GCP credit constraints and the time it takes to clean the data, I skipped this step and focused on the computational time of running the two algorithms.

Details of how to run each LDA method can be found in the Jupyter notebook. Online LDA took a total time of 4826 seconds with a 4 node master and two 8 node workers. Using the same architecture, Gibbs sampling took 22551 seconds.

It is clear that online LDA is the faster method and better suited to tight deadlines, however, the use of a distributed network allows Gibbs sampling to deliver results within half a day, even on some of the largest datasets.

9 Conclusion

In this paper, I examined various distributed algorithms that evaluate the quality of LDA models at two levels: efficiency and model accuracy. I looked first at an outline of the algorithms and considered how their structure would affect efficiency and accuracy. These algorithms included EM and online methods implemented in Spark as well as the Gibbs sampling method which I implemented in a distributed setting. I proceeded to set out metrics that would be used to quantitatively assess model accuracy.

Firstly, findings show that the EM algorithm does not scale up well to large corpora and so this algorithm is removed as a candidate for the best performer. In terms of model accuracy, results showed that original Gibbs sampling is a viable option for LDA analysis which leads to more accurate models than the online algorithm according to the metrics used in this paper. In terms of efficiency, however, the online algorithm performs significantly better. In the final section I scale up data volume and reach the conclusion that although online LDA is significantly more efficient than Gibbs LDA, distributed systems allow Gibbs sampling to provide results within a reasonable amount of time.

In conclusion, MCMC methods like Gibbs sampling are becoming a more common option with increasing computational power and the ability to distribute work across massive clusters. While past papers suggest that VI and MCMC methods lead to equally accurate results [8], the findings in this paper lead to the conclusion that Gibbs sampling outperforms VI methods, especially on large corpora and is therefore the preferred method provided a distributed framework is available. As the gap between VI and MCMC computational efficiency closes, MCMC algorithms will start to dominate the topic modelling space. Currently, mostly parallel variants of MCMC algorithms are used for LDA analysis. While these approximate algorithms perform with high accuracy, the sequential versions produce the best results. The challenge for future researchers will revolve around solutions to implement these sequential MCMC methods in a distributed framework as well as explore the potential role of GPUs in this area.

References

1

[1] Arthur Asuncion et al. “On smoothing and inference for topic models”. In: *arXiv preprint arXiv:1205.2662* (2012).

[2] David Blei, Andrew Ng, and Michael Jordan. “Latent Dirichlet Allocation”. In: *Journal of Machine Learning Research* 3 (May 2003), pp. 993–1022. DOI: 10.1162/jmlr.2003.3.4-5.993.

[3] Wray Buntine. “Estimating likelihoods for topic models”. In: *Asian Conference on Machine Learning*. Springer. 2009, pp. 51–64.

[4] Jonathan Chang et al. “Reading Tea Leaves: How Humans Interpret Topic Models”. In: *Neural Information Processing Systems*. Vancouver, BC, 2009. URL: http://umiacs.umd.edu/~jbg//docs/nips2009-rtl.pdf.

[5] “COVID-19 Open Research Dataset (CORD-19)”. In: (). DOI: doi:10.5281/zenodo.3715505. URL: https://pages.semanticscholar.org/coronavirus-research.

[6] Stuart Geman and Donald Geman. “Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images”. In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1984), pp. 721–741.

[7] Thomas L. Griffiths and Mark Steyvers. “Finding scientific topics”. In: *PNAS* (Apr. 2004). URL: https://www.pnas.org/content/101/suppl_1/5228/tab-figures-data.

[8] Matthew Hoffman, Francis R. Bach, and David M. Blei. “Online Learning for Latent Dirichlet Allocation”. In: *Advances in Neural Information Processing Systems 23*. Ed. by J. D. Lafferty et al. Curran Associates, Inc., 2010, pp. 856–864. URL: http://papers.nips.cc/paper/3902-online-learning-for-latent-dirichlet-allocation.pdf.

[9] Jey Han Lau, David Newman, and Timothy Baldwin. “Machine Reading Tea Leaves: Automatically Evaluating Topic Coherence and Topic Model Quality”. In: *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Gothenburg, Sweden: Association for Computational Linguistics, Apr. 2014, pp. 530–539. DOI: 10.3115/v1/E14-1056. URL: https://www.aclweb.org/anthology/E14-1056.

[10] David Newman et al. “Automatic Evaluation of Topic Coherence”. In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. USA: Association for Computational Linguistics, 2010, pp. 100–108. ISBN: 1932432655.

[11] Apache Spark. *LDA.scala*. https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/mllib/clustering/LDA.scala. 2020.

[12] Cheng Zhang et al. “Advances in variational inference”. In: *IEEE transactions on pattern analysis and machine intelligence* 41.8 (2018), pp. 2008–2026.

Appendix

	Word 1	Word 2	Word 3	Word 4	Word 5
Topic 1	cell	use	virus	et	al
Topic 2	cell	use	virus	et	al
Topic 3	cell	use	virus	et	al
Topic 4	cell	use	virus	et	al
Topic 5	cell	use	virus	et	al
Topic 6	cell	use	virus	et	al
Topic 7	cell	use	virus	et	al
Topic 8	cell	use	virus	et	al
Topic 9	cell	use	virus	et	al
Topic 10	cell	use	virus	et	al

Table 4. Distributed EM algorithm: Displays distribution of topics over words for the top 5 words in each topic.

	Word 1	Word 2	Word 3	Word 4	Word 5
Topic 1	protein	use	cell	rna	virus
Topic 2	use	cell	study	q	infection
Topic 3	health	case	disease	use	data
Topic 4	cell	virus	protein	viral	mouse
Topic 5	virus	cell	infection	al	mouse
Topic 6	may	disease	use	animal	infection
Topic 7	patient	study	infection	respiratory	clinical
Topic 8	de	la	en	el	que
Topic 9	virus	sequence	gene	use	al
Topic 10	et	al	protein	cell	use

Table 5. Single node online algorithm: Displays distribution of topics over words for the top 5 words in each topic.

	Word 1	Word 2	Word 3	Word 4	Word 5
Topic 1	virus	use	infection	study	di
Topic 2	cell	virus	use	protein	al
Topic 3	al	et	q	use	virus
Topic 4	patient	study	use	disease	may
Topic 5	de	la	en	el	los
Topic 6	der	die	und	use	patient
Topic 7	de	le	la	et	à
Topic 8	use	virus	health	disease	study
Topic 9	cell	use	protein	infection	gene
Topic 10	virus	cell	infection	use	protein

Table 6. Distributed Online algorithm: Displays distribution of topics over words for the top 5 words in each topic.

	Word 1	Word 2	Word 3	Word 4	Word 5
Topic 1	de	le	la	à	et
Topic 2	patient	study	infection	respiratory	virus
Topic 3	protein	virus	sequence	rna	viral
Topic 4	cell	mouse	infection	response	expression
Topic 5	use	cell	virus	antibody	sample
Topic 6	de	la	en	compound	el
Topic 7	health	disease	use	outbreak	public
Topic 8	al	et	e	di	der
Topic 9	use	model	data	number	method
Topic 10	may	disease	animal	cause	infection

Table 7. Single node Gibbs algorithm: Displays distribution of topics over words for the top 5 words in each topic.

	Word 1	Word 2	Word 3	Word 4	Word 5
Topic 1	de	le	la	et	à
Topic 2	virus	sequence	use	sample	strain
Topic 3	cell	use	mouse	virus	antibody
Topic 4	may	disease	animal	cause	cat
Topic 5	de	la	en	el	los
Topic 6	patient	study	infection	respiratory	use
Topic 7	health	disease	use	risk	public
Topic 8	cell	virus	protein	infection	viral
Topic 9	protein	rna	al	et	structure
Topic 10	model	use	data	number	time

Table 8. Distributed Gibbs algorithm: Displays distribution of topics over words for the top 5 words in each topic.