LDA for Large Scale Data - Final Report

Arpit Rajendra Shah
arpishah@iu.edu
Indiana University, School of Informatics and Computing
Bloomington, Indiana

Karen Sanchez Trejo
karsanc@iu.edu
Indiana University, School of Informatics and Computing
Bloomington, Indiana

Shilpa Singh
shilsing@iu.edu
Indiana University, School of Informatics and Computing
Bloomington, Indiana

1 INTRODUCTION

When an organization wants to obtain useful information from unstructured text documents in order to make business decisions, NLP techniques of classifying and clustering become very important. Latent Dirichlet Allocation (LDA) is a generative probabilistic model for collections of discrete data, which can be used as a unsupervised clustering algorithm for collections of documents. This model can be computationally expensive because text corpora usually contains a large amount of data, thus the need of improving the scalability and finding which implementation has the best computational performance.

LDA represents documents as mixtures over topics and each topic as a distribution over words [4]. The generative model is the following:

(1) For each topic index $k \in \{1, ..., K\}$ draw a word distribution $\beta_k \sim Dir(\eta_k)$
(2) For each document $d \in \{1, ..., M\}$:
  (a) Draw document's topic distribution $\theta_d \sim Dir(\alpha)$
  (b) For each index word $i \in \{1, ..., N_d\}$:
    (i) Choose a topic $z_{d,i} \sim Mult(\theta_d)$
    (ii) Choose a word $w_{d,i} \sim Mult(\beta_{z_{d,i}})$

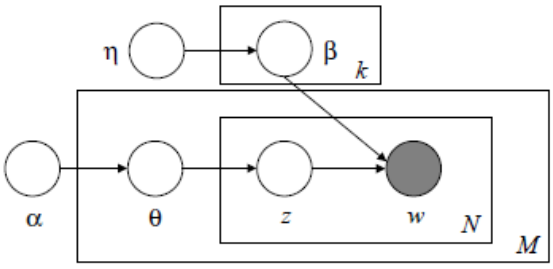

Figure 1: Graphical model representation of LDA [4]

The parameter β is the topic distribution over words, it is a matrix of the form $(K \times V)$, where V is the size of the vocabulary and $\beta_{k,n}$ is the probability of word n appearing in topic k. The parameter θ is the document's distribution over topics, it is a matrix of the form $(M \times K)$ and each $\theta_{d,k}$ is the probability that document d contains topic k. The parameter $z_{d,i}$ is the topic of the ith-word in document d. $w_{d,i}$ is the word in position i on the document d. The hyperparameters α and η are the parameters of the Dirichlet priors.

In order to estimate the parameters of the model we need to calculate the posterior probability of the hidden variables:

$$p(z, \theta, \beta | w, \alpha, \eta) \tag{1}$$

In the case of LDA the posterior is intractable, so a variety of approximate algorithms have been developed [3]. This project will compare the two algorithms that are implemented in Spark, finding which one has the best computational performance.

2 ARCHITECTURE AND IMPLEMENTATION

2.1 Algorithms

We describe the two algorithms that were used in this project: Online Variational Inference which is referred as Online and MAP Estimation that is referred as EM.

2.1.1 Online Variational Inference (Online). In the paper where LDA was first presented [4] the parameters of the model were calculated through Variational Inference, which consists in approximating the real posterior (eq. 1) with a simpler distribution $q(z, \theta, \beta)$. This distribution is indexed by a set of free parameters:

$$q(z_{d,i} = k) = \phi_{dw_{di}k},\ q(\theta_d) = Dir(\theta_d|\gamma_d),\ q(\beta_k) = q(\beta_k|\lambda_k)$$

$$q(z, \theta, \beta | \phi, \gamma\lambda) = \prod_{k=1}^{K} q(\beta_k|\lambda_k) \prod_{d=1}^{M} \left[q(\theta_d|\gamma_d) \prod_{n=1}^{N} {}_d q(z_{d,n}|\phi_{d,n}) \right] \tag{2}$$

The Kullback-Leibler (KL) divergence measures the closeness of the two distributions:

$$KL(q||p) = E_q \left[log \frac{q}{p} \right] \tag{3}$$

We can't directly minimize the KL divergence but we can do the equivalent: maximizing the evidence lower bound (ELBO):

$$L = E_q \left[log\, p(z, \theta, \beta, w | \alpha, \eta) - log\, q(z, \theta, \beta) \right] \tag{4}$$

To find the tightest ELBO we use a variational EM procedure:

- E-step: Maximizes the ELBO with respect to γ and ϕ.

$$\phi_{dik} \propto \beta_{kw_i} exp(\psi(\gamma_{dk})) \tag{5}$$

where $\psi()$ is the first derivative of the $log\Gamma$

$$\gamma_{dk} = \alpha + \sum_{n=1}^{N_d} \phi_{dik} \tag{6}$$

- M-step: Maximizes the ELBO with respect to λ.

$$\lambda_{kw} = \eta + \sum_{d=1}^{M} \sum_{n=1}^{N_d} \phi_{dnk} w_{dn} \tag{7}$$

Each iteration requires a full pass through the entire corpus, so in a large dataset it could require a lot of memory, also it is not useful when there is new data constantly arriving. That is why Online Variational Inference was proposed by Hoffman et. al [6]. This algorithm updates the parameters processing minibatches of documents in each step of the EM algorithm, which improves the scalability and it is useful for streaming data.

2.1.2 MAP Estimation (EM). We can also estimate the parameters using MAP (maximum a posterior) estimation, which finds the mode of the posterior distribution. For this to be possible we have to assume uniform Dirichlet priors (i.e. all values in the α and η vectors must be identical). Then the parameters are estimated with the following EM steps:

- E-step: Update $\phi_{dik} = P(z_{di} = k)$

$$\phi_{dik} \propto \frac{(N_{ik} + \eta - 1)(N_{kd} + \alpha - 1)}{(N_k + V\eta - V)} \tag{8}$$

- M-step: Update θ and β

$$\beta_{ik} = \frac{N_{ik} + \eta - 1}{N_k + V\eta - V} \tag{9}$$

$$\theta_{kd} = \frac{N_{kd} + \alpha - 1}{N_j + K\alpha - K} \tag{10}$$

where N_{id} is the number of observations for word type i in document d, $N_{ik} = \sum_d N_{id}\phi_{dik}$, $N_{kd} = \sum_i N_{id}\phi_{dik}$, $N_k = \sum_i N_{ik}$ and $N_d = \sum_k N_{kd}$.

It has been proved that if we assume uniform Dirichlet priors and we use MAP to estimate the parameters of LDA we obtain a PLSI (Probabilistic Latent Semantic Indexing) model [5]. So in conclusion, the Spark EM algorithm is in fact giving us a PLSI model, which tends to overfit because it doesn't consider the Dirichlet priors [4].

2.1.3 Hyperparameters. The setting of the hyperparameters (α, η) is very important to obtain a good model. Selecting a high α value (>1) means that each document is likely to contain a mixture of most of the topics vs selecting a low value (<1) means that it is more likely that a document contains just few topics. The equivalent happens with η, high values mean that each topic has most words and low values mean each topic contains few words. [8]

We can also pick α to be asymmetric, i.e., that each entry of the α vector has a different value. By doing this, we expect that certain groups of words will occur more frequently than others in every document. For example, in machine learning documents, words like "model", "data" and "algorithm" are very likely in every document, we can address this by using asymmetric priors [8]. As we mention before EM can't use this kind of priors, they have to be uniform.

In the case of MAP estimation (EM), hyperparameters lower than 1 cause MAP to have negative probability, that is why in the Spark implementation the hyperparameters must be greater than 1, following Asuncion et.al. [3], who recommended a +1 adjustment for this algorithm.

2.2 Dataset

We used Yelp dataset [9], which contains up to 5GB of data from Yelp users with a vocabulary of 262,144 words and 6,685,900 reviews. At the beginning of this project we were using PubMed dataset abstracts to run experiments, but this data is already pre-processed, so we switched datasets in order to construct a complete pipeline.

2.3 Distributed approach through Spark LDA implementation

We used Spark for the implementation. This provides a simple interface that abstract the complicated aspects of MapReduce programming, so we can build our distributed processing jobs. All distributed computations are done on RDDs (Resilient Distributed Datasets), which are linear data structures that are distributed across the nodes in the cluster and utilizes RAM of the machines in the clusters. When data is kept in memory, disk serialization/deserialization is greatly reduced. This allows us to distribute iterative algorithms much faster because we don't have to wait for the data to write to disk after each iteration [1].

We used the RDD-based LDA algorithm developed in Spark. It takes as input a collection of documents as vectors of word counts and the following parameters:

- k: Number of topics
- optimizer: EMLDAOptimizer or OnlineLDAOptimizer
- docConcentration: Dirichlet parameter for prior over documents' distributions over topics (α).

- topicConcentration: Dirichlet parameter for prior over topics' distributions over words (η).
- maxIterations: Limit on the number of iterations.

The implementation supports different inference algorithms via the setOptimizer function. The EMLDAOptimizer learns clustering using MAP Estimation, while OnlineLDAOptimizer uses iterative mini-batch sampling for Online Variational Inference [2].

2.4 High-level Design

We implemented text pre-processing, model creation and evaluation in a single pipeline though Spark MLlib and Feature Transformation Apis. The Feature Transformation Apis were used to perform basic NLP processing like Tokenzation,removing Stopwords and Countvectorization.We also used spark-core nlp for POStagging and filtering only nouns in the text since it has been shown that this improves the model.[7].

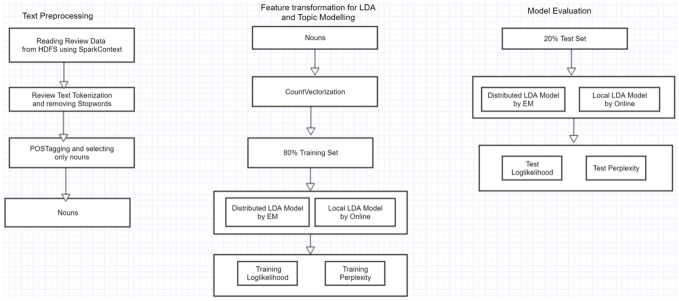

Figure 2: Implementation Overview

3 EXPERIMENTS

The experiments that we carried out aim to compare the two algorithms (EM and Online) in two ways: Computational performance and model quality. We divided the dataset in 80% training and 20% testing, and we measured the quality of the model with the perplexity of the test dataset, the lower the perplexity, the better the model.

3.1 Convergence

First, we want to ensure that the algorithms are converging. EM only supported 100 iterations, after that it gives an overflow error, with Online we tried up to 150 iterations without getting this error. We can observe (see fig. 3) that the EM algorithm is converging faster than Online. However, after 100 and 150 iterations each, Online gives better test perplexity (EM:21.86 vs Online:8.01).

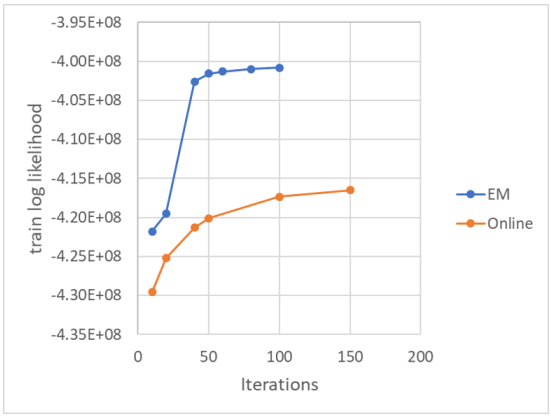

Figure 3: Train log likelihood vs number of iterations of EM (α=1.5, η=1.5, k=10) and Online (α=0.5, η=0.5, k=10). Settings: 12 executors, 1GB data.

3.2 Tuning hyperparameters

We want to get the hyperparameters (α and η) that give the best test perplexity for each algorithm. The experiments consist in holding fixed one parameter (either α or η), the number of topics k and the number of iterations. Then we varied the free parameter to observe when the test perplexity converges. This happens with EM for α=3.5 and η=2 (see fig. 4) and with Online for α=0.5 and η=0.1(see fig. 6), as we can see the parameters are very different for each algorithm. We also varied the number of topics (k), in fig. 5 we can see that the test perplexity increases as the number of topics increase, so we can say that the data is better represented with a low number of topics.

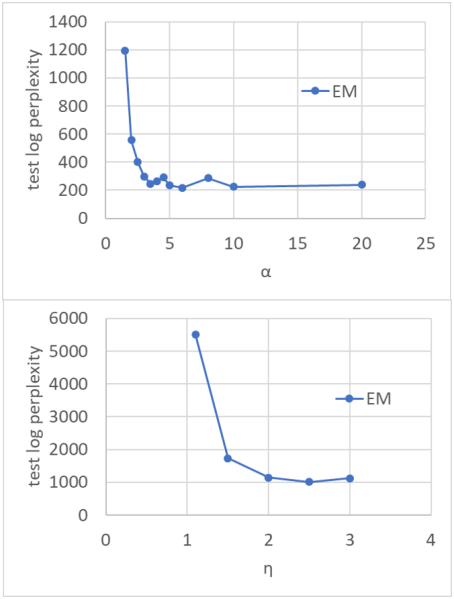

Figure 4: Test log perplexity as a function of parameter value. Top: EM (fixed η=1.1, K=20, iterations=100), Bottom: EM (fixed α=1.1, K=20, iterations=100). Settings: 12 executors, 1GB data.

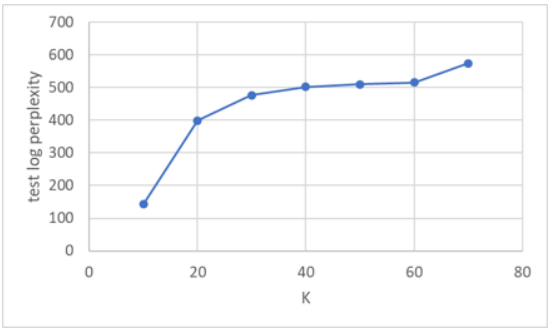

Figure 5: Test log perplexity as a function of number of topics (K). EM (fixed α=3, β=1.1, iterations=100), Settings: 12 executors, 1GB data.

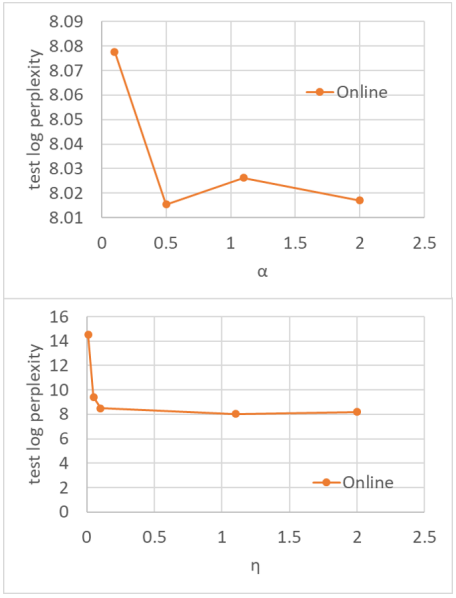

Figure 6: Test log perplexity as a function of parameter value. Top: Online (fixed α=0.05, K=20, iterations=100), Bottom: Online (fixed η=0.5, K=10, iterations=100). Settings: 12 executors, 1GB data.

3.3 Performance

We compare EM and Online in terms of computational performance. In figure 7 we can observe that the EM algorithm is faster in training, this could be explained to the fact that it is a simpler algorithm than the Online, which for example calculates the derivative of the gamma function in each iteration. It also has to do with the Spark implementation and the resources each algorithm uses when running.

Figure 7: Execution time (seconds) as a function of number of iterations. Settings: 12 executors, 1GB data.

In figure 8 we can observe that both algorithms take a lot of time to calculate the likelihood and perplexity, this is related to the Spark implementation. Online is the slowest, since the function to calculate likelihood requires the documents as input, in comparison, the EM algorithm calculates the likelihood at the time of training.

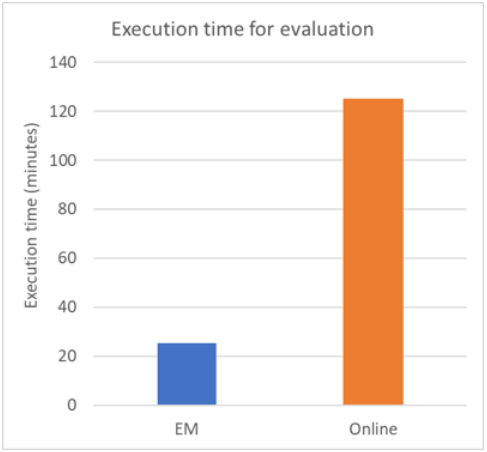

Figure 8: Bar plot of time (minutes) that took to calculate the train log likelihood and test log perplexity. Settings: 12 executors, 1GB data.

When we vary the number of executors for 1GB, we can see that 12 is the optimal for EM (see fig. 9). The resource allocation strategy for running spark experiments was to divide the total number of cores (only 24 taken as more cores were causing more network shuffling) and memory(only 62GB taken) on a node among the executors where initially each executor got most of the cores (20) and memory(60GB) resulting in a smaller number of executors(3) throughout the cluster. Then we decreased the number of allocated cores to 1 for each executor and memory (5GB) to each executor resulting in a greater number of executors (30) in the cluster. The optimal performance we got for EM is between these 2 extreme conditions where the cores allocated to each executor was 3 and memory 15GB with total number of executors

as 12. The reason for the medium sized executor being the most optimal was that when we create too less executors, we impose a limit on the degree of parallelism across map partitions as even though they have more cores to execute tasks in parallel , they are forced to work on the same data partition which is a condition enforced by spark. But, when we create more executors, though we have more executors to map to these data partitions, the memory allocated to each executor is reduced which caused lot of shuffling and delays in garbage collection step in the spark execution.If we keep the memory of each executor more and increase the number of executors,there is more network shuffling. So, the optimal value was achieved for medium sized executor where there were enough executors to map to data partitions in parallel without causing any shuffling in the execution. For Online there was not much difference in the execution time as compared to EM, when we started to allocate more resources to each executor and decrease the number of executors. The execution time for all the executor settings was approximate 30 minutes (for 100 iterations in 1GB) without POS tagging and 5 hours with POS tagging on 1 GB of data. This was because of instantiating new sentence object for nlp processing inside every record of the data frame, which was computationally expensive.

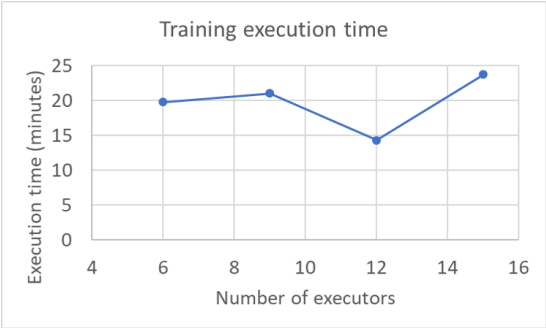

Figure 9: Training execution time (minutes) vs number of executors. Settings: EM(α=3.5, η=2, K=10, iterations=100), 1GB data.

Fixing the number of executors to 12, we calculated the execution time depending on the size of the file. We can see that the Online algorithm takes more time and increases faster than EM.

Figure 10: Training execution time (minutes) as a function of data size. Settings: EM(α=3.5, η=2, K=10, iterations=100), Online(α=0.5, η=0.1, K=10, iterations=100), 12 executors.

3.4 Topic Model Results

After tuning the parameters of the model and finding in which values the test perplexity converges, we show the results of the topic model for this parameters (see fig. 11 and 12). This topics were found after selecting only nouns during pre-processing. We think that in general, EM and Online give similar results.

EM									
Topic 1		**Topic 2**		**Topic 3**		**Topic 4**		**Topic 5**	
night	0.041	pizza	0.038	place	0.175	breakfast	0.027	food	0.125
people	0.035	cheese	0.034	service	0.098	coffee	0.027	order	0.061
drinks	0.028	chicken	0.033	food	0.058	selection	0.024	time	0.059
beer	0.020	salad	0.031	love	0.049	cream	0.022	table	0.030
kids	0.015	sauce	0.024	staff	0.046	options	0.020	minutes	0.025
music	0.014	bread	0.018	restaurant	0.037	buffet	0.016	fries	0.023
time	0.013	steak	0.016	experience	0.030	chocolate	0.013	server	0.023
group	0.013	meat	0.014	prices	0.027	cake	0.013	wait	0.023
drink	0.013	meal	0.014	everything	0.026	dessert	0.013	burger	0.023
party	0.013	side	0.013	atmosphere	0.023	eggs	0.013	sadwich	0.021
Topic 6		**Topic 7**		**Topic 8**		**Topic 9**		**Topic 10**	
service	0.037	time	0.037	room	0.033	work	0.018	chicken	0.027
customer	0.031	years	0.028	area	0.029	hair	0.018	food	0.027
location	0.029	care	0.021	hotel	0.017	shop	0.014	sushi	0.024
store	0.023	staff	0.018	parking	0.015	price	0.014	rice	0.024
time	0.020	appointment	0.013	strip	0.013	experience	0.013	soup	0.019
home	0.019	office	0.013	water	0.012	deal	0.012	restaurant	0.018
manager	0.018	year	0.013	trip	0.009	thanks	0.012	dishes	0.018
money	0.017	week	0.012	walk	0.009	card	0.011	pork	0.018
people	0.015	work	0.012	pool	0.009	company	0.011	fish	0.017
business	0.014	experience	0.011	vegas	0.008	guys	0.009	price	0.016

Figure 11: Top 10 words with EM (α=3.5, η=2, k=10, iterations=100). Settings: 12 executors, 1GB data.

Online									
Topic 1		**Topic 2**		**Topic 3**		**Topic 4**		**Topic 5**	
show	0.071	time	0.017	hair	0.041	jarrett	0.003	pinball	0.011
church	0.019	service	0.014	time	0.027	pedro	0.003	tait	0.009
stage	0.015	place	0.011	place	0.023	service	0.002	service	0.009
music	0.013	staff	0.009	nails	0.019	time	0.002	machines	0.007
filipino	0.011	people	0.008	salon	0.019	maga	0.001	montreal	0.007
audience	0.011	customer	0.008	massage	0.015	krav	0.001	beatles	0.007
tickets	0.010	room	0.007	nail	0.014	place	0.001	games	0.005
comedy	0.009	store	0.007	appointment	0.012	gazette	0.001	restaurant	0.005
seats	0.009	work	0.007	love	0.012	clutch	0.001	bien	0.005
halo	0.008	experience	0.006	experience	0.011	sunburst	0.001	vraiment	0.005
Topic 6		**Topic 7**		**Topic 8**		**Topic 9**		**Topic 10**	
boots	0.016	matcha	0.009	food	0.036	naan	0.017	dosa	0.014
hershey	0.007	linq	0.003	place	0.025	masala	0.015	dosas	0.003
shoes	0.007	pizza	0.003	service	0.017	yelp	0.015	food	0.003
pair	0.004	sehr	0.003	time	0.013	http	0.015	orchestra	0.003
place	0.003	promanade	0.003	chicken	0.011	paneer	0.014	warren	0.003
heel	0.003	auch	0.002	restaurant	0.010	biz_photos	0.012	bojangles	0.003
cowboy	0.003	nincht	0.002	order	0.009	tikka	0.011	recommend	0.002
shoe	0.003	oder	0.002	menu	0.009	biryani	0.009	whataburger	0.002
zipper	0.002	hotel	0.002	pizza	0.007	chicken	0.009	lenny	0.002
statue	0.002	service	0.002	sauce	0.006	food	0.006	shaat	0.002

Figure 12: Top 10 words with Online (α=0.5, η=0.1, k=10, iterations=150). Settings: 12 executors, 1GB data.

4 CONCLUSION

We found that EM converges faster, however Online gives better results in terms of test perplexity. The best test perplexity that we got with the parameters that we tried was 21.86 with EM (alpha=1.5, beta=1.5, k=10, iterations=100) vs 8.01 with Online (alpha=0.5, beta=0.5, iterations=150).

Regarding the computational performance, EM was faster in training, this could be explained to the fact that it is a simpler algorithm than Online, it also has to do with the Spark implementation and the resources each algorithm uses when running. Both algorithms take a lot of time to calculate the likelihood and perplexity, this is related to the Spark implementation.

Looking deeply into both algorithms (Online and EM) we realized that they are in essence very different, on one hand Online is a Variational Inference approach to estimate the parameters of LDA. On the other hand, the EM algorithm is a MAP estimation of LDA, which we found it is in fact just a regularized PLSI model, meaning that can only consider symmetric priors and tends to overfit because it is a point estimate. In comparison, the Online algorithm can accept asymmetric prior for document's distribution over topics (α) and manage overfitting better because it is a Bayes estimation. All of this is confirmed in our experiments.

We also found that EM algorithm benefited from medium sized executors in terms of execution time,but Online algorithm was agnostic to the different executor settings.

In conclusion, with the Online algorithm we got better results (in terms of test perplexity), it doesn't get an overflow error and takes asymmetric priors. EM had worse test perplexity, it gets an overflow error after 100 iterations and doesn't consider asymmetric priors. In general, we could say Online is a better algorithm, but it is also the slowest. So, we should decide between the two depending on our priorities and resources.

5 ACKNOWLEDGMENT

We would like to thank professor Judy Qiu, Selahattin Akkas and our mentor Bo Peng for providing us the guidance to accompish this project.

REFERENCES

[1] Alex Minnaar. 2019. Distributed Online Latent Dirichlet Allocation with Apache Spark. http://alexminnaar.com/distributed-online-latent-dirichlet-allocation-with-apache-spark.html

[2] Apache Spark. 2019. MLlib: RDD-based API. https://spark.apache.org/docs/latest/mllib-clustering.html#latent-dirichlet-allocation-lda

[3] Arthur Asuncion, Max Welling, Padhraic Smyth, and Yee Whye Teh. 2009. On smoothing and inference for topic models. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press, 27–34.

[4] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.

[5] Mark Girolami and Ata Kabán. 2003. On an equivalence between PLSI and LDA. In *SIGIR*, Vol. 3. 433–434.

[6] Matthew Hoffman, Francis R Bach, and David M Blei. 2010. Online learning for latent dirichlet allocation. In *advances in neural information processing systems*. 856–864.

[7] Fiona Martin and Mark Johnson. 2015. More efficient topic modelling through a noun only approach. In *Proceedings of the Australasian Language Technology Association Workshop 2015*. 111–115.

[8] Hanna M Wallach, David M Mimno, and Andrew McCallum. 2009. Rethinking LDA: Why priors matter. In *Advances in neural information processing systems*. 1973–1981.

[9] Yelp Inc. 2019. Yelp Dataset. https://www.yelp.com/dataset