STA 663: Final Writeup

Cole Juracek and Pierre Gardan

April 2020

1 Abstract

Latent Dirichlet Allocation (LDA) is a generative, probabilistic model for discrete data, which is traditionally text. LDA is a hierarchical Bayesian model where we assume the words in each document are drawn according to document-specific mixture distributions of finite topics. Moving one level up the hierarchy, each document draws its mixture distribution from an infinite mixture of topic probabilities contained in the simplex. There are many goals for LDA, but they all trace back to computation of the posterior of this model; given documents of words, we wish to obtain the posterior probabilities of topic assignment per word, topic distribution per document, and word distribution per topic.

2 Background

For this project, we will be referencing "Finding Scientific Topics" (Griffiths et al.) and "Fast Collapsed Gibbs Sampling for Latent Dirichlet Allocation" (Porteous et al.). They both cover the same material, however the latter describes the algorithm in more explicit detail. Consider the following graphical model (picture from Porteous):

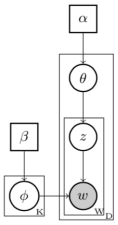

Figure 1: Graphical model for LDA.

For a given corpus, α and β are fixed hyperparameters that control the sparsity of topics. The goal of LDA then is to compute the posterior probabilities of the other latent variables:

$$p(\theta, Z, \phi|W) \tag{1}$$

For a complex hiererachical model such as this, there is no tractable closed form solution, and we cannot appeal to any simple ways of getting at the posterior. There are two main approaches of approximating the posterior:

1. Sampling based (MCMC)
2. Optimization (Variational Bayes)

This paper and project concerns choice (1). We seek to use a Monte Carlo simulation to approximate the posterior by constructing a Markov chain whose stationary distribution is the posterior. We will construct the Markov chain through the use of Gibbs sampling.

A naive construction of Gibbs sampling using full conditionals, while technically correct, will not mix in a reasonable amount of time. We may help the chain mix faster by integrating out θ and ϕ, and constructing a chain based on the z values alone. This is what is known as a *collapsed* Gibbs sampler. In other words, we are only looking for the marginal posterior of the topics conditioned on all of the other topics in the document, or:

$$p(z_{ij}|z^{-ij}, \mathbf{w}, \alpha, \beta) \tag{2}$$

The Markov chain here will converge in a relatively short number of iterations, which is desired; each iteration of this sampler updates every word in every document, which can be very expensive depending on the size of the corpus. Once the chain has converged, we may choose the topic per word as the most frequently occurring value in the chain. Finally, we may obtain estimates for θ, ϕ as well using the topic values.

LDA may be used for a variety of problems. One such problem is that of document modelling. Given a text corpus, we would like to find a way to represent the information it contains in a compressed form, while preserving the statistical structure within and between documents. In LDA, each document is represented by a θ vector containing the mixture distributions over K topics. We may use this mixture to determine how homogenous a document is compared to others in the same corpus. Aside from that, we may also use these θ vectors for classification, summarization, and similarity scoring.

We may also use LDA to discover the latent topics in our corpus, along with the most probable words for any topic. It is important to note here that the statistical structure imposed on these documents pays no attention to what words a topic *should* contain; the fact that we are able to recognize the topics produced as useful insights into the corpus is simply a byproduct of the algorithm running.

In regards to the different implementations of LDA, Porteuous mentions that "the variational approach is arguably faster computationally, but the Gibbs sampling approach is in principle more accurate since it asymptotically approaches the correct distribution." While we will not be writing a variational method, we

will be comparing with Scikit's implementation that does utilize a variational approach. We opted for the sampling based method as we were already familiar with the notion of MCMC and wanted to get results first and foremost.

3 Description

3.1 High-level

As stated above, we seek to obtain the marginal posterior distribution of a topic conditioned on all the other topics in the document. We construct a Markov chain of these topics using Gibbs sampling. A new sample is chosen first by computing the probability of that word belonging to that topic. This probability is influenced by 2 factors (not including the word in consideration):

1. How many times times that topic appears in the same document
2. A ratio of how many times that word appears in that topic to the number of times that topic appears at all

(1) encourages topic similarity within the document. If topic 3 appears frequently in a given document, there is a higher probability that a new word will be assigned to topic 3. The "amount" this matters is determined by the α parameter.

(2) encourages words to belong to the same topic. If "baseball" is assigned to topic 3 all 10 times, there is a higher probability that a new instance of the word "baseball" will be assigned to topic 3. Note that even if the word never currently appears in a particular topic, there is still a chance to be assigned to that topic next. The amount that the current topic distribution of a given word influences the next assignment is also dictated by the β parameter, in addition to the size of the word vocabulary.

Once we have these probabilities, we normalize by dividing by the sum of the densities, and draw a random sample. Gibbs sampling will allow us to explore the latent space of topics, while also tending toward a topic configuration that has high posterior density. Finally, we may calculate approximations of the other latent variables using the current topic assignment:

- ϕ_{wk}: The probability of word w in the kth topic. Note that every word has a probability in every topic. For a topic identified as "sports", the word "baseball" will likely be very high. In contrast, if another topic is identified as "academia", the probability of baseball will likely be very low.
- θ_{kj}: The mixture component of topic k in document j. Different documents will have different distributions of mixture components. Some may be almost entirely assigned to a single topic, while others will exhibit equal mixtures over several topics.

3.2 Technical

Now we dive into the technical details of the algorithm. First, we begin with some notation:

$N_{wkj} = \#\{i : x_{ij} = w, z_{ij} = k\}$, or the number of times word w belongs to topic k in document j. Missing indices are summed out, so we obtain the following notation:

- $N_{kj} = \sum_w N_{wkj}$: The number of times topic k appears in document j
- $N_{wk} = \sum_j N_{wkj}$: The number of times word w belongs to topic k
- $N_k = \sum_w \sum_j N_{wkj}$: The number of times topic k appears in the corpus.

Let the superscript $-ij$ indicate that the word i from document j has been excluded from the relevant count.

Finally, let $\mathbf{x} = x_{ij}$: The set of all words in document j (where i, j are implicitly defined in the context of the z_{ij} we are working with). Now recall the Markov chain we are trying to construct. We want to obtain the posterior distribution of topic i in document j, additionally conditioned on all of the other topics in the document. Mathematically, this posterior distribution can be written as:

$$p(z_{ij}|\mathbf{z}^{-ij}, \mathbf{x}, \alpha, \beta) \tag{3}$$

Thus we can calculate the probability of any topic for z_{ij} by

$$p_{z_{ij}}(k) = \frac{1}{Z} a_{kj} b_{wk} \tag{4}$$

Where

$$a_{kj} = N_{kj}^{-ij} + \alpha$$
$$b_{wk} = \frac{N_{wk}^{-ij} + \beta}{N_k^{-ij} + W\beta}$$

And Z is the normalizing constant, which we obtain for a discrete distribution by summing the densities of each value of k, or:

$$Z = \sum_k^K a_{kj} b_{wk} \tag{5}$$

With the method for sampling new topics now achieved, we can construct a Markov chain via the algorithm below:

z_{ij} = [] for all i, j
for $iter \leftarrow 1$ to $niter$ **do**

  for $j \leftarrow 1$ to D **do**

    for $i \leftarrow 1$ to n_j **do**
      Densities = []
      for $k \leftarrow 1$ to K **do**
        Append $p(z_{ij} = k)$ (4) to Densities
      end for
      Densities = Densities / sum(Densities)
      New topic = Sample from Densities
      Append new topic to Markov chain of z_{ij}
    end for
  end for
end for
z_{ij} = mode z_{ij}

The time complexity of this algorithm is not ideal; it's somewhere on the order of magnitude of $O(n_{iter} * D * n_j * K)$. There's no getting around this for a sampling based approach; we have to iterate through every word in every document for every possible topic. Luckily, the algorithm converges in a relatively short number of iterations.

Once we have either achieved convergence or reached the max number of iterations, we may use the Markov chain of topics to produce estimates of per-document topic distributions (θ_j) and per-topic word distributions (ϕ_k):

$$\hat{\phi}_{wk} = \frac{N_{wk} + \beta}{N_k + W\beta}$$
$$\hat{\theta}_{kj} = \frac{N_{kj} + \alpha}{N_j + K\alpha}$$

4 Describe optimization for performance

Initially the code we had written was very intractable, and only worked on small data sets. Due to the nature of the Gibbs sampling algorithm, it's very punishing on adding any additional loops due to the already nested layers of the algorithm. Smart use of dictionaries to keep track of relevant counts allowed us to effectively utilize the space-time tradeoff and achieve reasonable performance on larger data sets.

Profilling was used to improve the performance of our algorithm. Gibbs sampling being sequential, we vectorized as many steps as we could and implemented several modifications to make marginal speed gains such as for loops arrangements or the use of scalars for basic operations. We used data dictionaries and lists to proceed through the analysis efficiently. Numba was considered

to improve the overall speed. However, since version 0.45, numba has introduced new experimental ways to implement lists and dictionaries. As a result, it is still quite restrictive which did not allow us to execute necessary steps such as nested dictionaries or nested lists. Cython was explored as well but did not make the algorithm faster and was therefore disregarded. Finally, given the nature of gibbs sampling, implementing parallelization is technically "impossible". However, an approximation by distributing documents equally through P processors and updating the global counts after each iteration of the samplers can be done. Since this improvement in speed does not match our original algorithm, we did not proceed with parallelization but instead used the gensim package to implement it for comparison (see Comparative Analysis section). We also note the "fast" Gibbs sampler that Porteous recommends. This was never in the scope of this project; we just wanted to implement the simple Gibbs sampler proposed by Griffiths and the beggining of Porteous. The faster Gibbs sampler is certainly a consideration for future considerations though.

5 Applications to simulated data sets

Given the characteristics of Latent Dirichlet Allocation, there are no simulated datasets giving known outputs. However, we were able to use the 20NewsGroup dataset extracted from the scikit package. We imported the training portion of the dataset made of more than 11,000 news reports originally classified in twenty categories. Our algorithm was implemented on this dataset with twenty topics. The top words generated for each topic were compared to the actual twenty categories. We used the parameters recommended by Porteous et al. for the prior of topic word distribution ($\beta = 0.01$) and the prior of document topic distribution ($\alpha = 2/K$, $K = 20$ here).
After data preprocessing to eliminate common stop words (i.e. 'the', 'in',..), numbers and to "stem" some common word endings (i.e. 'able', 'ing', ...), our algorithm took about 7.5 minutes to run with 10 iterations of the sampler for each word. On the outputs of figure 1, we can clearly see the similarities between the five top words generated and the original categories. For example, topic 9 represents the "soc.religion.christian", topic 7 illustrates "sci.space". Overall, we can match more than three quarters of these topics/categories. Printing the ten top words could have help us make even more connections. Therefore, we can say that this application shows overall accuracy of our collapsed Gibbs sampling.

6 Applications to real data sets

We applied LDA on a collection of articles from Reuter (reut2-000, available at http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html). This file is made of 1,000 news articles published by Reuter. We determined 5 arbitrary topics to be generated through 10 iterations of the sampler for each word. We

alt.atheism	
comp.graphics	
comp.os.ms-windows.misc	
comp.sys.ibm.pc.hardware	
comp.sys.mac.hardware	
comp.windows.x	
misc.forsale	
rec.autos	
rec.motorcycles	
rec.sport.baseball	
rec.sport.hockey	
sci.crypt	
sci.electronics	
sci.med	
sci.space	
soc.religion.christian	
talk.politics.guns	
talk.politics.mideast	
talk.politics.misc	
talk.religion.misc	

(a) Original Categories

1	2	3	4	5	6	7	8	9	\
team	space	key	play	use	one	use	drive	one	
db	use	use	game	speed	know	one	control	make	
game	system	chip	first	scsi	peopl	document	hard	get	
year	launch	encrypt	period	drive	even	turkish	disk	use	
win	support	secur	power	water	say	p	car	know	

10	11	12	13	14	15	16	17	18
use	armenian	q	think	new	gun	god	say	peopl
one	good	presid	go	appear	like	jesu	peopl	state
problem	get	center	peopl	cover	one	christian	one	right
like	like	new	know	copi	weapon	mean	go	kill
wire	right	avail	mr	book	firearm	believ	think	gun

19	20
x	window
file	run
use	use
program	imag
entri	thank

(b) Distribution of top words among topics

Figure 1: Comparison of generated topics with actual categories

chose 5 topics as topics became too similar when the number of topics increased. We used again the Porteus recommended parameters and compared it to the Griffiths recommendations $\alpha = 50/K$, $\beta = .1$. The algorithm took 23 seconds to execute. The topics' top-5 words can be seen below. Topics seem to converge around finance, the sock market, money, industry, oil and trade for both choices of parameters.

1	2	3	4	5
share	mln	market	bank	pct
compani	vs	us	pct	year
dlr	dlr	oil	billion	billion
inc	ct	industri	dlr	govern
stock	net	price	issu	price

(a) Porteus; $\alpha = \frac{2}{5}$, $\beta = .01$

1	2	3	4	5
us	pct	tonn	bank	mln
industri	bank	export	year	dlr
price	march	pct	pct	vs
market	billion	govern	week	ct
mln	manag	oil	share	net

(b) Griffiths; $\alpha = \frac{50}{5}$, $\beta = .1$

Figure 2: Reuter

We also wanted to apply our LDA algorithm on a more data set much larger at scale. For this, we turn to the NIPS document set implemented by Porteous et al., which contains nearly 2 million words across 1,500 documents, and 12000 unique words. Unfortunately, our algorithm could not handle the size of this input. It took too long for even 1 iteration of the Gibbs sampler on the full data set. We wanted to apply LDA on a data set found in the paper though, and so we took a random sample of 10 percent of the data. This is prone to producing

skewed topics in our data set. We accept this due to the scale of the data set we are working with:

	1	2	3	4	5	6	7	8	9	10
0	stimulus	isolate	stimulus	keywords	stimulus	relaxes	numbers	isolate	alan	identical
1	numbers	dueling	denoise	specifying	eigenfunction	stimulus	prot	numbers	eigenfunction	isolate
2	quantities	tained	inclusion	numbers	denoise	mon	eigenfunction	prot	fails	netw
3	timings	penalizing	eigenfunction	imprecise	ability	sition	richard	reconstructs	jacobs	numbers
4	isolate	depended	purkinje	graf	leslie	tained	stimulus	numer	stimulus	adoption

We ran our algorithm on the NIPS data with $\alpha = K/2, \beta = 0.01$. It does not seem to perform as well here. It certainly picks up topics related to math (eigenfunction, numbers), but the rest seems to lack coherence. This is almost certainly due to the much smaller data set that we are forced to work with in this section. Additionally, it could be lack of convergence on account of needing to run this sampler for fewer iterations than previous data sets.

7 Comparative analysis with competing algorithms

We compared our collapsed gibbs sampler of LDA with two different methods: variational bayesian inference and parallelized collapsed gibbs sampling or PLDA (an approximation of collapsed gibbs sampling). To do so, we used existing packages scikit-learn and gensim. Scikit-learn provides a variational bayesian inference method while we executed PLDA through gensim. Note that we used our own preprocessing method to compare analysis on the same data. We left the default parameters, only tuning the number of topics for the different dataset. We compared these three methods on 20NewsGroup and Reuter.

For the 20NewsGroup dataset, with 20 topics and 10 iterations, scikit-learn was faster as it took 2.25 minutes to run. However, even though it may be subjective, it seems that the top-5 words displayed for each topic do not seem to match the actual categories as well as our algorithm. The implementation of gensim took about 7 minutes and the topics generated were quite comparable to the ones of our collapsed gibbs sampler (see figure 3). Note that the numbers appearing before words with gensim represent the proportion of time a word characterize the topic when this topic appears.

For the Reuter dataset, with 5 topics and 10 iterations, scikit learn and gensim both took about 4 seconds to execute. We see the convergence of the same topics between our algorithm and the gensim implemetation (money, finance, oil or trade). The scikit output also gives us topics around money, finance and

```
#0: new san april center divis
#1: la pt van trade turk
#2: do ps pro gateway propheci
#3: like armenian time car know
#4: space db launch orbit satellit
#5: use file program window avail
#6: vs gm ranger pit det
#7: key encrypt chip secur de
#8: driver vote acceler nj card
#9: cartridg cure loop vitamin eh
#10: stephanopoulo packag labor vat dole
#11: keyboard batteri voltag acid stress
#12: conductor coli sail b6 gnd
#13: use like think make peopl
#14: null undefin sy void aluminum
#15: drive use card disk problem
#16: god christian jesu peopl believ
#17: int char valu part na
#18: game team play player year
#19: event alloc valu pointer colormap
```

(a) Scikit

```
0   0.054*"key" + 0.025*"secur" + 0.022*"input" + 0.021*"technolog" + 0.018*"encrypt"
1   0.093*"team" + 0.060*"play" + 0.058*"player" + 0.051*"win" + 0.050*"ship"
2   0.053*"vs" + 0.023*"wire" + 0.023*"wing" + 0.023*"texa" + 0.020*"red"
3   0.075*"god" + 0.048*"true" + 0.044*"jesu" + 0.028*"church" + 0.024*"bibl"
4   0.040*"moral" + 0.035*"greek" + 0.034*"action" + 0.034*"event" + 0.030*"judg"
5   0.056*"use" + 0.029*"window" + 0.027*"system" + 0.025*"entri" + 0.021*"drive"
6   0.020*"govern" + 0.020*"nation" + 0.017*"jew" + 0.016*"state" + 0.015*"left"
7   0.574*"x" + 0.025*"output" + 0.022*"display" + 0.018*"char" + 0.018*"valu"
8   0.053*"button" + 0.045*"school" + 0.044*"prevent" + 0.040*"weapon" + 0.040*"press"
9   0.093*"leagu" + 0.064*"score" + 0.029*"playoff" + 0.014*"ranger" + 0.013*"math"
10  0.158*"thank" + 0.061*"keyboard" + 0.054*"pleas" + 0.044*"appreci" + 0.034*"advanc"
11  0.017*"new" + 0.013*"public" + 0.011*"report" + 0.011*"first" + 0.009*"unit"
12  0.089*"space" + 0.037*"satellit" + 0.035*"launch" + 0.033*"smith" + 0.022*"highest"
13  0.040*"length" + 0.039*"outsid" + 0.023*"usenet" + 0.021*"signatur" + 0.018*"notion"
14  0.026*"year" + 0.022*"good" + 0.018*"like" + 0.018*"look" + 0.016*"game"
15  0.062*"mr" + 0.048*"respond" + 0.034*"music" + 0.023*"copyright" + 0.021*"convent"
16  0.011*"brad" + 0.005*"adb" + 0.000*"compat" + 0.000*"one" + 0.000*"instal"
17  0.039*"file" + 0.033*"program" + 0.024*"use" + 0.018*"avail" + 0.017*"inform"
18  0.088*"israel" + 0.057*"isra" + 0.050*"cap" + 0.034*"beat" + 0.027*"van"
19  0.015*"know" + 0.014*"peopl" + 0.011*"think" + 0.011*"use" + 0.011*"like"
```

(b) Gensim

Figure 3: Comparison of generated topics for 20NewsGroup

oil. However, it seems to have a topic around politics not appearing in our implementation.

```
#0: oil price opec ton dlr
#1: pct bank year billion dlr
#2: vs mln ct dlr net
#3: reagan coffe quota presid deleg
#4: dlr compani share inc mln
```

(a) Scikit

```
0  0.079*"mln" + 0.056*"vs" + 0.045*"dlr" + 0.034*"ct" + 0.026*"pct"
1  0.020*"pct" + 0.019*"dlr" + 0.013*"compani" + 0.013*"mln" + 0.013*"share"
2  0.024*"oil" + 0.017*"product" + 0.014*"price" + 0.013*"ton" + 0.010*"tonn"
3  0.012*"us" + 0.011*"trade" + 0.011*"govern" + 0.010*"last" + 0.009*"year"
4  0.032*"bank" + 0.009*"compani" + 0.008*"japan" + 0.008*"debt" + 0.008*"dlr"
```

(b) Gensim

Figure 4: Comparison of generated topics for Reuter

8 Conclusion

We have successfully been able to implement an algorithm for Latent Dirichlet Allocation. LDA is inherently an unsupervised learning algorithm, so by verifying that the topics produced "make sense", we have verified our correct implementation of the algorithm. This project has taught us many things including:

- Hierarchical models
- Collapsed Gibbs sampling
- Bayesian posterior inference
- Natural language processing
- Package creation in Python

Further steps could be taken to perform posterior analysis. One particularly promising approach is using the θ components for document similarity and returning similar documents to one given. Speed is also a concern; the faster Gibbs sampler could be one avenue for this, as well as alternate methods based around variational Bayes.

9 References:

1. David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. J. Mach. Learn. Res. 3, null (March 2003), 993–1022.

2. Griffiths, Thomas & Steyvers, Mark. (2004). Finding Scientific Topics. Proceedings of the National Academy of Sciences of the United States of America. 101 Suppl 1. 5228-35. 10.1073/pnas.0307752101.

3. Ian Porteous, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. 2008. Fast collapsed gibbs sampling for latent dirichlet allocation. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '08). Association for Computing Machinery, New York, NY, USA, 569–577.

4. Newman, D., Asuncion, A., Smyth, P.,and Welling, M. 2007. Distributed inference for latent dirichlet allocation. In Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS'07). 1081–1088.

5. Schofield, A., Magnusson, M., Thompson, L., & Mimno, D. (2017). Pre-Processing for Latent Dirichlet Allocation.