Exploration of Variational Inference and Monte Carlo Markov Chain Models for Latent Dirichlet Allocation of Wikipedia Corpus

Duha Aldebakel DALDEBAK@UCSD.EDU

Yu Cao Y6CAO@UCSD.EDU

Anthony Limon A1LIMON@UCSD.EDU

Rui Zhang R2ZHANG@UCSD.EDU

DSC180B A06

Editor:

Abstract

Topic modeling allows us to fulfill algorithmic needs to organize, understand, and annotate documents according to the discovered structure. Given the vast troves of data and the lack of specialized skillsets, it is helpful to extract topics in an unsupervised manner using Latent Dirichlet Allocation (LDA). LDA is a generative probabilistic topic model for discrete data, but unfortunately, solving for the posterior distribution of LDA is intractable, given the numerous latent variables that have cross dependencies. It is widely acknowledged that inference methods such Markov Chain Monte Carlo and Variational Inference are a good way forward to achieve suitable approximate solutions for LDA. In this report, we will explore both these methods to solve the LDA problem on the Wikipedia corpus. We find that better performance can be achieved via preprocessing the data to filter only certain parts-of-speech via lemmatization, and also exclude extremely rare or common words. We improved on the Expectations-Maximization (EM) Algorithm used for variational inference by limiting the number of iterations in the E step even if sub-optimal. This leads to benefit of faster runtimes and better convergences due to fewer iterations and avoidance of local minima. Finally, we explore early stopping runtimes on under-parameterized LDA models

to infer the true dimensionality of the Wikipedia vocabulary to solve for topics. While the English language has around a million words, our findings are that it only takes around fifteen thousand words to infer around twenty major topics in the dataset.

1. Introduction

As more data and information in the world becomes available, it not only provides the opportunity to learn, interact with, and apply that knowledge, but it makes it more difficult to conveniently organize, understand, and extract usefulness out of this data. Thus, we have algorithmic needs to help us with this kind of task involving data such as massive collections of electronic text. Specifically, topic modeling provides us with methods for automatically organizing, understanding, and summarizing these large collections of text data without actually having to read through every document of these massive text archives. Topic modeling allows us to discover the hidden thematic structure in data, such as text data, to annotate documents according to the discovered structure. We can then use these annotations to organize, summarize, and search the documents. One particular topic model is Latent Dirichlet Allocation (LDA).

The intuition behind LDA is the assumption that documents exhibit multiple topics, as opposed to the assumption that documents exhibit a single topic. We can elaborate on this by describing the imaginary generative probabilistic process that we assume our data came from. LDA first assumes that each topic is a distribution over terms in a fixed size vocabulary. LDA then assumes documents are generated as follows:

1. A distribution over topics is chosen, for each document

2. For each word in a document, a topic from the distribution over topics is chosen

3. A word is drawn from a distribution over terms associated with the topic chosen in the previous step.

In other words, a document might have high associations with topics x, y and z. For a particular word, We might choose a topic x to be expressed. Based on this topic x we choose a word from the distribution over terms associated with topic x. This is repeated for every word in a document and is how our document is generated. We repeat this for the next document in our collection, and thus a new distribution over topics is chosen and its words are chosen in the same process. It is important to note that the topics across each document remain the same, but the distribution of topics and how much each document exhibits the topics changes. Another important observation to point out is that this model has a bag-of-words assumption, in other words, the order of the words doesn't matter. The generative process isn't meant to retain coherence, but it will generate documents with different subject matter and topics.

Now that we have explained the generative process, we can reiterate the statistical problem that is we cannot observe the hidden structure, we only assume it exists. Thus, we want to solve this problem by inferring all of the values of the hidden variables: the topic proportions associated with each document, the topics associated with each word, and the distribution over terms that forms the documents in a collection.

LDA as a graphical model (in figure 1) allows us to describe the generative process as well as define a factorization of the joint probability of all of the hidden and observed random variables. It can help us infer the hidden variables given the observations by writing down the algorithms that can solve this problem. The only variable that we observe is the words in every document, represented by $W_{d,n}$, and therefore, its circle is shaded. The other variables are latent variables and not shaded. The boxes represent multiplicity since there are D documents, K topics, and N words in every document, and correspondingly the number of variables and thus multiplied.

The joint probability defines a posterior in which we want to infer from a collection of

documents, the topic assignments for each word $z_{d,n}$, the topic proportions for each document θ_d, and the topic distributions for each corpus β_k. We can then use those posterior inferences to perform varying tasks. In summary, the hidden structure is uncovered by computing the posterior, and then the uncovered structure can be used to perform tasks. This is done by using all the hidden variables that we assume existed in the collections of data and discovered through the posterior distribution.

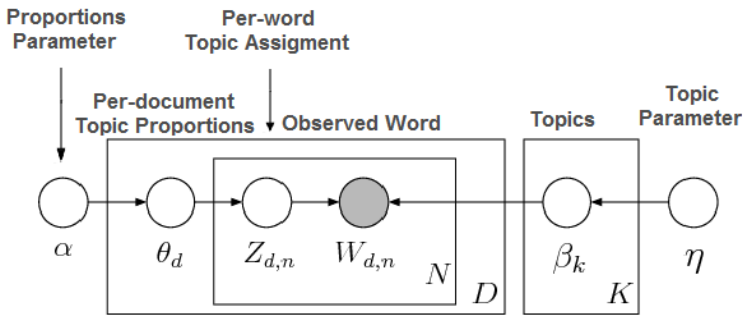


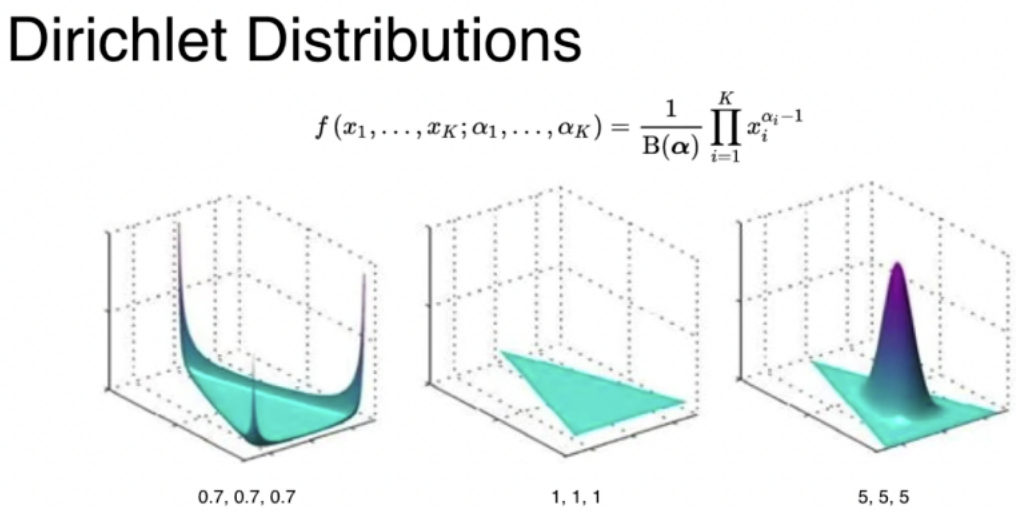
Figure 1: Graphical model of LDA

Building off our work from Q1, we find that solving the LDA posterior directly is intractable due to the dependence between the latent parameters from the two pathways of topics and topic assignments to the observed variable. Fortunately, we have options for approximate inference techniques such as variational inference (VI) and Monte Carlo Markov Chains (MCMC), and we will explore both of these in this report.

1.1 Probability distributions of LDA model

This section will go into further detail about the probability distributions governing the LDA model. From Figure 1, we can see α on the far left and η on the far right. Both of these are hyperparameters of the Dirichlet distribution, from which we draw θ_d for the topic proportions and β_k for the terms in topics respectively.

These parameters should be smaller than 1, as they are concentration parameters. When they are larger than 1, the distribution becomes focused in the middle, which means that the chances are that every topic will be meaningful in a document for α, or every term is meaningful for a given topic for η. In practice, we know that only a section of topics or words are meaningful, and so we use $\alpha = 0.1$ and $\eta = 0.01$.



The name of this topic model, Latent Dirichlet Distribution, probably comes from the fact that the Dirichlet distribution plays such an important role, and that it is latent and not observed directly.

The Dirichlet distributions are conjugate priors to the multinomial distribution, meaning that we draw samples from the two Dirichlet distributions to get θ_d for the topic proportions and β_k for the terms in topics respectively. We then first draw $Z_{d,n}$ for every word in the document from the multinomial distribution parameterized by θ_d to get one exact topic out of the K topics for every word. Conditioning on this topic, we draw from the relevant multinomial distribution parameterized by the β_k matching the topic $Z_{d,n}$, to get the exact word or term that is popular with the same topic.

2. Methods

2.1 Data Collection

2.1.1 Previous work

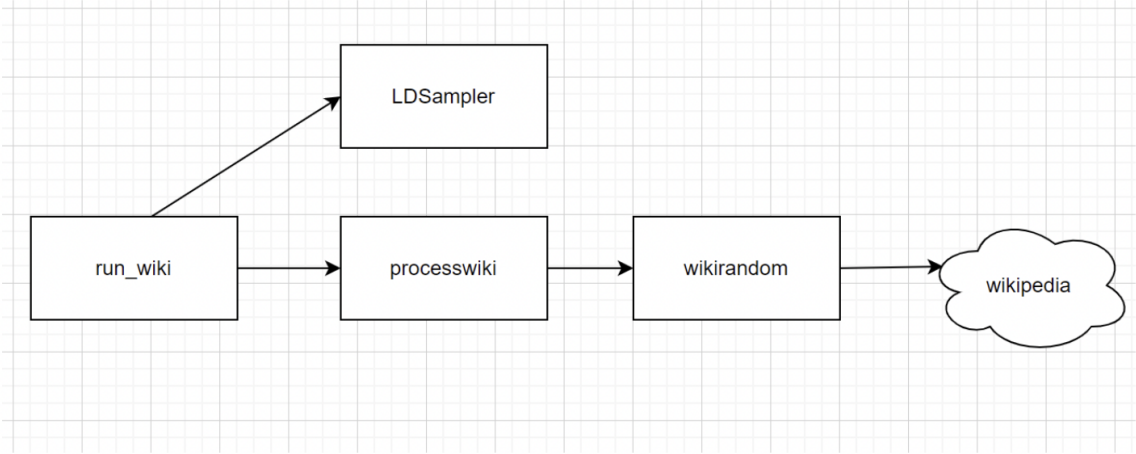


Figure 2: Retooling existing paradigm

We explored existing work done by Sam Patterson and Yee Whye Teh, who have kindly published their code on their webpage. Their existing code is in Python 2 and the structure of the library is laid out in the figure above.

Data is in the form of Wikipedia Articles, which are downloaded on the fly from the web as needed by processwiki. Since data is not saved for future runs, and due to rate-limiting throttling from Wikipedia.com, it takes a while to run. Due to the randomness of generating data, exact experiments cannot be replicated. The original authors suggested that the code could be modified in the future to download many articles via threads and to cache the data, but this was not yet implemented.

2.1.2 Our implementation

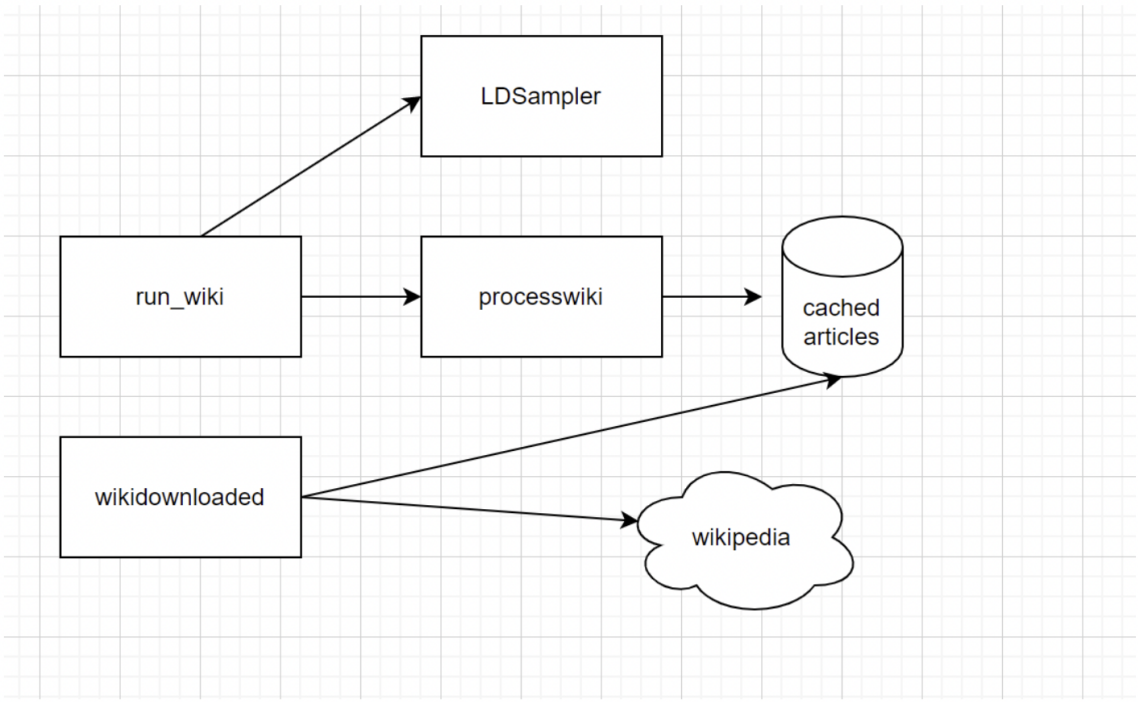


Figure 3: Current paradigm

Our implementation will be to create a standalone program, wikidownloader that would download articles from Wikipedia in the background, and saved them as compressed pickle files. We will be using the latest libraries in python 3 as best practice, and also updated the code to reflect the latest URLs and HTML tags that have evolved over the years.

Reginald Munn

From Wikipedia, the free encyclopedia

Lieutenant-Colonel **Reginald George Munn**, CMG (20 August 1869 – 12 April 1947) was a British Indian Army officer and English first-class cricketer who played a single first-class game for Worcestershire against Marylebone Cricket Club (MCC) in 1900; he was bowled by Dick Pougher for 2 in his only innings.

Munn was born in Madresfield, Worcestershire. He was commissioned a second lieutenant in the Derbyshire Regiment on 23 March 1889, and promoted to lieutenant on 1 November 1890. The following year, he was admitted to the Indian Staff Corps in September 1891,[1] and attached to the 36th Sikhs. He served with the Chitral Relief Force in 1895, and on the North West Frontier of India in 1897–98. He was promoted to captain on 23 March 1900, and from November 1901 served as Aide-de-camp to Colonel Charles Egerton, Commander of the Punjab Frontier Force.[2] He was mentioned in a despatch from Egerton dated 4 July 1902, following operations against the Mahsud Waziris.[3]

Munn served through the First World War, and was appointed a Companion of the Order of St Michael and St George (CMG) in the 1919 New Year Honours.[4]

Munn died aged 77 in Virginia Water, Surrey.[5]

References [edit]

1. ^ "No. 26390". *The London Gazette*. 7 April 1893. p. 2121.
2. ^ Hart´s Army list, 1903
3. ^ "No. 27462". *The London Gazette*. 8 August 1902. pp. 5089–5092.
4. ^ "No. 31097". *The London Gazette* (Supplement). 1 January 1919. p. 82.
5. ^ "Deaths". *The Times*. 15 April 1947. p. 1.

Figure 4: An example of a Wikipedia article

An example of a Wikipedia article is shown in the figure above. Several decisions have to be made in data collection. For example, do we collect text data from tables and graphs? Do we collect data from the "References" section?

External links [edit]

- Reginald Munn at ESPNcricinfo
- Statistical summary from CricketArchive

This biographical article related to the British Army is a stub. You can help Wikipedia by expanding it.

This biographical article related to an English cricket person born in the 1870s is a stub. You can help Wikipedia by expanding it.

Figure 5: An example of a Wikipedia article post-script

Finally, the postscript of the article provides hints of the topic under description. We believe it would be a pitfall to capture that since it tells us that topic, which in this case might be British Army and English cricket. Therefore, we only capture the main text on the Wikipedia page, and we above the references, external links, or footnotes.

2.2 Data Preprocessing

2.2.1 Tokenization

Our first step in preprocessing is tokenization. Tokenization is the process of taking the article text as a string, converting to lowercase, removing punctuation and other non-alphabets, and splitting the remaining text into a list of words. These words can then be assigned an integer ID so that the corpus can be transformed into a sparse matrix of word counts. An example of the tokenization process is in the figure below.

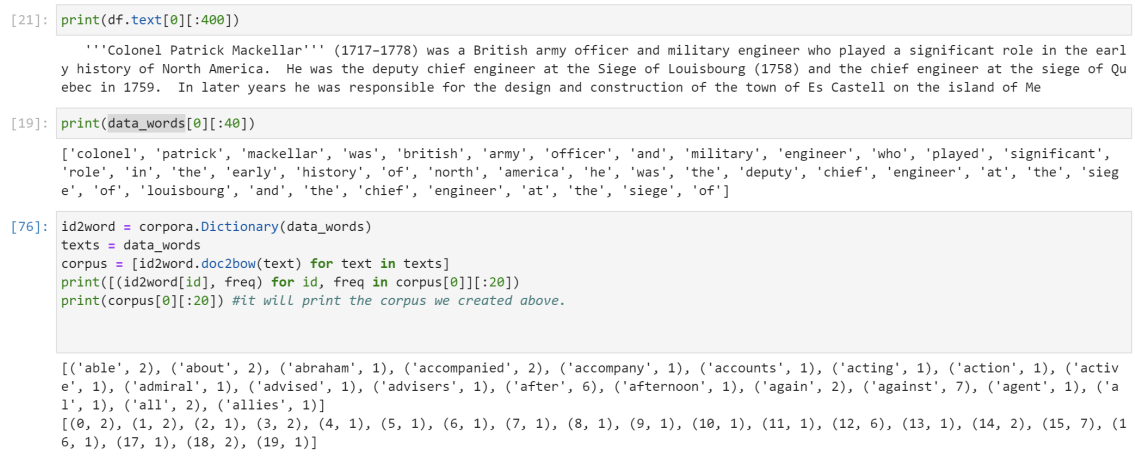


Figure 6: An example of the tokenizaton process

2.2.2 N-grams

N-grams are N consecutive tokens in the text. Tokens are basically unigrams while bi-grams are two consecutive words. N-grams are useful when two words frequently occur together, such as "New York", and we merged them into a single entity as they are only meaningful

together. We use the Spacy library and its linguistic model to form meaningful bigrams. For example, in this article about geology, only the terms 'family' and 'gelechiidae' are merged into a bigram 'family_gelechiidae' since they are linked together and improves the meaning of the bags of N-grams as the geological "family" is now distinguished from the more common use of the word "family".

```
print(data_words_nostops[0][:40])
```

```
['megacraspedus', 'cuencellus', 'moth', 'family', 'gelechiidae', 'described'
rance', 'spain', 'forewings', 'uniform', 'mouse', 'grey', 'margin', 'whitish
s', 'mouse', 'grey']
```

```
print(data_words_bigrams[0][:40])
```

```
['megacraspedus', 'cuencellus', 'moth', 'family_gelechiidae', 'described', '
ce', 'spain', 'forewings', 'uniform', 'mouse', 'grey', 'margin', 'whitish',
'mouse', 'grey']
```

Figure 7: An example of Bi-Gram

2.2.3 Lemmatization

Lemmatization is a process of grouping together the inflected forms of a word. The first step is to apply a language model to tag each word into a part-of-speech (PoS). We can then assign the base form for each word. For example, the lemma of "was" is "be", and the lemma of "plays" is "play".

We use the python spacy library for Lemmatization. It supports more than 64 languages, and so it allows us to extend our analysis to articles in other languages in the future. It follows the Universal Part-of-Speech (UPoS) tagset (Petrov et al., 2012), and lists the following 17 categories: adjective(ADJ), adposition (ADP), adverb (ADV), auxiliary (AUX), coordinating conjunction (CCONJ), determiner (DET), interjection (INTJ), noun (N), numerical (NUM), particle(PART), pronoun (PRON), proper noun (PROPN), punctuation (PUNCT), subordinating conjunction (SCONJ), symbol (SYM), verb (VERB) and other

(X).

With the tags, we can remove the parts of speech that do not give valuable information about the article. For example, if we believe that verbs and adjectives are not informative, we will avoid including words that are tagged as VERB or ADJ.

In a preliminary run with a corpus of 50k articles, we obtained a log perplexity score of -15.2 with n-gram tokens, but a better log perplexity score of -14.0 with lemmatization.

2.3 Performance Measures of LDA models

Recall that LDA is an unsupervised learning model, and there are no target labels to match. This means that typical comparisons between actual and desired outputs cannot be done, along with loss ratios and other measures. Instead, our measure of goodness on an output would have to be based on likelihood measures on unseen data to test our approximate inferred posterior distributions. This is related to the concept of entropy as explained in the subsections below. Since the likelihood becomes exponentially small as the number of words increases, it is customary to normalize it to a per-word likelihood in measures such as perplexity.

2.3.1 Entropy

Information entropy measures the average level of unpredictability inherent in a random variable. For example, a biased coin that always lands on heads (or a trick coin that only has heads) has an entropy of 0, since it is very predictable. On the other hand, a fair coin that has a 50-50 split between heads and tails will have an entropy of 1, and would have the highest entropy of all coins including the biased ones.

Formally, entropy (H(X)) of random variable X is:

$$H(X) = E[-log(p(X))]$$

and for a discrete domain:

$$H(X) = -\sum p(x_i) log(p(x_i))$$

The plot of entropies of a coin with various biases is shown in the figure below.

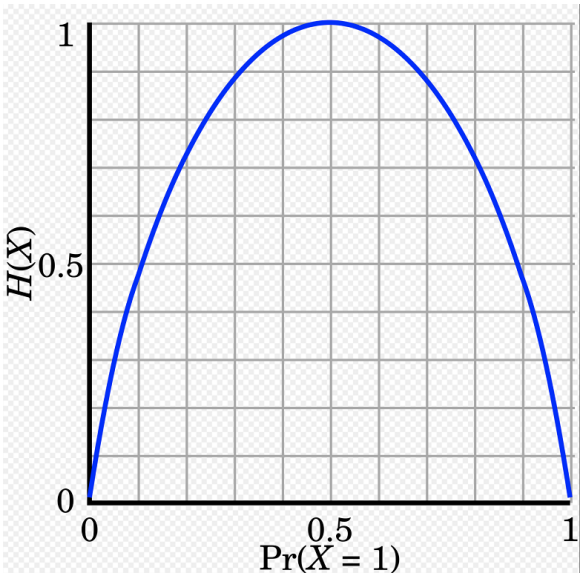


Figure 8: Plot of entropy of a coin with different biases

2.3.2 Cross-entropy

Often, we do not actually know the target probability distribution p(X), and we approximate it with another probability distribution q(X). When this occurs, it is helpful to define cross-entropy as a distance measure between the two distributions.

$$H(p, q) = E_{p(X)}[-log(q(X))]$$

For a discrete domain:

$$H(p, q) = -\sum p(x_i) log[q(x_i)]$$

Note that the sum is maximized when $p(x_i)$ matches $q(x_i)$ for every x_i. Taking into account the negative sign, therefore, the cross-entropy H(p,q) is minimized when p=q. In that case, H(p,q) becomes the entropy H(p).

2.3.3 Kullback -Leibler divergence

The posterior distribution is usually denoted by $p(\theta|x)$ while the variational posterior is denoted by $q(\theta|x)$. The variational posterior $q(\theta|x)$ is the closest member within the variational approximate family Q to the target posterior $p(\theta|x)$. That is, $q \in Q$.

To define this "closeness", the Kullback–Leibler divergence (D_{KL}) is used to measure the distance between these two probability distributions. It is often used interchangeably as a likelihood measure. D_{KL} is also known as relative entropy and is defined by the following integral:

$$D_{KL}(q(\theta)|p(\theta|x)) = \int_{-\infty}^{\infty} q(\theta|x) log(\frac{q(\theta|x)}{p(\theta|x)})\, d\theta \tag{1}$$

In expectations notation, this is equivalent to

$$D_{KL}(q(\theta)|p(\theta|x)) = E_q(log(q(\theta)) - E_q(log(p(\theta|x))) \tag{2}$$

Then the best candidate $q(\theta|x)$ would be:

$$q(\theta|x) = argmin_\theta E_q(log(q(\theta|x)) - E_q(log(p(\theta|x))) \tag{3}$$

To avoid having to calculate $p(\theta|x)$, using Bayes' rule, note

$$D_{KL}(q(\theta)|p(\theta|x)) = E_q(log(q(\theta) - log(p(x,\theta)) + log(p(x)) \tag{4}$$

Defining Evidence Lower Bound or ELBO(q) as:

$$ELBO(q) = L(q) = -E_q(log(q(\theta) - log(p(x,\theta))) \tag{5}$$

We get the following relationship between KL and ELBO:

$$D_{KL}(q(\theta)|p(\theta|x)) = -ELBO(q) + log(p(x)) \tag{6}$$

$$log(p(x)) = D_{KL}(q(\theta)|p(\theta|x)) + ELBO(q) \tag{7}$$

Therefore, instead of minimizing $D_{KL}(q(\theta)|p(\theta|x))$, we can equivalently maximize ELBO(q), since they both sum to a constant.

Note that ELBO has two terms. The first term is just the entropy of q, while the second term is related to the cross entropy between p and q.

$$ELBO(q) = H(q) - H(q,p) \tag{8}$$

2.4 Perplexity of a LDA model

Perplexity is related tightly related to ELBO, with two main differences. (1) It flips the sign such that the desired level is a minimum rather than maximum. (2) It calculates the measure on a per-word level, rather than on the entire document, therefore normalizing extremely small likelihoods due to compounding imposed on longer documents versus shorter documents.

According to Hofmann [12], perplexity refers to the log-averaged inverse probability on unseen data. The perplexity of our LDA model can be defined as the exponential of the cross-entropy between $\tilde{p}$ and q. That is,

$$perplexity = e^{H(\tilde{p},q)} = e^{-\sum \tilde{p}(x)log(q(x))} \tag{9}$$

Note that since we don't know p, we use $\tilde{p}$ instead, which is the empirical distribution of samples drawn from the true distribution p.

Assuming there are N sample tokens drawn, the formula simplifies to,

$$perplexity = e^{\sum_{x \in \tilde{p}} (log(q(x))^{-1})/N} \tag{10}$$

which explains why Hoffman calls it the "log-averaged inverse probability".

Cancelling the log with the exponential, we see that the equation is equivalent to the geometric mean of the inverse token probabilities. This is another popular form of the definition of perplexity but the two forms are mathematically equivalent.

$$perplexity = \sqrt[N]{\prod_{x \in \tilde{p}} (1/log(q(x))} \tag{11}$$

Since this is a measure on token probabilities, a longer or shorter sentence in the unseen test set will not affect the measure.

In practice, we optimize our variational inference model using KL divergences, but we report perplexity as an afterthought by dividing the log probabilities by the number of words and then using it as the negative exponent:

$$perplexity = 2^{-ELBO_Perword(q)} \tag{12}$$

```
def log_perplexity(self, chunk, total_docs=None):                    [docs]
    """
    Calculate and return per-word likelihood bound, using the `chunk` of
    documents as evaluation corpus. Also output the calculated statistics. incl.
    perplexity=2^(-bound), to log at INFO level.

    """
    if total_docs is None:
        total_docs = len(chunk)
    corpus_words = sum(cnt for document in chunk for _, cnt in document)
    subsample_ratio = 1.0 * total_docs / len(chunk)
    perwordbound = self.bound(chunk, subsample_ratio=subsample_ratio) / (subsample_ratio * corpus_w
    logger.info("%.3f per-word bound, %.1f perplexity estimate based on a held-out corpus of %i doc
                (perwordbound, numpy.exp2(-perwordbound), len(chunk), corpus_words))
    return perwordbound
```

Figure 9: Exponent of perplexity is -ELBO per word

```
def bound(self, corpus, gamma=None, subsample_ratio=1.0):                    [docs]
    """
    Estimate the variational bound of documents from `corpus`:
    E_q[log p(corpus)] - E_q[log q(corpus)]

    `gamma` are the variational parameters on topic weights for each `corpus`
    document (=2d matrix=what comes out of `inference()`).
    If not supplied, will be inferred from the model.

    """
    score = 0.0
    _lambda = self.state.get_lambda()
    Elogbeta = dirichlet_expectation(_lambda)

    for d, doc in enumerate(corpus):  # stream the input doc-by-doc, in case it's too large to fit
        if d % self.chunksize == 0:
            logger.debug("bound: at document #%i", d)
        if gamma is None:
            gammad, _ = self.inference([doc])
        else:
            gammad = gamma[d]
        Elogthetad = dirichlet_expectation(gammad)

        # E[log p(doc | theta, beta)]
        score += numpy.sum(cnt * logsumexp(Elogthetad + Elogbeta[:, id]) for id, cnt in doc)

        # E[log p(theta | alpha) - log q(theta | gamma)]; assumes alpha is a vector
        score += numpy.sum((self.alpha - gammad) * Elogthetad)
        score += numpy.sum(gammaln(gammad) - gammaln(self.alpha))
        score += gammaln(numpy.sum(self.alpha)) - gammaln(numpy.sum(gammad))

    # compensate likelihood for when `corpus` above is only a sample of the whole corpus
    score *= subsample_ratio

    # E[log p(beta | eta) - log q (beta | lambda)]; assumes eta is a scalar
    score += numpy.sum((self.eta - _lambda) * Elogbeta)
    score += numpy.sum(gammaln(_lambda) - gammaln(self.eta))

    if numpy.ndim(self.eta) == 0:
        sum_eta = self.eta * self.num_terms
    else:
        sum_eta = numpy.sum(self.eta, 1)

    score += numpy.sum(gammaln(sum_eta) - gammaln(numpy.sum(_lambda, 1)))
    return score
```

Figure 10: Code for function bound calculates the ELBO

2.5 LDA implementation with Gibbs sampling

In Q1 we explored MCMC methods, specifically the Metropolis-Hastings algorithm, to solve the problem of computing high-dimensional integrals. We learned that this particular class of sampling methods was especially effective in performing probabilistic inferences which can be applied to solve problems specifically within the field of Bayesian inference and learning. We applied this particular MCMC method for sampling posterior distributions, since the posterior probability distribution for parameters given our observations proved to be intractable, hence the use of sampling algorithms for approximate inference.

LDA is similar in that its aim is to infer the hidden structure of documents through posterior inference. Since we cannot exactly compute the conditional distribution of the hidden variables given our observations, we can use MCMC methods to approximate this intractable posterior distribution. Specifically, we can use Gibbs sampling to implement LDA. Gibbs sampling is another MCMC method that approximates intractable probability distributions by consecutively sampling from the joint conditional distribution:

$$P(W, Z, \phi, \theta|\alpha, \beta).$$

This models the joint distribution of topic mixtures θ, topic assignments Z, the words of a corpus W, and the topics ϕ. In the approach we use, we leverage the collapsed Gibbs sampling method in not representing ϕ or θ as parameters to be estimated, and only considering the posterior distribution over the word topic assignments $P(Z|W)$. By using Dirichlet priors on ϕ and θ, this means α and β are conjugate to the distributions ϕ and θ. This allows us to compute the joint distribution by integrating out the multinomial distributions ϕ and θ:

$$P(Z|W, \alpha, \beta).$$

Then because W is observed we only need to sample each $z_{m,n}$, that is, the topic assignment for the n'th word in the m'th document, given all other topic assignments and the

observations, not including the current topic assignment.

$$P(z_{m,n}|Z_{\neg m,n}, W, \alpha, \beta)$$

This conditional distribution is required to construct a Markov chain by sampling the next state based on the current state, this MCMC approach is known to converge to the target distribution, or in our case the posterior. Each state is an assignment of values to the variables being sampled, in our case the topic assignment z, and the next state is achieved by sampling all variables from their distribution conditioned on the current values of all other variables and the observations. Thus, we obtain a conditional distribution where for a single word w in document d that is assigned topic k, the conditional distribution becomes the sum of two ratios.

$$P(z_{m,n} = k|Z_{\neg m,n}W, \alpha, \beta) \propto \frac{\sigma_{m,k}^{\neg m,n} + \alpha_k}{\sum_{i=1}^{K} \sigma_{m,i}^{\neg m,n} + \alpha_i} * \frac{\delta_{k,w_{m,n}}^{\neg m,n} + \beta_{w_{m,n}}}{\sum_{r=1}^{V} \delta_{k,r}^{\neg m,n} + \beta_r}$$

The first term being the ratio of word w in topic k, multiplied by the second term which equals the ratio of topic k in document d. The product is a vector which assigns a weight to each topic k, and it's normalization is the probability of the topic assignment of the next state based on the current state. For our context, the next state is the assignment of the current word to a topic k. For example, a word w with a high ratio in topic k and a topic k with a high ratio in document d is indicative that the topic assignment k for that word w is more likely for that word w in document d.In order to compute this full conditional distribution we keep tack of these statistics as we sweep through all the documents in a collection. And our estimated of ϕ and θ can be implemented and calculated with the following:

$$\theta_{j,k} \approx \frac{\sigma_{j,k} + \alpha_k}{\sum_{i=1}^{K} \sigma_{j,i} + \alpha_i}$$

$$\phi_{k,v} \approx \frac{\delta_{k,v} + \beta_v}{\sum_{r=1}^{V} \delta_{k,r} + \beta_r}$$

Where $\sigma_{j,k}$ is the number of times we see topic k in document j, and $\delta_{k,v}$ is the number of times topic k has been chosen for word type v. This is how we derive the final estimates for θ and ϕ, and we set implement this algorithmically to compare MCMC sampling methods against other inference methods.

2.6 LDA implementation with Mean-Field Variational Inference

We ran some baseline tests using the Gensim python library initially but later reverted to the original onlineldavb.py package of functions for fitting LDA with online variational Bayes by Matthew D. Hoffman [13] so as to better understand and control the algorithm.

To fit LDA, the variational family chosen is a Mean-field Variational family that otherwise mirrors the distributions of the posterior. Recall that the posterior is intractable due to the dependence between variables arising from the two pathways of topic proportions and term proportions to get to the actual word, as shown in figure 1. Mean-field Variational Inference assumes that all the variables are independent so that the variational posterior can be fully factored. Namely, the variational family from which we fit q assumes that β, θ, z are independent and can be fully factored, such that:

$$q(\beta_k, \theta, z) = \prod_k q(\beta_k) \prod_d q(\theta_d) \prod_n q(z_{dn}) \tag{13}$$

To find the best q within this family to fit the posterior p, we introduce variational parameters that we tweak as we attempt to maximize the likelihood of the model to the Wikipedia corpus.

The variational parameter for topic assignments is ϕ.

$$q(z_{di} = k) = Multinomial(\phi_{dwk}) \tag{14}$$

The variational parameter for topic proportions is γ.

$$q(\theta_d) = Dirichlet(\theta_d; \gamma_d) \tag{15}$$

The variational parameter for topics is λ.

$$q(\beta_k) = Dirichlet(\beta_k; \lambda_k) \tag{16}$$

Note that probability distributions of two Dirichlet and one Multinomial have not changed from the original posterior p. Note that the posterior p also has

$$z_{di} \sim Multinomial(\theta)$$

$$\theta_d \sim Dirichlet(\alpha)$$

$$\beta_k \sim Dirichlet(\eta)$$

and these distributions are mirrored in q, but the big difference is the dependence of the terms in p (such as between θ and β) has been broken.

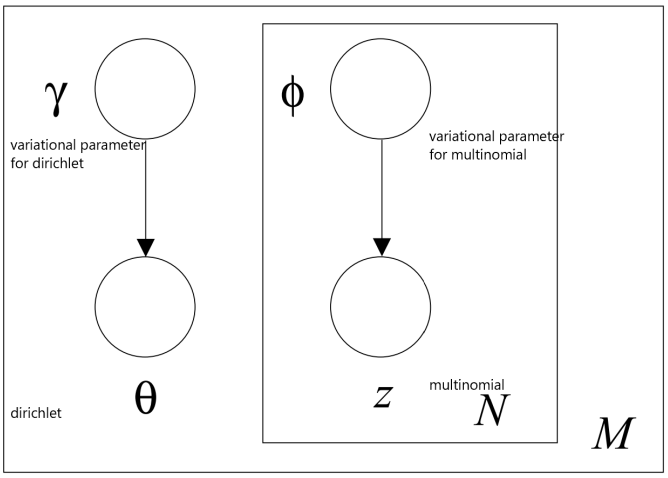


Figure 11: Graphical model of MFVI for LDA

The figure above shows the template for MFVI, where the arrows between variables are now broken and do not show the dependence. Instead, the arrows we see are between the variational parameters and the parameters. (The figure only illustrates for θ and z, and but the same is true for β that is outside of this figure.)

From this template, Hoffman developed an Expectations Maximization (EM) algorithm, which consists of E steps and a M steps. At each stage or sub-stage, due to our assumption of independence, we solve for the best $q(z_{di})$, $q(\theta_d)$, or $q(\beta_k)$ while holding the other variables constant and assuming that they are correctly specified. The E step solves this for topic proportion (left pathway in figure 1) by assuming that topics is correctly specified. It does this by varying ϕ and γ to optimize $q(z_{di})$ and $q(\theta_d)$. The algorithm loops until no further improvement is shown. Then, the M step solves for the topics (right pathway in figure 1) by assuming that topic proportions are correctly specified. It does this by varying λ to optimize $q(\beta_k)$. The full algorithm is shown in the pseudo-code below.

Algorithm 1 Batch variational Bayes for LDA

Initialize $\boldsymbol{\lambda}$ randomly.
while relative improvement in $\mathcal{L}(\boldsymbol{w}, \boldsymbol{\phi}, \boldsymbol{\gamma}, \boldsymbol{\lambda}) > 0.00001$ **do**
  E step:
  for $d = 1$ to D **do**
    Initialize $\gamma_{dk} = 1$. (The constant 1 is arbitrary.)
    repeat
      Set $\phi_{dwk} \propto \exp\{\mathbb{E}_q[\log \theta_{dk}] + \mathbb{E}_q[\log \beta_{kw}]\}$
      Set $\gamma_{dk} = \alpha + \sum_w \phi_{dwk} n_{dw}$
    until $\frac{1}{K} \sum_k |\text{change in} \gamma_{dk}| < 0.00001$
  end for
  M step:
  Set $\lambda_{kw} = \eta + \sum_d n_{dw} \phi_{dwk}$
end while

Figure 12: MFVI EM Algorithm for LDA

2.6.1 Our improvements to the MFVI algorithm

We make two tweaks to the MFVI algorithm as described above.

Firstly, we find it wasteful to repeat the E step indefinitely to find the perfect variational parameters ϕ and γ under the assumption that $q(\beta_k)$ is correctly specified because we know that that assumption is certainly false at the start of the algorithm when λ is assigned randomly. The focus should be to process as many documents as possible for an online algorithm and not be caught up with "perfect as the enemy of good". As the algorithm converges upon many passes, we will expect that the E step will not require many iterations anyway, as it will just be incremental changes at that point. Therefore, we changed the code to take the maximum number of E step iterations as a parameter, so that we can vary it from say 5 to 80. The results are shown in the results section of this write-up.

Secondly, we are motivated by Shen, Gao, and Ma [11] to find means to under-parameterize the MFVI model so that we can run experiments on early stopping, and in doing so, get hints about the true dimensionality of the underlying processes. This would answer the key question as to how many critical words are really needed to distinguish topics from the corpus, and bear light as to whether a human looking at the top 20 or 100 words can do this effectively. We do this under-parameterization by keeping the structure of the algorithm but no longer changing some of the λ variational parameters effectively fixing them

at baseline probabilities. We remove variational parameters from the most popular words first, as they are assumed to have the most commonality and hence less useful to distinguish topics. This parameterization removal does not change the size of the dataset nor the likelihood calculations so we can do an apples-to-apples comparison across under and over parameterized models. More details and results are available in the results section of this write-up.

3. Results

3.1 Preprocessing Experiments

3.1.1 Minimal Preprocessing - Tokenization and N-grams

With minimal preprocessing, we remove punctuation and tokenize every article into unigrams and bigrams. There are 504 thousand words in the dictionary for which we count unigrams and bigrams frequencies. The large sparse matrix takes considerable time to run. In the figure below, we show the evolution of the likelihood and perplexity scores over the first 40 minutes of runtime on a machine with 4 CPU cores.

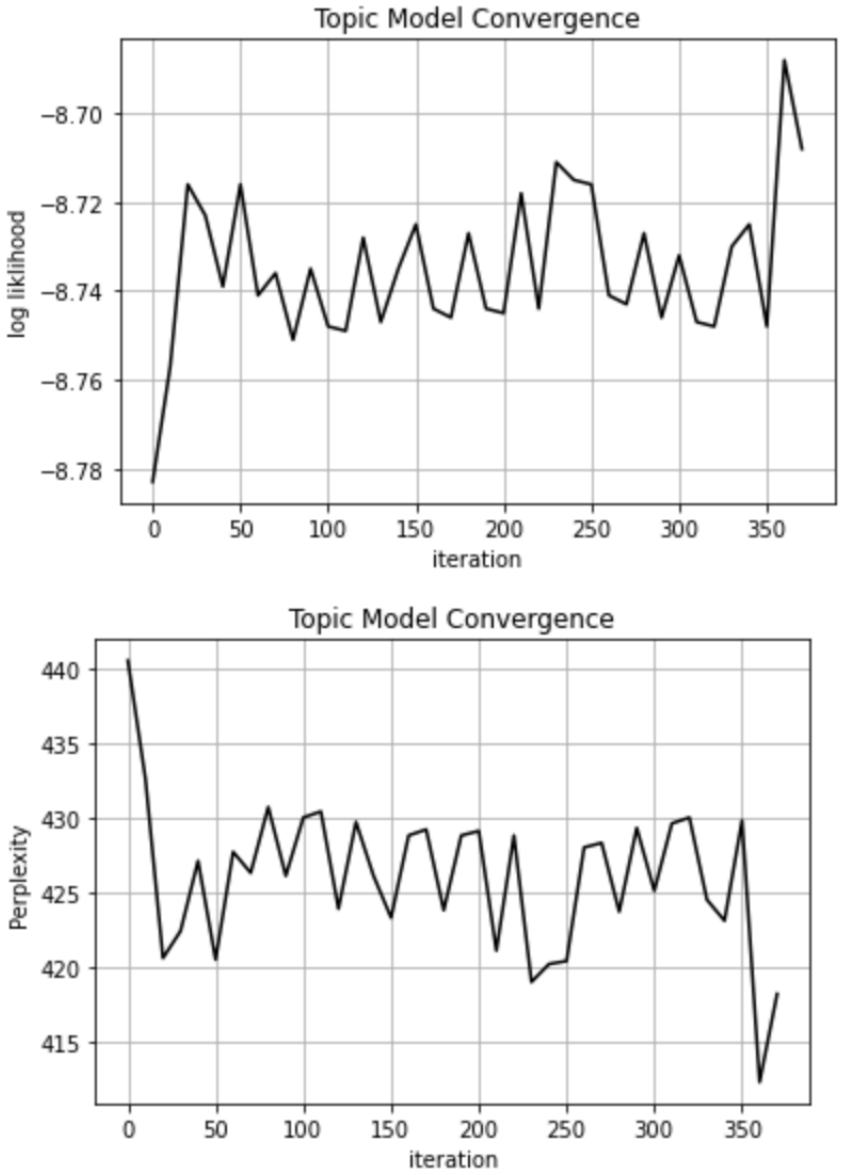


Figure 13: Optimal K value for Wikipedia dataset

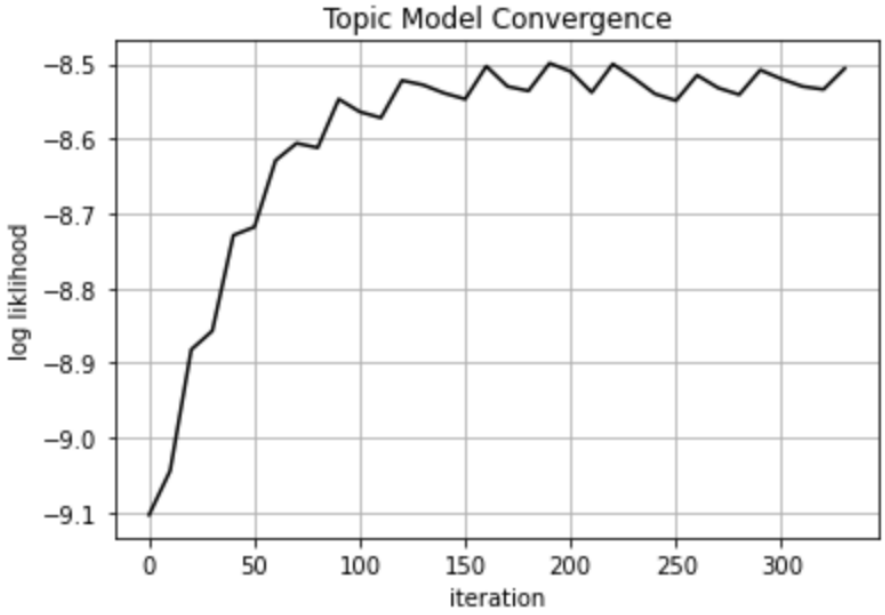
As evidenced above, the performance in terms of time (40 minutes) or convergence (with perplexity stuck at around 420 units) is not very good due to a large number of latent parameters and tokens.

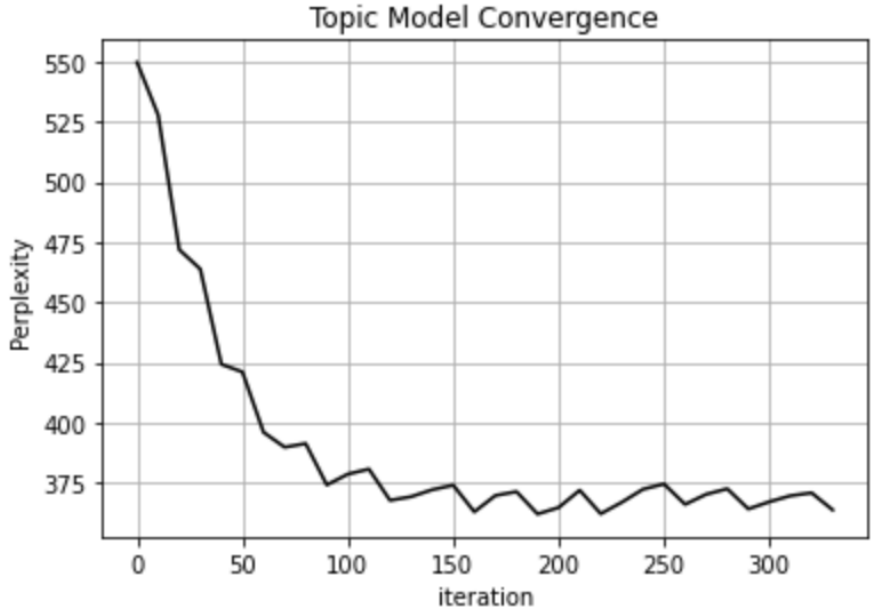
3.1.2 Extensive Preprocessing - Lemmatization and Dictionary Compression

We also ran the same experiment with extensive preprocessing. In addition to the above, We applied the lemmatization process with the spaCy library to infer the part-of-speech using the Universal Part-of-Speech (UPoS) tagset as per Petrov et al., 2012. It lists the following 17 categories: adjective (ADJ), adposition (ADP), adverb (ADV), auxiliary (AUX), coordinating conjunction (CCONJ), determiner (DET), interjection (INTJ), noun (N), numerical (NUM), particle(PART), pronoun (PRON), proper noun (PROPN), punctuation (PUNCT), subordinating conjunction (SCONJ), symbol (SYM), verb (VERB) and other (X). Of these categories, we choose to only retain nouns, verbs, adverbs, and adjectives as meaningful tokens for the purpose of inferring topics.

In addition, we dealt with a large number of rare yet uninformative tokens. We removed tokens that only appear in less than 0.1% of all documents. This reduced the number of tokens in the dictionary to 31 thousand.

The run time and convergence are both greatly improved. In the figure below, we show the evolution of the likelihood and perplexity scores over the first 3 minutes of runtime on a machine with 4 CPU cores. It is likely that convergence already occurred in around 2 minutes, as compared to 40 minutes in the prior example. In addition, we achieved a better perplexity score of around 375 as opposed to around 420 in the previous experiment.





Note: Perplexity estimate based on a held-out corpus of 4 documents

Figure 14: Likelihood and Perplexity evolution by iterations

3.2 Optimal value of K for Wikipedia dataset

We ran the gensim algorithm on the lemmatized Wikipedia dataset using different values of K to find the best number of topics as measured by the coherence score. Our finding is that a value of around 25 topics is optimal.

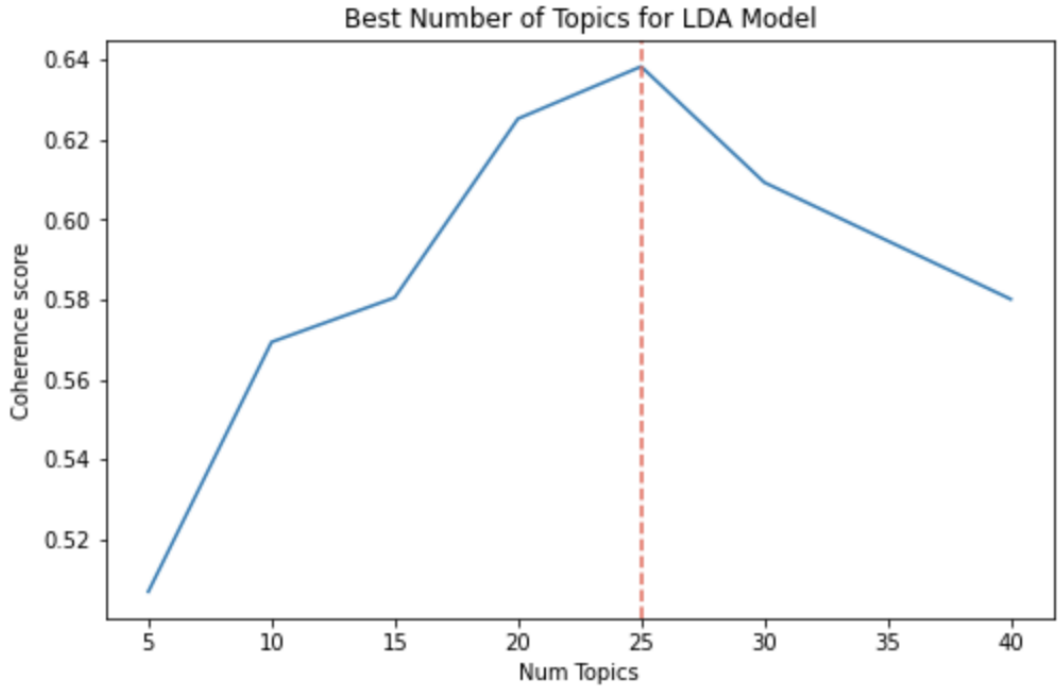


Figure 15: Optimal K value for Wikipedia dataset

3.3 Improving on EM Algorithm via limiting E step iterations

As explained in the methodology section above, we modified the code to impose a maximum number of iterations in the E-step so that the online algorithm can more quickly process more documents in the Wikipedia corpus while not trying to overly achieve the perfect topic proportions estimate when the topics matrix is itself a very crude estimate of reality.

We vary the maximum number of iterations between 5 and 80 as we find that, in practice, the number of iterations does not generally exceed 80 under the original conditions of

breaking the loop upon no incremental changes to γ. As expected, as can be seen in figure 16 below, the running time until early stopping (as defined in the next section) is faster as the maximum number of iterations is dropped to 5.

We are positively surprised that the computational performance is also proved for our training set of 24 thousand documents, as measured by perplexity on the holdout set of 1 thousand documents at early stopping. The figure shows a higher ELBO when the number of iterations is smaller, along with the error bars from multiple runs. We believe that this is a positive change to the original algorithm as it balances the computation of the E step with the M step more parsimoniously, and it prevents ELBO from getting stuck at a local maxima while the topics matrix is still in infancy.

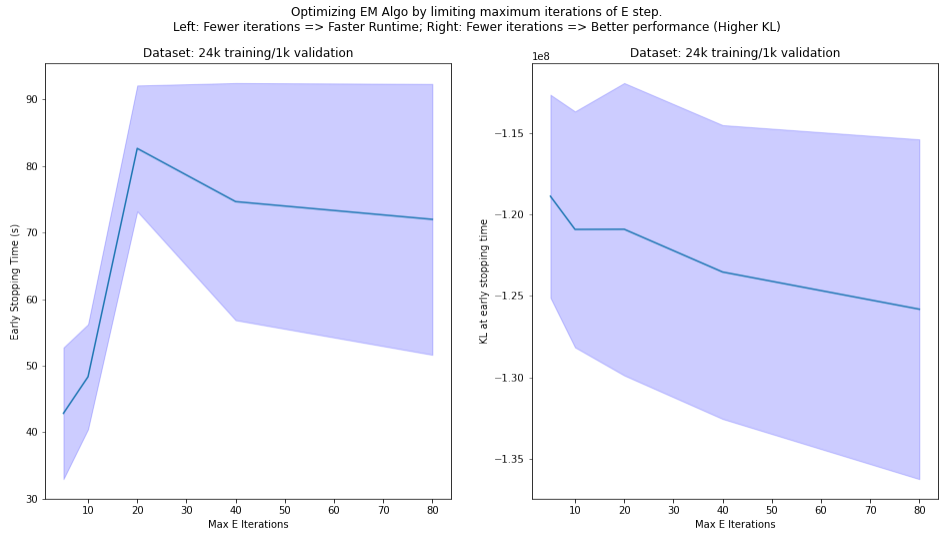


Figure 16: Optimal MaxE value for Wikipedia dataset

3.4 Optimal Early Stopping on Under and Over-Parametrization

Along the lines of Shen, Gao, and Ma [11], we explore optimal early stopping on under and over-parametrization of our LDA model. According to Shen, Gao, and Ma, early stopping is a simple and widely used method to prevent over-training neural networks, and they

developed theoretical results to show that (1) optimal early stopping time increases with sample sizes and (2) optimal early stopping time increases with model dimension when the model is under-parameterized, and then decreases with model dimension when it is over-parameterized.

While they focused on neural networks, we extend their findings with empirical results from our MFVI algorithm for LDA. Similar to them, we define early stopping when moving average performance on a validation set shows no improvement, with the caveat that we are measuring performance using perplexity scores for our unsupervised learning model without labeled data.

We find this research particularly exciting because it answers the question of the dimensions of the underlying data. The LDA model has been largely specified by a black box, with parameters governed by conjugate priors or hyperparameters, but it doesn't answer the human question of how many words are needed to master a unique topic. When viewing the results of an LDA model, human behavior would be to look at the relative frequencies of the top 20 words of each topic, but there is no requirement that the separation of topics lies in the next 20, 200 or even 2000 words. The experiment on early stopping times will enable us to answer the question as to how many words are needed to uniquely define and master a topic, just like the widely held view that it takes a human 10,000 hours to master a new skill (even if we feel good after 20 hours!)

We tried unsuccessfully to under-parameterize the model via trimming the Dirichlet distribution or increasing its concentration to fewer topics per document. The general result was an increase in early stopping time as parametrization increased until the original full model is reached. We believe that this is because handicapping the model unnaturally just forces early stopping as it finds a local minimum.

We find success by the more natural imposition on the number of parameters that are fit to discover the terms in topics, namely by reducing the number of variational parameters λ in our mean-field variational inference model. Note that we are not actually changing the dataset, which is now fixed after the preprocessing steps, as doing so would make the comparison over different datasets rather than different model parameterization. Instead, we under-parameterized our model by removing the variational parameters of the more common words (that are probably common across topics) to a baseline assumption.

Recall that we have already removed common or non-meaningful words (as defined by lemmatization tags) from our corpus. We have also removed extremely rare words so that topics are not learned from outliers. Along those lines, we found experimentally that the crux of separation of topics lies in the less frequently used (and yet above a rarity threshold) specialized words for each topic. These words should be included as parameters in our model, but it is less useful to consider common words that are shared by several or all topics. By sorting word frequencies on the corpus, we can vary the number of variational parameters of terms in topics that we are inferring by setting the more common words to baseline probabilities. This has the effect of setting our model to the desired dimension d as measured over parameters for topic vocabulary. We then run experiments on various model dimensions d and expect that the longest early stopping times will correspond to p=d where p is the natural dimension or number of unique words required to separate the K topics.

Our findings largely mirror the direction of Shen, Gao, and Ma in their neutral network experiment. The results are presented in the figure below. Similar to Shen, Gao, and Ma, we find better model outcomes (in our case measured by perplexity scores) for large model sizes. In addition, we find that small model sizes that are under-informative parameterizations and of low model capacity reach early stopping quickly. As we increase the model size, the stopping time increases until a peak and thereafter decreases, where the longest

times are measured to be at 10 thousand words for the 25 thousand document dataset, and 17.5 thousand words for the 50 thousand document dataset. If we can infer similarly from the prior work, where the peak is the underlying data dimension p equals the model dimension d, it would mean that while the English Language has around 1 million words, only around 15 thousand words are needed to distinguish the various topics on Wikipedia. Indeed, most of the convergence in perplexity has already occurred at such model sizes, but like prior work, a larger model beyond p is still helpful in incremental convergence as well as in running times.

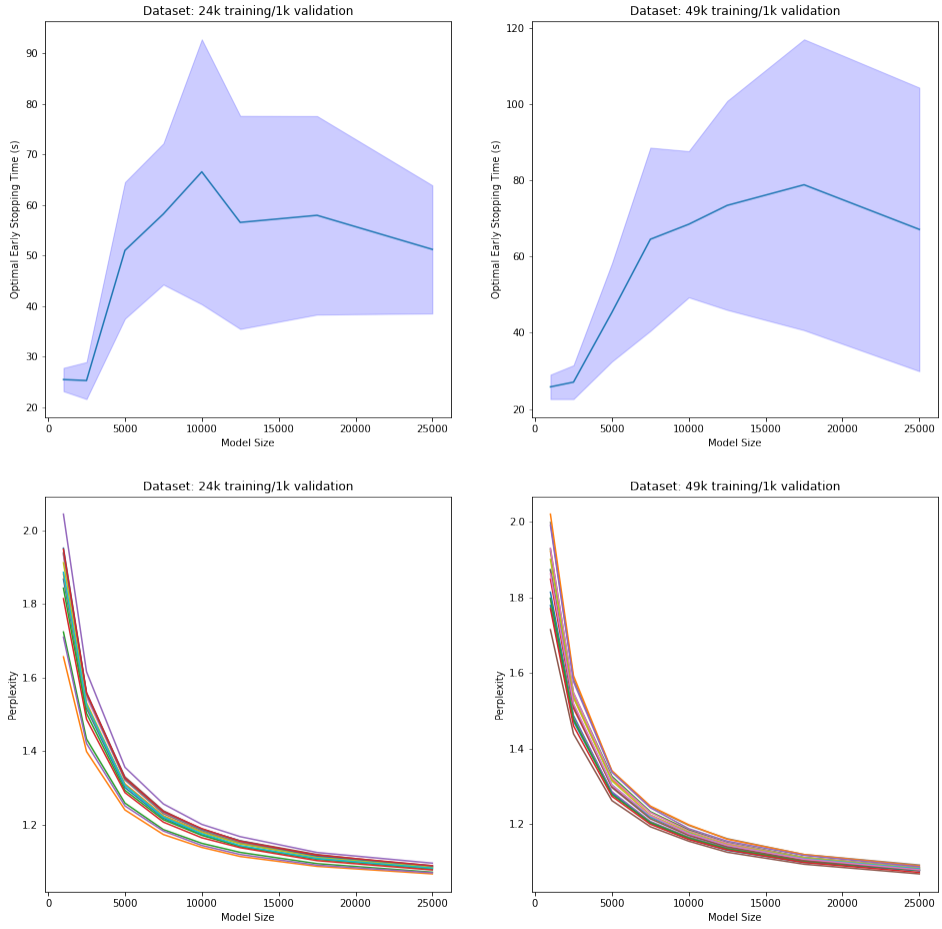


Figure 17: Early Stopping Times and Perplexity for Underinformed Models

3.5 Results from Gibbs sampling

We also developed code from scratch to perform Gibbs sampling to estimate the posterior. We obtain perplexity scores that are roughly similar to the scores from MFVI, showing that both methods achieve comparable and relatively good performance. However, it took 38 minutes to run the code, as opposed to just 3-4 minutes for MFVI. It is expected that

MFVI will have faster performance, and perhaps it is the better method in the world of exponentially rising data sizes.

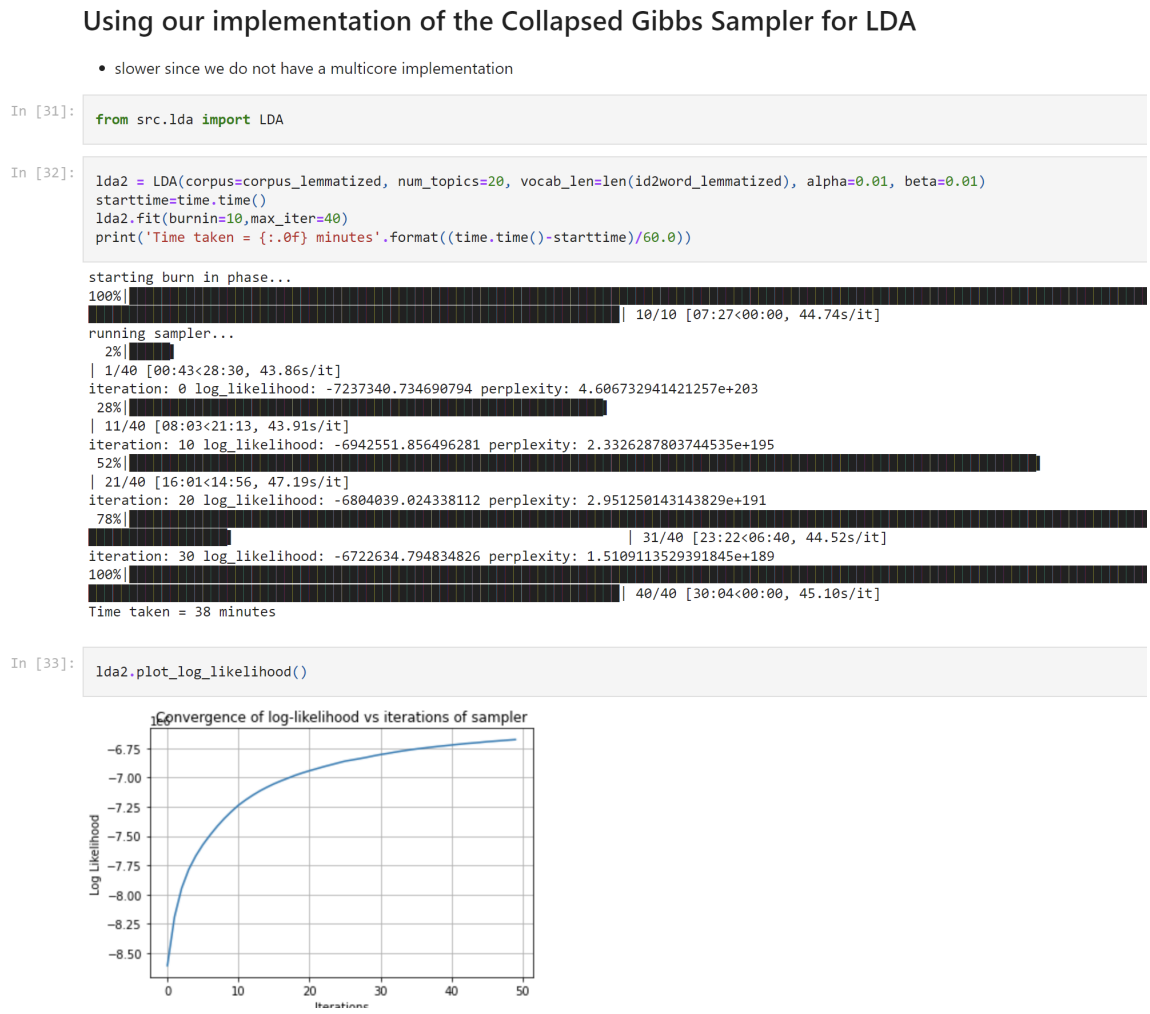


Figure 18: Results from Gibbs Sampling

4. Discussion and Conclusions

4.1 Human interpretation of topics

The following word clouds generated below summarizes how a human can perceive the topic based on the relative word probabilities that are higher for certain terms in the topic. For

example, topic 0 below appears to be about music, topic 5 below is about education and topic 6 below is about sports.

Figure 19: Music topic

Figure 20: Academia

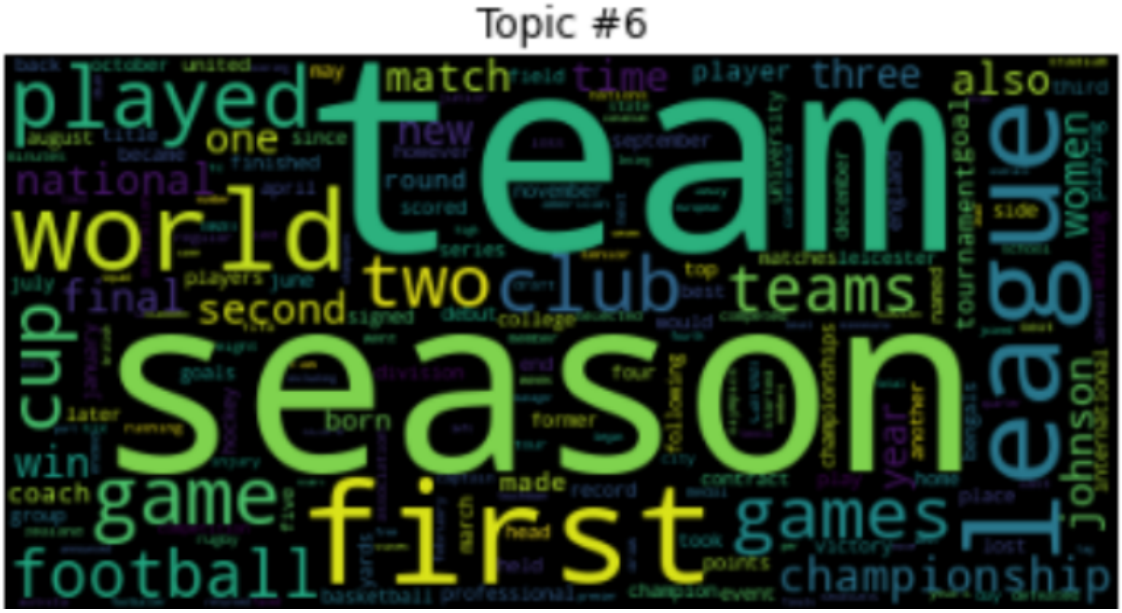


Figure 21: Sports

4.2 Machine interpretation of topics

While a human might see the topics and their corresponding terms above obvious ex-post, it is not so obvious ex-ante. Our experiments on the optimum number of topics show that the best number as measured by coherence scores to is about 25. It is not clear if a human would choose this same number, and different people might choose to create more or fewer topics based on their discretions.

In addition, our experiment with under and over parameterizations and early stopping shows that around 15 thousand words are used to distinguish topics. This is far higher than a human can process in a typical word cloud which usually has less than 50 words.

In fact, according to a recent psychology research paper [14], Divergent Association Tasks was given to thousands of participants to name 10 nouns that are completely unrelated to one another. Humans have varying degrees of creativity, and this serves as a creativity score when the words are maximally semantically distant on average. However, most of us find it

very hard to list the perfect 10 words. Based on this, it might not be easy for a human to create the perfect 25 topics that an LDA model can. After the first few topics of say sports, education, and music, how easy is it to list 22 more topics? Would "biology" be a subtopic of education or is it a new topic? Would "poems" be similar to "music", or should we have "literature" as another topic? If psychological experiments show a difficulty at 10 nouns, we believe that it is better for a machine to learn the topics as it ensures maximum topic separation and lowest perplexity scores.

4.3 Final thoughts

Based on our findings above, we believe that LDA is an attractive model for unsupervised learning of topics from the Wikipedia corpus, and it is best for computers rather than humans to perform this cognitively difficult task. Preprocessing is key for performance, and lemmatization works better than raw tokens as the meaning behind the tokens is inferred and cataloged. In addition, we would prefer a MFVI model as compared to a MCMC model – both have similar outputs but the former has a much faster run time. As for the MFVI model, we propose limiting the E steps of the algorithm as a means for optimizing the run time performance, while not sacrificing or possibly even leading to better perplexity scores.

References

[1] Blei, D. M., Ng, A. Y. Jordan, M. I. (2003). Latent dirichlet allocation. J. Mach. Learn. Res., 3, 993–1022. doi: http://dx.doi.org/10.1162/jmlr.2003.3.4-5.993

[2] Geigle, C. (2016). Inference Methods for Latent Dirichlet Allocation.

[3] Upton, G., Cook, I. (2008) Oxford Dictionary of Statistics, OUP. ISBN 978-0-19-954145-4.

[4] Grimmer, J. (2011). An introduction to bayesian inference via variational approximations. Political Analysis, 19(1), 32–47. https://doi.org/10.1093/pan/mpq027

[5] Hoffman, M. D. (2013). Stochastic Variational Inference.

[6] Information on https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletA

[7] Information on https://towardsdatascience.com/end-to-end-topic-modeling-in-python-latent-dirichlet-allocation-lda-35ce4ed6b3e0

[8] Hoffman and Gelman. Journal of Machine Learning Research 15 (2014) 1351-1381. Adaptively Setting Path Lengths in Hamiltonian Monte Carlo.

[9] Information on https://arxiv.org/pdf/1608.03995.pdf

[10] Stanković, Ranka and Šandrih, Branislava and Krstev, Cvetana and Utvić, Miloš and Škorić, Mihailo. Machine Learning and Deep Neural Network-Based Lemmatization and Morphosyntactic Tagging for Serbian

[11] Ruoqi Shen, Liyao Gao, Yi-An Ma. On Optimal Early Stopping: Over-informative versus Under-informative Parametrization

[12] Hofmann. Unsupervised Learning by Probabilistic Latent Semantic Analysis. Machine Learning, 42, 177–196, 2001

[13] Matthew D. Hoffman (2010) onlineldavb.py: Package of functions for fitting Latent Dirichlet Allocation (LDA) with online variational Bayes (VB). Can be downloaded under GNU General Public License from https://github.com/blei-lab/onlineldavb/blob/master/onlineldavb.py

[14] Jay A. Olson et al. Naming unrelated words predicts creativity. PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES Vol. 118 — No. 25. June 22, 2021