

# 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. 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  $\theta_d$ , and the topic distributions for each corpus  $\beta_k$ . 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.

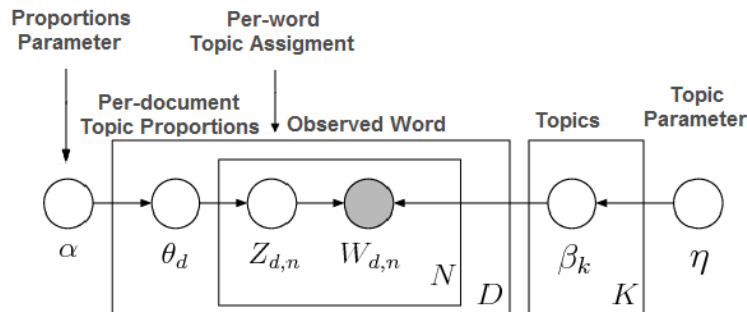


Figure 1: Graphical model of LDA

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

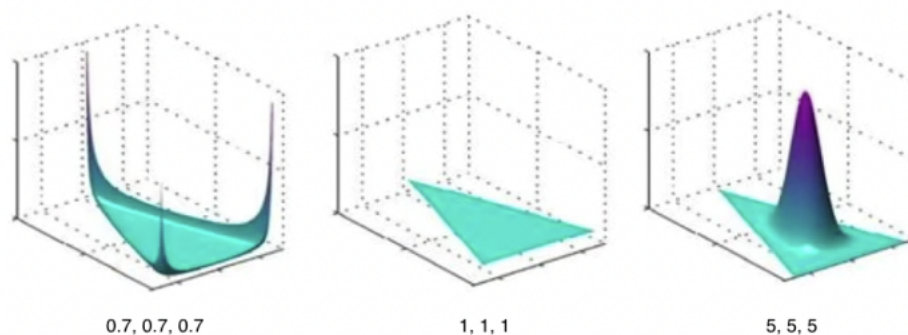
This section will go into further detail about the probability distributions governing the LDA model. From Figure 1, we can see  $\alpha$  on the far left and  $\eta$  on the far right. Both of these are hyperparameters of the Dirichlet distribution, from which we draw  $\theta_d$  for the topic proportions and  $\beta_k$  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  $\alpha$ , or every term is meaningful for a given topic for  $\eta$ . In practice, we know that only a section of topics or

words are meaningful, and so we use  $\alpha = 0.1$  and  $\eta = 0.01$ .

## Dirichlet Distributions

$$f(x_1, \dots, x_K; \alpha_1, \dots, \alpha_K) = \frac{1}{B(\alpha)} \prod_{i=1}^K x_i^{\alpha_i - 1}$$



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  $\theta_d$  for the topic proportions and  $\beta_k$  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  $\theta_d$  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  $\beta_k$  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

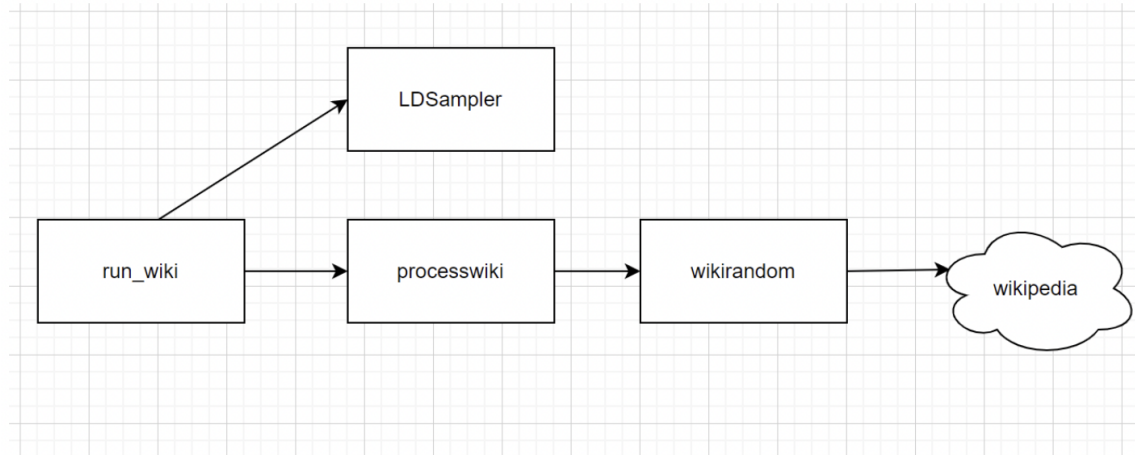


Figure 2: Retooling existing paradigm

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

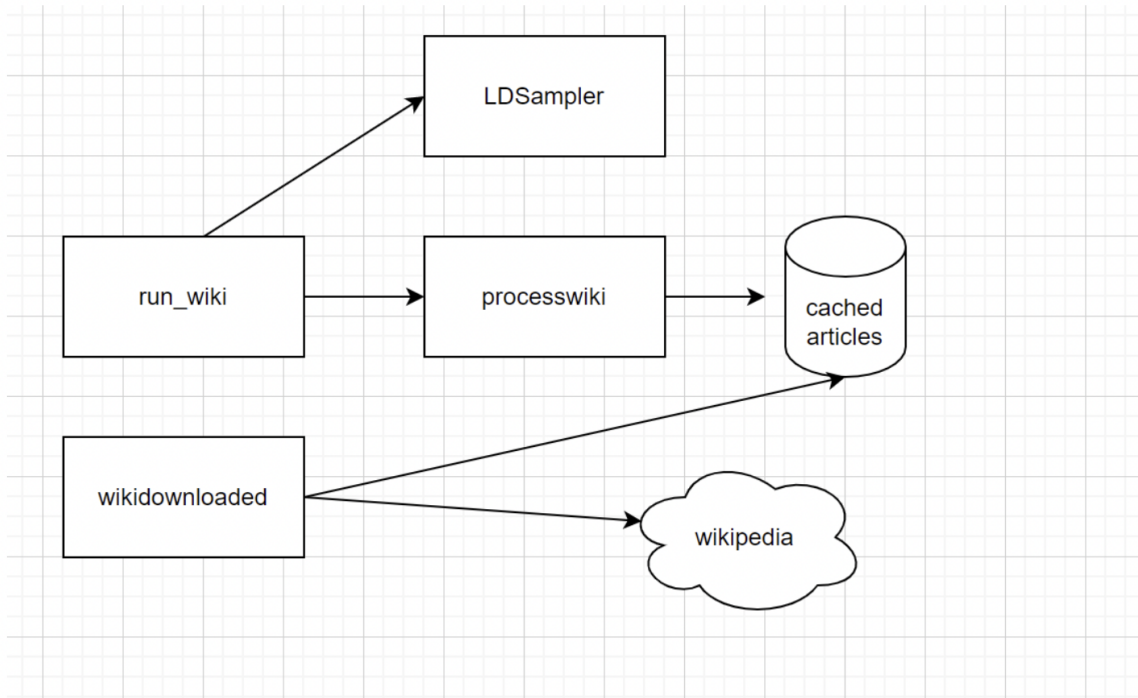


Figure 3: Current paradigm

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,<sup>[1]</sup> 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](#).<sup>[2]</sup> He was [mentioned in a despatch](#) from Egerton dated 4 July 1902, following operations against the [Mahsud Waziris](#).<sup>[3]</sup>

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](#).<sup>[4]</sup>

Munn died aged 77 in [Virginia Water, Surrey](#).<sup>[5]</sup>

## References [[edit](#)]

- ↑ "No. 26390" [↗](#). *The London Gazette*. 7 April 1893. p. 2121.
- ↑ Hart's Army list, 1903
- ↑ "No. 27462" [↗](#). *The London Gazette*. 8 August 1902. pp. 5089–5092.
- ↑ "No. 31097" [↗](#). *The London Gazette* (Supplement). 1 January 1919. p. 82.
- ↑ "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

Tokenization is the process of taking the article text as a string, converting to lowercase, removing punctuation and other non-alphabets, and splitting 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.

```
[21]: print(df.text[0][:400])

'''Colonel Patrick Mackellar''' (1717-1778) was a British army officer and military engineer who played a significant role in the early history of North America. He was the deputy chief engineer at the Siege of Louisbourg (1758) and the chief engineer at the siege of Quebec in 1759. In later years he was responsible for the design and construction of the town of Es Castell on the island of Me

[19]: print(data_words[0][:40])

['colonel', 'patrick', 'mackellar', 'was', 'british', 'army', 'officer', 'and', 'military', 'engineer', 'who', 'played', 'significant', 'role', 'in', 'the', 'early', 'history', 'of', 'north', 'america', 'he', 'was', 'the', 'deputy', 'chief', 'engineer', 'at', 'the', 'siege', 'of', 'louisbourg', 'and', 'the', 'chief', 'engineer', 'at', 'the', 'siege', 'of']

[76]: id2word = corpora.Dictionary(data_words)
texts = data_words
corpus = [id2word.doc2bow(text) for text in texts]
print([(id2word[id], freq) for id, freq in corpus[0]][:20])
print(corpus[0][:20]) #it will print the corpus we created above.

[('able', 2), ('about', 2), ('abraham', 1), ('accompanied', 2), ('accompany', 1), ('accounts', 1), ('acting', 1), ('action', 1), ('active', 1), ('admiral', 1), ('advised', 1), ('advisers', 1), ('after', 6), ('afternoon', 1), ('again', 2), ('against', 7), ('agent', 1), ('all', 1), ('allies', 1)]
[(0, 2), (1, 2), (2, 1), (3, 2), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 6), (13, 1), (14, 2), (15, 7), (16, 1), (17, 1), (18, 2), (19, 1)]
```

Figure 6: An example of the tokenization process

### 2.2.2 N-GRAMS

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

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

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.

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

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.

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

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 track of these statistics as we sweep through all the documents in a collection. And our estimated of  $\phi$  and  $\theta$  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  $\theta$  and  $\phi$ , and we set implement this algorithmically to compare MCMC sampling methods against other inference methods.

### 3. Results

#### 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.

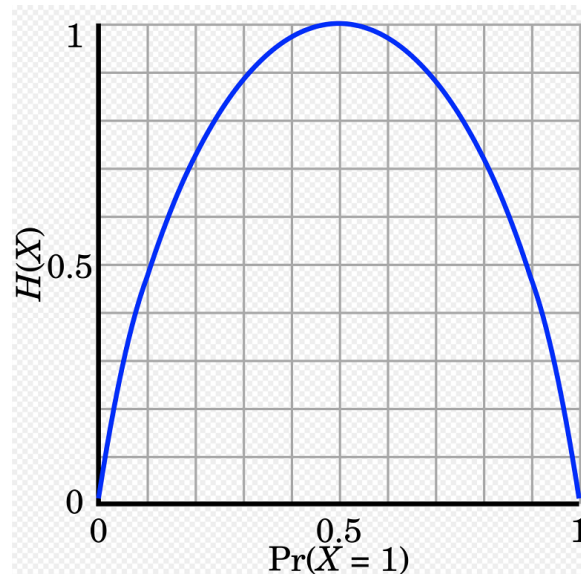


Figure 7: Plot of entropy of a coin with different biases

### 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  $H(p)$ .

### 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.  $D_{KL}$  is also known as relative entropy and is defined by the following integral:

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

In expectations notation, this is equivalent to

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

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

$$q(\theta|x) = \operatorname{argmin}_{\theta} E_q(\log(q(\theta|x)) - E_q(\log(p(\theta|x)))) \quad (3)$$

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

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

Defining Evidence Lower Bound or ELBO(q) as:

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

We get the following relationship between KL and ELBO:

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

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

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 sec-

ond term is related to the cross entropy between  $p$  and  $q$ .

$$ELBO(q) = H(q) - H(q, p) \quad (8)$$

### 3.4 Perplexity of a LDA model

Perplexity refers to the log-averaged inverse probability on unseen data.

Reference: HOFMANN. Unsupervised Learning by Probabilistic Latent Semantic Analysis.

Machine Learning, 42, 177–196, 2001 <https://link.springer.com/content/pdf/10.1023/A:1007617005950.pdf>

The perplexity of our LDA model can be defined as the exponential of the cross-entropy between  $\tilde{p}$  and  $q$ . That is,

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

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,

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

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.

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

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

### 3.4.1 GENSIM IMPLEMENTATION OF PERPLEXITY

We seen above that perplexity can be expressed as a exponential of a negative likelihood measure on a test set.

Given d unseen documents w, the log likelihood can be calculated as

$$L(w) = \log[p(w|\Phi, \alpha)] = \sum_d \log[p(w_d|\Phi, \alpha)]$$

The Gensim implementation of perplexity differs in the way the likelihood is estimated. Instead of cross-entropy as the exponent, they used the ELBO instead.

$$2^{ELBO(q)} = 2^{H(q) - H(q,p)} \quad (12)$$

Perhaps the mentor can clarify the difference.

```

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 8: 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`:
     $E_q[\log p(\text{corpus})] - E_q[\log q(\text{corpus})]$ 

    `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]
        Elogtheta_d = dirichlet_expectation(gammad)

        #  $E[\log p(\text{doc} \mid \theta, \beta)]$ 
        score += numpy.sum(cnt * logsumexp(Elogtheta_d + Elogbeta[:, id]) for id, cnt in doc)

        #  $E[\log p(\theta \mid \alpha) - \log q(\theta \mid \gamma)]$ ; assumes alpha is a vector
        score += numpy.sum((self.alpha - gammad) * Elogtheta_d)
        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

    #  $E[\log p(\beta \mid \eta) - \log q(\beta \mid \lambda)]$ ; 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

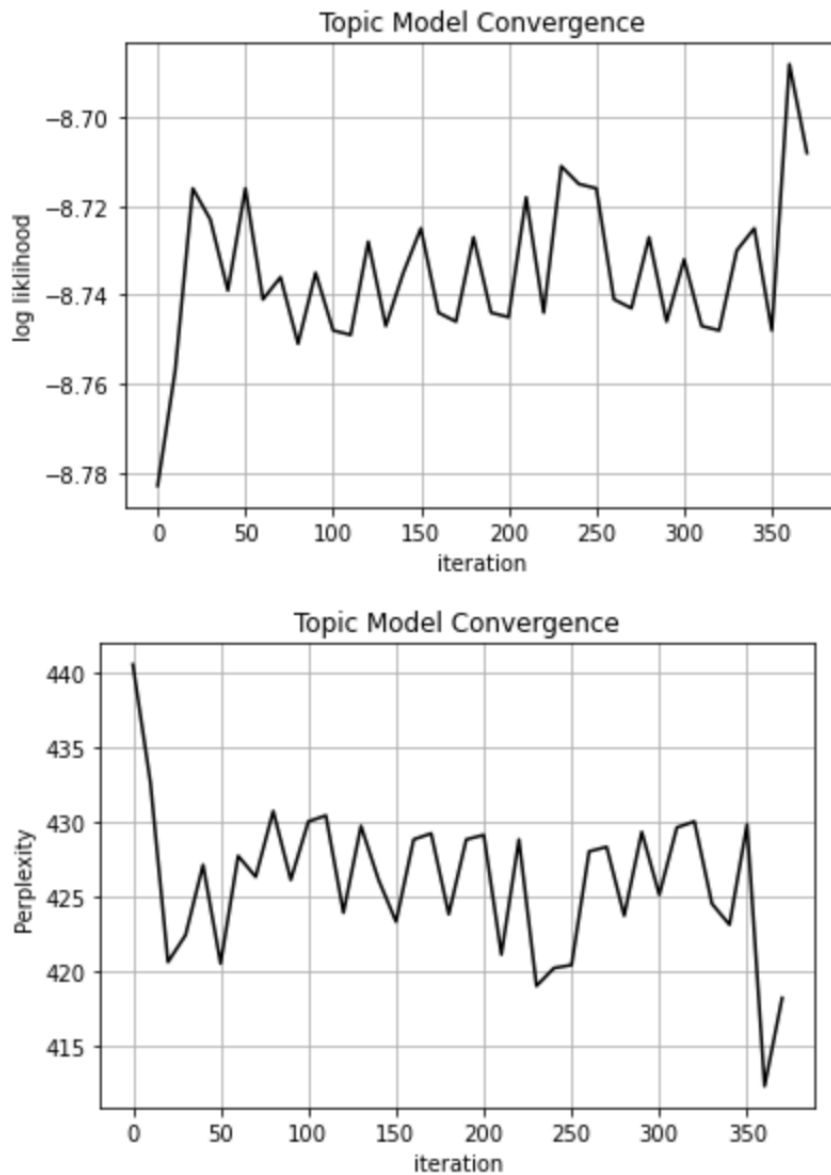
```

Figure 9: Code for function bound calculates the ELBO

### **3.5 Likelihood and Perplexity**

#### **3.5.1 MINIMAL PREPROCESSING**

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.



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

Figure 10: Optimal K value for Wikipedia dataset

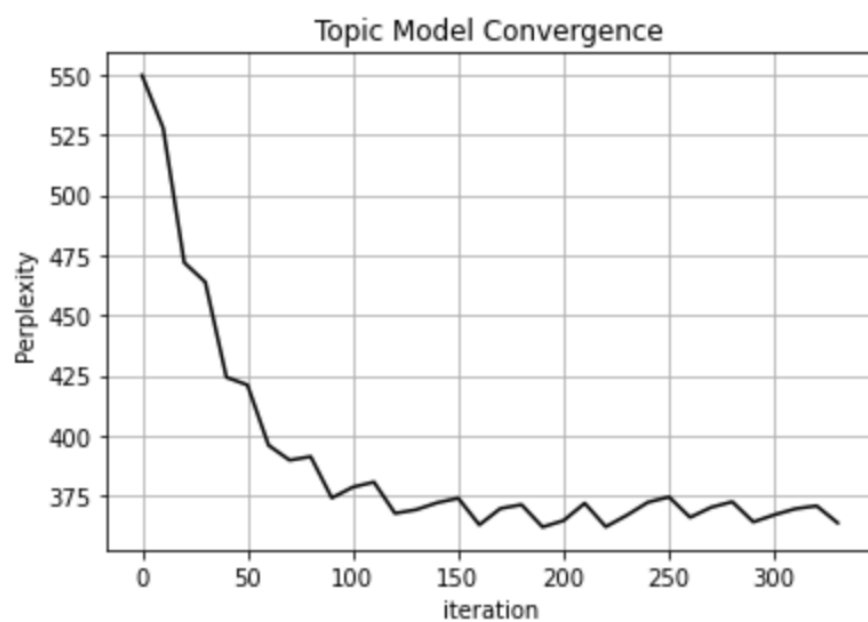
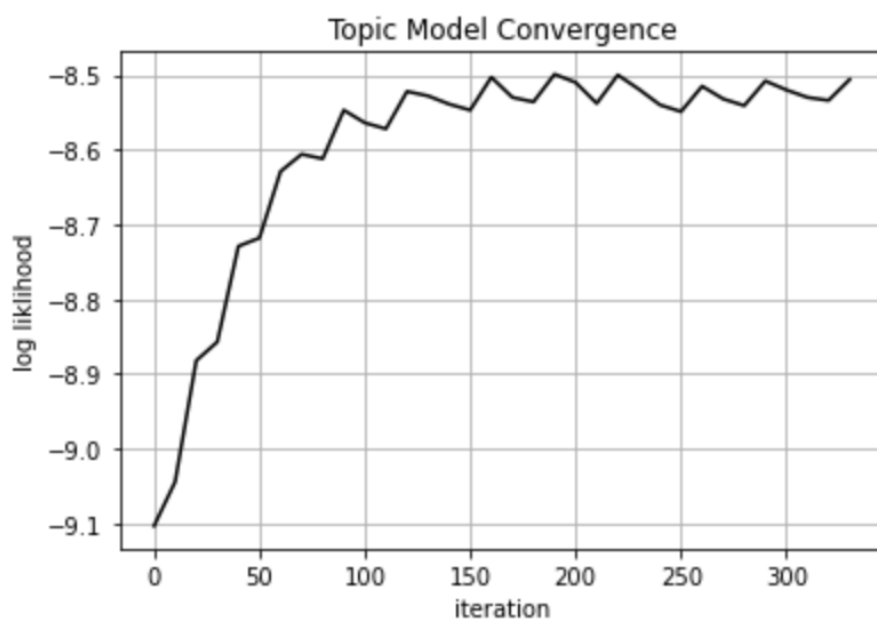
### 3.5.2 LEMMATIZATION AND DICTIONARY COMPRESSION

We also ran the same experiment with extensive preprocessing. In addition to 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 the 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 is 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.



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

Figure 11: Likelihood and Perplexity evolution by iterations

### 3.6 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 findings is that a value of around 25 topics is optimal.

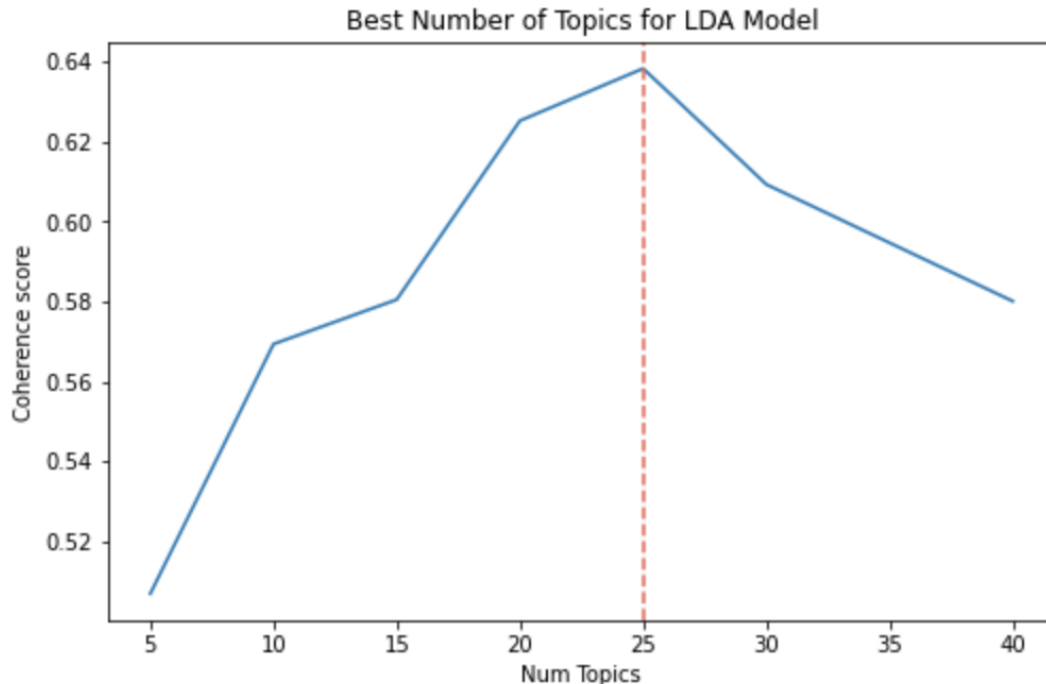


Figure 12: Optimal K value for Wikipedia dataset

### 3.7 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 words in the vocabulary that is considered by 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 param-

eterization. Instead, we under-parameterized our model by truncating the probabilities of common words 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 parameters of words 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.

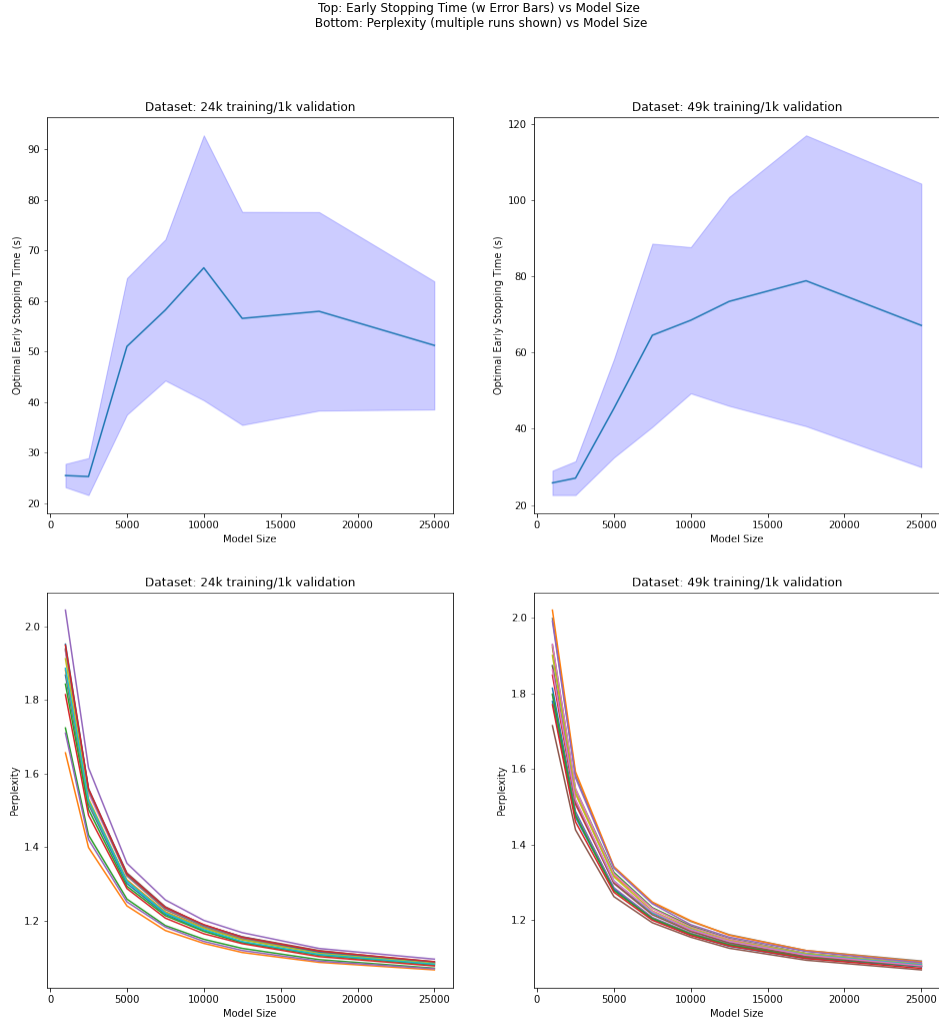


Figure 13: Early Stopping Times and Perplexity for Underinformed Models

#### 4. LDA demo

In this section, we would like to present the how we use LDA to model a large collection of text data. We need to manually identify the topic

## Topic #0



Figure 14: Music topic

## Topic #5



Figure 15: Academia

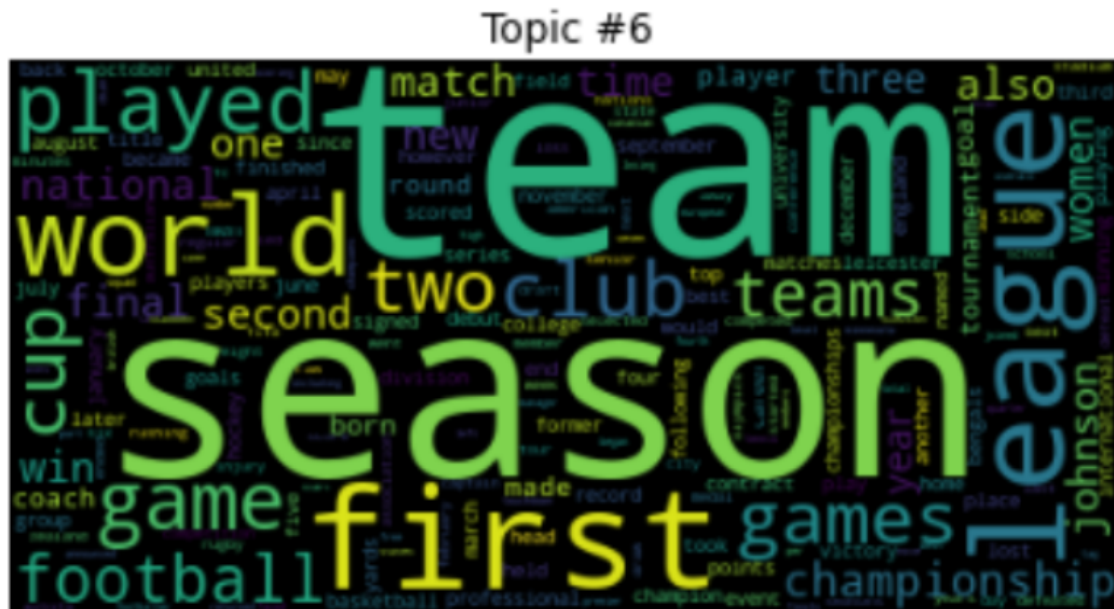


Figure 16: Sports

## 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] Information on [https://www.researchgate.net/publication/341574872\\_Machine\\_Learning\\_and\\_Deep\\_Neural\\_Networks\\_Based\\_Lemmatization\\_and\\_Morphosyntactic\\_Tagging\\_for\\_Serbian](https://www.researchgate.net/publication/341574872_Machine_Learning_and_Deep_Neural_Networks_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