DSC 180B A06 write-up

Duha Aldebakel daldebak@ucsd.edu

Yu Cao y6cao@ucsd.edu

Anthony Limon a1limon@ucsd.edu

Rui Zhang r2zhang@ucsd.edu

Editor:

Abstract

We explore both Markov Chain Monte Carlo algorithms and variational inference methods for Latent Dirichlet Allocation (LDA), a generative probabilistic topic model for data such as text data. LDA is a generative probabilistic topic model, meaning we treat data as observations that arise from a generative probabilistic process including hidden variables, i.e. structure we want to find in the data. Topic modelling allows us to fulfill algorithmic needs to organize, understand, and annotate documents according to the discovered structure. For text data, hidden variables reflect the thematic structure of a corpus that we don't have access to, we only have access to our observations which are the documents of the collection themselves. Our aim is to infer this hidden structure through posterior inference, that is, we want to compute the conditional distribution of the hidden variables given our observations, and we use our knowledge from Q1 about inference methods to solve this problem.

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 these kind of tasks 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 each and 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
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, We might choose a topic x from a distribution over topics. 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 actually 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 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 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, 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.

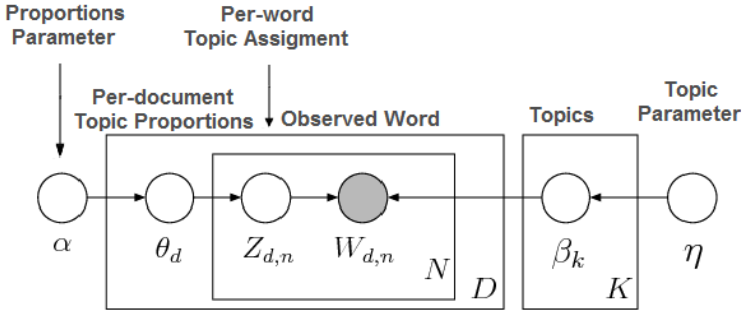


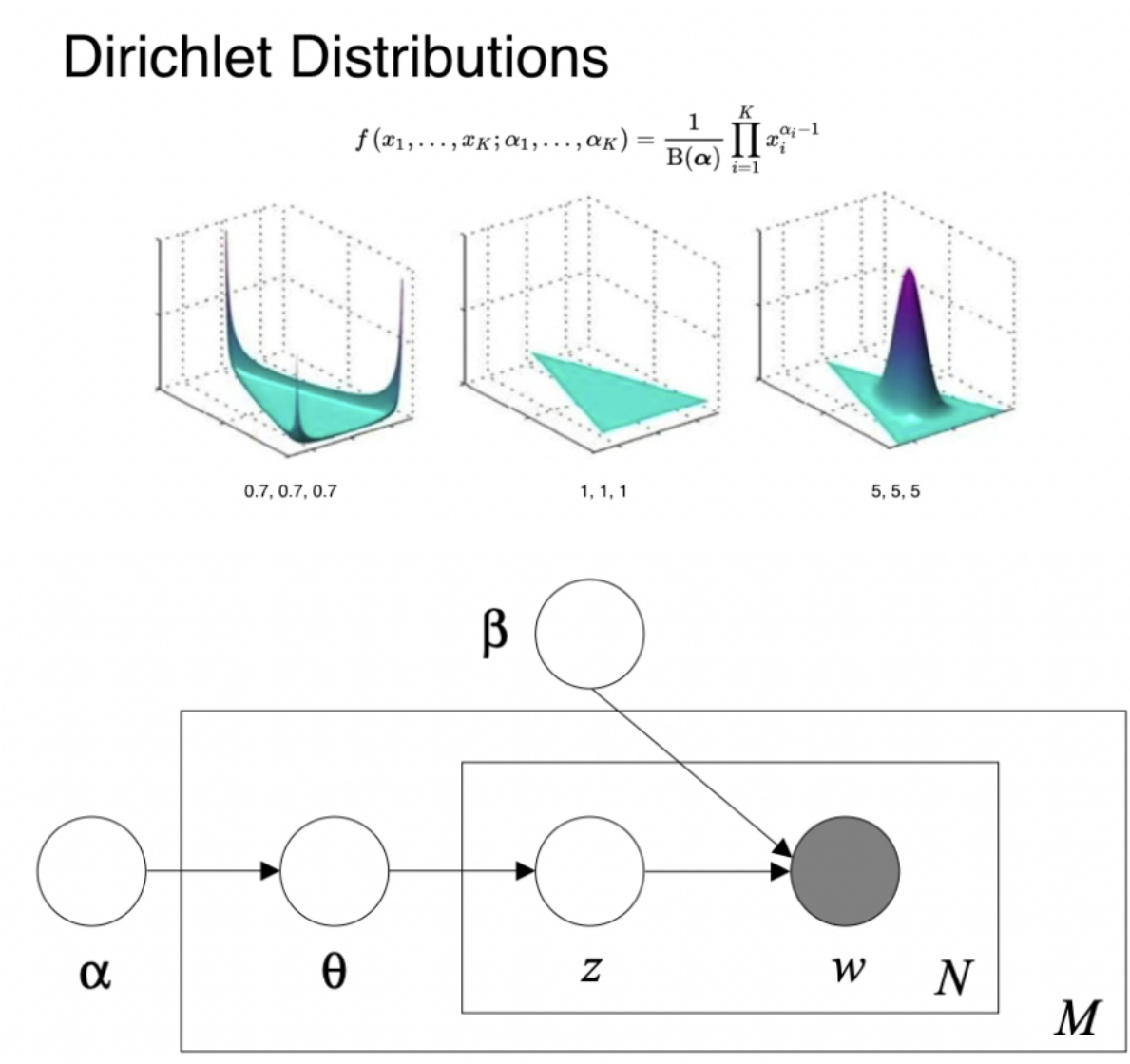
Figure 1: Graphical model of LDA

Building off our work from Q1, the posterior distribution can be computed in many different ways. Of course we now know the posterior distribution cannot be computed directly for complicated models such as LDA, which gives rise to approximate inference techniques such as variational inference from which we learned of in Q1.

Now we will explain a bit about the LDA algorithm by considering the hyper parameters of the model, α and β. Our task for this semester is a pivot away from the previous semester where we explored the statistical and computational trade-offs in estimating logistic regression posteriors. For this semester, we will explore learning Latent Dirichlet allocation (LDA) models using variational and EM methods.
Latent Dirichlet allocation (LDA) is a generative probabilistic model of a corpus, such as a collection of documents. There are three levels to this hierarchical Bayesian model. Each document is modeled as a finite mixture of topics. Each word is associated with a topic that is modeled as an infinite mixture over an underlying set of topic probabilities. The actual words within each document are generated based on the conditional probabilities governing that topic. Given this generative probabilistic model, there are several efficient approximate inference techniques to estimate the latent parameters, such as variational and EM methods. LDA can be used for document modeling, text classification, and collaborative filtering. Some researchers have reported better results with LDA models as compared to traditional latent semantic indexing models.
The parameter α should be smaller than 1. As the α becomes larger than 1, the 'articles' will move toward the middle, while the α becomes smaller, the 'articles' will go to corners and edges. Based on how close an 'article' is to a croner, we can determine the probability of its topics based on the location and Dirichlet Distributions. Note that within each 'article', there is another distribution of percentages for topics, which means there is a multinomial distribution for the distribution of articles and topics.
As the number of topics increases, the dimension increases instead of the number of edges. β on the other hand is the distribution of the topics in relation with the words in articles. The distribution is the reverse of the distribution of α where the axis and the data points are switched. So instead of having topics in each corner of the distribution, there is a word in each corner and topics will be put into the location where they are closest to certain words. Intuition

Based on the two distributions discussed, two variables will be established based on the probability of topics based on articles and words based on topics. We then randomly pick words within the distribution and put them together to generate a new document. Note that the chance that the output document is the same as the original input documents is very low. However, with various different settings of the model and comparison of many generated documents, we can pick the one with the highest probability for the original document which is the one where the documents are correctly located.

This is the graphical representation from the assigned paper "Latent Dirichlet Allocation". α is the Dirichlet distribution of article and topics and β is the one with topics and words. Based on α, we get θ which is the multinomial distribution for picking a topics for documents. Based on β, we get the distributions for picking words. Then we get z from θ which is list of topics. We combine distribution from β and z, we obtain a list of words (One



words for each selected topics). We will then compare the corpse of words to the original document for maximizing the probability

2. Methods

2.1 Data Collection

2.1.1 Previous work

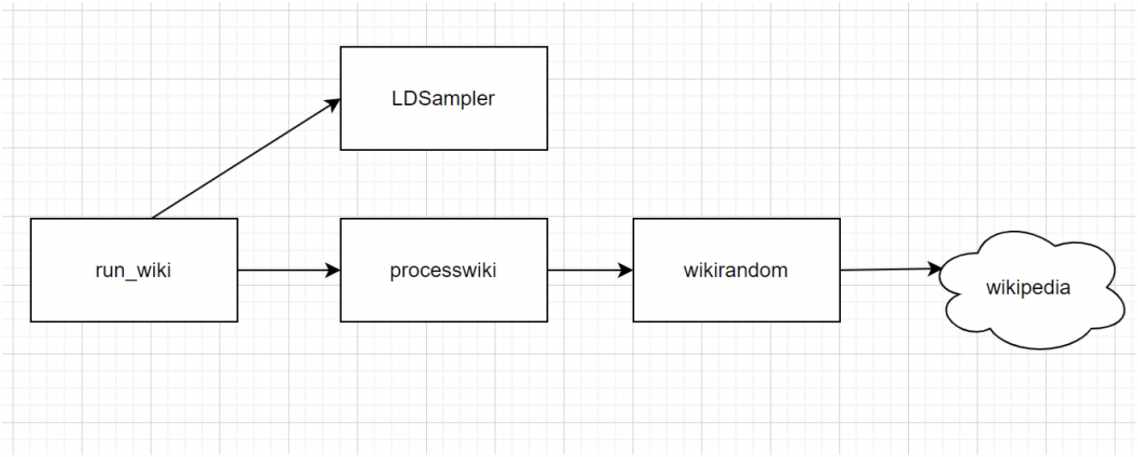


Figure 2: Retooling existing paradigm

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 is 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

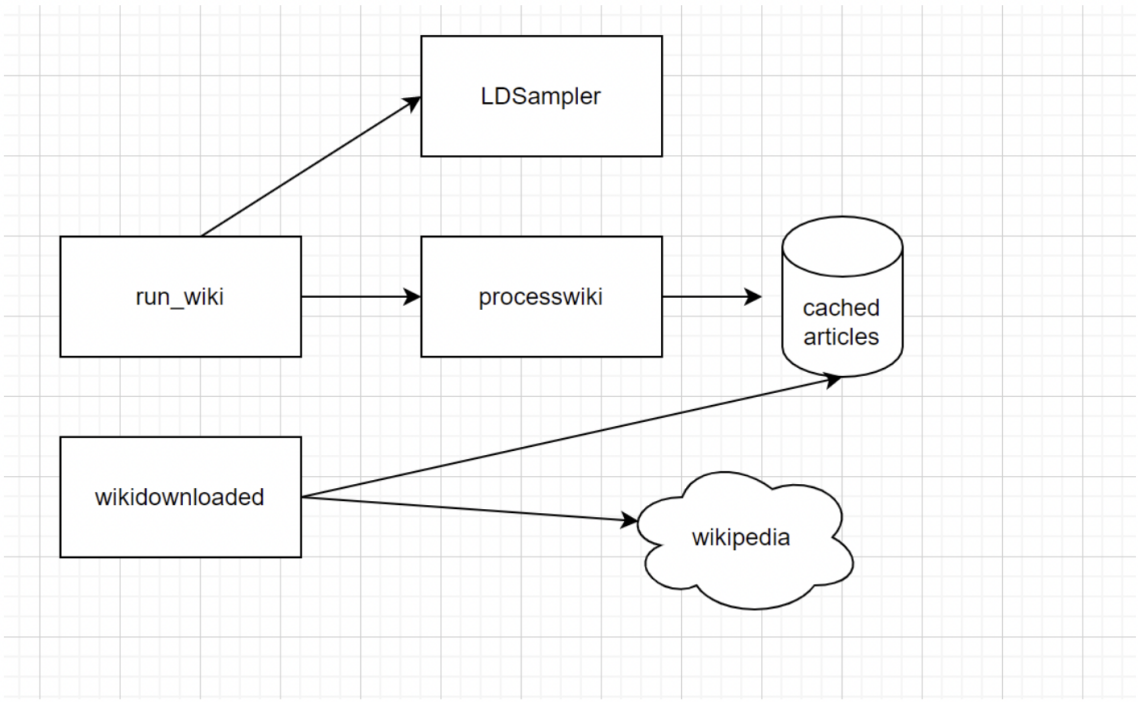


Figure 3: Current paradigm

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

At example of a wikipedia article is shown in the figure above. There are several decisions that 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 post script 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 Engish criket.

2.1.3 Current progress

- Create wikidownloader in Python3, which downloaded articles from wikipedia.
  - Changed some search and parsing parameters for the latest website format.
  - I used the “request” library instead of the depreciated urllib2 Python2 library.
  - Used threadings.
    - Thread locks to ensure mutual exclusivity of shared data
    - Thread events to terminate threads
  - At 5 threads, I received 429 errors from Wikipedia, which basically means they are ghosting us.
    - When 429 errors are received, the rate gets much slower. (150 documents per minute or less)
    - It is better not to trigger with 4 threads. (300 documents per minute)
  - Data is large and disk latency is also an issue.
  - Solution: we serialize the data (article and titles) using python 3’s pickle, and read/save it to file using a stream that includes BZ2 compression.
- Current cached data is about 60k documents and 75 Mbytes in a compressed file.
  - More can be downloaded but this is sufficient for the experiment under default parameters. (Needs 50k)

```
Load old data
We have 60295 documents
42/1000 downloaded. Last title was Salmo_akairos
87/1000 downloaded. Last title was Frasier_(season_1)
130/1000 downloaded. Last title was Bun_(hairstyle)
173/1000 downloaded. Last title was Neocyttus
215/1000 downloaded. Last title was Calhoun,_Georgia
```

- Getting rest of the code to run
  - Latest Anaconda Python 3
  - Used script “2to3” to convert python 2 to python 3
    - Replace print “hello” to print(“hello”) etc
  - Made other fixes
    - Make parser arguments optional so they accept the default value
    - File saved includes invalid characters so simplified filename
  - Changed processwiki to refer to cached dataset

```
def online_wiki_docs():
    datacount=0
    # with open('data.pickle', 'rb') as f:
    with bz2.BZ2File('datapickle.bz2', 'rb') as f:  #Use datacompression BZ2
        data= pickle.load(f)
    while True:
        #yield get_random_wikipedia_article()
        yield (data[0][datacount],data[1][datacount])
        datacount+=1
        if datacount>=len(data[0]):
           datacount=0
```

- Currently code runs and converges

```
LDSampler iteration 988, docs so far 49400, log pred -6.52408, time 0.358
LDSampler iteration 989, docs so far 49450, log pred -7.18363, time 0.279996
LDSampler iteration 990, docs so far 49500, log pred -7.23893, time 0.252985
LDSampler iteration 991, docs so far 49550, log pred -7.33376, time 0.295004
LDSampler iteration 992, docs so far 49600, log pred -6.92362, time 0.351001
LDSampler iteration 993, docs so far 49650, log pred -6.48408, time 0.209987
LDSampler iteration 994, docs so far 49700, log pred -7.31041, time 0.179001
LDSampler iteration 995, docs so far 49750, log pred -7.27323, time 0.311981
LDSampler iteration 996, docs so far 49800, log pred -7.29773, time 0.215835
LDSampler iteration 997, docs so far 49850, log pred -7.1703, time 0.261033
LDSampler iteration 998, docs so far 49900, log pred -7.34724, time 0.359001
LDSampler iteration 999, docs so far 49950, log pred -7.39689, time 0.215974
```

```
5.0000000000000000e+03 -7.8034621275896725e+00
1.0000000000000000e+04 -7.5252322456784126e+00
1.5000000000000000e+04 -7.4039378103311186e+00
2.0000000000000000e+04 -7.3054975880476566e+00
2.5000000000000000e+04 -7.2868140483432384e+00
3.0000000000000000e+04 -7.2714407694008249e+00
3.5000000000000000e+04 -7.2546954478127849e+00
4.0000000000000000e+04 -7.2182612172470825e+00
4.5000000000000000e+04 -7.2095532864533833e+00
```

-
- Next goal is to understand the LDA algorithm in the code.
-

2.2 Data Preprocessing Approach One

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 a 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 earl
y history of North America.  He was the deputy chief engineer at the Siege of Louisbourg (1758) and the chief engineer at the siege of Qu
ebec 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', 'sieg
e', '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), ('activ
e', 1), ('admiral', 1), ('advised', 1), ('advisers', 1), ('after', 6), ('afternoon', 1), ('again', 2), ('against', 7), ('agent', 1), ('a
l', 1), ('all', 2), ('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), (1
6, 1), (17, 1), (18, 2), (19, 1)]
```

Figure 6: At example of the tokenizaton 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 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.

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.

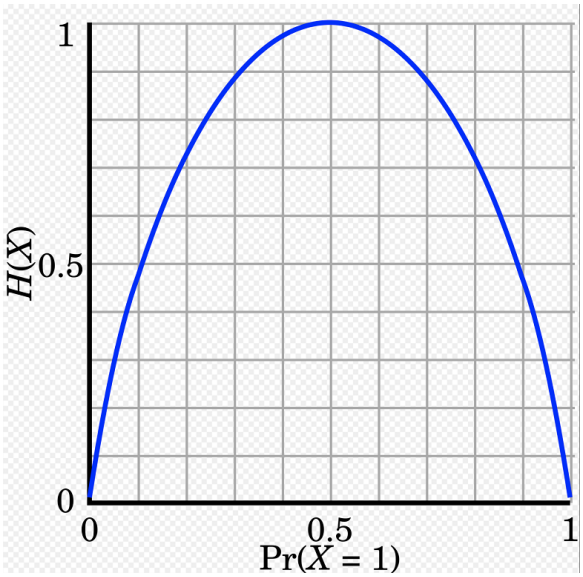


Figure 7: Plot of entropy of a coin with different biases

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:

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

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.

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

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,

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

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.

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

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

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.

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

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(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 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.

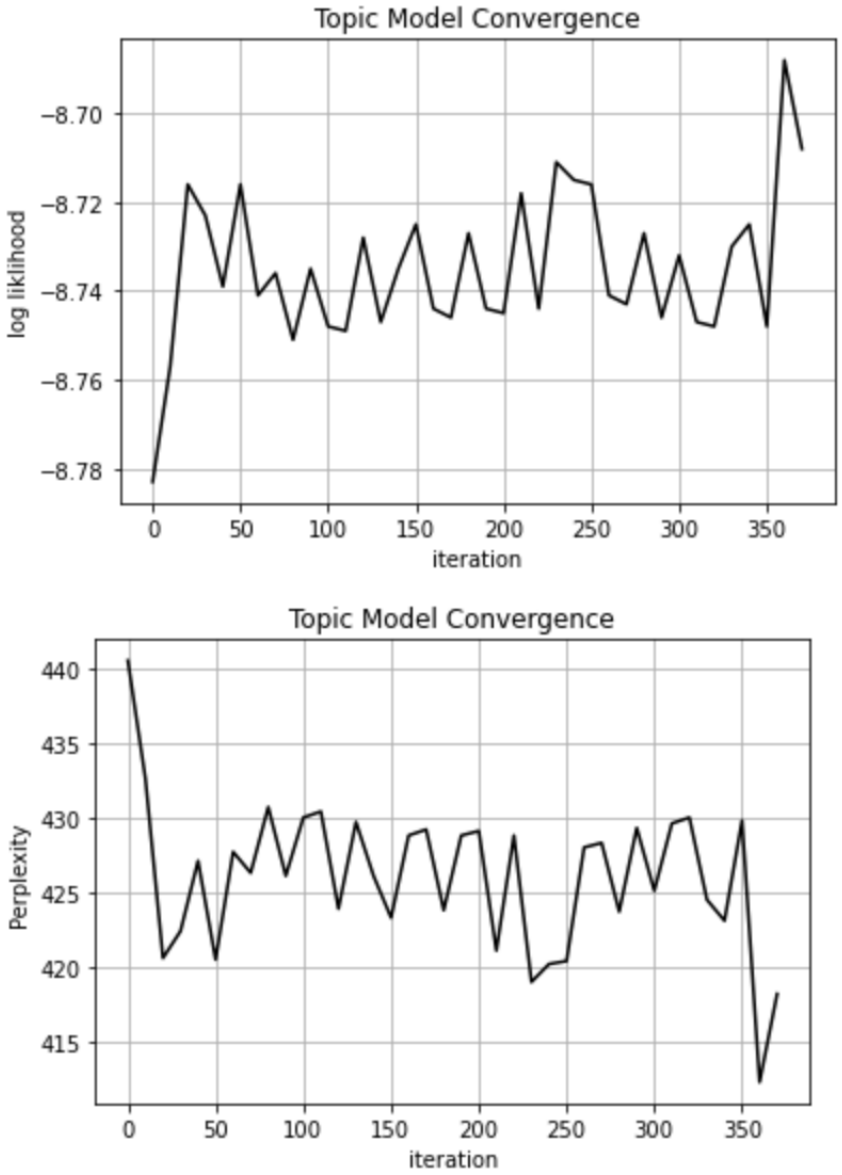


Figure 10: Optimal K value for Wikipedia dataset

3.5.2 Lemmatization and Dictionary Compression

We also ran the same experment 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.

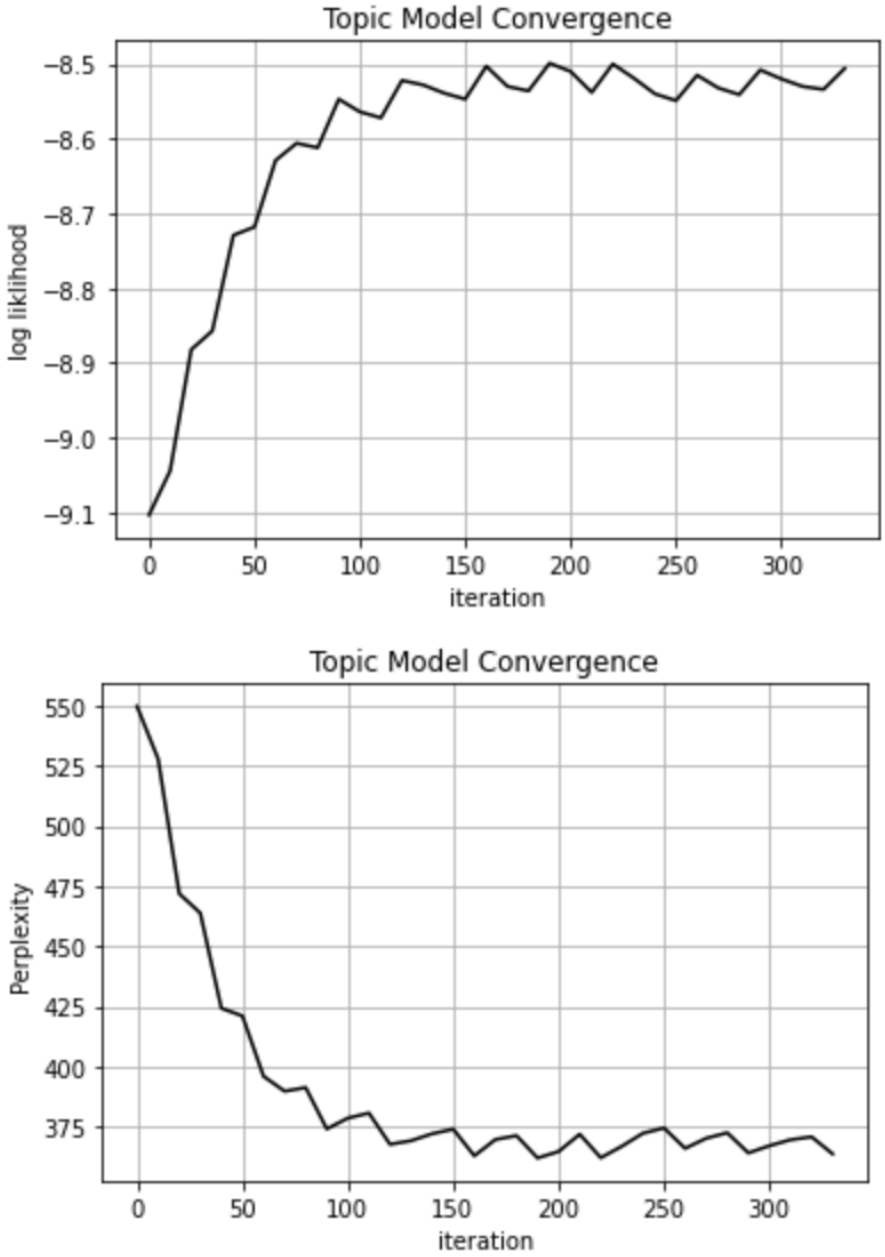


Figure 11: Likelihood and Perplexity evolution by iterations

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.

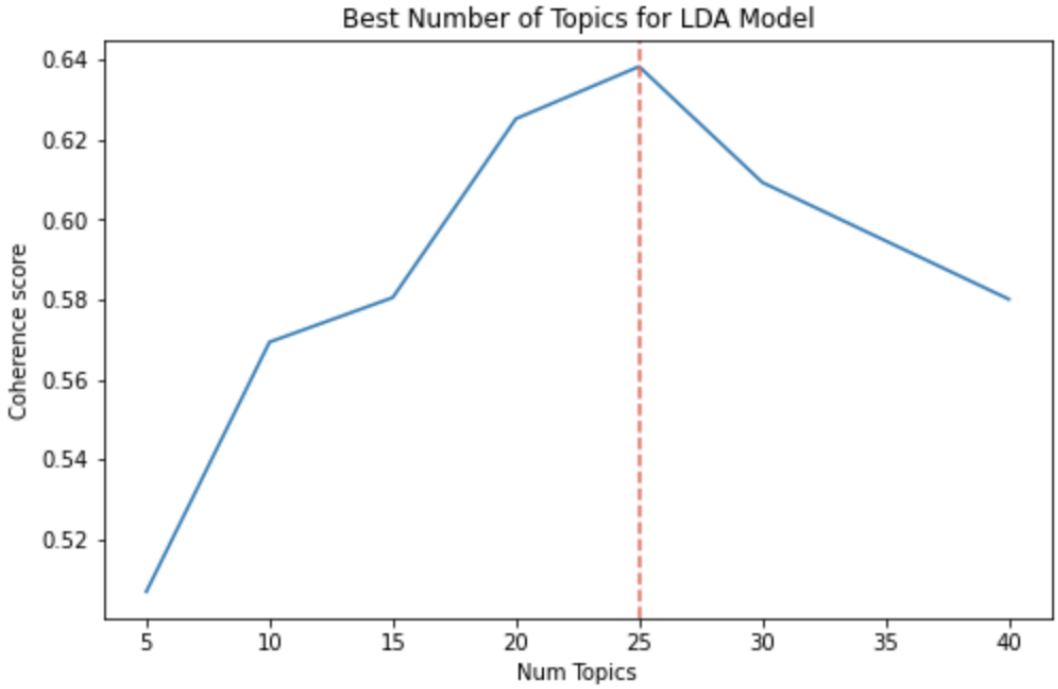


Figure 12: Optimal K value for Wikipedia dataset

4. Discussion

We will be implementing LDA using a variety of existing natural language processing libraries. For the purpose of evaluation, we decide to use Gensim for building our LDA model. It contains several useful functions by which we could use to evaluate the interpretability our topics. Hoping with using such metrics, we can fine tune the parameters of our model. However, using Gensim library comes with a challenge: we need to implement the text preprocessing code from scratch, which we have already completed during past few weeks. But we are not sure how we going to tune the parameters since the estimated training time will be very long. Say if we are going to train a LDA model based on 2 millions articles, then the time taken to train a single model would be at least several hours, if not several days. We have to scale up the size of the documented for one reason, that the topics are scattered sparsely for the articles in Wikipedia, If our our sample size is too small, our topic model may not well represent the true topics distribution. We are thinking about having 40 50 topics manually labelled.

We also need to carefully analyze our model after we train it. Specifically, we need to understand the topics we are seeing and hand label it meaning. The LDA model will not label the topics for us; Instead, it will gives us the topics-words distribution. The label must be inferred based on model outputs. Therefore, We will manually examine and label each topic of our model.

Analysis being involved (tentative): 1) word clouds analysis. 2) Explore the relationship between topics via dimentionality reduction 3) extend our analysis to bigram or trigram. 4) Topic influence analysis. See the dominant topics across our corpus.

Application: Use the LDA model that we train to label or "understand" the unseen text data. I am thinking about news article/quora question/reddit blogs.

If time permit, Train some auxiliary models that would help us perform tasks other than labelling the article. For example, sentiment analysis and Text summerization could be useful. If there is a trained model available, we can just directly use them. In best case scenario, Our goal is to build an APP that could extract high level abstract features of long, complex article.

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$_{M}achine_{L}earning_{a}nd_{D}eep_{N}eural_{N}e$
$Based_{L}emmatization_{a}nd_{M}orphosyntactic_{T}agging_{f}or_{S}erbian$