CUED - Engineering Tripos Part IIB 2021-2022 **Module Coursework**

Module	4F13	Title of report	Latent Dirichlet Allocation
Date submitted: 29/11/2021		Assessment for this module is ☒ 100% / ☐ 25% coursework of which this assignment forms 33 %	

UNDERGRADUATE STUDENTS ONLY		POST GRADUATE STUDENTS ONLY			
Candidate number:	5585G	Name:		College:	

Feedback to the student ☐ **See also comments in the text**		Very good	**Good**	Needs improvmt
CONTENT	**Completeness, quantity of content:** Has the report covered all aspects of the lab? Has the analysis been carried out thoroughly?			
	Correctness, quality of content Is the data correct? Is the analysis of the data correct? Are the conclusions correct?			
	Depth of understanding, quality of discussion Does the report show a good technical understanding? Have all the relevant conclusions been drawn?			
	Comments:			
PRESENTATION	**Attention to detail, typesetting and typographical errors** Is the report free of typographical errors? Are the figures/tables/references presented professionally?			
	Comments:			

Indicative grades are not provided for the FINAL piece of coursework in a module

Assessment (circle one or two grades)		A*	A	B	C	D
Indicative grade guideline		>75%	65-75%	55-65%	40-55%	<40%
Penalty for lateness:		*20% of maximum achievable marks per week or part week that the work is late.*				

Marker: Date:

1 Latent Dirichlet Allocation

1.1 Question 1:

In the multinomial text model we consider words w_n drawn from a size M vocabulary $\mathcal{V}$ according to associated probability parameters $p(w_n = v_i|\boldsymbol{\beta}) = \beta_i$. Given a set of word counts $\{c_i\}_{i=1}^M$ as training data, maximum likelihood estimates for the model parameters are

$$\hat{\beta}_i^{ML} = \frac{c_i}{\sum_{i=1}^M c_i} \tag{1}$$

which is just the proportion of total observations that were the i^{th} word. Given a training matrix A whose structure is explained in the handout, ML parameters were calculated as follows

```
# one beta parameter per word
counts = np.zeros(V.shape[0])
# count each word individually
for i in range(counts.shape[0]):
    counts[i] = np.sum(np.where(A[:,1]==i, A[:,2], 0))
# as a proportion
beta_ml = counts/np.sum(counts)
```

Figure 1: Python code for $\hat{\beta}^{ML}$

The twenty highest probability words are shown below

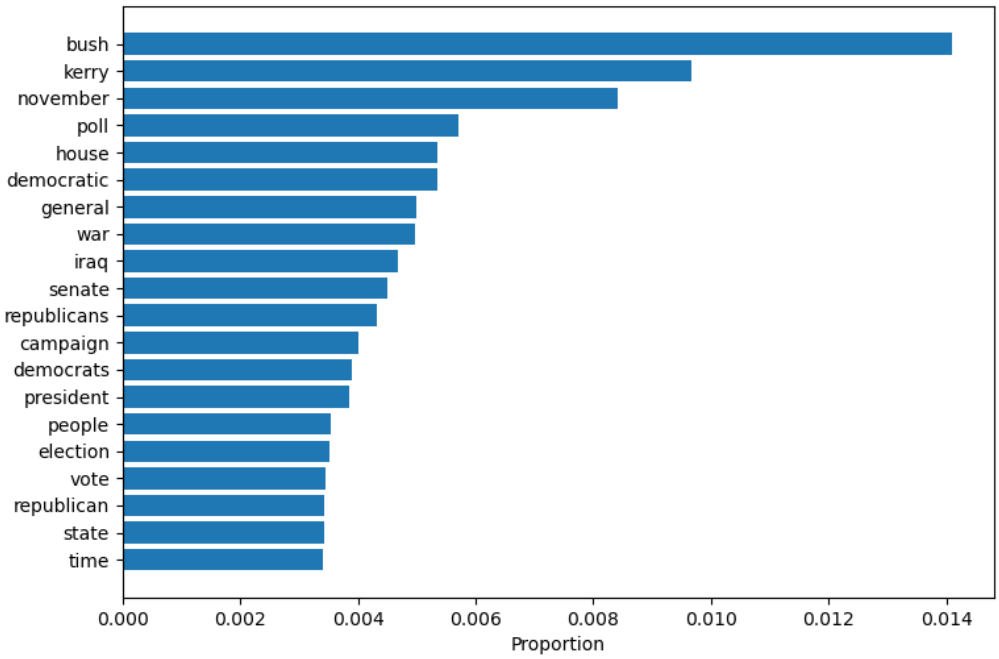

Figure 2: Twenty highest probability words using data in A

Below is an illustration of Zipf's law - frequency of occurence is inversely proportional to ranking as demonstrated on a log plot. Note how the low end of rankings drop to zero as the maximum likelihood model assigns zero probability to the unobserved words.

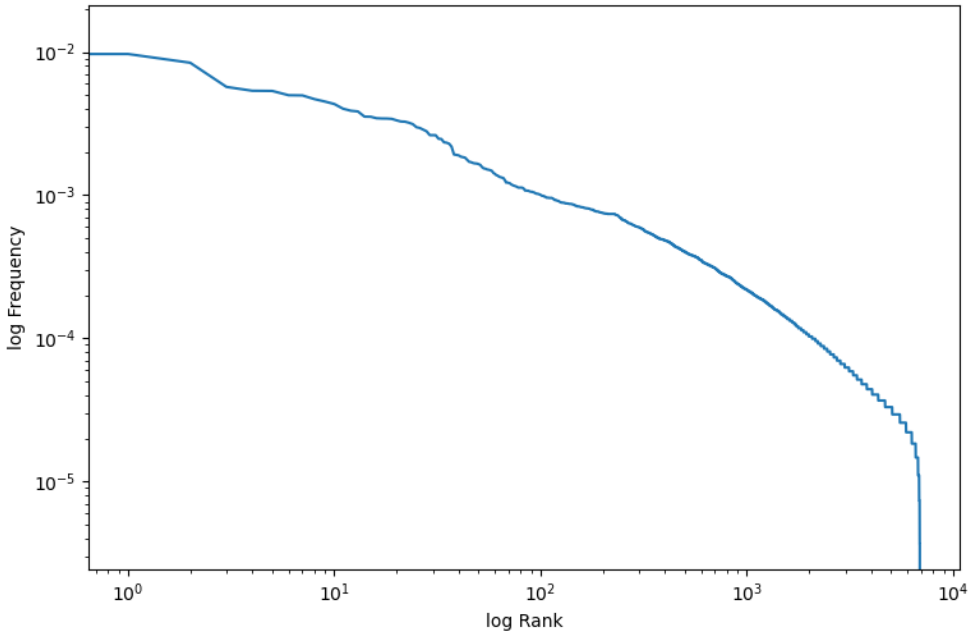

Figure 3: Zipf's Law

Considering the log probability for a test set

$$\log p(\boldsymbol{c}|\boldsymbol{\beta}) = \sum_{i=1}^{M} c_i \log \beta_i$$

Then if any words present in the test set were unobserved in the training set ($\beta_i = 0$ when $c_i \neq 0$) the log probability will diverge to $-\infty$. The log probability will be maximised if c_i is zero except for the most frequent word in the training set, i.e. just a repeated sequence of the most likely word.

This divergence to $-\infty$ is undesirable behaviour since assigning zero probability to a sequence containing unseen words ignores the fact that the training data may not contain every word in the vocabulary.

1.2 Question 2

This pathology can be addresed using Bayes rule with a Dirichlet prior. We consider here a symmetric Dirichlet prior with concentration parameter α on $\boldsymbol{\beta}$

$$p(\boldsymbol{\beta}|\alpha) \propto \prod_{i=1}^{M} \beta_i^{\alpha-1}$$

Combining this with our likelihood function using Bayes' rule we get

$$p(\boldsymbol{c}|\boldsymbol{\beta}) \propto \prod_{i=1}^{M} \beta_i^{c_i}$$

$$p(\boldsymbol{\beta}|\boldsymbol{c},\alpha) \propto \prod_{i=1}^{M} \beta_i^{\alpha+c_i-1} \tag{2}$$

Hence our posterior distribution for $\boldsymbol{\beta}$ conditional on the observed counts and the prior parameter is also a Dirichlet with parameter vector $\boldsymbol{c}+\boldsymbol{\alpha}$. The predictive mean for each word under this posterior (Dirichlet) distribution is

$$\mathbb{E}_{p(\boldsymbol{\beta}|\boldsymbol{c},\alpha)} c_i = \frac{c_i+\alpha}{\sum_{j=1}^{M}(c_j+\alpha)} \tag{3}$$

Therefore, this prior has inflated the count of each word by the prior parameter α. Importantly, this means that unobserved words during training are no longer assigned zero probability, and so the log test set probability no longer diverges to $-\infty$. For small α we heavily weight the observed data, so that the maximum-a-priori values of β are very close to the ML values (i.e. very small β_i for rare words and larger β_i for more common words). For increasing α we assign more weight to the prior - this has the effect of increasing the probability of rare words and vice versa, and in the limit $\alpha \to \infty$ we retrieve a uniform distribution over the entire vocabulary ($\beta_i = \frac{1}{M}$).

1.3 Question 3

The categorical distribution function is suitable for document modelling since the combinatorial factor in the multinomial distribution is a function of the word counts only, and the word counts are fixed, hence it is inconsequential to our inference. The multinomial test set probability in terms of counts $\boldsymbol{c}$ is

$$p(\boldsymbol{c}|\boldsymbol{\beta}) = \frac{(\sum_{j=1}^{M} c_j)!}{\prod_{i=1}^{M}(c_i!)} \prod_{i=1}^{M} \beta_i^{c_i} \tag{4}$$

Dropping the combinatorial factor results in the categorical model, so taking logs, we have

$$\log p(\boldsymbol{c}|\boldsymbol{\beta}) = \sum_{i=1}^{M} c_i \log \beta_i \tag{5}$$

We train β_{post} on the training counts, $\boldsymbol{c}_{(A)}$ and calculate the log test probability with respect to a particular test document, $\boldsymbol{c}_{(\text{test})}$. Therefore our log test probability is

$$\log p(\boldsymbol{c}_{(\text{test})}|\boldsymbol{c}_{(A)},\alpha) = \sum_{i=1}^{M} c_i^{(\text{test})} \log \frac{c_i^{(A)}+\alpha}{N+M\alpha} \tag{6}$$

Where $N = \sum_{i=1}^{M} c_i^{(A)}$. The corresponding perplexity for this is

$$p_{(\text{test})} = \exp\Big(-\frac{\log p(\boldsymbol{c}_{(\text{test})}|\boldsymbol{c}_{(A)},\alpha)}{N_{(\text{test})}}\Big) \tag{7}$$

Where $N_{(\text{test})} = \sum_{i=1}^{M} c_i^{(\text{test})}$

With $\alpha = 1$, the log test set probability and per-word perplexity for document 2001 are

Document ID	$\log p(\boldsymbol{c}_{(\text{test})}\|\boldsymbol{c}_{(A)},\alpha)$	$p_{(\text{test})}$
2001	-3688.62	4373.11

Per word perplexity is a length-normalised measure of how well the model predicts the observed sequence. It is a decreasing function of the log probability, so higher probability models have a lower perplexity for fixed sentence length. For a uniform multinomial the perplexity is

$$p = \exp(-\frac{N}{N}\log\frac{1}{M}) = M \tag{8}$$

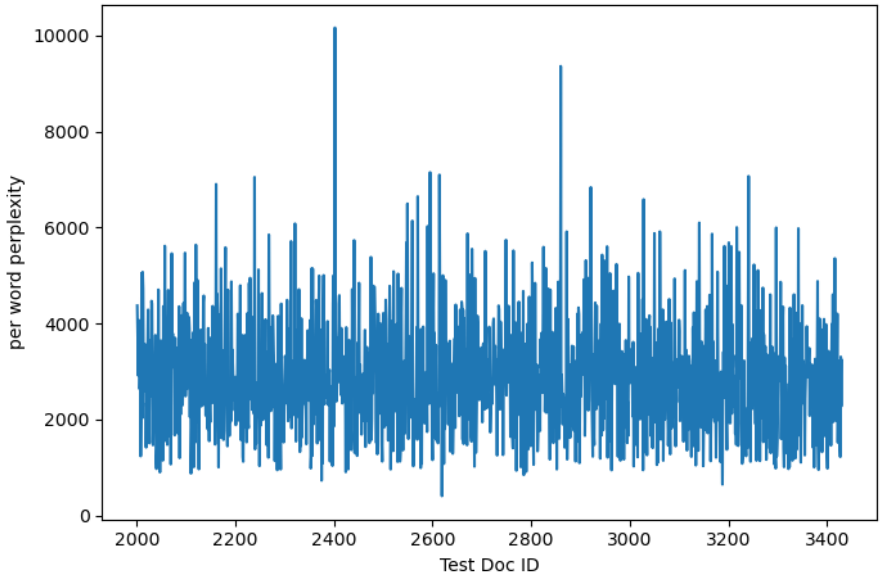

Figure 4: Per-word perplexities for each test document

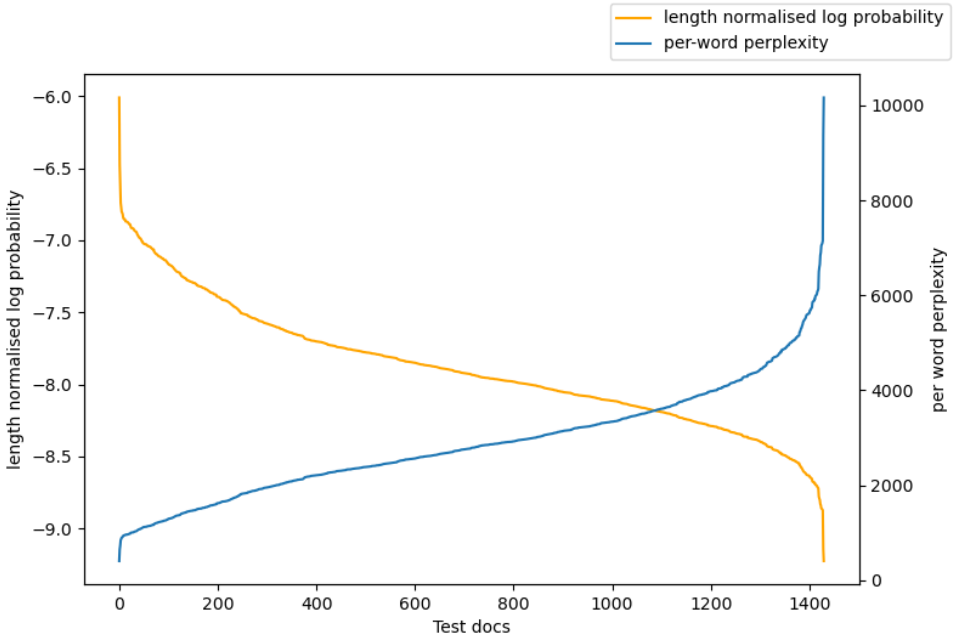

Figure 5: Test docs sorted in increasing perplexity order, along with the corresponding length normalised log probability $\frac{l}{N}$

.

Figure (5) shows that sequences with (log) lower probability per word have a higher perplexity.

1.4 Question 4

In the Bayesian mixture of multinomials model, the mixing proportions are the posterior probabilities of each mixture component

$$\pi_k = \frac{c_k + \alpha}{\sum_{i=1}^{K}(c_j + \alpha)} \tag{9}$$

Where c_k is the total number of documents assigned to topic k, α is a prior topic pseudocount parameter and K is the number of distinct topics.

```
# skeleton for mixing proportions at each iteration
pis = np.zeros((K, num_iters_gibbs))
for iter in range(num_iters_gibbs):
    # implement posterior component probabilities
    pis[:, iter] = (sk_docs.flatten()+alpha)/np.sum(sk_docs.flatten()+alpha)
```

Figure 6: Python code for finding mixing proportions at each gibbs iteration

We can plot the evolution of these mixing proportion for $K = 20$ over 50 iterations of the collapsed gibbs sampler.

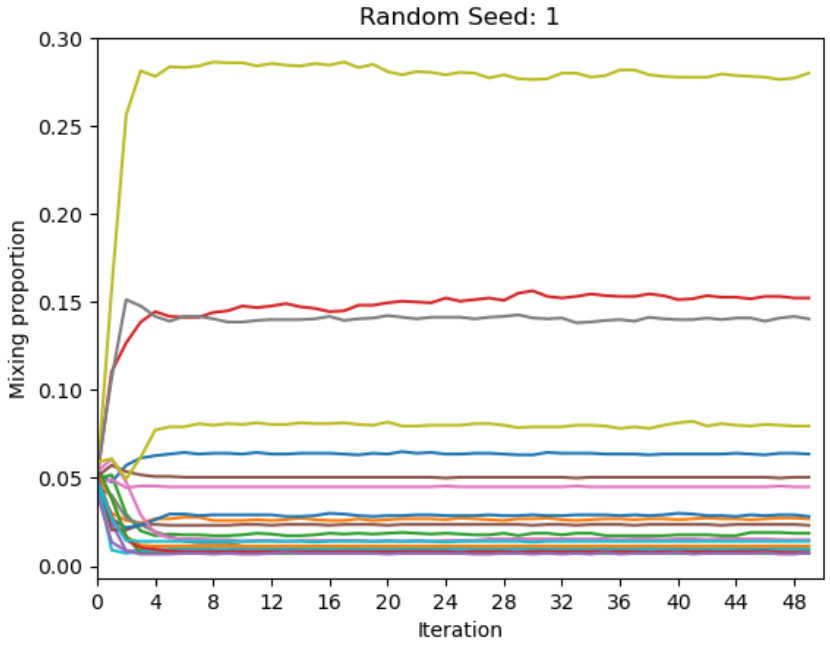

Figure 7: Evolution of mixing proportion with gibbs iterations

These mixing proportions appear to be steady, but this does not imply that they have converged to the target posterior, nor that the chain explores the posterior well. If the chain does converge to the invariant distribution, then the initial random seed should not affect the steady-state mixing proportions.

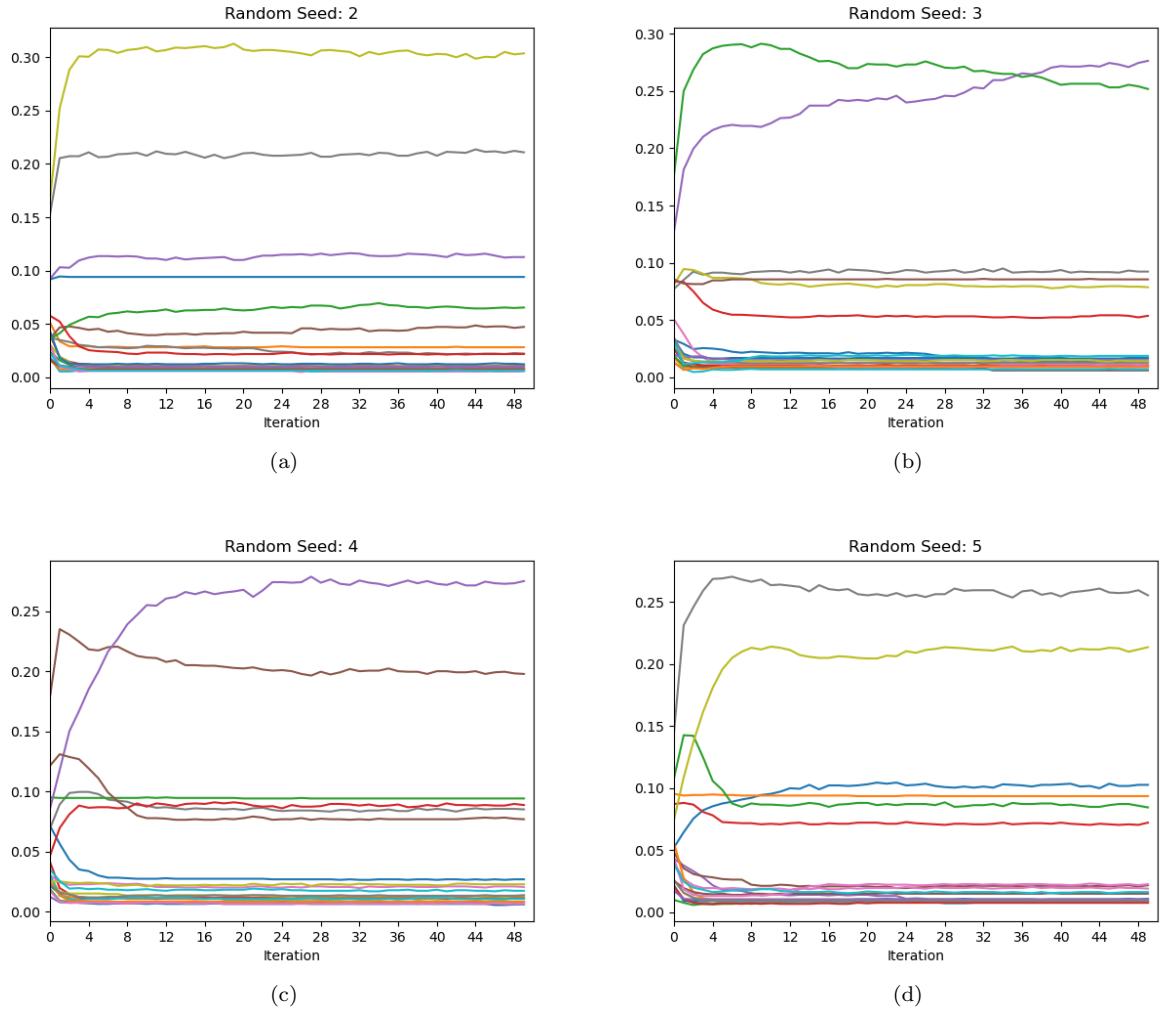

Figure 8: BMM gibbs sampler initialised at different points

Figure (8) makes it clear that the gibbs sampler is not exploring the whole posterior as the chain behaviour is dramatically different depending on the starting point. This may be down to the curse of dimensionality causing very concentrated areas of probability mass in this 20-dimensional posterior.

The lowest per-word perplexity achieved for these mixture models was $p = 2092.15$ for seed 5. The other values were only very slightly higher than this. Clearly these mixture models are able to describe the observed data better than the single topic model where the geometric mean (since we are in the log domain) of the perplexities in the test set is $p = 2657.34$.

1.5 Question 5

In contrast to the mixture model, the Latent Dirichlet Allocation models assigns each word in a document its own topic (rather than the entire document). The collapsed gibbs sampler is used again here. The collapsed sampler marginalises over the component parameters $\boldsymbol{\beta}$ and mixing proportions $\boldsymbol{\pi}$ so that we only sample the latent state allocations. This introduces dependence into the samples and leads to a *rich get richer* property of the mixing proportions.

The code to calculate intermediate mixing proportions is similar to that of the previous question, except we must first sum the topic counts in each document since we no longer treat a document as a single topic.

Again, the gibbs sampler does not appear to converge and demonstrates the rich-get-richer behaviour as initially one topic sees a large rise in mixing proportion while the others mostly decrease.

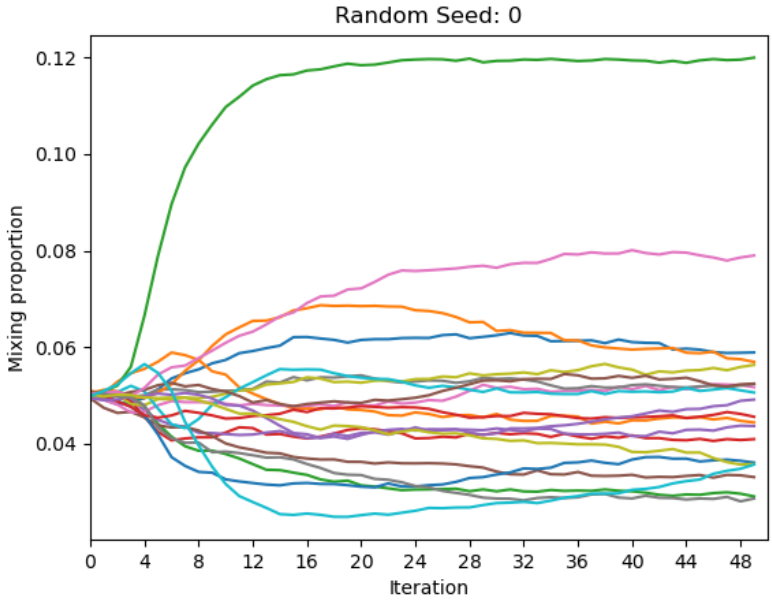

Figure 9: Evolution of mixing proportion with gibbs iterations

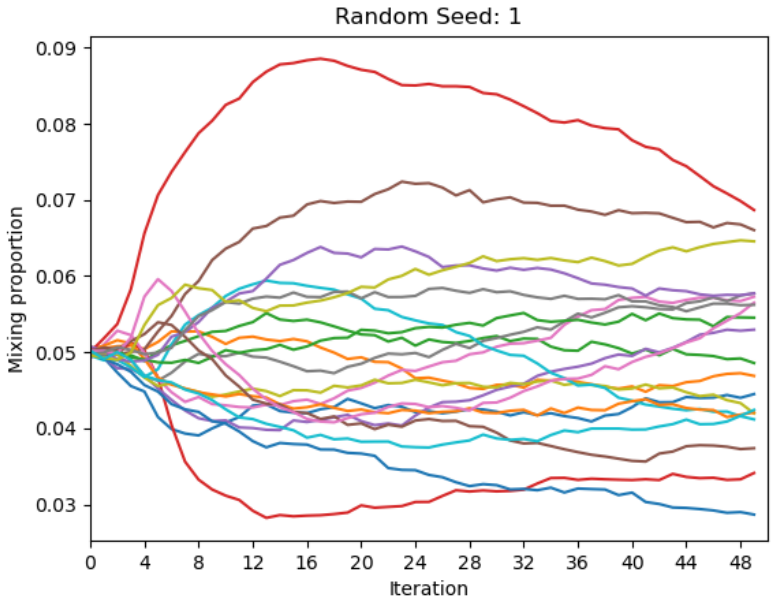

Figure 10: Evolution of mixing proportion with gibbs iterations

For random seed 0, the per-word perplexity after 50 iterations was $p = 1647.83$ which is lower than the mixture model and the single topic model. Since there is no convergence, we cannot say that 50 iterations is adequate, but we can observe that the model is finding a reasonable local optimum for each random seed as the perplexity is decreasing. Therefore, we can choose one of these local optima as our model and explore its performance further.

Assigned to each topic k is a set of word probabilities $\boldsymbol{\beta}^{(k)}$

$$\beta_i^{(k)} = \frac{c_i^k + \gamma}{\sum_{j=1}^{M}(c_j^k + \gamma)} \tag{10}$$

One way to understand how these probabilities are distributed is through the word entropy measured in bits

$$H_k = -\sum_{i=1}^{M} \beta_i^{(k)} \log_2 \beta_i^{(k)} \tag{11}$$

Higher entropy implies that the distribution is closer to uniform, hence a lower entropy distribution assigns higher probability to specific words.

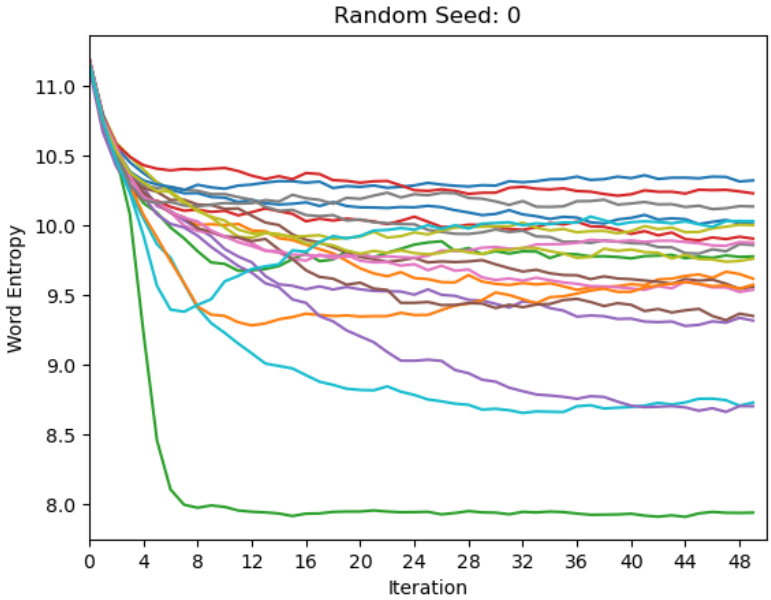

Figure 11: Evolution of word entropy with gibbs iterations

The entropies all decrease initially, but most of them stay roughly constant after about 8 gibbs iterations. After 50 gibbs iterations, the entropy values for $\alpha = \gamma = 0.1$ and random seed 0 were as shown in the table below.

Topic	H_k (bits)	Topic	H_k (bits)
1	10.02	11	10.32
2	9.62	12	9.60
3	9.80	13	7.92
4	9.84	14	10.26
5	9.32	15	8.70
6	9.60	16	9.37
7	9.52	17	9.89
8	9.84	18	10.15
9	10.03	19	9.74
10	8.74	20	10.04

Figure 12: Entropy values in bits for each topics. Higher values imply more uncertain outcomes in word selection.

The top five word probabilities for the lowest and highest entropy distributions (topics 13 and 11 respectively) are

Topic 13	Word Probability	Topic 11	Word Probability
1	0.3074	1	0.0139
2	0.0675	2	0.0127
3	0.0630	3	0.0103
4	0.0611	4	0.0081
5	0.0608	5	0.0080

Figure 13: Most common words in the most extreme entropy topics

As expected, the low entropy topic has specialised very heavily in the most common word, whereas the higher entropy topic is more balanced.

Wordcount: 987 according to Overleaf