Probabilistic Machine Learning: Latent Dirichlet Allocation

Abhinav Heble

December 9, 2021

1 Abstract

Assigning documents to topics using machine learning techniques is a widely explored and important task. This report explores two methods for allocation - Gibbs sampling for a mixture of multinomials model and LDA - and investigates the per-word perplexities (a measure of performance) and potential pitfalls of the methods. Both the training and test data used are word counts in political articles from US publications in 2004.

2 The Maximum Likelihood Multinomial Over Words

It is useful to first explore methods for finding the log probability of a test document using the word counts in the training documents. The maximum likelihood multinomial over words, with N_d total words and the n-th word w_{nd} in document d, is

$$argmax_\beta \;\; p(\boldsymbol{w}|\boldsymbol{\beta}) = \prod_{n=1}^{D}\prod_{n=1}^{N_d}\beta_{w_{nd}} \qquad subject\ to \;\; \sum_{m=1}^{M}\beta_m = 1 \tag{1}$$

Taking logs and maximising using a Lagrangian multiplier, with c_m being the total count of word m and n being the total number of words,

$$argmax_\beta \;\; log(p(\boldsymbol{w}|\boldsymbol{\beta})) = \sum_{m=1}^{M} c_m log(\beta_m) \implies \beta_m = \frac{c_m}{n} \tag{2}$$

Listing 1 takes the result in Equation 2 to calculate the probability of the words, the top 20 of which are shown in Figure 1.

```
word_count, total_count = np.zeros(num_words), 0 #initialise specific and total word counts
for i in A: total_count+=i[2]; word_count[i[1]-1]+=i[2] #iterate through words
beta=list((word_count[x]/total_count,x) for x in range(num_words)) #fill betas
beta.sort(reverse=True,key=lambda x:x[0]) #sort words by probability
```

Code Listing 1: Code to calculate the maximum likelihood multinomial over all words.

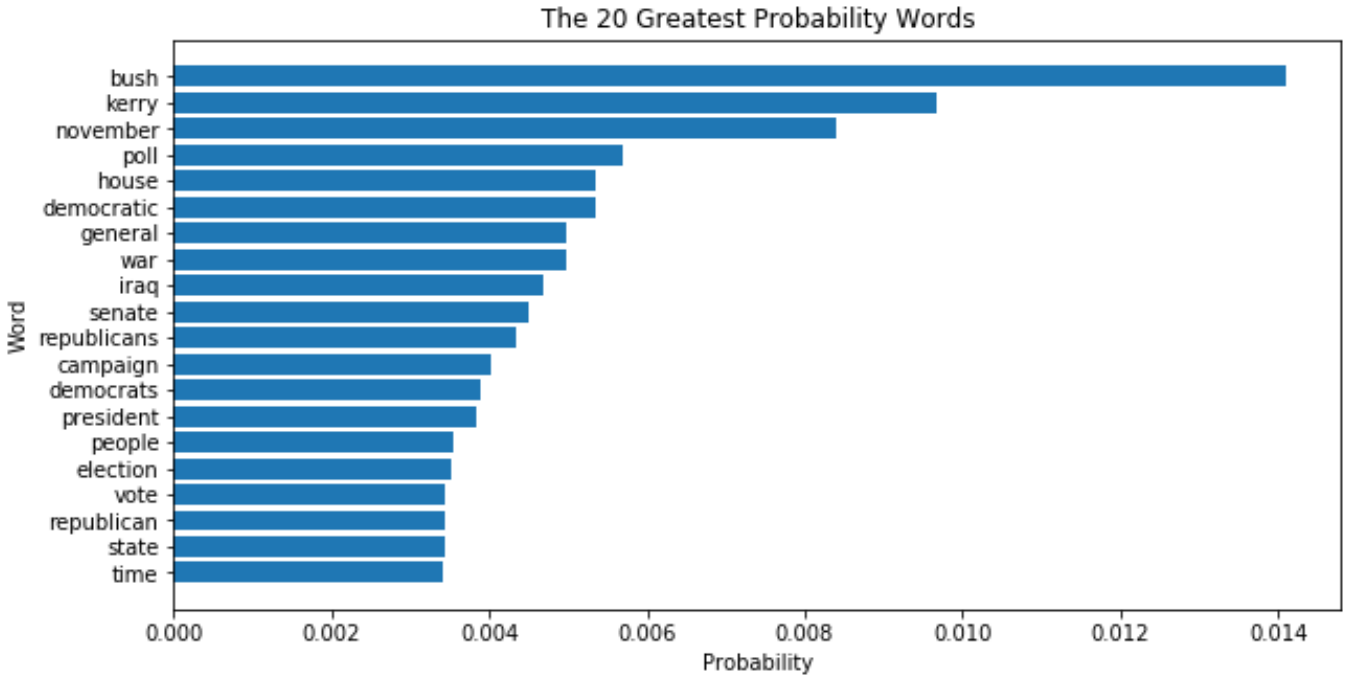

Figure 1: A horizontal bar graph showing the 20 words with the largest probability using the maximum likelihood multinomial over all the words.

The test set log probability is calculated by summing the log probabilities (the beta values) of the words in the test set. As the probabilities are less than one, the highest test set log probability for this model is the single most probable word, bush, with a value of -4.27. If a new word is encountered that does not exist in the training data, then the associated beta value would be zero, giving a lower bound on the test set log probability of negative infinity. Possible implications of these bounds is the need for a per-word probability for measuring performance and pseudo-counts for each word to prevent an infinite probability.

3 Bayesian Inference Using A Symmetric Dirichlet Prior

A symmetric Dirichlet prior, with concentration parameter α, has the form

$$p(\boldsymbol{\beta}) = Dir(\boldsymbol{\beta}|\alpha) = \frac{1}{B(\alpha)} \prod_{m=1}^{M} \beta_m^{\alpha-1} \tag{3}$$

Using this prior and the likelihood, the posterior distribution is

$$p(\boldsymbol{\beta}|\boldsymbol{w}) \propto p(\boldsymbol{w}|\boldsymbol{\beta})p(\boldsymbol{\beta}) \propto \prod_{m=1}^{M} \beta_m^{c_m} \prod_{m=1}^{M} \beta_m^{\alpha-1} = \prod_{m=1}^{M} \beta_m^{c_m+\alpha-1} \tag{4}$$

This is another Dirichlet distribution, $p(\boldsymbol{\beta}|\boldsymbol{w}) = Dir(\boldsymbol{\beta}|c_1+\alpha, ..., c_M+\alpha)$, demonstrating that the prior and posterior are conjugate distributions. The predictive probability for word i is

$$p(x=i|\boldsymbol{w}) = \int p(x=i|\boldsymbol{\beta})p(\boldsymbol{\beta}|\boldsymbol{w})d\boldsymbol{\beta} = \frac{\alpha+c_i}{\sum_{m=1}^{M}(\alpha+c_m)} = \frac{\alpha+c_i}{M\alpha+n} \tag{5}$$

Using a symmetric Dirichlet prior is the same as adding a uniform pseudo-count, α, to the word counts and finding the maximum likelihood solution for β . Figure 2 shows the predictive word probabilities for different values of α, n, M and c. When α approaches zero, the posterior distribution tends to a multinomial distribution. For rare words, there is a possibility of zero predictive probability while common words have a high probability. For large α, the prior dominates the predictive distribution, resulting in similar probabilities for all words.

Predictive Word Probabilities For Varying Alpha, Total Words, Unique Words and Word Count

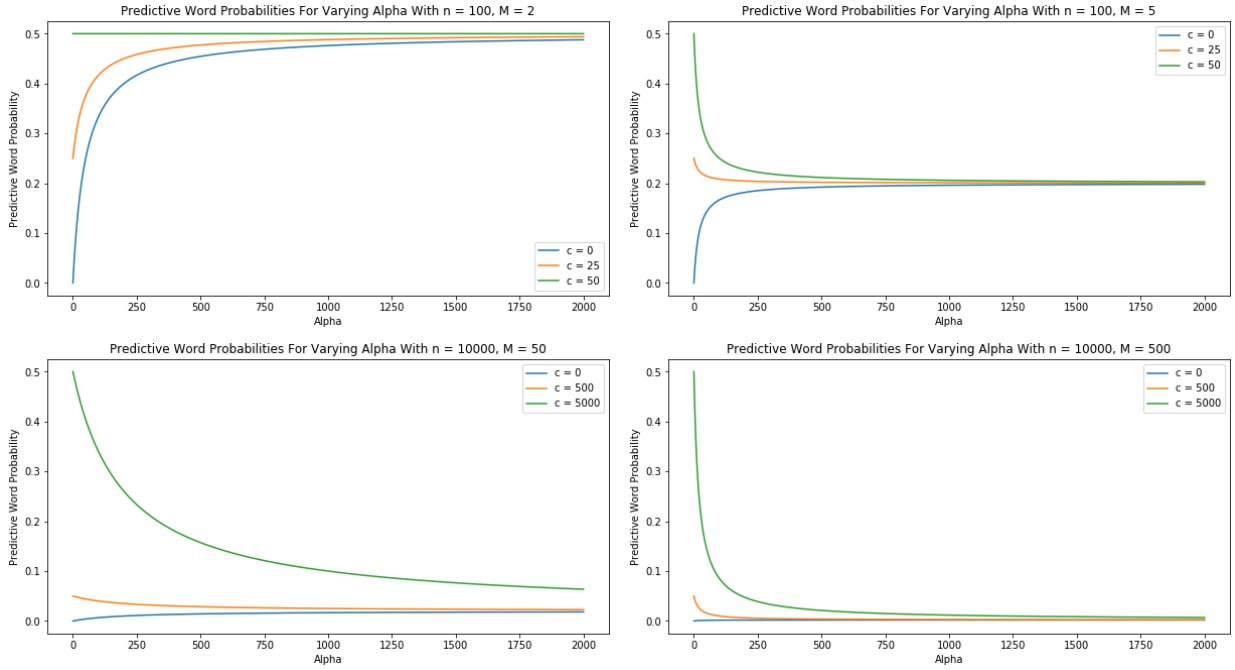

Figure 2: Plots showing the predictive word probabilities using a symmetric Dirichlet prior with varying alpha for four different values of n and M and three different values of c for each graph.

4 Applying The Bayesian Model To Test Documents

The probability of the test document can be calculated using the multinomial distribution:

$$p(\boldsymbol{w}|\boldsymbol{\beta}) = \frac{n!}{\prod_{m=1}^{M} c_m!} \prod_{m=1}^{M} \beta_m^{c_m} \tag{6}$$

Since we have fixed and known word counts for both the training and test documents, the combinatorial factor in the multinomial distribution can be dropped for convenience to give the categorical distribution:

$$p(\boldsymbol{w}|\boldsymbol{\beta}) = \prod_{m=1}^{M} \beta_m^{c_m} \tag{7}$$

Using a symmetric Dirichlet prior, we obtain the log posterior probability similar to Task B:

$$log(p(\boldsymbol{w_{test}}|\boldsymbol{w_{train}}, \alpha)) = log(\prod_{m=1}^{M} \beta_{m,train}^{c_{m,test}}) = \sum_{m=1}^{M} c_{m,test} log(\frac{\alpha + c_{m,train}}{M\alpha + n}) \tag{8}$$

The per-word perplexity can also be calculated:

$$perplexity = exp(\frac{-log(p(\boldsymbol{w_{test}}|\boldsymbol{w_{train}}, \alpha))}{\sum_{m=1}^{M} c_{m,test}}) \tag{9}$$

```
words_a, words_b = defaultdict(int),defaultdict(int) #initialise word dictionaries
logprob, perp = [], [] #initialise output arrays
for (doc_id,word_id,count) in B:
    if doc_id==2001: words_b[word_id]+=count; totb+=count #add word counts
for alpha in alphas:
    tot_p=0
    for w in wordsb.keys():
```

```
        tot_p+=words_b[w]*np.log((alpha+words_a[w])/((n*alpha)+tot_w)) #calculate log probability
    logprob.append(tot_p) #append to probability array
    perp.append(np.exp(-tot_p/tot_b)) #calculate perplexity
```

Code Listing 2: Code to calculate the log probability and per word perplexity for the document with ID 2001.

Figure 3 uses Listing 2 and Equations 8 and 9 to plot the probability and per-word perplexity for the document with ID 2001 for alpha between 0 and 500. A lower per-word perplexity gives a better fit, so alpha = 9 seems optimal from the graph. For smaller alpha, the posterior distribution is skewed towards common words and for larger alpha, the alpha term begins to dominate over the word counts, resulting in poorer distinctions. The results using alpha = 9 are shown in Table 1.

Log Probability And Per-Word Perplexity For The Document With ID 2001 With Varying Alpha

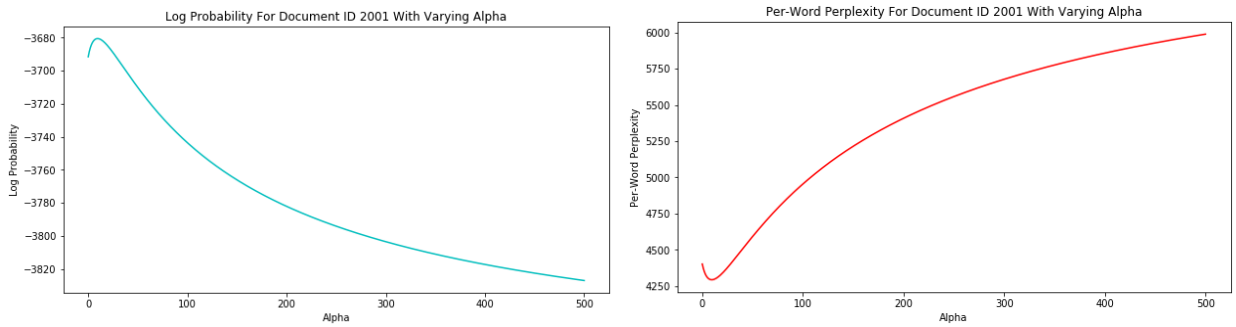

Figure 3: Plots showing the log probability and per-word perplexity of the document with ID 2001 using a symmetric Dirichlet prior with varying alpha.

Log Probability	Per-Word Perplexity
-3681	4295

Table 1: Table showing the log probability and per-word perplexity for the test document with ID 2001 with alpha = 9.

Figure 4 shows the log probabilities and per-word perplexities for all test documents given. The documents contain different word counts, so the perplexities vary between them since rarer words lead to a poorer fit. A uniform multinomial uses the same beta for all words, $\frac{1}{M}$, giving the per-word perplexity

$$perplexity = exp(\frac{-log(p(\boldsymbol{w_{test}}|\boldsymbol{w_{train}}, \alpha))}{\sum_{m=1}^{M} c_{m,test}}) = exp(-log(\frac{1}{M})) = M = 6906 \tag{10}$$

This value is independent of the test documents provided they contain no new words, as the uniform multinomial assumes all words are equally likely.

Log Probabilities And Per-Word Perplexities For All Test Documents

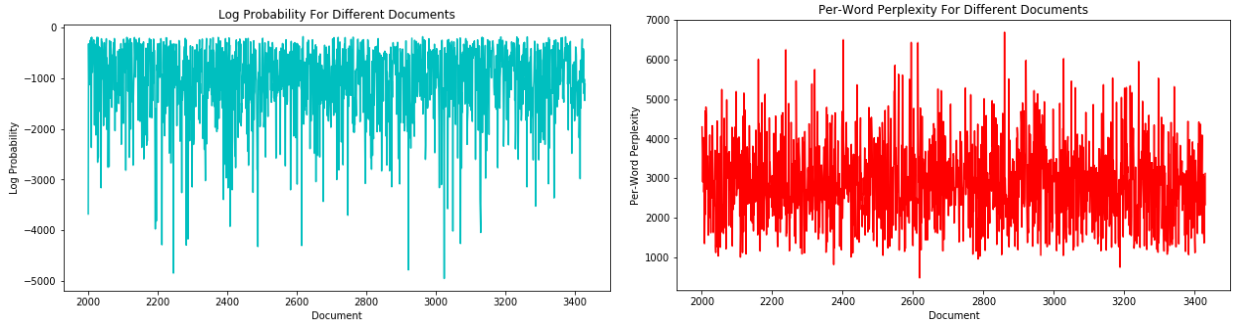

Figure 4: Plots showing the log probabilities and per-word perplexities for all test documents with alpha = 9.

5 Gibbs Sampling For A Mixture Of Multinomials Model

The Gibbs sampler uses the mixing proportions

$$p(\boldsymbol{\theta}|\boldsymbol{z}, \alpha) \propto p(\boldsymbol{\theta}|\alpha)p(\boldsymbol{z}|\boldsymbol{\theta}) \propto Dir(\boldsymbol{c} + \alpha) \tag{11}$$

Here, c_k is the document count for mixture k, leading to the mixture component posterior probability, with D documents,

$$\theta_k = \frac{\alpha + c_k}{\sum_{j=1}^{K}(c_j + \alpha)} = \frac{\alpha + c_k}{K\alpha + D} \tag{12}$$

```
mix_prop = np.zeros((num_iters_gibbs, K)) #initialise mixing proportions matrix
for m in range(K): mix_prop[itera,m] = (alpha+sk_docs[m])/((alpha*K)+D) #fill for each iteration
```

Code Listing 3: Code to calculate the mixing proportions at each iteration of Gibbs sampling.

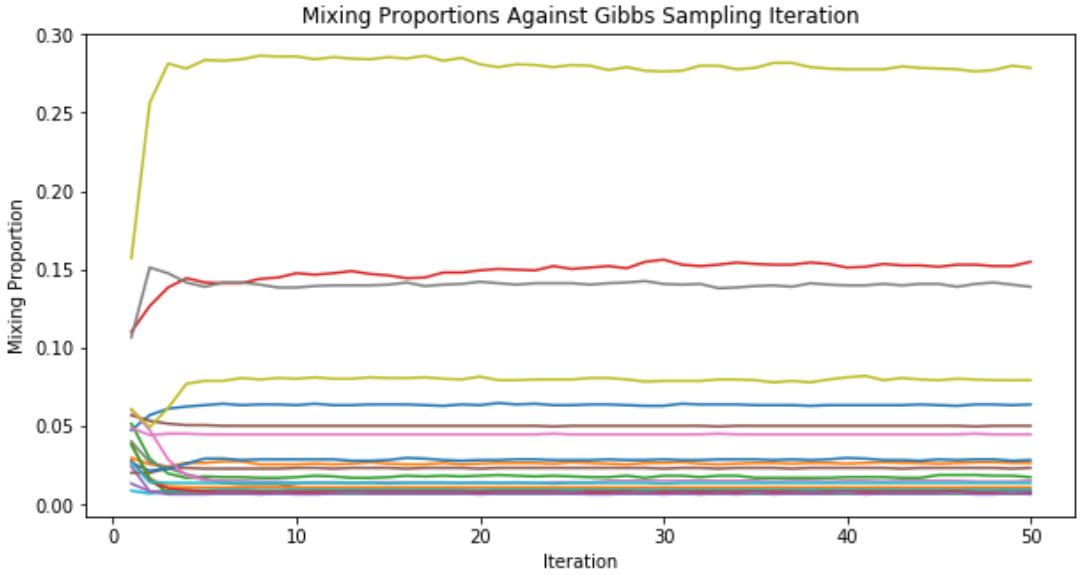

Figure 5: A graph showing the evolution of the K mixing proportions over 50 iterations of Gibbs Sampling using the random seed 1.

The evolution of the mixing proportions for 50 iteration, shown in Figure 5 using the code in Listing 3, suggest that the mixing proportions fluctuate around certain values after an initial rapid fluctuation. The reduction in fluctuations after a burn-in period suggests rapid convergence to a set of mixing proportions, but the final mixing proportions must be independent of the starting seed for convergence. Figure 6 shows the same sampler run with different starting seeds, indicating that convergence has not occurred with 50 iterations, but rather the sampler gets stuck in local optima. This result is echoed by Table 2, which shows that the perplexity is different for the different starting seeds. Exploring the top words for each topic, seeds 4 and 5 have a topic with Knowles, percent and Murkowski as the top words unlike seeds 2 and 3. On the other hand, 2 and 3 have a topic with Coburn, Carlson and Senate as the top words unlike 4 and 5. This again implies that there exist multiple local optima with similar perplexities but different assignments. The difficulty of convergence can be explained by the fact that the posterior distribution has a high dimensionality, meaning exploring the entire set of posterior distributions for the global minimum perplexity incurs a high cost.

Seed			
2	3	4	5
2152	2124	2116	2101

Table 2: Table showing the perplexity for the test documents after 50 iterations of Gibbs sampling using the starting seeds 2, 3, 4 and 5.

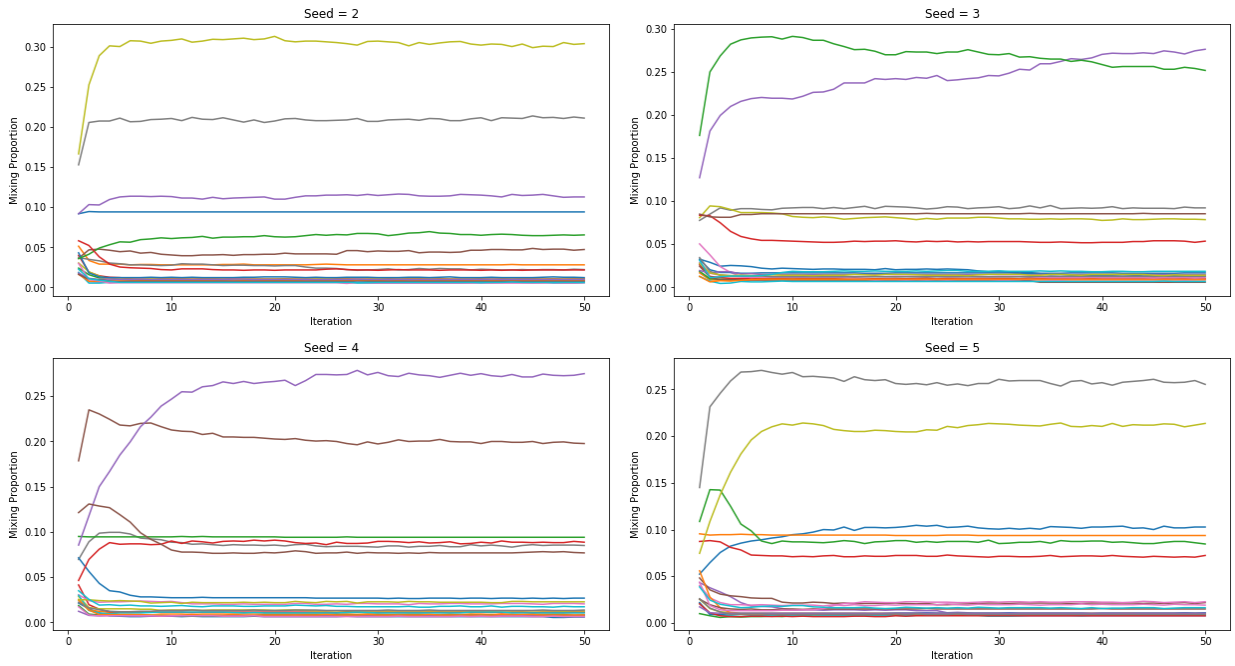

Figure 6: Plots showing the evolution of the mixing proportions over 50 iterations of Gibbs sampling with the seeds 2, 3, 4 and 5.

6 Latent Dirichlet Allocation

Unlike the previous model, the LDA model allows for words within a single document to belong to different topics. The predictive distribution can be written as a product between the topic distribution and the word distribution:

$$p(z_{nd} = k|z_{-nd}, w, \gamma, \alpha) \propto \frac{\alpha + c^k_{-nd}}{\sum_{j=1}^{K}(\alpha + c^j_{-nd})} \frac{\gamma + \tilde{c}^k_{-w_{nd}}}{\sum_{m=1}^{M}(\gamma + \tilde{c}^k_{-m})} \tag{13}$$

```
mix_prop = np.zeros((num_gibbs_iters, K)) #initialise mixing proportions matrix
for m in range(K): #fill for each iteration
    mix_prop[itera,m] = (alpha+np.sum(skd, axis=1)[m])/((alpha*K)+np.sum(np.sum(skd, axis=1)))
```

Code Listing 4: Code to calculate the topic posteriors at each iteration of Gibbs sampling.

Listing 4 uses Equation 13 to calculate the topic posteriors over 50 iterations of Gibbs sampling. The resulting plot, shown in Figure 7, indicates that a single topic dominates the posterior - an example of the rich-get-richer property of Gibbs sampling.
The perplexity for the documents using this method is 1644 - a significant improvement on the mixture of multinomials model, suggesting the documents share common words from the same topic. The topic posteriors fluctuate around different values depending on the initial seed, seen in Figure 8, suggesting the sampler gets stuck at local minima with similar perplexities. Convergence is impractical due to the high cost, so 50 iterations is sufficient to reach a local minimum with an acceptable perplexity.

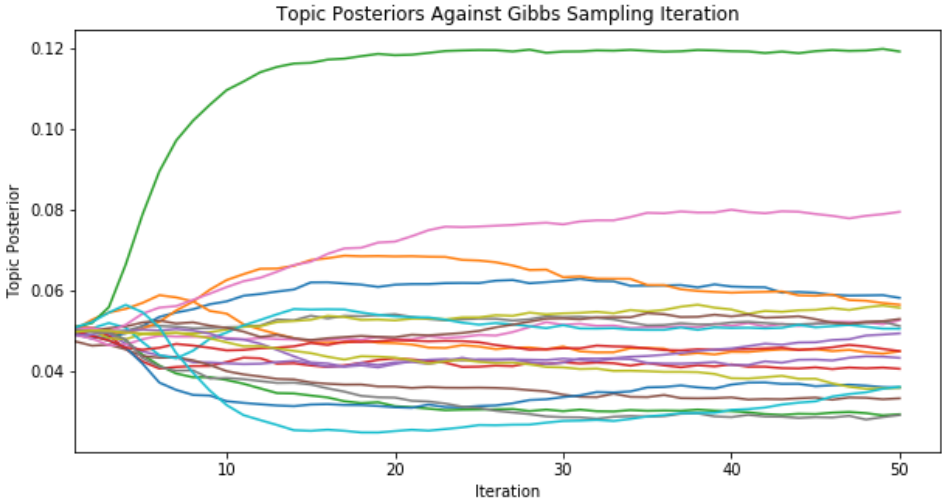

Figure 7: A plot showing the evolution of the topic posteriors over 50 iterations of Gibbs sampling.

Topic Posteriors Against Gibbs Sampling Iteration For Different Starting Seeds

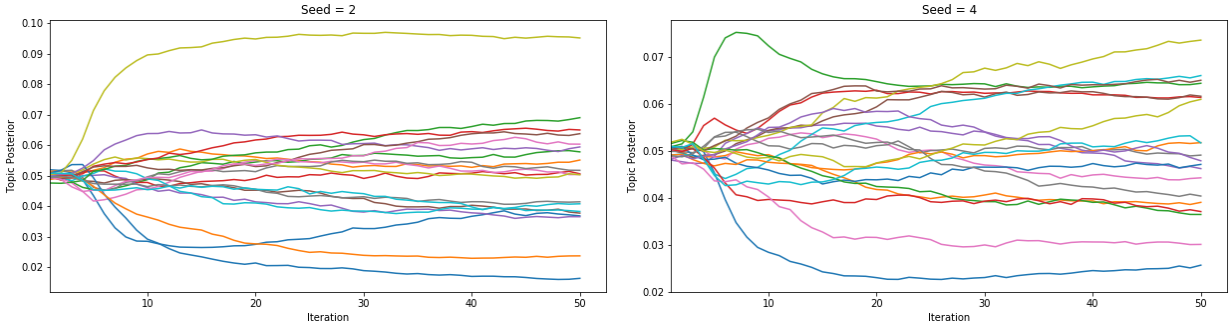

Figure 8: Plots showing the evolution of the topic posteriors over 50 iterations of Gibbs sampling for the starting seeds 2 and 4.

The word entropy is a measure of uncertainty and is calculated by:

$$entropy(k) = -\sum_{m=1}^{M} \beta_{km} ln(\beta_{km}) \tag{14}$$

Due to the use of the natural logarithm, the entropy is measured in nats. Listing 5 uses Equation 14 to plot the word entropies over 20 iterations, shown in Figure 9. After an initial drop, the entropies fluctuate around a local minimum, consistent with Figures 7 and 8. The topics with a lower entropy are more certain about word distribution, containing fewer words with larger probabilities.

```
entropy = np.zeros((num_gibbs_iters, K)) #initialise entropy matrix
for m in range(K): #iterate through the topics
    for word in range(W): #iterate through the words
        beta=(gamma+swk[:,m][word])/((gamma*W)+np.sum(swk[:,m])) #calculate beta
        entropy[iter,m]-=beta*np.log(beta) #calculate the word entropy
```

Code Listing 5: Code to calculate the word entropy for each of the topics at each iteration of Gibbs sampling.

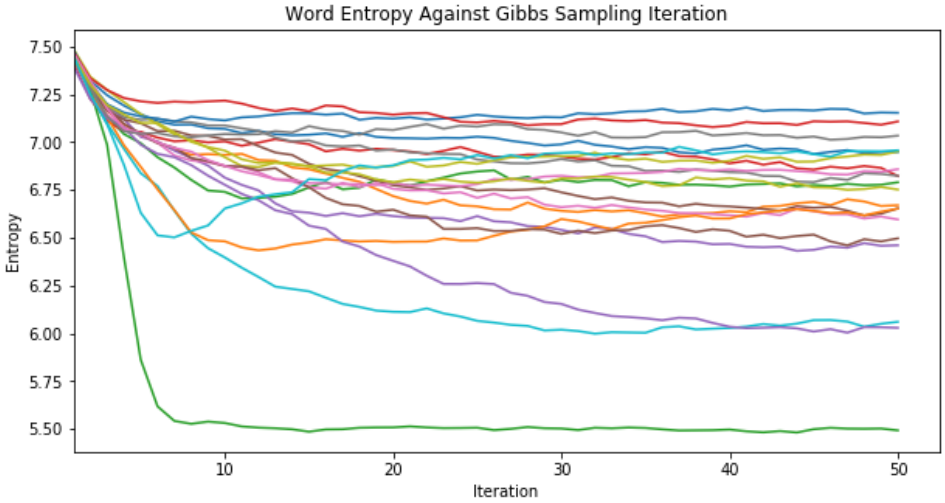

Figure 9: A plot showing the evolution of the word entropies for the topics over 50 iterations of Gibbs sampling.