ENGINEERING TRIPOS PART IIB

4F13 Coursework #3: Latent Dirichlet Allocation

Candidate 5606G

997 Words

December 3, 2021

1 Part A: Maximum Likelihood Multinomial (MLM)

Python A.1:

```
for index in range(W):
    word_identified = np.where(A[:, 1] == index+1)
    occurrences = np.array(A[word_identified, 2])
    wordcounts[index] = np.sum(occurrences)
probabilities = wordcounts/np.sum(wordcounts)
```

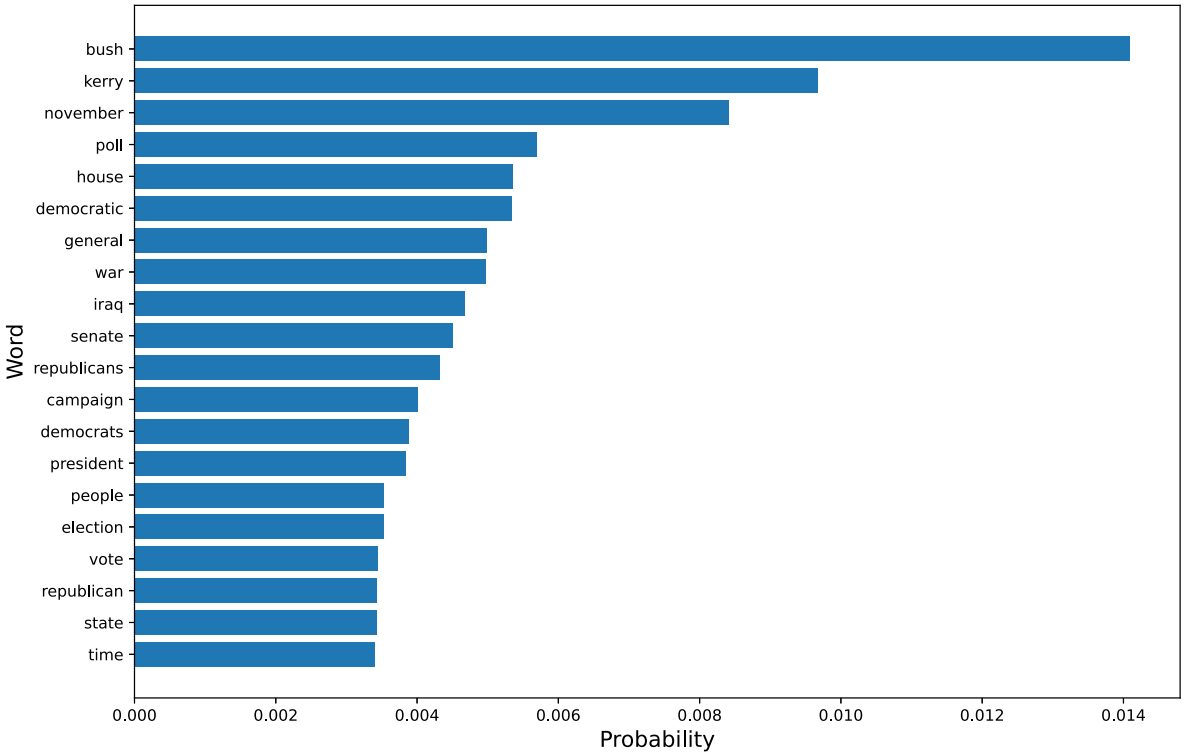

Figure 1: Probabilities of Top 20 Most Likely Words using MLM

$$p_i = \frac{c_i}{\sum_j c_j} \quad (1) \qquad p(\mathbf{k}|\pi, n) = \frac{n!}{k_1!k_2!...k_m!} \prod_{i=1} \pi_i^{k_i} \quad (2)$$

We calculate the parameters of our MLM model by dividing the total occurrences of a word by the total words in the dataset, as in eq. 1. Words not in the dataset are therefore assigned a probability of 0. Any test documents with even a single word not in our training dataset will have a probability of 0 - a log probability of $-\infty$. Probabilities under the multinomial are calculated using eq. 2, which has a maximum where the frequency of each word is as close to its probability of occurring as possible. The height of the peak increases with smaller word counts, and the maximum log probability for a test set is a one-word document of the most highly occurring word, i.e. 'bush'. This has a natural log probability of -4.26.

2 Part B: Bayesian Inference

Python B.1:

```
wordcounts += alpha
total_words = np.sum(wordcounts)
probabilities = wordcounts/total_words
```

Python B.2:

```
# P is the log probabilities of words
def loglik(P, wordcounts):
    return np.sum(P * wordcounts)
```

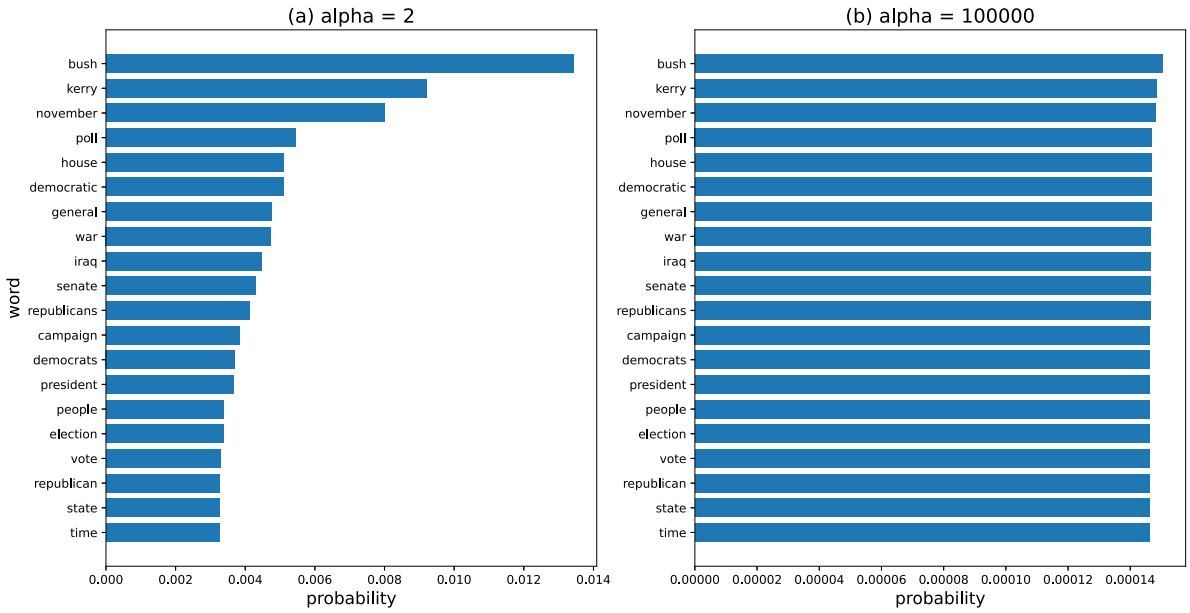

Figure 2: Probabilities of Top 20 Most Likely Words using Bayesian Inference under different alphas

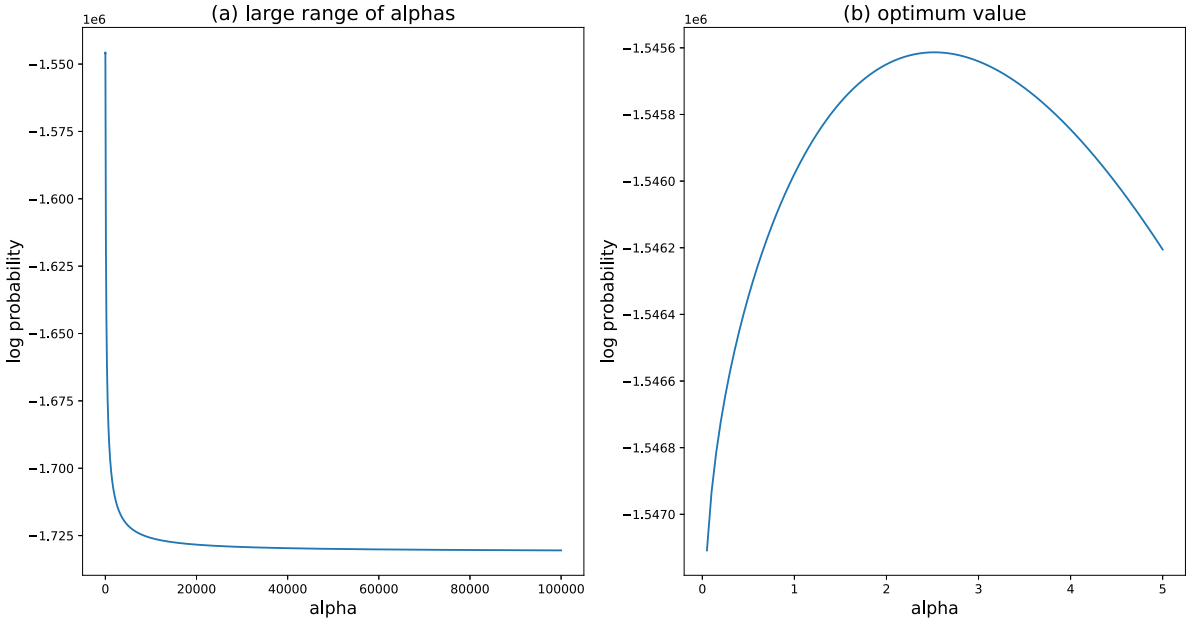

Figure 3: Log Probability of Test Data under different settings of Alpha

In this model, we have included a symmetric Dirichlet model as our prior. This is equivalent to seeing α occurrences of each word before seeing the training data. The expression for the MLM model is given in eq. 1, and eq. 3 is for the Bayesian model. The parameter α is the number of pseudocounts of each word we are adding. The models are equivalent when $\alpha = 0$, and the rank of words from highest to lowest remains the same. The Bayesian model guarantees every word to have a non-zero probability when $\alpha \neq 0$. Fig. 2 shows that our distribution over the top 20 words looks similar at small α, but is almost a uniform distribution under large α. Increasing α reduces the weight on our likelihood, to the point where it becomes irrelevant. To find an optimum value for alpha, I plotted the log probability of our test data under varying values of alpha. Fig. 3(b) shows our optimum value to be around 2.5. This makes sense as we have some probability weighted on all words, but the training data still gives us information on the probabilities of words.

$$p_i = \frac{\alpha + c_i}{\sum_j \alpha + c_j} \tag{3}$$

3 Part C: Perplexity under the Bayesian Model

Python C.1:

```
N_d = np.sum(doc_wordcounts)
ll_d = loglik(P, doc_wordcounts)
d_perplexity = np.exp(-ll_d/N_d)
```

Document	Alpha	Log Probability	Perplexity
2001	2.5	-3685.435403999346	4341.562366283298
All in B	2.5	-1545613.4755187833	2678.984207551639

Table 1: Log Probabilities and Perplexities of Test Documents under the Bayesian Categorical Model

The Multinomial distribution gives us the probability of getting a particular set of word counts for a document. The Categorical distribution gives us the probability of a particular sequence of words occurring in a document (i.e. the order matters). They are equivalent up to a constant factor. We use the categorical distribution for table 1 since we want the probability of a particular test document, rather than the probability of a particular set of counts.

Perplexity is used as a measure of performance on text models, and is calculated by eq. 4. Higher perplexity means higher uncertainty in an outcome. Higher perplexities of models on test documents implies lower performance, as the model is more uncertain on the outcome. The perplexity of a document for a uniform multinomial would be the size of the vocabulary, which is 6906 for our total dataset. Documents have different perplexities due to having different probabilities of occurring under the model, which is related to how well they match the training data. Table 1 shows the perplexity across B is much smaller than document 2001, indicating that the documents in B on average match the training data much more than document 2001.

$$\text{perplexity} = \exp(-l/n) \tag{4}$$

4 Part D: Gibbs Sampling for a Mixture of Multinomials

Python D.1:

```
sk_docs_array[:,iter+1] = sk_docs[:,0]
sk_mix_array = (sk_docs_array + alpha)/(D + W * alpha)
```

We attempt to sample from the posterior distribution of latent topic assignments to documents using a collapsed Gibbs Sampler. The posterior is given in eq. 7, but we can marginalise over θ to give us eq. 8 used in our sampler. Here we have dependency on the likelihoods of documents given their topics, and the topic assignments of all other documents due to marginalising out θ. We can then calculate our mixing proportions at each iteration by the counts of documents assigned to each topic plus a pseudocount α for each topic given by a uniform Dirichlet prior (eq. 9). These have been plotted in fig. 4 for different random seeds.

$$p(z_d = k|\mathbf{w}_d, z_{-d}, \beta, \alpha) \propto \theta_k p(\mathbf{w}_d|\beta_k) \tag{7}$$

$$p(z_d = k|\mathbf{w}_d, z_{-d}, \beta, \alpha) \propto p(\mathbf{w}_d|\beta_k)\frac{\alpha + c_{-d,k}}{\sum_{j=1}^{K}\alpha + c_{-d,j}} \tag{8}$$

$$\theta_k = \frac{\alpha + c_k}{\sum_j \alpha + c_j} \tag{9}$$

A Gibbs Sampler has converged when the samples are representative of the posterior distribution. This distribution is independent of any initial condition, so we would expect our Gibbs samples, after burn-in and thinning, to look the same regardless of starting point. We see in fig. 4(a) that our mixing proportions appear to converge after around 10 iterations and then oscillate around fixed values. These can be seen as exploring around a region of high probability, where variation is caused by some documents flipping between topics, as seen in fig. 5(a). This can be caused by the topics in question having a similar meaning. However, we see that different seeds cause completely different regions to be reached. The Gibbs sampler never escapes one region to explore another, which we can see from fig. 6. This means that the sampler isn't exploring the entire posterior but converges to one region. This makes sense as a region is connected to the topics being assigned certain 'meanings', which will vary between regions. In some ways, it is useful to converge to one categorisation and do multiple runs to compare categorisations. Table 2 shows that the distributions reached between runs have practical use, as they're all similarly reasonable at explaining the test data.

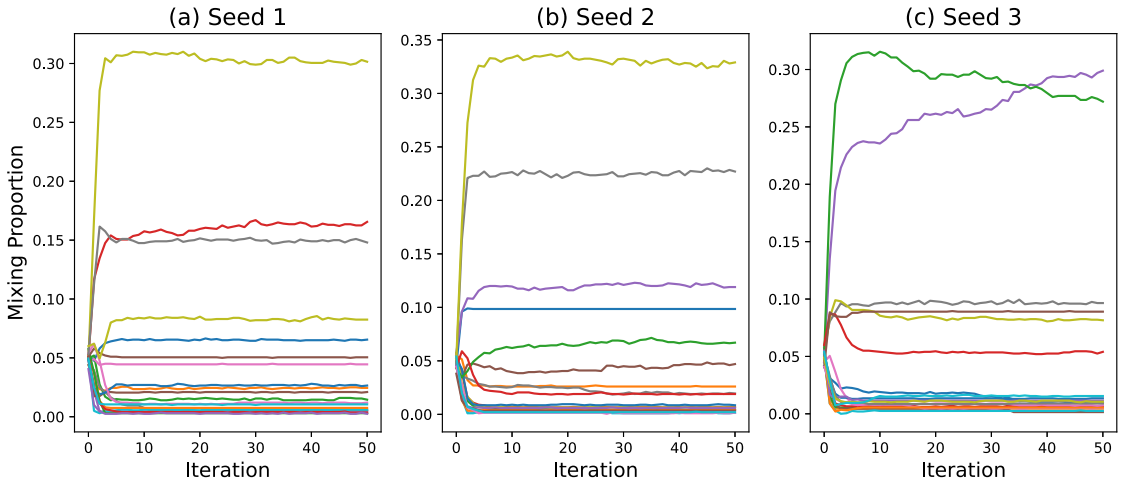

Figure 4: Mixing Proportions over 50 Gibbs Samples of a Mixture of Multinomials Model (BMM)

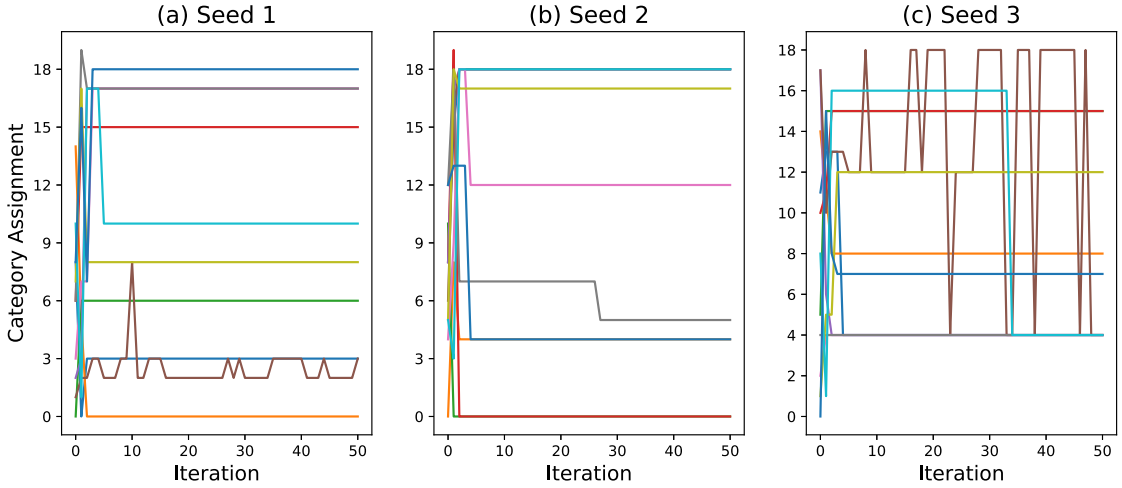

Figure 5: Topic Assignments of First 10 Documents over 50 Gibbs Samples

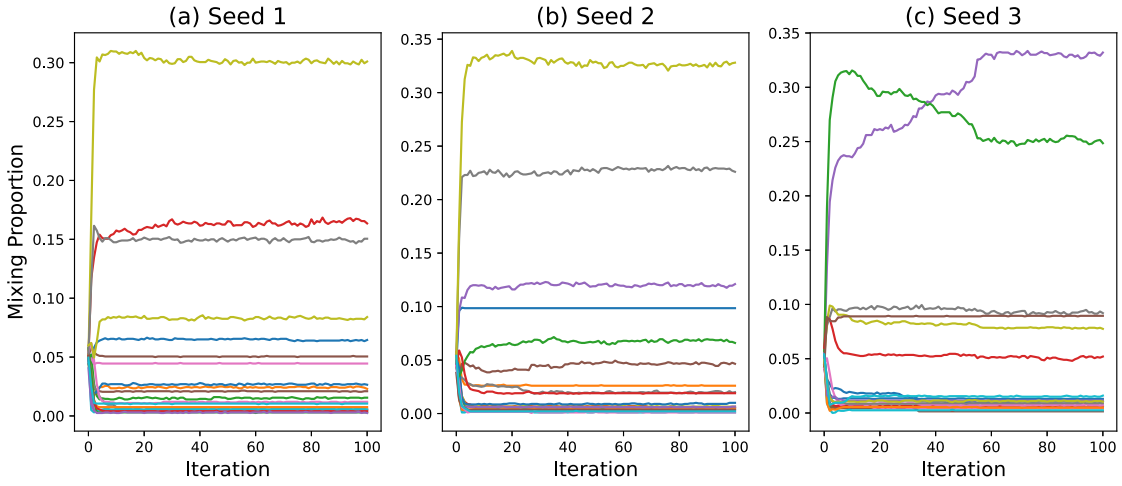

Figure 6: Mixing Proportions over 100 Gibbs Samples of a Mixture of Multinomials Model (BMM)

Model	MLM	Bayesian	Gibbs Seed 1	Gibbs Seed 2	Gibbs Seed 3
Perplexity	-	2679	2093	2151	2126

Table 2: Perplexity over All Test Documents under Different Models

5 Part E: Latent Dirichlet Allocation (LDA)

Python E.1:

```
posteriors = (sk_array+alpha)/(sk_array.sum(axis=0)+K*alpha)
```

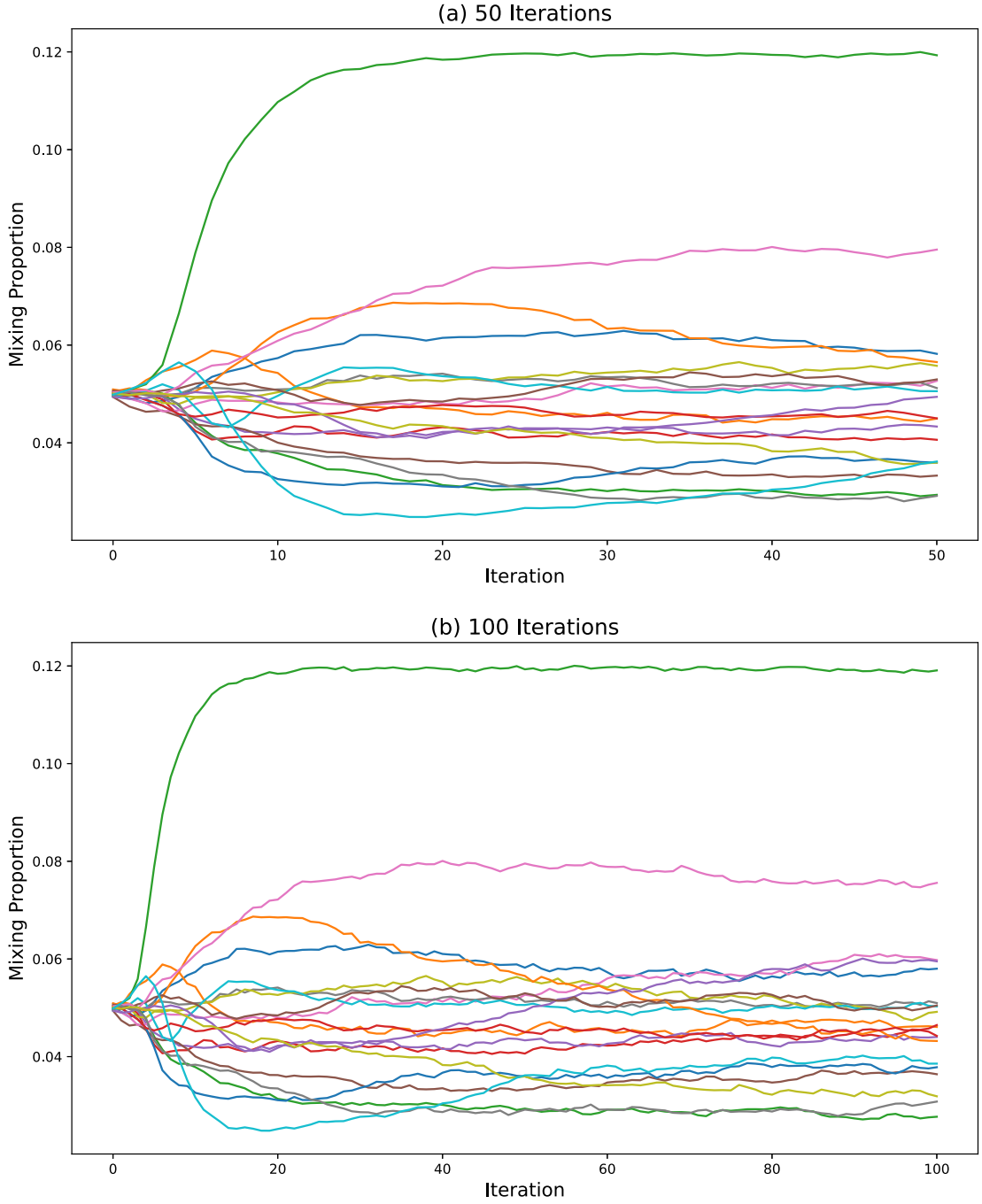

Figure 7: Mixing Proportions over iterations of a Gibbs Run using an LDA Model

The LDA model assigns topics to individual words in documents. The posterior we sample from is given in eq. 10. We calculate our mixing proportions similarly to part (d), except using word counts instead of document counts. The result is plotted in fig. 7. It shows us that most topics converge to a certain proportion after 25 iterations, but a couple take much longer, as shown in fig. 7(b). This could be due to the overlap in vocabularies in each topic. A word strongly associated with 2 topics means it can be unstable in the amount of times it is assigned to each. Gibbs sampling exhibits the 'rich get richer' quality, where a word's probability in being assigned to a topic increases as more words are assigned to it. This is shown in eq. 10, as the counts in the numerators cause the probability to increase as they increase. This could be what is causing some topics to take longer to converge.

$$p(z_{nd} = k|\{z_{-nd}\}, \{w\}, \gamma, \alpha) \propto \frac{\alpha + c^k_{-nd}}{\sum_{j=1}^{K} \alpha + c^j_{-nd}} \times \frac{\gamma + \tilde{c}^k_{-w_{nd}}}{\sum_{m=1}^{M} \gamma + \tilde{c}^k_{-m}} \tag{10}$$

Fig. 8 shows our perplexity decreases smoothly with increased iterations and to significantly smaller values than previous models. We tend towards a value of around 1633 after 90 iterations. 50 iterations gives us 1644, which is adequately close.

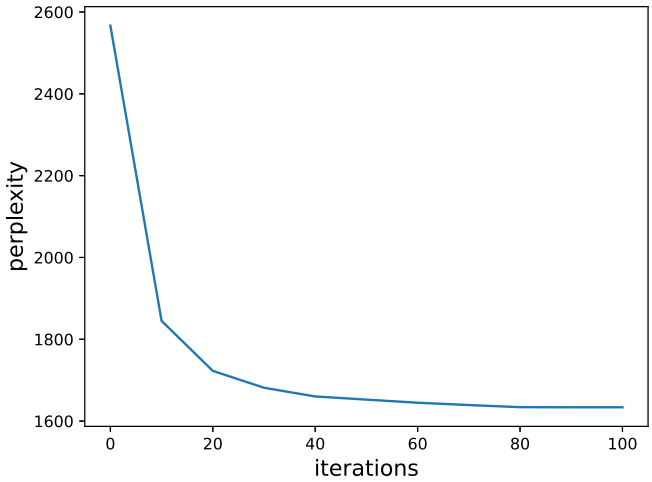

Figure 8: Perplexity over All Documents in B with Increasing Iterations

Python E.2:

```
for i in range(num_gibbs_iters+1):
    swk = np.load('swk_{}.npy'.format(i))
    probabilities = (swk+gamma)/(swk.sum(axis=0)+W*gamma)
    log_probs = np.ma.log2(probabilities).filled(0)
    entropies = -np.sum(probabilities*log_probs, axis=0)
    entropy_array[:, i] = entropies
```

The word entropy of a topic is the average uncertainty when drawing a word from it. We calculate it using eq. 11, where the probabilities are given by word counts with pseudocounts as done before. The entropy value is in bits, since we use base 2 in our log. It is equivalent to the average number of bits needed to encode a word from the distribution, assuming an optimal code is used. Fig. 9 shows us all our topics start with high entropies and reduce as they are better defined. Most topics converge around a mean between 9 and 10.5 bits with some doing better. The convergence stays relatively consistent from 50 iterations to 100. These look similar across different runs. Topics with the lowest entropies have the most well-defined underlying meanings, as they have higher counts on words with higher probabilities.

$$H(x) = -\sum_{i=1}^{n} p(x_i) \log_2(p(x_i)) \tag{11}$$

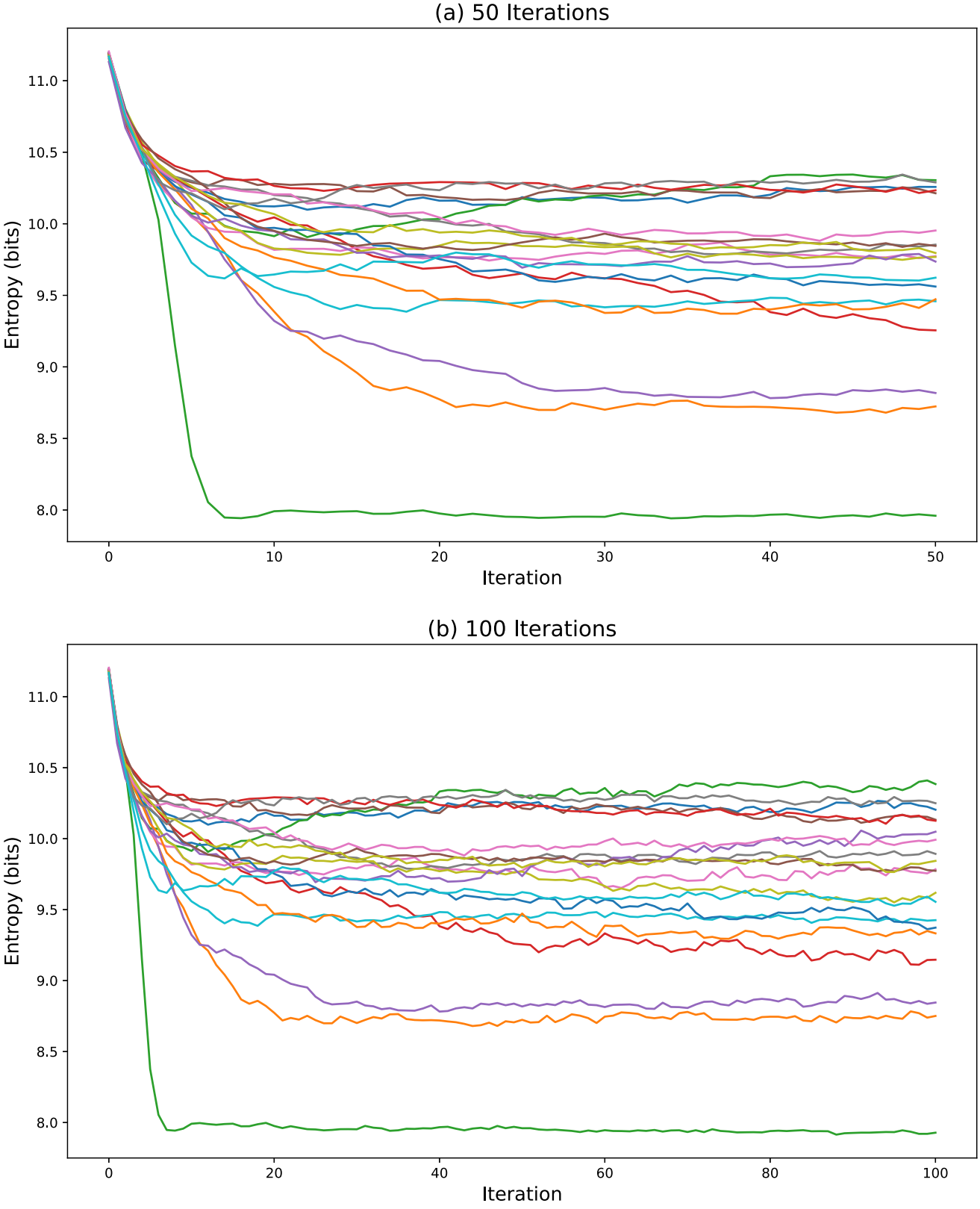

Figure 9: Word Entropies of Each Topic over Gibbs Iterations