1. Maximum Likelihood

   The maximum likelihood model assigns probability of each word equal to the fraction of the collection of words is that word.

$$p_i = \frac{c_i}{\sum_j c_j} \tag{1}$$

   This model gives the twenty most probable words as shown in figure 1.

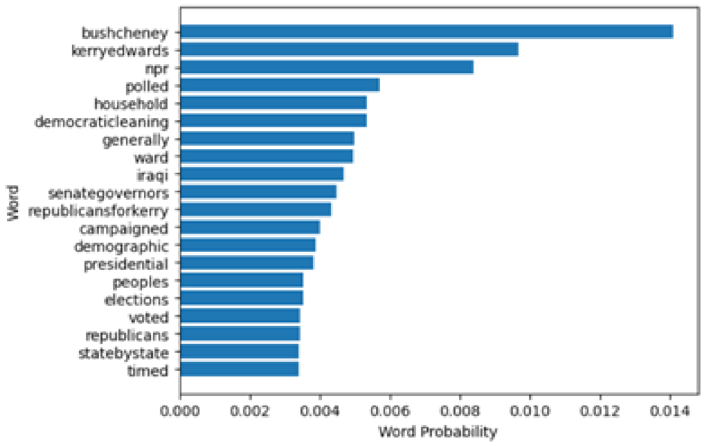

Figure 1: The twenty most likely words in the document collection using a ML approach

   The most common word `bushcheney` has probability 0.014, meaning that the test set with the highest log probability is one that only contains `bushcheney` and will have per-word log probability log(0.014).

$$\frac{1}{N}\log(p) = \frac{1}{N}\sum_{i=0}^{N-1}\log(p_i) \leq \frac{1}{N}\sum_{i=0}^{N-1}\log(p_{max}) = \log(p_{max}) \tag{2}$$

   Any word not in the training set will be assigned probability zero, meaning a test set with any of these words will have probability zero. It is undesirable to have possible test sets with a probability zero, and a maximum possible probability so low. We conclude that a maximum-likelihood model is insufficient.

2. Bayesian Predictive

   Bayesian inference introduces a Dirichlet prior to account for words that are not in the training set. The posterior distribution is then simply the sum of the Dirichlet parameters and the count vector re-normalized, this is because the Dirichlet distribution is conjugate to the multinomial. Since there is no prior information to suggest that any word is more likely than another a symmetric Dirichlet distribution is used which gives the special case

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

   The difference between this posterior and the likelihood in equation 1 is the Dirichlet parameter α which is equal to the number of pseudo-counts we assign to each word plus one. From equation 3 it can be seen that as $\alpha \to \infty$ the posterior tends to a uniform distribution and as $\alpha \to 0$ it is simply the likelihood from equation 1. The effect of this on common words is shown in 2, increasing alpha reduces the probability of more common words, a higher alpha leads to less influence from the training data counts on the model. The effect on rarer words is shown in figure 2, the words are near uniform anyway, so increasing alpha simply increases their probability. A large alpha leads to higher probability for words that were not present in the training set which leads to better generalisation but if alpha is increased too much then we are not using the information from the training set and there was little point training the model at all.

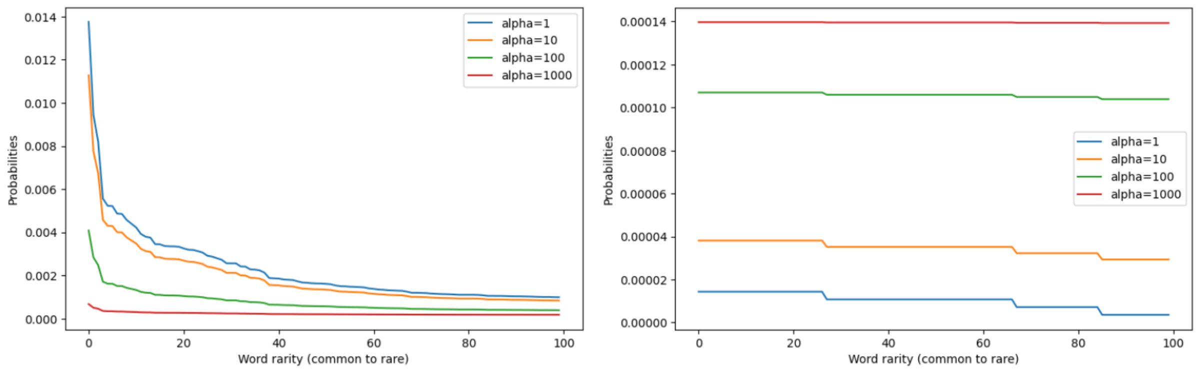

Figure 2: Posterior probability for varying alpha for common words (left) and rare words (right)

3. Testing Bayesian Predictions

   With this Bayesian model, we can evaluate the log-probability by using either a categorical (equation 4) or multinomial (equation 5) distribution.

$$p(\mathbf{w}_d|\boldsymbol{\beta}) = \prod_{i=1} \beta_i^{k_i} \tag{4}$$

$$p(\mathbf{w}_d|\boldsymbol{\beta}) = \frac{n!}{c_1!c_2!...c_m!} \prod_{i=1} \beta_i^{k_i} \tag{5}$$

	categorical	multinomial
$\alpha = 1$	-3689	-1692
$\alpha = 10$	-3681	-1684
$\alpha = 100$	-3744	-1748
$\alpha = 1000$	-3852	-1856

Table 1: Log probabilities of test document 2001

The multinomial distribution considers only the counts of each word, with no consideration for order, the categorical distribution however is specific to an order of words. This is why the multinomial has higher log-probability because there are fewer overall outputs the model could generate. Since a document would have no meaning as simply a count of words, the categorical distribution is required for log-probability calculation. Figure 3 shows the perplexity for document 2001 for a range of alpha values. We observe a minima, where the increase in probability for unseen words balances the trained probabilities from the training data to generate the most likely documents. Different documents will differ in perplexity, as each has a different word distribution, and those with a higher proportion of common words are more likely to have been generated from the model giving them a lower perplexity.

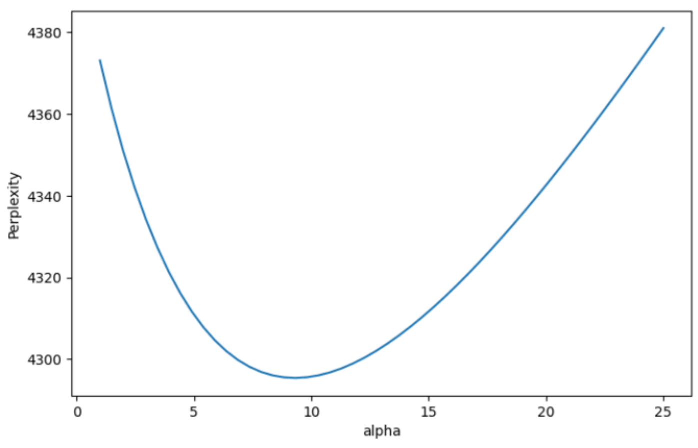

Figure 3: Perplexity for document 2001 for a range of alpha

Figure 4 shows the perplexity of the joint distribution of all words from all documents, it is lower than for document 2001 showing that document 2001 was a rarer document were it generated from the model. If we used a uniform multinomial rather than training the model, the perplexity has an upper bound equal to the size of the vocabulary, which is greater than from the Bayesian model.

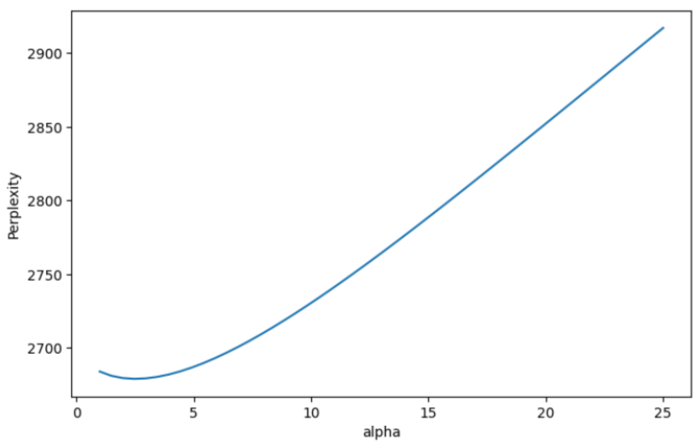

Figure 4: Perplexity for entire collection for a range of alpha

4. Bayesian Mixture Model (BMM)

Rather than computing word probabilities across the whole document set, we can assign latent variables (topics) to each document with each having its own word distribution. The topics are categorically distributed, and the word distributions are categorically distributed conditioned on the topics.

$$z_d \sim Cat(\theta)$$
$$w_{nd}|z_d \sim Cat(\beta_{z_d})$$

For this model, computing the posterior distribution on document topics is intractable. We instead use Gibbs sampling to sample parameters in turn with all others fixed. This leads to convergence to the correct topic distribution across documents. Figure 5 shows the convergence of mixing proportions for each topic (proportion of documents assigned to each topic) for $K = 20$ different topics. We observe convergence here after about 15 iterations. However, by changing the random seed in python different posterior distributions are generated, this implies the Gibbs sampler is not reaching a true posterior and is instead converging to local minima. Also many of the topics have low posterior densities, it is likely we could reduce the number of topics without affecting the model performance.

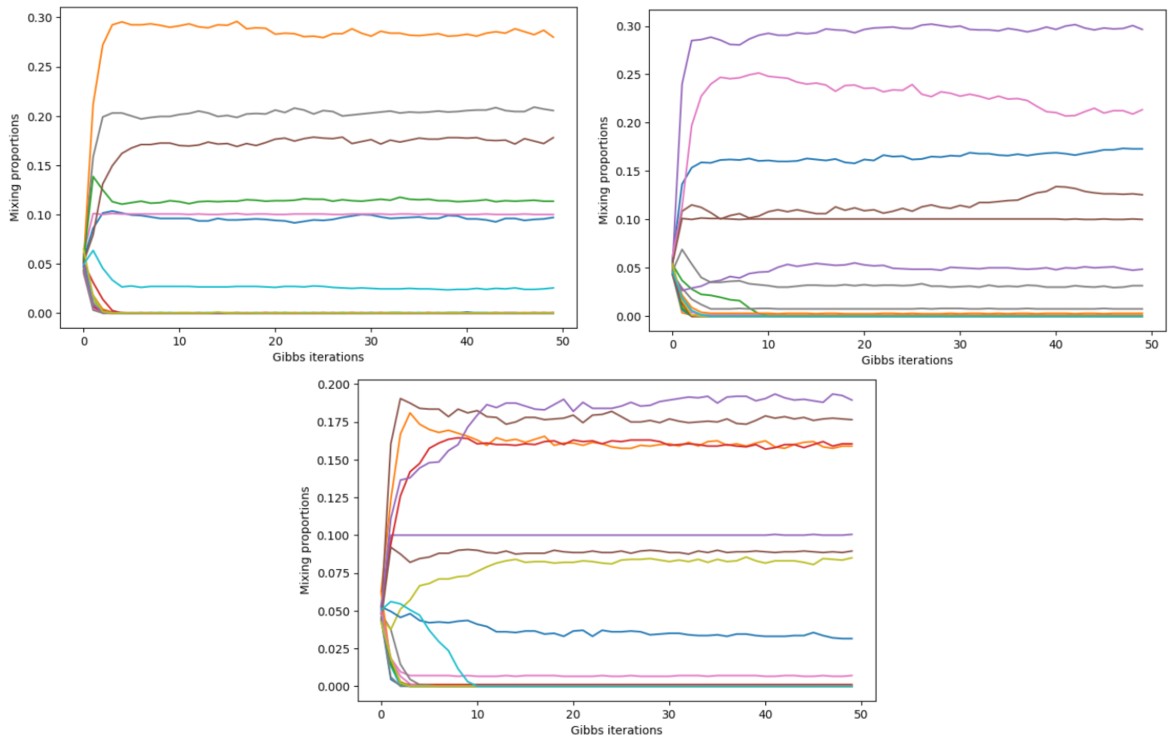

Figure 5: Mixing proportions for BMM model with increasing Gibbs iterations

5. Latent Dirichlet Allocation (LDA)

Another approach is to assign topics to individual words within a document. This allows a single document to have multiple topics (which is more realistic). Again, we use a Gibbs sampling approach to compute the posterior, as it is otherwise intractable. Considering the joint distribution of every word in the document collection yields the evolution of the posterior shown in figure 6.

We again observe that many topics have very low posterior probabilities, it is unlikely that 20 topics were necessary. Most of the posterior density is taken up by a few of the topics, we can attribute this to the "rich get richer" property of this sampling approach. Table 2 shows the perplexity after 50 Gibbs iterations, alongside those for BMM and Bayesian Predictive.

Bayesian Predictive	BMM	LDA
2679	2044	1915

Table 2: Perplexities of different approaches

LDA boasts the lowest perplexity here, the model is the most complex and so is likely to fit the data best: giving the highest log-probability of the test documents. The model is good after 50 Gibbs iterations, however we can see from figure 6 that not all topics have converged, for example topics 14 and 15 appear to be crossing towards 50 samples, it may have been desirable to run the sampler for slightly longer. The evolution of entropy with Gibbs samples as shown in figure 7 is

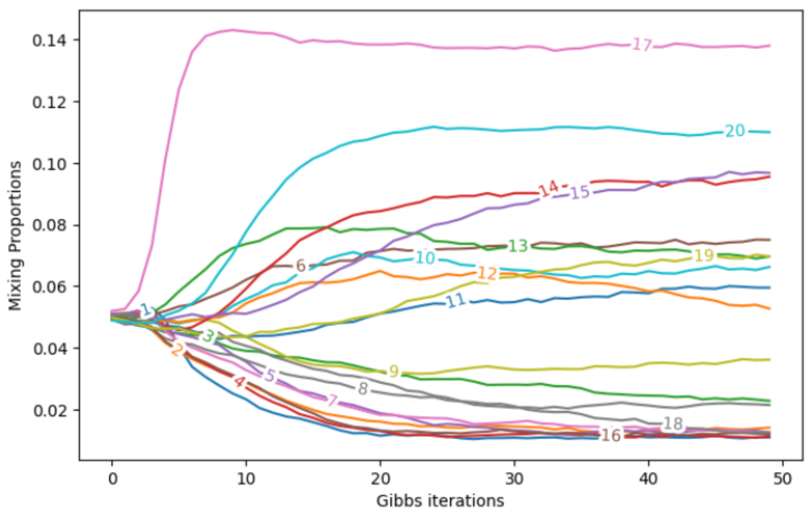

Figure 6: Evolution of posterior distribution for LDA

also of interest. Entropy decreases with more samples as expected, but for some topics it does not decrease much. It is not surprising to observe that topics with higher log-probabilities have a lower entropy, these topics are more prolific in the document collection, and thus hold more information about it.

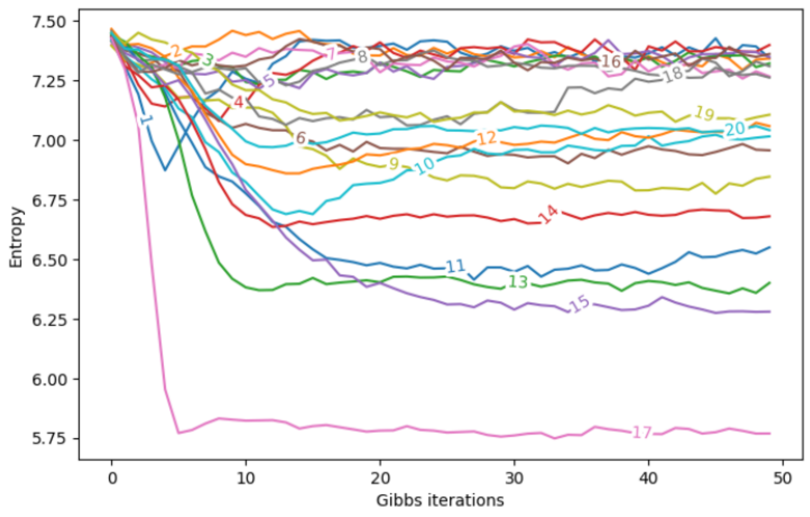

Figure 7: Evolution of topic entropy

6. Appendix

```
# generate array with index = wordId, value = count
for i in range(N+1):
    train_vocab_counts[i] = np.sum(train_counts[np.where(train_words==i)[0]])

# get 20 most likely words and their probabilities
indexes = np.argsort(train_vocab_counts)[::-1]
top_words = V[indexes[:20]][:,0]
train_probs = train_vocab_counts[indexes[:20]]/np.sum(train_vocab_counts)
```

Listing 1: Part A

```
# compute bayesian probabilities
bayes_train_counts = counts + np.full(len(V)+1,alpha)
bayes_train_probs = bayes_train_counts/np.sum(bayes_train_counts)
```

Listing 2: Part B

```
def log_prob_cat(train_prob, test_counts):
    out = 0
    for i in range(1,len(test_counts)):
        out += test_counts[i]*np.log(train_prob[i])
    return out

def get_perp(train_prob, test_counts):
    l = log_prob_cat(train_prob, test_counts)
    return np.exp(-l/np.sum(test_counts))
```

Listing 3: Part C

```
# in bmm.py in gibbs loop
sd_iters[iter] = sd

# in main code
perp, swk, sd_iters = BMM(A, B, 20, 1, 1)
counts = np.zeros((50, 20))
for i in range(50): # iterations
    for t in sd_iters[i]: #topics
        counts[i][int(t)] += 1
ax.plot(counts/np.sum(counts[0]))
```

Listing 4: Part D

```
# in lda.py in gibbs loop
sk_iters[iter] = sk
calc_entropy = lambda p : -1*np.dot(p, np.log(p))
for i in range(K):
    entropy[iter][i] = calc_entropy(swk[:,i]/np.sum(swk[:,i]))

# in main code
sk_iters, perp, entropy = LDA(A, B, 20, 1, 1)
```

Listing 5: Part E