4F13 - COURSEWORK #3: LATENT DIRICHLET ALLOCATION

CCN: 5654F

1. Task A: Multinomial Maximum Likelihood

The maximum likelihood multinomial assumes each word is drawn from one categorical distribution: $w_{nd} \sim \text{Cat}(\boldsymbol{\beta})$, where $\boldsymbol{\beta}$ is a vector containing the probability each word will be used. The maximum likelihood (ML) estimate for $\boldsymbol{\beta}$ for training set $\mathcal{A}$ is simply the empirical word frequency for each word, as seen in Equation 1, where $\boldsymbol{c}$ is the vector of counts of distinct word occurences in the document set. $\boldsymbol{c}$ is calculated as shown in Listing 1, and then divided by the total word count over the document, giving us $\hat{\boldsymbol{\beta}}_{ML}$.

$$\begin{aligned}\hat{\boldsymbol{\beta}}_{ML} &= \underset{\beta}{\text{argmax}} \log \prod_{d=1}^{D} \prod_{n}^{N_d} \text{Cat}(w_{nd}|\beta) \\ &= \underset{\beta}{\text{argmax}} \log \text{Mult}(c_1, \ldots, c_M|\beta, N) = \frac{\boldsymbol{c}}{N}\end{aligned} \tag{1}$$

```
word_counts = np.zeros(W, dtype=int)
for doc_num in range(D):
    doc__word_indices = np.where(A[:, 0] == doc_num + 1)
    word_indices = np.array(A[doc_word_indices, 1])
    word_counts[word_indices - 1] += np.array(A[doc__word_indices, 2])
```

Listing 1. Task A code excerpts

The most and least probable words are then plotted in two bar plots, as shown in Figure 1. The highest log probability possible is generated when the test set contains only the most probable word: "bush". For a test set L words long this is a log probability of $\log \hat{\boldsymbol{\beta}}_{ML,\max}^{L} = L \log 0.0141 \approx -4.3L$ (0.0141 is the probability of "bush"). The lowest log probability occurs when any of the words in the test set did not appear in the training set, giving a probability of 0 and a log probability of $-\infty$. This is a very oversimplistic model as it cannot handle words that do not appear in the training set.

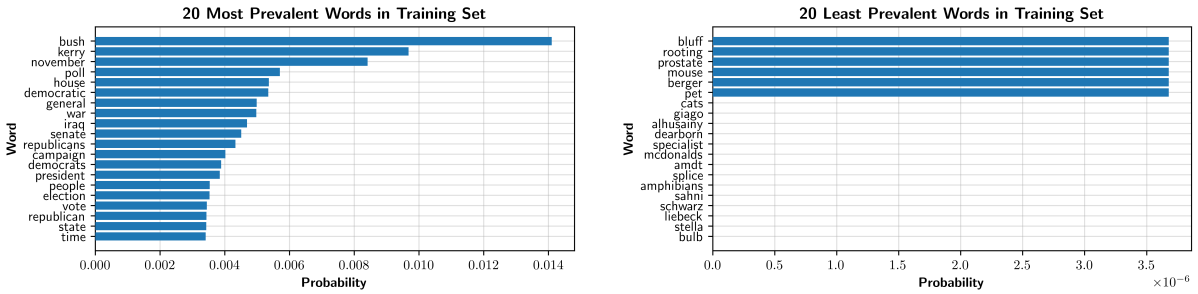


Figure 1. Task A Figures

2. Task B: Dirichlet Prior

The model was improved by adding a symmetric Dirichlet Prior: $\boldsymbol{\beta} \sim \text{Dir}(\boldsymbol{\beta}, \alpha)$, with concentration parameter α. Using Bayes Rule, the posterior distribution is calculated as in Equation 2, and as this is also a Dirichlet (due to the Dirichlet being a conjugate prior to the categorical), the maximum a posteriori (MAP) predictive distribution is as in Equation 3. The effect of the prior is to add a pseudocount of α to each count, by adding the lines in Listing 2.

Date: Wednesday 15th November 2023.

$$P(\boldsymbol{\beta}|\mathcal{A}) \propto P(\boldsymbol{\beta}) \cdot P(\mathcal{A}|\boldsymbol{\beta}) \propto \prod_{m=1}^{M} {\beta_M}^{c_m} \prod_{m=1}^{M} {\beta_M}^{c_m+\alpha-1} \tag{2}$$
$$= \mathrm{Dir}(\boldsymbol{\beta}|\boldsymbol{c} + \alpha \cdot \mathbf{1})$$

$$\hat{\boldsymbol{\beta}}_{MAP} = \frac{\boldsymbol{c} + \alpha \cdot \mathbf{1}}{\sum_{l=1}^{M} c_l + \alpha} \tag{3}$$

```
prior_counts = alpha * np.ones(W)
posterior_counts = word_counts + prior_counts
total_count = np.sum(posterior_counts)
```

Listing 2. Task B code excerpts

Figure 2 shows different posterior predictive distributions for different values of α, for the top 20 and bottom 20 words (for $\alpha = 10$). The change is clear for the bottom 20 words: unseen words in training have a non-zero probability of occuring. As α increases the word probabilities become more uniform across the vocabulary. When $\alpha = 0$ the prior does nothing.

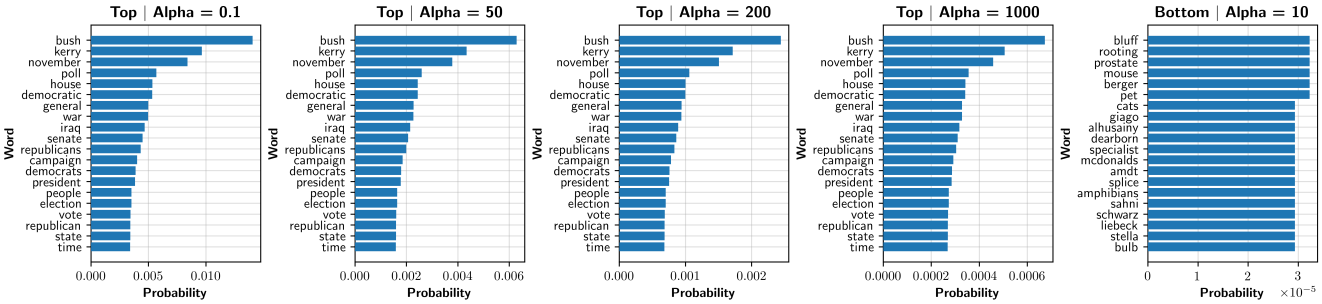


Figure 2. Dirichlet Prior - Posterior Predictive Distributions

3. Task C: Log Probabilities

To calculate the log probability $L(d)$ of a test document d using the Bayesian Model, Equation 4 is used. $\{w^*_{nd}\}_{n=1}^{N_d}$ is the set of words in d. c^*_m is the count of word m in d. The data we are given howvever is just the word counts for each document, not the word order, which is obviously an important part of document analysis. To account for this a combinatorial term could be added as in Equation 5. However, we can consider all word orderings of documents to be equally likely for the moment and therefore this can be ignored and the categorical distribution function is used.

With α set to 0.1, the log probabiility of Document 2001 was calculated to be -3691.2, with 440 total words.

Also of use is the per-word perplexity, which is a measure of performance, and the definition is given in Equation 6. Listing 3 shows how the log probability and perplexity for Document 2001 were calculated.

$$L(d) = \log P\left(\{w^*_{nd}\}_{n=1}^{N_d} | \mathcal{A}\right) = \log \prod_{n=1}^{N_d} P\left(w^*_{nd}|\mathcal{A}\right) = \sum_{n=1}^{N_d} \log P\left(w^*_{nd}|\mathcal{A}\right) \tag{4}$$
$$= \sum_{m=1}^{M} c^*_m \log P\left(w^* = m|\mathcal{A}\right) = {\boldsymbol{c}^*}^T \log \hat{\boldsymbol{\beta}}^*$$

$$P\left(\{w^*_{nd}\}_{n=1}^{N_d} | \mathcal{A}\right) = \frac{N_d!}{\prod_{m=1}^{M} c^*_m!} \prod_{n=1}^{N} P(w^*_{nd}|\hat{\boldsymbol{\beta}}^*) \tag{5}$$

$$p(d) = \exp\left(-\frac{L(d)}{N_d}\right) \tag{6}$$

Table 1 contains the per-word perplexities for Document 2001 of the test set $\mathcal{B}$ and the whole of $\mathcal{B}$, as well as the perplexity of $\mathcal{B}$ when using a uniform multinomial. This is just equal to the number of distinct words in $\mathcal{B}$, as a perplexity of n implies a confidence the same as if an

```
log_posterior_frequencies = np.log(posterior_frequencies)
test_doc_log_probability = np.dot(test_doc_word_counts, log_posterior_frequencies)
test_doc_perplexity = np.exp(-1 * test_doc_log_probability / test_doc_total_words)
```

LISTING 3. Task C code excerpts

—	Document 2001	All Documents	Uniform Multinomial
Per-word Perplexity	4399.0	2697.1	6906

TABLE 1. Task C Perplexities

n-sided dice was choosing each word. Different documents have different perplexities due to the fact that they contain a different subset of the whole word set that is trained over. A document that has a higher proportion of words that were more common in the training set will have a higher log probability and therefore a lower (better) perplexity. Document 2001 has a much higher perplexity than $\mathcal{B}$ as a whole, and therefore must contain a higher proportion of words that are scarcer in $\mathcal{A}$ than the average.

4. Task D: Bayesian Mixture of Multinomials

Figure 3 shows an overview of the model used in Task D. $z_d \in 1, \ldots, K$ is a latent variable denoting the topic of each document. The probability of a document being about topic k is $\theta_k \in \boldsymbol{\theta}$, which is initialised with a symmetric Dirichlet prior.

The python function `BMM` was provided which returns the test set perplexities and multinomial weights over words using gibbs sampling. The function was modified to return the number of documents assigned to each topic at each iteration by adding a new return array which contains a copy of `sk_docs` at each iteration, as seen in Listing 4.

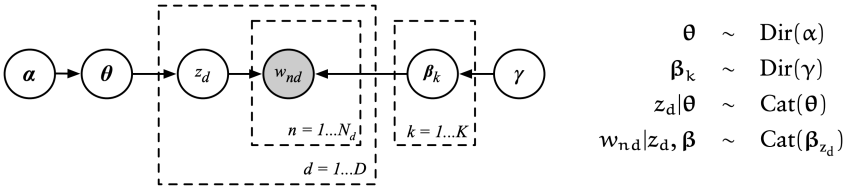


FIGURE 3. Bayesian Mixture of Multinomials Model[1]

```
sk_docs_history = np.zeros(
    (num_mixture_components, num_iters_gibbs + 1), dtype=int
)
sk_docs_history[:, 0] = np.copy(sk_docs)[:, 0]
# ...
for iteration in tqdm(range(num_iters_gibbs)):
    # Do gibbs ....
    sk_docs_history[:, iteration + 1] = np.copy(sk_docs)[:, 0]
```

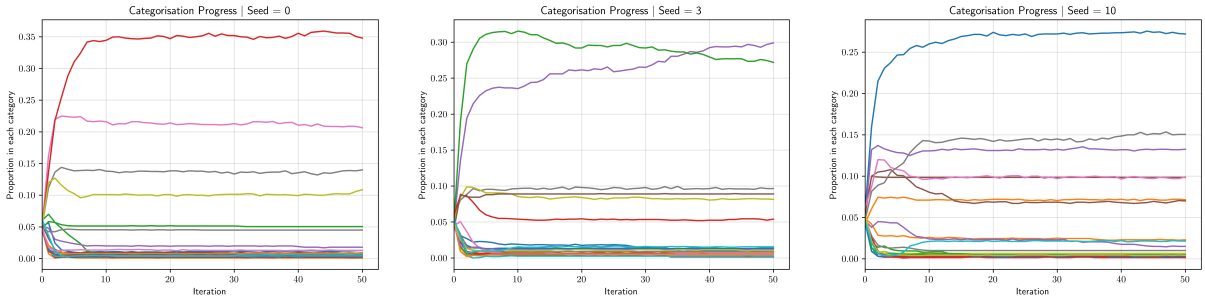
LISTING 4. Task D code excerpts

The model could then be trained using Gibbs Sampling over the training set using $\alpha = 10$ for the prior on mixture components and $\gamma = 0.1$ for the prior on words. The θ_ks can be calculated in the same way as βs earlier, shown in equation 7, although $\boldsymbol{c}$ now is a vector of the topic counts.

$$\boldsymbol{\theta}_k = \frac{\boldsymbol{c} + \alpha \cdot \mathbf{1}}{\sum_{l=1}^{M} c_l + \alpha} \tag{7}$$

Figure 4 shows the topic allocation progress for 3 different values of `np.random.seed`. For the most part, each reaches a stationary distribution, implying convergence of the model to a local minimum. However, different seeds yield different distributions, which are not explainable

[1]CUED 4F13 Lecture notes (course site)



(A) Seed = 0 (B) Seed = 3 (C) Seed = 10

FIGURE 4. BMM Progress

by relabelling. Also, the vast majority of the 20 clusters are pretty much ignored (i.e. the model does not "believe" there are 20 distinguishable clusters). It cannot therefore be said that the algorithm is finding the same global optimum each time, and therefore is not truly exploring the "true" posterior. However, the test set perplexities for the three seeds are 2111, 2124, and 2116, which (averaged) is a $\sim 21\%$ improvement over the model in Tasks B and C. Therefore the model must work well to a certain extent.

5. Task E: Latent Dirichlet Allocation

In Latent Dirichlet Allocation (LDA), each word in a document can be drawn from a different topic, and therefore each document now has its own distribution over topics $\boldsymbol{\theta}_d$. Figure 5 shows the model, and the difference between LDA and BMM can be seen as there is now a $\boldsymbol{\theta}_d$ for each document, rather than only one $\boldsymbol{\theta}$ for the whole model. This increases the number of parameters significantly, making the need for Gibbs Sampling greater.

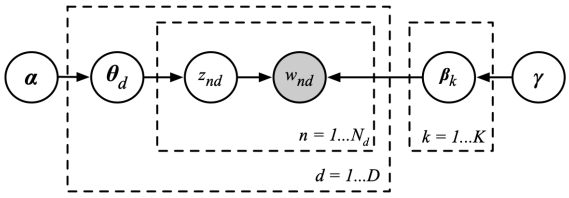


FIGURE 5. Latent Dirichlet Allocation Model[2]

The provided python function `LDA` was modified in a similar way to in Task D to return the topic posteriors at each Gibbs iteration, and also to return the word entropies for each topic at each iteration.

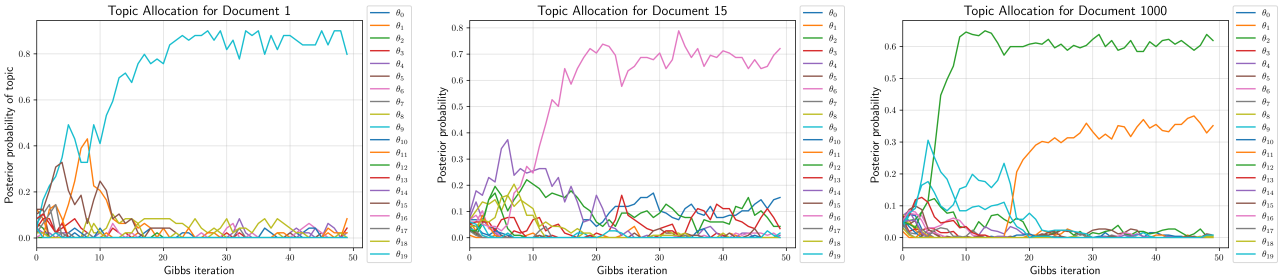
The model was run with parameters $\alpha = 0.1, \gamma = 0.1$ as before, giving a per-word perplexity of 1644, by far the lowest of all the models so far. This is as desired - we are increasing the complexity of the model but still decreasing the log likelihood of the test set - therefore the model is not overfitting to the training set.

Figure 6 shows the topic posteriors as a function of the Gibbs iteration for 3 different documents. The Gibbs sampler exhibits the *rich get richer* property - the more words in a document already assigned to a topic, the more a word is likely to be assigned to that topic. This explains why each document has one topic than dominates. It is possible that more than 50 iterations would be useful, as the topic allocations do not seem to have reached a stationary distribution. It was unfortunately not possible to run for more than 50 iterations to test this as the function was taking over 30 minutes to run.

The oscillations that persist may also be due to high similarity between multiple topics, which could be due to having too many topics, meaning a word has a similar probability of being drawn into either topic.

The word entropy can be calculated using equation 8. This is the expected uncertainty associated with each observation. Figure 7 shows the word entropies for each topic against

[2]See Footnote 1



(A) Document 1 (B) Document 15 (C) Document 354

FIGURE 6. LDA Topic Progress

the Gibbs iteration. Initially, all the topics have the same entropy. After some iterations the entropy decreases, as the topics are reaching a distinct, non-uniform distribution - the topics are becoming more specific as training occurs, as expected. This means fewer words are required to "describe" the topic, or the topic has a smaller vocabulary - therefore the entropy decreases. The entropies reach fairly stable values after 50 iterations, implying that not many, if any, more iterations are required to train the model any better.

$$H_k = -\sum_{m=1}^{M} \beta_{km} \log(\beta_{km}) \tag{8}$$

The natural logarithm is used instead of the standard base 2 logarithm as that allows for easier comparison with the perplexity. Using the natural logarithm results in units of nats, not bits. The log-perplexity is a comparable unit to the entropy, as they will be the the same if measuring an infinitely long document on a single topic. The log-perplexity of the test set is $\log 1644 = 7.40$, which is higher than all the topic entropies, as expected as the test set is a blend of all the topics, and therefore less specific than any single topic.

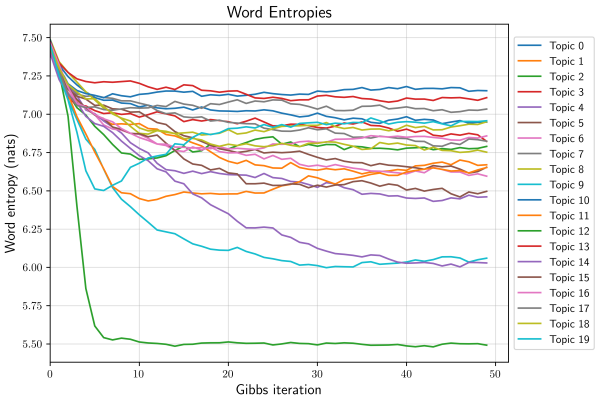


FIGURE 7. Entropy Evolution