# 4F13 Probabilistic Machine Learning
# Coursework #3: Latent Dirichlet Allocation

Candidate Number: 5504C
Word count: 995

December 1, 2023

## 1 Exercise a)

Each word $w_{nd}$ in the training dataset is independently sampled from a uniform categorical distribution that is defined by the parameter vector $\boldsymbol{\beta}$, denoted as $\text{Cat}(\boldsymbol{\beta})$. The likelihood function and maximum log likelihood estimator is then derived as:

$$\hat{\boldsymbol{\beta}}_{\boldsymbol{ML}} = \arg\max_{\boldsymbol{\beta}} \left[\log p(\mathbf{w}|\boldsymbol{\beta})\right] = \log[\prod_{d=1}^{D}\prod_{n=1}^{N_d} P(w_{nd}|\boldsymbol{\beta})] = \sum_{m=1}^{M} c_m \log \beta_m$$

$$\text{subject to} \quad \sum_{m=1}^{M} \beta_m = 1$$

And by adding Lagrangian multiplier, we can get the solution to the optimization problem with the cost:

$$\mathcal{L} = \sum_{m=1}^{M} c_m \log \beta_m + \lambda \left(1 - \sum_{m=1}^{M} \beta_m\right) \quad \text{where} \quad \beta_m = \frac{c_m}{N}$$

$c_m$ is the count of word $m$ and $N$ represents the total count of all words. Hence, the maximum likelihood solution is the multinomial distribution where all words appears in empirical frequency, i.e. individual word count over total word count. The 20 most probable words is shown in Figure.1.

According to Figure.1, the configuration with the highest probability would be one where all words are ['bush'], which is the most probable word in the training set. And it is calculated to be $\log(0.1410) = -4.26$. On the other hand, the lowest probability scenario for the test set would occur if it includes words that never appeared in the training set, leading to a log probability of $\log(0) = -\infty$.

The situations underscore a significant shortcoming in the model's ability to deal with unseen data. It ascribes a zero probability to any word not observed in the training set. Such an approach is evidently inadequate; it is unreasonable for a viable test set to be assigned a probability of zero. Consequently, this indicates that relying solely on the maximum likelihood estimation is not adequate for effective modeling.

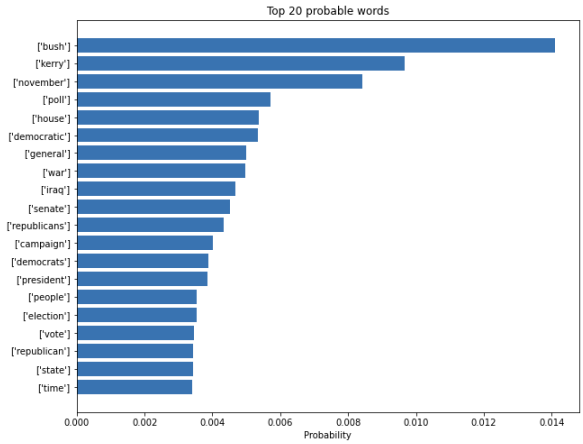



Figure 1: Top 20 probable words using ML estimator

## 2 Exercise b)

By incorporating prior knowledge about the distribution of parameters , we can then perform the Bayesian inference, with the likelihood being the multinomial distribution. The prior we apply in the Bayesian model is Dirichlet distribution, where $p(\boldsymbol{\beta}) \sim \text{Dir}(\alpha)$. And then the Dirichlet distribution is then found to be conjugate prior to the categorical distribution as the prior and posterior has the same closed form:

$$\begin{aligned} p(\boldsymbol{\beta} \mid \mathbf{w}) &\propto p(\boldsymbol{\beta}) \cdot p(\mathbf{w} \mid \boldsymbol{\beta}) \\ &= \text{Dir}(\alpha) \cdot \prod_{p=1}^{M} \beta_p^{c_m} \\ &= \left( \frac{1}{B(\alpha)} \prod_{m=1}^{M} \beta_m^{\alpha_m - 1} \right) \cdot \prod_{m=1}^{M} \beta_m^{c_m} \\ &= \frac{1}{B(\alpha)} \prod_{m=1}^{M} \beta_m^{(\alpha_m + c_m) - 1} \\ &= \text{Dir}(\alpha + c) \end{aligned}$$

And the predictive distribution of unseen word under the Bayesian model is:

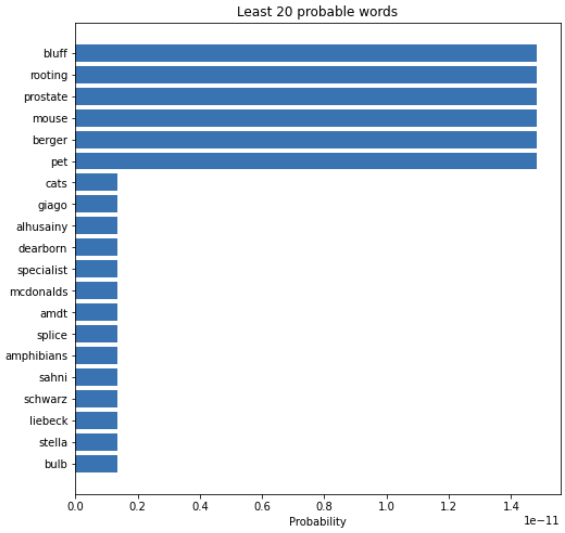
$$\begin{aligned} p\left(w^* = i \mid \mathbf{w}\right) &= \int p\left(w^* = i, \boldsymbol{\beta} \mid \mathbf{w}\right) d\boldsymbol{\beta} \\ &= \int p\left(w^* = i \mid \boldsymbol{\beta}, \mathbf{w}\right) \cdot p(\boldsymbol{\beta} \mid \mathbf{w}) d\boldsymbol{\beta} \\ &= \int \beta_i p\left(\beta_i \mid \mathbf{w}\right) d\beta_i \\ &= \frac{\alpha_i + c_i}{\sum_{m=1}^{M} \alpha_m + c_m} \end{aligned}$$

$\mathbf{w}$ is the training dataset. Whereas the predictive distribution of maximum likelihood model is:

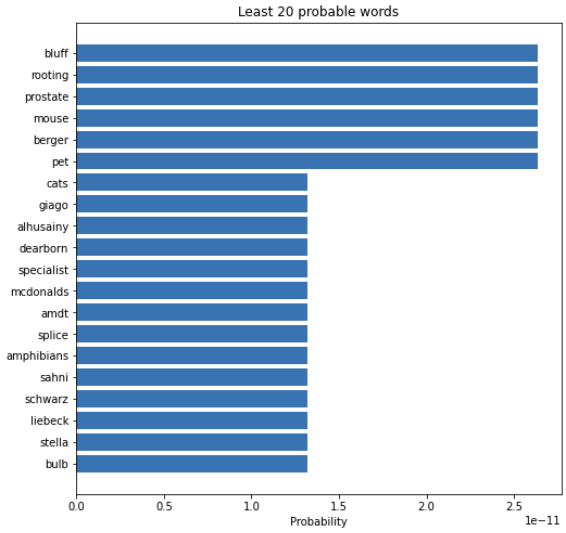
$$p\left(w^* = i \mid \hat{\boldsymbol{\beta}}_{\boldsymbol{ML}}\right) = \frac{c_i}{\sum_{m=1}^{M} c_m}$$

For rare words, the maximum likelihood approach often results in significantly low or even zero probability, posing challenges when encountering unseen words during testing. Bayesian inference addresses this issue

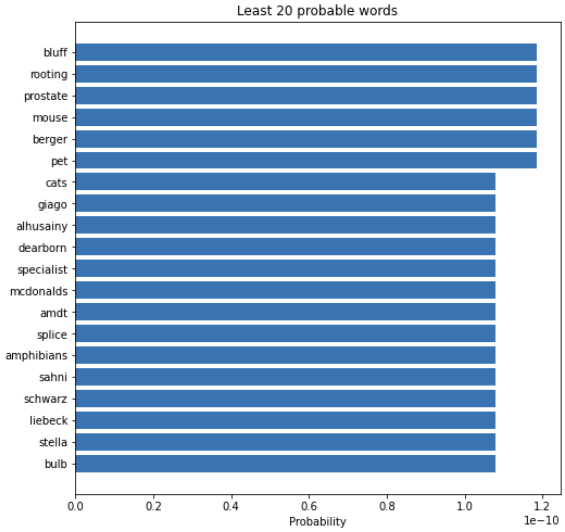
adeptly. At low $\alpha$ values, it assigns small, non-zero probabilities to rare or unseen words, enhancing the model's capacity to handle such occurrences. As the value of $\alpha$ is increased shown in Figure.2, the probabilities for rare words are further elevated, ensuring that these words are not neglected, thus draws the distribution closer to uniform. This Bayesian approach with a Dirichlet prior, therefore, offers a more nuanced and effective method for word probability prediction, adeptly balancing the treatment of both common and rare words, and enhancing the model's overall robustness and adaptability compared to the more straightforward maximum likelihood method.



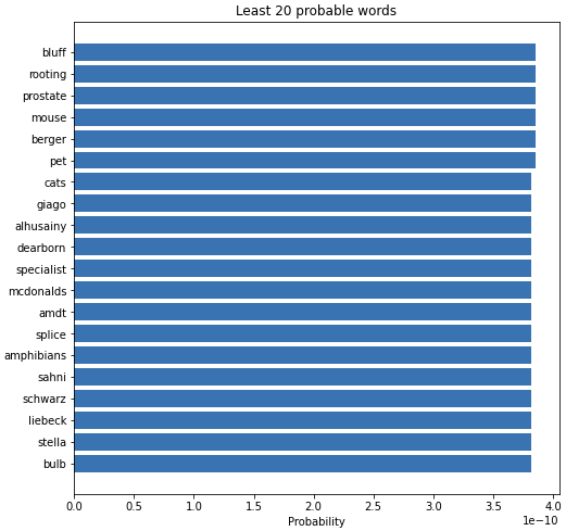

(a) Least probable words with $\alpha = 0.1$




(b) Least probable words with $\alpha = 1$




(c) Least probable words with $\alpha = 10$




(d) Least probable words with $\alpha = 100$

Figure 2: Least probable words with increasing $\alpha$

# 3 Exercise c)

For the Bayesian model, the log probability for the test document with ID 2001 is computed through the predictive distribution in the previous section as:

$$\log p(d \mid \mathrm{w}) = \log p\left(\{w_{nd}^*\}_{n=1}^{N_d} \mid \mathbf{w}\right) = \log \prod_{n=1}^{N_d} P\left(w_{nd}^* \mid \mathbf{w}\right) = \sum_{n=1}^{N_d} \log \frac{\alpha + c_n}{\sum_{m=1}^{M} (\alpha + c_m)}$$

Note that the counts mentioned in the expression are all from training set $A$. When computing the log probability of a test document, the categorical distribution function is preferred because it accounts for the sequence of words, unlike the multinomial distribution which only considers word counts without regard to order. Per-word perplexity is a measure of how well a probability model predicts a sample and can be computed by:

$$\text{per-word perplexity } = \exp\left(-\frac{l}{n}\right) = \exp \frac{-\log p(d \mid \boldsymbol{w})}{N_d}$$

With $\alpha$ set to 1, the results for log probability and per-word perplexity for document 2001 and all documents in test set are shown in Table.1. The perplexity of a document varies based on its unique composition of word frequencies, with documents featuring more common words typically showing higher log-probabilities and thus lower perplexities. Consequently, the higher perplexity observed for document 2001 compared to the average in the test set $B$ suggests it contains a greater proportion of rare words relative to the common words in the training set.

| | document with ID 2001 | all documents in $B$ |
|---|---|---|
| log probability | -3688.62 | -1545978.59 |
| per-word perplexity | 4373.11 | 2683.98 |

Table 1: Results for single document and all documents

The expected perplexity in a probability distribution represents the number of equally likely outcomes. For a uniform multinomial distribution with N categories, the expected perplexity is also N, as each category has an equal probability of being selected. In the context of language models, if our vocabulary has M words and we treat each word as a category in a multinomial distribution, the perplexity is basically M.

# 4 Exercise d)

The Bayesian documents mixture model clusters documents by assuming they arise from a mixture of topics (latent variable), each with its own word distribution. It uses Dirichlet-distributed priors for both word distributions per topic $\beta_k$ and topic distributions per document $\theta$, with document-topic assignments $z$ drawn from a categorical distribution. Gibbs sampling is applied to infer the latent topic structure from the observed words in documents. The structure is shown in Figure.3.

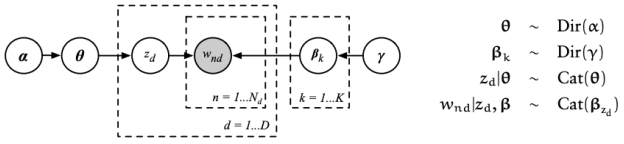



Figure 3: Bayesian documents mixture model in 4F13 handout

With $\alpha$ set to 0.1, Gibbs sampling is conducted on the training set $\mathcal{A}$ to generate samples for $\theta$. The mixing

proportions is then computed by:$\theta_k = \frac{\alpha+c_k}{\sum_{i=1}^{K}(\alpha+c_i)}$. The evolution of mixing proportions of the mixture component against iterations of Gibbs sampler is drawn in Figure.4.

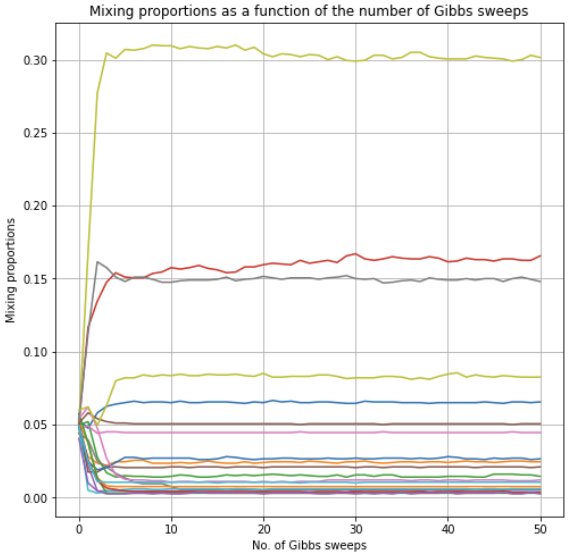
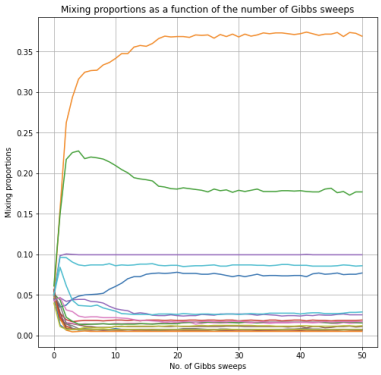
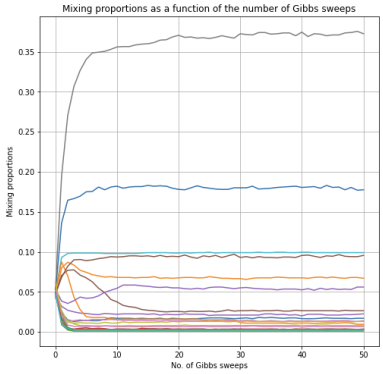



Figure 4: Mixing proportions as a function of the number of Gibbs sweeps

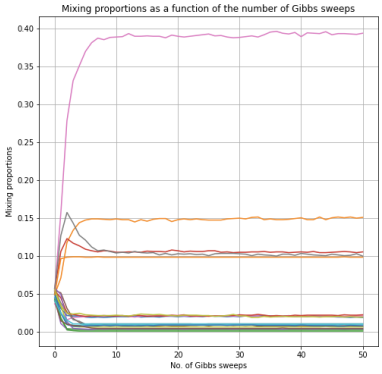
The diagrams indicate that the mixing proportions stabilize after several iterations, which may suggest convergence. However, this could represent a local rather than global optimum. Different initializations, achieved by setting various seeds in Python, such as *np.random.seed*(), can lead to divergent stationary distributions. Each of the three Gibbs processes shown in 5 reaches stability, but they each converge to different distributions, a situation that remains unchanged even after random relabeling of categories. This divergence implies that rather than converging to the definitive posterior, the process tends to settle at what can be considered an acceptably adequate local optimum.




(a) Mixing proportions as a function of the number of Gibbs sweeps with seed 41




(b) Mixing proportions as a function of the number of Gibbs sweeps with seed 150




(c) Mixing proportions as a function of the number of Gibbs sweeps with seed 278

Figure 5: Mixing proportions as a function of the number of Gibbs sweeps with different initializations

# 5 Exercise e)

The LDA (Latent Dirichlet Allocation) model extends the BMM model by allowing each document to be a blend of topics. Every single word $w_{id}$ from each document $d$ can be drawn from a unique topic $z_{id}$, where $z_{id} \sim \text{Dir}(\theta_d)$. The structure is shown in Figure.6.

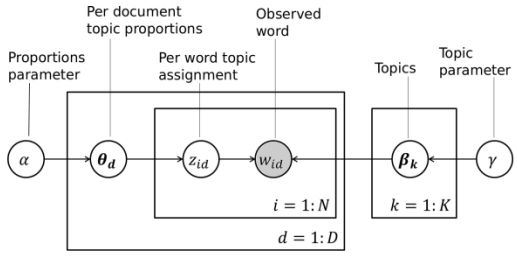



Figure 6: Latent Dirichlet Allocation model from 4F13 handout

With K set to 20 topics, we then perform Gibbs sampling for up to 50 iterations to estimate the posterior distribution of topics within the documents of set A. The posterior samples are drawn similarly as last part. The resulting topic posteriors against iterations is shown in Figure.7. The individual documents behave differently. Certain topics become more dominant as they accumulate more words; this is a characteristic feature of Gibbs sampling in topic modeling, where topics that already have a larger share of words are more likely to gain new words. Evaluating the topic posteriors over successive Gibbs iterations reveals that after 50 cycles, the topic distributions appear to stabilize, suggesting that this number of iterations is sufficient for the model to converge.

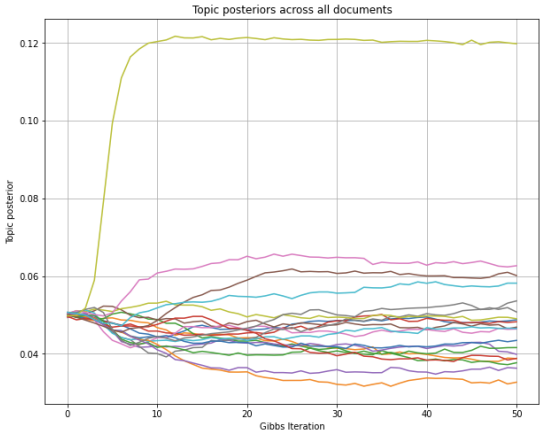



Figure 7: Topic posteriors across all documents

The per-word perplexity of different models is then summarized in Table.2. The perplexity values calculated for test set $B$ post-iteration reflect the model's ability to accurately predict new data, with a noted decrease in perplexity as the model advances from basic Bayesian inference to the more complex LDA approach, indicating enhanced performance.

| | Maximum Likelihood | Bayesian Inference | BMM | LDA |
|---|---|---|---|---|
| per-word perplexity | $\infty$ | 2683.98 | 2100.73 | 1701.10 |

Table 2: per-word perplexity for different models

Additionally, the word entropy across topics, indicative of the model's certainty in topic assignment, diminishes over iterations, indicating increasingly distinct topic definitions. This is shown in the Figure.8. The downward trend in word entropy stabilizes after 50 iterations, validating the notion that this quantity of sweeps is appropriate for dependably establishing the model's parameters. Also, if a topic has low entropy, it probably has a few words that occur very frequently, which are highly representative of that particular topic.

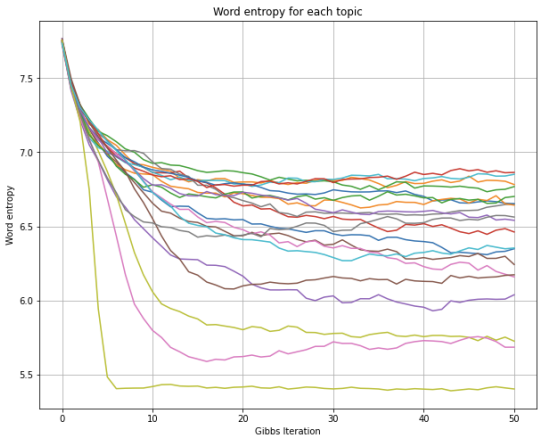



Figure 8: Word entropy for each topic

# 6 Appendix

## 6.1 (d)

```
for k in range(K):
    ll = np.dot(np.log(swk[w-1, k] + gamma) - np.log(sk_words[k] + gamma * W), c.T)

    lb[k] = np.log(sk_docs[k] + alpha) + ll
b = np.exp(lb - np.max(lb))  # exponentiation of log probability plus constant
```

## 6.2 (c)

```
for k in indices:
    k = int(k)
    for i in range(int(a[k])):
        z[w, k] -= 1  # remove word from count matrix for doc d
        Skd[k] -= 1
        b = (alpha + Skd) * (gamma + swk[w, :]) \
            / (W * gamma + sk)
        kk = sampleDiscrete(b, np.random.rand())
        z[w, kk] += 1  # add word with new topic to count matrix for doc d
        Skd[kk] += 1
```