Latent Dirichlet Allocation

Probabilistic Machine Learning

Abhishek Shenoy

Total words = 998
[Excluding tables, figures, code, equations, abstract and references]

Abstract

This report explores topic modelling with the aim of comparing different models including maximum likelihood (ML), Bayesian word probabilities (BWP), Bayesian mixture of multinomials model (BMM) and latent Dirichlet allocation (LDA).

1 Part A

Training data A is used to find the maximum likelihood multinomial over words. If $w_{nd} \in \{1, ..., M\}$ denotes the nth word in document d with M unique words, the aim is to learn the parameters β where $w_{nd} \sim \text{Cat}(\beta)$. The likelihood of the parameters in the multinomial model is given by Equation 1.

$$\begin{aligned} \mathcal{L}(\beta) &= \text{Mult}(c_1, ..., c_M|\beta) \\ c_m &= \sum_{n,d} \mathbb{I}(w_{nd} = m) \end{aligned} \tag{1}$$

The maximum likelihood estimate (MLE) can then be calculated using Equation 2.

$$\begin{aligned} \mathcal{L}(\beta) &\propto \prod_{m=1}^{M} \beta_m^{c_m} \\ \log \mathcal{L}(\beta) &= \sum_{m=1}^{M} c_m \log \beta_m + \text{const.} \\ \text{Enforcing } \sum_m \beta_m &= 1 \text{ by using Lagrange multipliers} \\ \tilde{\mathcal{L}}(\beta, \lambda) = \log \mathcal{L}(\beta, \lambda) &= \sum_{m=1}^{M} c_m \log \beta_m + \lambda(1 - \sum_m \beta_m) \end{aligned} \tag{2}$$

$$\begin{aligned} \frac{d\tilde{\mathcal{L}}}{d\beta_m} &= \frac{c_m}{\beta_m} - \lambda = 0 \rightarrow \beta_m \lambda = c_m \\ \frac{d\tilde{\mathcal{L}}}{d\lambda} &= 1 - \sum_m \beta_m = 0 \rightarrow \sum_m \beta_m = 1 \\ \therefore \beta_m &= \frac{c_m}{\sum_m c_m} \end{aligned} \tag{3}$$

The 20 largest fractional word counts are given in Figure 1. Listing 1 provides shows the implementation of Equation 3.

The simple multinomial model constructed using the training data will not perform well on characterising other data. Any test document with a word w_{nd} not in the training data (ie. any new word) will have a probability of $p(w_{nd}|\beta) = 0$ (overall log probability will be $\log(0) = -\infty$). Hence, this model does not generalize well to other datasets.

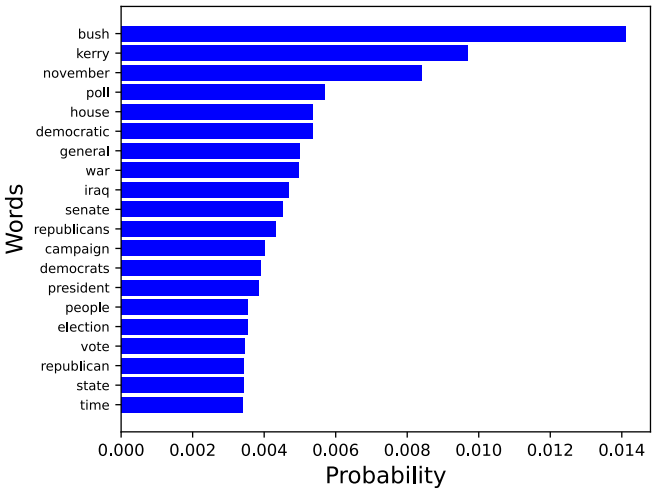


Figure 1. Word probabilities for 20 most common words in training dataset

The lowest log probability of a test set is $-\infty$. The greatest probability will be given by a document with the single word, $\hat{w}$ that is the most frequent in the training data, giving the log probability $\log p(\hat{w}|\beta) = \mathbf{\log \max_i \beta_i}$.

In this case, the most common word in the training data is *bush* hence giving a maximum log probability of $\mathbf{-4.26}$ in this case. Note that a single document is unrepresentative of typical documents.

```
# Number of unique words
M = max(A[:, 1])

# Number of training documents
D = max(A[:, 0])

# Total number of words
total_count = sum(A[:, 2])

beta = np.zeros(M)
# Count for each word index
for i in range(M):
    count_i = sum(A[A[:,1] == i, 2])
    beta[i] = count_i / total_count

# Sort to get 20 highest probs
n = 20
sorted_index = beta.argsort()[-n:]
word_names = [V[i-1][0] for i in sorted_index]
sorted_beta = [beta[i] for i in sorted_index]
```

Listing 1. Computation of word probabilities

2 Part B

Here we use a Bayesian approach for which a symmetric Dirichlet prior (conjugate prior) is applied over β to determine the Bayesian word probabilities (BWP). The posterior for β given observed data $\mathcal{D}$ is calculated as follows.

$$\begin{aligned} p(\beta|\mathcal{D}) &\propto p(\beta)p(\mathcal{D}|\beta) \\ &\propto \text{Dir}(\beta|\alpha)\text{Mult}(c_1, ..., c_M|\beta) \\ &\propto \prod_{m=1}^{M} \beta_m^{\alpha-1} \prod_{m=1}^{M} \beta_m^{c_m} \\ &\propto \text{Dir}(\beta|c_1 + \alpha, ..., c_M + \alpha) \end{aligned}$$

The resulting posterior can be used to find the predictive distribution for a word.

$$\begin{aligned} p(w = k|\mathcal{D}) &= \int p(w|\beta)p(\beta|\mathcal{D})d\beta \\ &= \int \int \beta_k p(\beta|\mathcal{D})d\beta_{i=1:M,i\neq k}d\beta_k \\ &= \int \Big[p(\beta|\mathcal{D})d\beta_{i=1:M,i\neq k}\Big]d\beta_k \\ &= \mathbb{E}_{q(\beta_k)}[\beta_k] \end{aligned}$$

Note that $q(\beta_k) \sim \text{Beta}(\alpha+c_k, \sum_{m,m\neq k}(\alpha+c_m))$. Hence the standard result is obtained.

$$p(w = k|\mathcal{D}) = \frac{\alpha + c_k}{\sum_m (c_m + \alpha)} \tag{4}$$

From Equation 4, the use of a symmetric Dirichlet distribution can be interpreted as the introduction of pseudo-counts for each word α. Hence $\alpha = 0$ is equivalent of a MLE approach (Eq 4 becomes identical to the MLE Eq 3) ie. the prior gives no information. The MLE prediction will also be recovered when $c_m >> \alpha$.

Increasing the concentration parameter α means that the prior is more confident that each element of β is $1/M$ as the distribution narrows. Large α indicates a large number of pseudo-counts for each word dominating any observations meaning that it is more certain of β_m being equiprobable.

The predictive word probabilities using BWP is different for common and rare words. For a common word, k ($c_k >> \alpha$), there will not be a significant change in the predictive distribution as $\frac{\alpha+c_k}{\sum_m (c_m+\alpha)} \approx \frac{c_k}{\sum_m (c_m)}$, assuming a small α. For a rare word ($c_j << \alpha$), the pseudo-counts make up a significant proportion of all observed counts for that word. Increasing α means that the pseudo-counts will start to dominate the observed counts forcing the probability toward $1/M$.

The predictive probabilities for rare words are therefore higher than the ML model which is beneficial as unseen words in the test dataset will not result in zero probability. The α parameter must be sufficiently large to give an adequate positive probability to rare words whilst being sufficiently small to avoid overwhelming the actual observed counts.

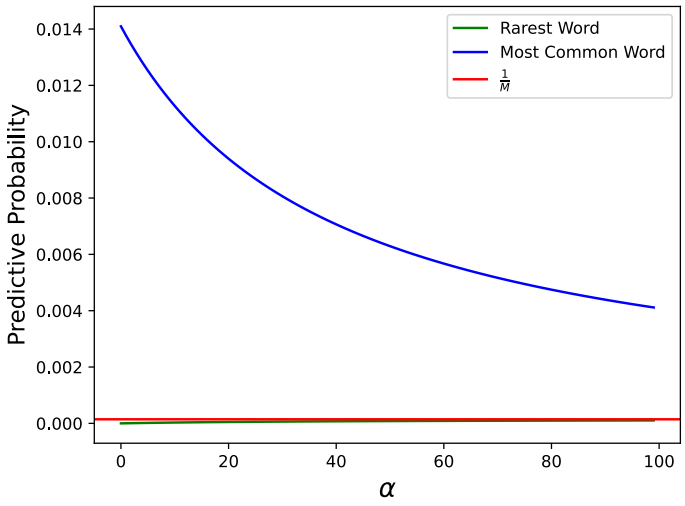


Figure 2. Predictive probability for common and rare word with varying concentration α using BWP

Figure 2 shows the impact of the Bayesian approach on the rarest and most common word predictive probabilities with varying α. If the total number of words is N, the probabilities of words with frequency $< N/M$ increases with α. The probabilities of words with frequency $> N/M$ decreases with α.

```
small, large = [], []
for alpha in range(100):
    large_val = (alpha + largest_count)/(
      total_count + alpha*M)
    large.append(large_val)

    small_val = (alpha + smallest_count)/(
      total_count + alpha*M)
    small.append(small_val)
```

Listing 2. Predictive probabilities with BWP

3 Part C

Equation 5 gives the probability of observing document d with N_d words using BWP.

$$\begin{aligned} p(\{w_{nd}{}_{n=1}^{N_d}\}|\mathcal{D}) &= \prod_{n=1}^{N_d} p(w_{nd}|\mathcal{D}) \\ &= \prod_{n=1}^{N_d} \frac{c_{w_{nd}} + \alpha}{\sum_m (c_m + \alpha)} \end{aligned} \tag{5}$$

The categorical distribution function is used instead of the multinomial distribution since we want the probability of the specific document. Using the multinomial distribution would account for all the documents that have the same frequency counts.

	Log Probability	Num of Words	Per-Word Perplexity
Doc 2001	-3691	440	4399
All Test Docs	-1546958	195816	2697

Table 1. Log probability and per-word perplexity for document 2001 and test documents

Table 1 shows the per-word perplexity for both document 2001 and the test documents computed using Listing 3.

```
word_ids, obs = [], []
for item in B:
    if item[0] == 2001:
        word_ids.append(item[1])
        obs.append(item[2])

training_counts = [counts[word_id] if word_id <
    len(counts) else 0 for word_id in word_ids]

# Log probability for document 2001
log_prob = 0
for i, count in enumerate(training_counts):
    new_prob = (alpha + count)/(total_count + (
      alpha*M))
    log_prob += np.log(new_prob) * obs[i]

# Per-word perplexity for document 2001
num_words = sum(obs)
perplexity = np.exp(-1*log_prob / num_words)

##############################

word_ids, obs = B[:, 1], B[:, 2]
all_counts = [counts[word_id] if word_id < len(
    counts) else 0 for word_id in word_ids]

# Log probability for all test documents
log_prob = 0
for i, count in enumerate(all_counts):
    new_prob = (alpha + count)/(total_count + (
      alpha*M))
    log_prob += np.log(new_prob) * obs[i]

# Per-word perplexity for all test documents
num_words = sum(obs)
perplexity = np.exp(-1*log_prob / num_words)
```

Listing 3. Log probability and per-word perplexity for document 2001 and test documents

$$\begin{aligned} \text{Perplexity} &= \exp \frac{-1}{N_d} \log p(w_{1:N_d}|M) \\ &= p(w_{1:N_d}|M)^{\frac{-1}{N_d}} \\ &= \left[\frac{1}{M^{N_d}}\right]^{\frac{-1}{N_d}} \\ &= M \end{aligned} \tag{6}$$

The perplexities are different for different documents because they contain different words. Documents with words common to the training dataset will have a higher probability per word (lower negative log probability) yielding a smaller perplexity.

With a uniform multinomial, every word will have the same probability of 1/M. Equation 6 shows that this gives a perplexity equal to the number of unique words M. Here M = 6906, which gives a perplexity of 6906 equivalent to drawing from a discrete uniform distribution with 6906 options.

4 Part D

Here we consider BMM for which collapsed Gibbs sampling is used to determine the mixing proportions θ (the posterior probability of a document belonging to a particular class) using $K = 20$ topics. The mixing proportion is computed using the counts $\tilde{c}_k$ of the number of documents assigned to topic k. With a symmetric Dirichlet prior on θ and each β (with parameters α and γ respectively), the posterior over the mixing proportions is as follows.

Let the latent variable z_d denote the topic assignment of document d.

$$\begin{aligned} p(\theta|\mathcal{D}, M, K) &= \int p(\theta, \underline{\boldsymbol{z}}|\mathcal{D}, M, K) d\underline{\boldsymbol{z}} \\ &= \int p(\underline{\boldsymbol{z}}|\mathcal{D}, M, K) p(\theta|\underline{\boldsymbol{z}}, K) d\underline{\boldsymbol{z}} \end{aligned}$$

An approximation can be made using Monte-Carlo with samples drawn $\underline{\boldsymbol{z}}^{(i)} \sim p(\underline{\boldsymbol{z}}|\mathcal{D}, M, K)$.

$$p(\theta|\mathcal{D}, M, K) \approx \frac{1}{N} \sum_{i=1}^{N} p(\theta|\underline{\boldsymbol{z}}^{(i)}, K)$$

Since only one sample of $\underline{\boldsymbol{z}}$ is used, this simplifies further.

$$p(\theta|\mathcal{D}, M, K) \approx p(\theta|\underline{\boldsymbol{z}}^{(1)}, K)$$

The probability of the mixing proportions given the latent variable can be written as a product of a Dirichlet and multinomial distribution.

$$\begin{aligned} p(\theta|\underline{\boldsymbol{z}}^{(1)}, K) &\propto p(\theta|K) p(\underline{\boldsymbol{z}}^{(1)}|\theta) \\ &\propto \text{Dir}(\alpha)\text{Mult}(\tilde{c}_1, ..., \tilde{c}_K|\theta) \end{aligned}$$

Note that $\tilde{c}_k = \sum_{d=1}^{D} \mathbb{I}(z_d = k)$. The product of a Dirichlet and Multinomial distribution gives a Dirichlet distribution, hence the sample counts can be used to calculate the expectation of the mixing proportions over the posterior.

$$\mathbb{E}_{p(\theta|\mathcal{D}, M, K)}[\theta_k] \approx \frac{\tilde{c}_k + \alpha}{\sum_{i=1}^{K} (\tilde{c}_i + \alpha)} \tag{7}$$

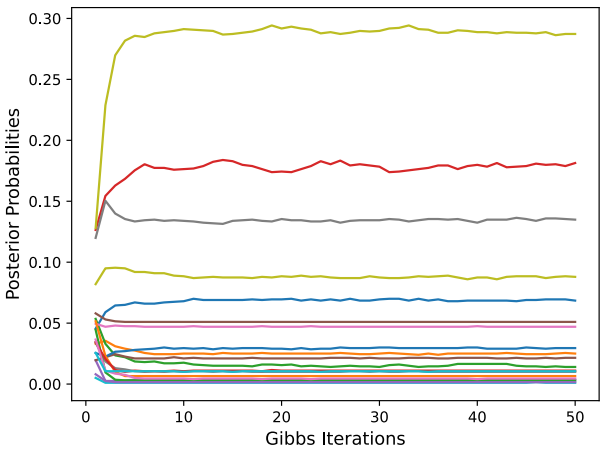
```
docs = sk_docs[:]
num = docs + alpha
den = np.sum(docs + alpha)
theta[:, iter] = (num/den).squeeze()
```

Listing 4. Calculation of mixture components in BMM

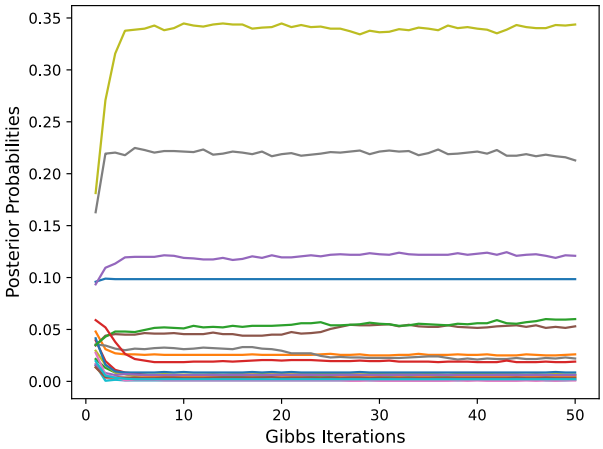
Listing 4 gives the implementation of Equation 7.

Running the Gibbs sampler ($\alpha = \gamma = 0.1$) for different seeds (Fig 3) shows that the final mixing posterior is dependent on the seed initialisation. In Figure 3, the topic indices are irrelevant as any set of mixture components can be permuted meaning that the topic posterior is multi-modal. Over 50 iterations, the Gibbs sampler cannot escape a particular mode and therefore does not explore other modes of the posterior. The Gibbs sampler does not converge as it does not sample from the entire posterior mainly due to the high dimensionality of the distribution.

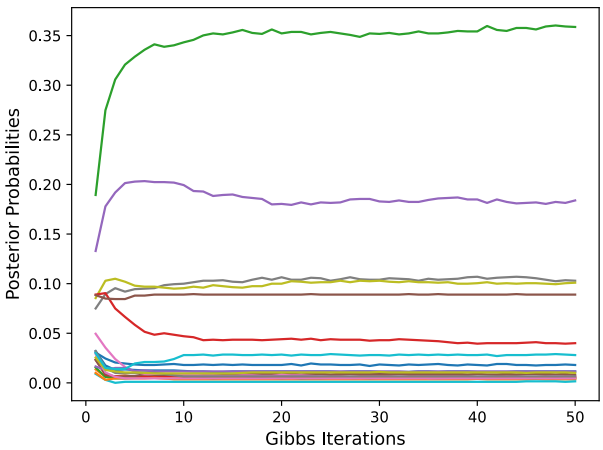
Convergence is reached when the distributions are similar. We can test for convergence by sampling for multiple seeds (as well as applying burn-in and thinning) and measuring the similarity of the posteriors using a metric such as KL divergence. The sampler did not converge over the 50 iterations. The per-word perplexity is similar for all seeds using BMM over the test dataset, at a value around $\approx \mathbf{2100}$, suggesting all local modes are equally suitable.



(a) Seed=1



(b) Seed=2



(c) Seed=3

Figure 3. Posterior mixing probabilities for BMM for different seeds

5 Part E

We again use collapsed Gibbs sampling with LDA with $K = 20$. LDA differs from BMM because it allows words in the same document to be assigned to different topics.

Figure 4 shows the development of the topic posteriors as well as the word entropy (measured in bits/word). Listing 5 shows the implementation of the calculation of the topic posterior.

From the fluctuation of specific topics, it is clear that the Gibbs sampler has not converged indicating that 50 iterations are not adequate. Furthermore the differences in the final topic posteriors for the different seeds suggest that the stationary distribution is not fully explored.

```
theta = np.zeros((K, gibbs_iterations))
        docs = skd[:, 2]
        num = docs + alpha
        den = np.sum(docs + alpha)
        theta[:, iter] = (num/den).squeeze()
```

Listing 5. Calculation of mixture components in LDA

```
beta = np.zeros((W,K))
word_entropy = np.zeros((K, gibbs_iterations))
        for k in range(K):
            arr = swk[:, k] + gamma
            beta[:, k] = (arr/sum(arr)).squeeze()

        for k in range(K):
            entropy = 0
            for i in range(W):
                entropy -= beta[i, k] * np.log2(
    beta[i,k])
            word_entropy[k, iter] = entropy
```

Listing 6. Calculation of word entropies in LDA

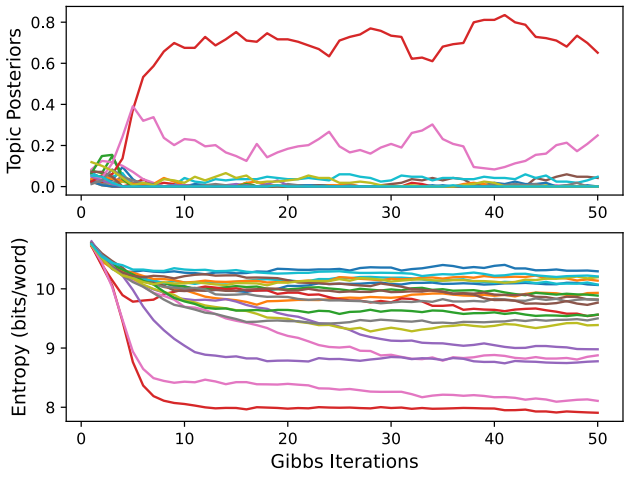
Listing 6 shows how entropy is calculated in each iteration. Figure 4 shows that the upper bound on the entropy $\log_2 6906 = 12.8$ bits/word is given by the maximum number of words $M = 6906$.

As each topic begins to specialize, entropy decreases. Specific words will have a higher predictive probability for a specific topic reducing uncertainty therefore giving a lower entropy. Certain topics achieve much lower entropies than others meaning that the predictive probabilities of certain words must be much greater for these topics (more specialised). Generally, the word entropies behave similarly across the seeds. The lower word-entropy topics have a highly concentrated probability on a single word whereas the higher word-entropy topics have a relatively uniform distribution.

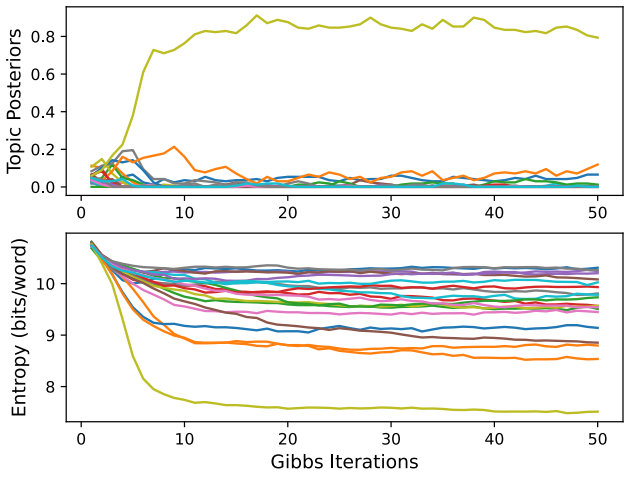
Model	Perplexity
ML	∞
BWP	2697
BMM	≈ 2100
LDA	≈ 1642

Table 2. Perplexity over test dataset for different models

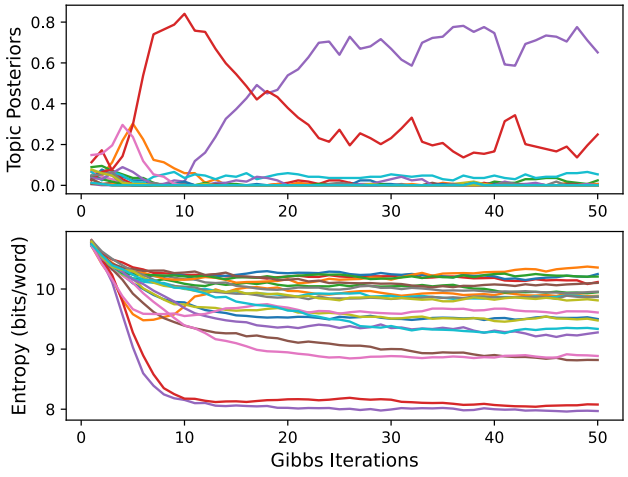
LDA has a much lower perplexity (averaged over multiple seeds) (Table 2), indicating a higher log probability of the test set. Hence LDA is better able to represent the unseen test data and generalises better than the other models. This is understandable as LDA allow words in the same document to be drawn from different topics.



(a) Seed=1



(b) Seed=2



(c) Seed=3

Figure 4. Topic posterior and word entropy for LDA model for different seeds