UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning and Machine Intelligence 2021-2022

Module Coursework Feedback

Module Title: Probabilistic Machine Learning

Module Code: 4F13

Candidate Number: J902C

Coursework Number: 3

I confirm that this piece of work is my own unaided effort and adheres to the Department of Engineering's guidelines on plagiarism. √

Date Marked: Click here to enter a date. Marker's Name(s): Click here to enter text.

Marker's Comments:

This piece of work has been completed to the following standard *(Please circle as appropriate)*:

	Distinction			**Pass**			**Fail (C+ - marginal fail)**		
Overall assessment (circle grade)	**Outstanding**	**A+**	**A**	**A-**	**B+**	**B**	**C+**	**C**	**Unsatisfactory**
Guideline mark (%)	90-100	80-89	75-79	70-74	65-69	60-64	**55-59**	**50-54**	**0-49**
Penalties	**10% of mark for each day, or part day, late (Sunday excluded).**								

The assignment grades are given **for information only**; results are provisional and are subject to confirmation at the Final Examiners Meeting and by the Department of Engineering Degree Committee.

Report coursework 3

4F13 - Probabilistic Machine Learning

November 30, 2021

Candidate no.: J902C

Word count: 1000

UNIVERSITY OF CAMBRIDGE
Department of Engineering

Trumpington Street
Cambridge CB2 1PZ
Tlf.: +44 01223 332600
http://www.eng.cam.ac.uk/

Table of contents

1 Question a) 1

2 Question b) 3

3 Question c) 5

4 Question d) 8

5 Question e) 10

1. Question a)

A document $d \in \{1, \ldots, D\}$ can be considered a collection of N_d words $w_{nd} \in \{1, \ldots, M\}$ drawn from a discrete categorical distribution with parameters $\boldsymbol{\beta}$, as modelled in grahic 1.1. The **parameters** $\boldsymbol{\beta}$ can be **fit by maximising the likelihood**:

$$\begin{aligned} \boldsymbol{\beta} &= argmax_{\boldsymbol{\beta}} \prod_{d=1}^{D} \prod_{n}^{N_d} Cat(w_{nd}|\boldsymbol{\beta}) = \\ &= argmax_{\boldsymbol{\beta}} Mult(\boldsymbol{c_1}, \ldots, \boldsymbol{c_M}|\boldsymbol{\beta}, N) \implies \\ \implies \hat{\beta}_m &= \frac{c_m}{N} = \frac{c_m}{\sum_{l=1}^{M} c_l}, \; m \in \{1, \ldots, M\}, \end{aligned} \tag{1.1}$$

where $c_m = \sum_{d=1}^{D} \sum_{n}^{N} \mathbb{I}(w_{nd} = m)$ is the total count of vocabulary word m.

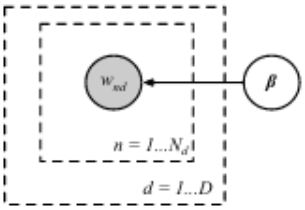

Figure 1.1: Simple graphical representation of a multinomial model

This can be done in python with the following lines:

```
totalWordCount=np.sum(A[:,2])
betas=[np.sum(A[A[:,1]==i][:,2])/totalWordCount for i in range(len(V))]
```

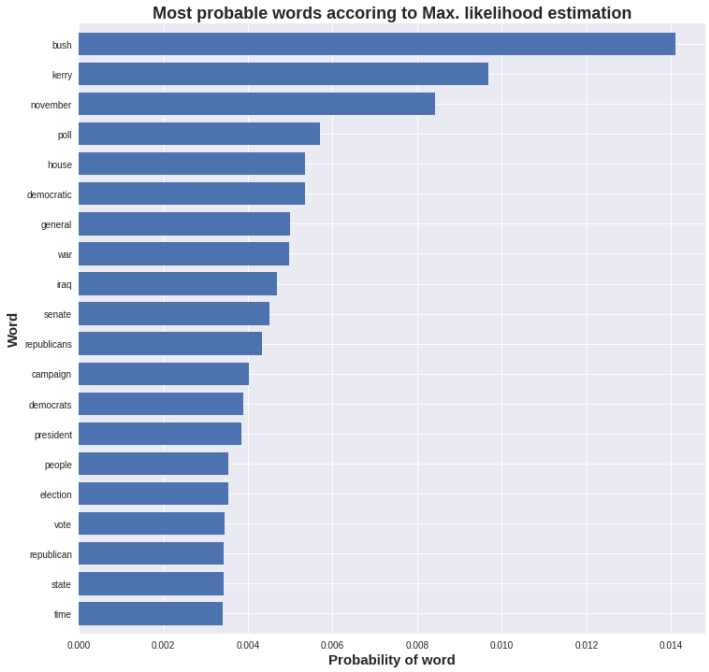

Figure 1.2: 20 Largest probability words, i.e., largest β_m values

The most probable words can be represented in a bar plot (figure 1.2). Additionally, the least probable words inferred using equation 1.1 are shown in graphic 1.3. There are **some words** with **probability zero** since **they're in the vocabulary but not in the training set**. The lowest 20 values of an array (betas) can be computed in python using:

```
sortedIndices = np.argsort(betas)
sortedNames = vocab[sortedIndices]
lowNames = sortedNames[:N]
lowBetas = betas[sortedIndices][:N]
```

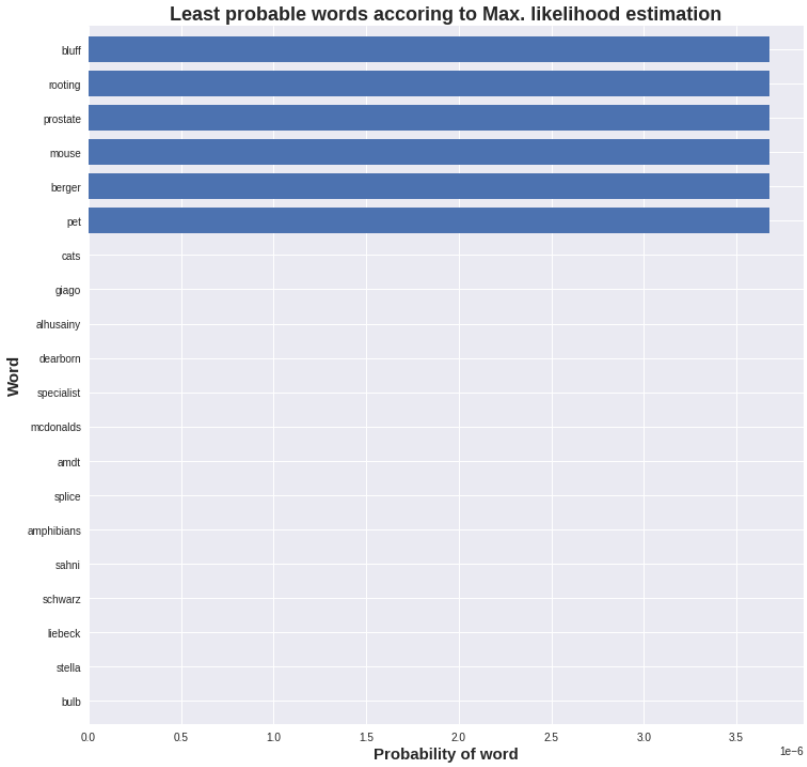

Figure 1.3: 20 Smallest probability words, i.e., smallest β_m values

In this model every document follows the same distribution, regardless of its meaning or topic. To **maximise** the test set log probability the test document should contain **only the most probable word** of the training set ('bush', so $logP('bush'|\boldsymbol{\beta}) = 0.014$). On the other hand, a test set on which every document has a word that is **not present** in the training set will have log probability equal $-\infty$:

$$logP(\mathbf{w}_{test}|\boldsymbol{\beta}) = \sum_{m=1}^{M} c_m^{test} logP(\mathbf{w}_{nd} = m|\boldsymbol{\beta}) = \sum_{m=1}^{M} c_m^{test} log(\beta_m), \tag{1.2}$$

where $\mathbf{w}_{test}$ are all the words in the test set.

2. Question b)

A better model uses a **prior distribution** on the parameter $\boldsymbol{\beta}$. In this case, a **Dirichlet distribution** with parameter α is used: $p(\boldsymbol{\beta}) = Dir(\boldsymbol{\beta}, \alpha) = B(\alpha)^{-1} \prod_{m=1}^{M} \beta_m^{\alpha-1}$.

The posterior distribution can be computed by multiplying the likelihood to the given prior:

$$p(\boldsymbol{\beta}|w) \propto p(w|\boldsymbol{\beta})p(\boldsymbol{\beta}) \propto \prod_{m=1}^{M} \beta_m^{c_m} \prod_{m=1}^{M} \beta_m^{c_m+\alpha-1} \implies p(\boldsymbol{\beta}|w) = Dir(\boldsymbol{\beta}|c_1+\alpha, \ldots, c_M+\alpha). \tag{2.1}$$

Since the **posterior is another Dirichlet distribution** (i.e. Dirichlet distribution is conjugate prior for multinomial distribution), the **predictive mean** can be calculated as:

$$\hat{\beta}_m = \mathbb{E}_{p(\boldsymbol{\beta}|w)}[\beta_m] = \frac{c_m + \alpha}{\sum_{l=1}^{M} c_l + \alpha} \tag{2.2}$$

So incorporating a symmetric Dirichlet prior is the same as adding a **pseudocount** α to each word in the vocabulary, and the maximum likelihood estimate is recovered for $\alpha = 0$. This is easily done in python using the following lines:

```
wordCount = [sum(A[A[:,1]==i][:,2]) for i in range(len(V))]
dirichParams = wordCount+alpha
dirichBetas = dirichParams/np.sum(dirichParams)
```

In figures 2.1 and 2.2 are compared the inferred word probabilities for the 20 most probable words and for the 20 least probable words (respectively), using both Bayesian inference (equations 2.1 2.2) and maximum likelihood estimation (question 1) and with a small $\alpha = 0.1$. In addiction, in figures 2.3 and 2.4 are compared the word probabilities for the 20 most probable words and for the 20 least probable words using both inference methods and a large $\alpha = 100$.

It can be seen (and mathematically derived) is that the new prior leads o **unseen words in training having non-zero probability**. Basically, Adding a pseudo-count **increases the probabilities of rare words and reduces those of common words**. For **small values of** α the **observed counts are dominant over pseudo-counts**, so the BI is similar to ML. For **large** α **values** the word probabilities are **more uniform** across all the vocabulary.

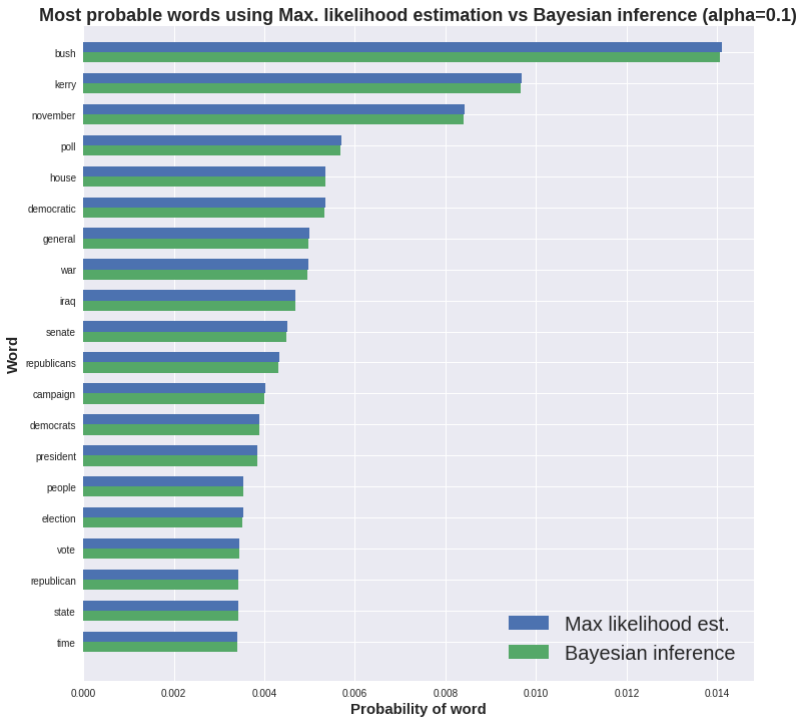

Figure 2.1: 20 largest probability words with $\alpha = 0.1$

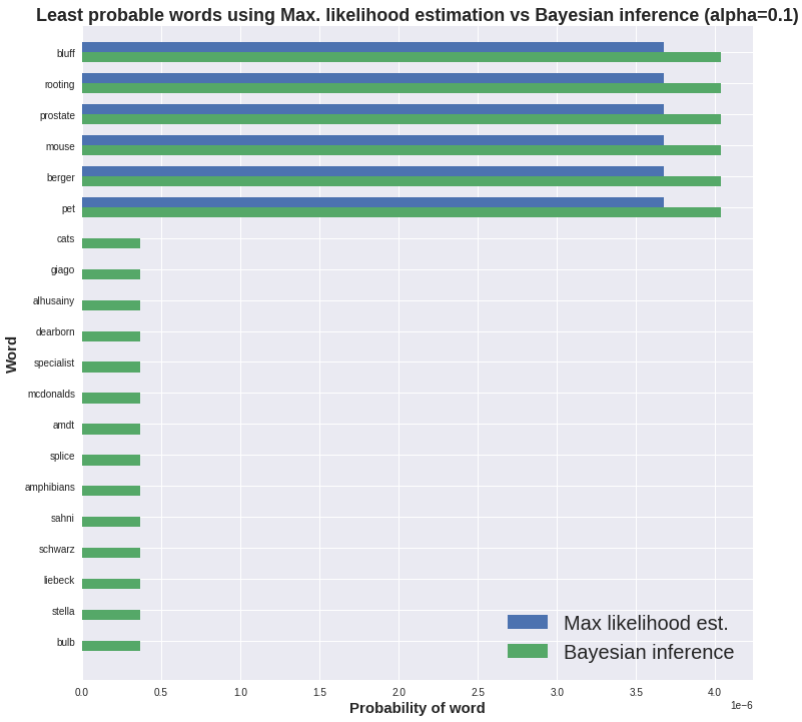

Figure 2.2: 20 smallest probability words with $\alpha = 0.1$

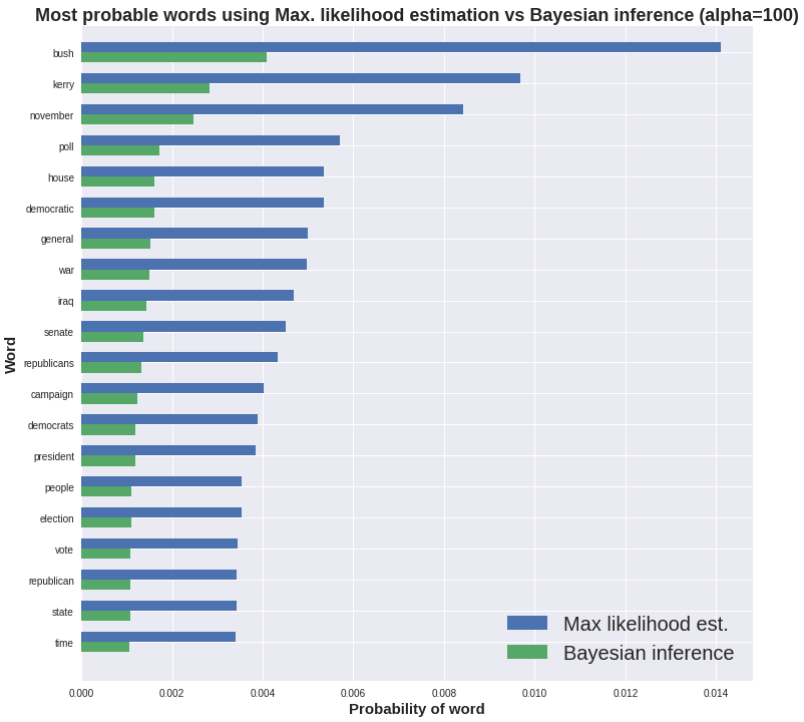

Figure 2.3: 20 largest probability words with $\alpha = 100$

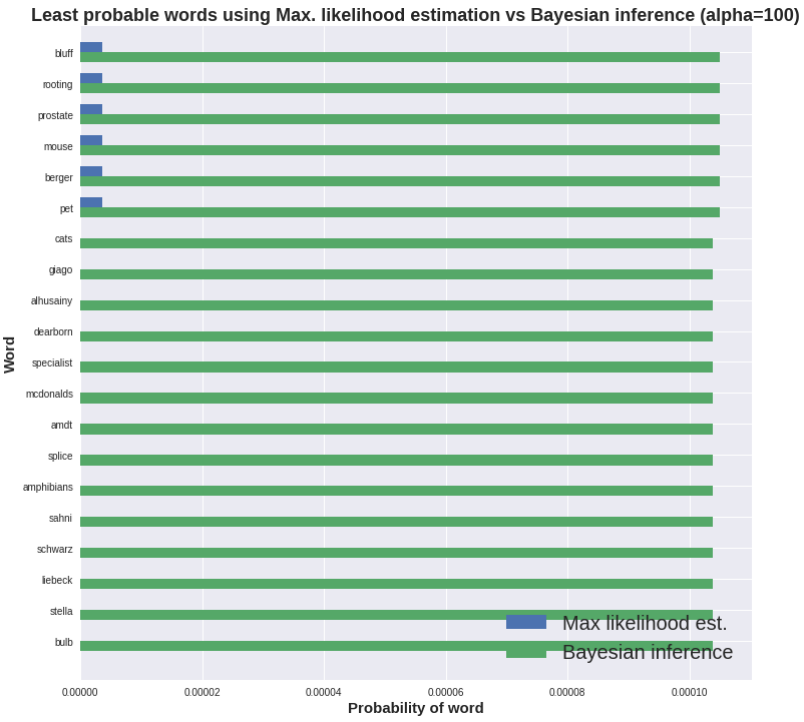

Figure 2.4: 20 smallest probability words with $\alpha = 100$

3. Question c)

As it's been explained before, the probability of a sequence of words can be computed by multiplying the probability of each word: $P(\mathbf{w}|\boldsymbol{\beta}) = \prod_{i=1}^{M} \beta_m^{c_m}$.

Since the data we're given just accounts for **word counts**, we need to multiply by a combinatorial factor **(multinomial distribution function**, equation 3.1) , to take into account all word combinations in a document. The combinatorial factor represents all the possible word sequences that could have generated the given word count. However, **all possible sequences are considered to be equally probable**, so we can drop the combinatorial factor **recovering the categorial distribution function**.

$$(c_1, \ldots, c_m|\boldsymbol{\beta}) = \frac{M!}{c_1!c_2!\ldots c_m!} \prod_{i=1}^{M} \beta_m^{c_m}. \tag{3.1}$$

The **performance of a model** on the test set can be measured using different metrics. One of them is **per-word perplexity (PWPP)** of the test set:

$$PWPP(d) = \exp\left(\frac{-1}{N_d} logP(\mathbf{w}_d|\boldsymbol{\beta})\right), \tag{3.2}$$

where N_d is the number of words of document d, $\mathbf{w}_d$ is the list of words in document d and $logP(\mathbf{w}_d|\boldsymbol{\beta}) = \sum_{m=1}^{M} c_m^d log(\beta_m)$.

Several PWPPs and log probabilities can be computed using equation 3.2 for document $d = 2001$, as shown in table 3.1.

α VALUE	Log-probability	Per-word Perplexity
0.1	-3691.22	4398.98
0.5	-3689.98	4386.72
1	-3688.62	4373.11
10	-3680.75	4295.55
100	-3744.23	4962.19

Table 3.1: Log-probabilities and perplexities of document $d = 2001$ for different α. The log probability and PWPP can be computed for each document given an α value.

Perplexity is a measurement of how well a probability model predicts a sample. In this case, the perplexity of a document **tries to capture the number of available choices for choosing a test word from all M vocabulary words**. Each word is weighted by the inferred vocabulary probabilities $\boldsymbol{\beta}$. Lower perplexity represents lower uncertainty for choosing the test word, so a small value relates to high probability under the model.

If a document consists of **common words**, the perplexity will be **small**. Similarly, **rare words increase perplexity**, being infinite if there is a word not present in the training set. That's the reason why different documents have different perplexities.

In figure 3.1 **all test documents perplexities are plotted** (with $\alpha = 0.1$ to infer $\boldsymbol{\beta}$), as well as the perplexity if a **uniform multinomial distribution** were used.

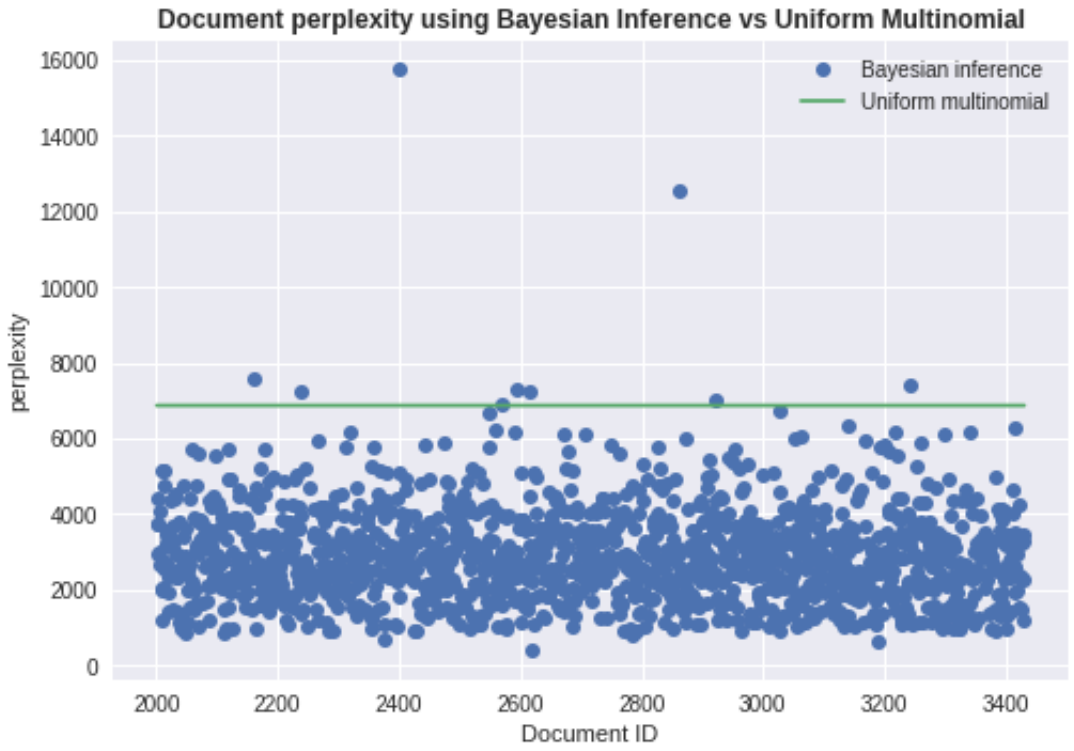

Figure 3.1: Perplexities for all test documents and uniform multinomial. For uniform multinomial, each for has probability of $1/M$, so the perplexity is exp(log$(M)) = 6906$.

The **PWPP for the whole test set** is calculated by averaging over all documents:

$$PWPP(testSet|\alpha = 0.1) = \exp\left(\frac{-1}{N_{test}}\sum_{d=1}^{N_{test}} log(P(\mathbf{w}_d|\boldsymbol{\beta}))\right) = 2697.11 \tag{3.3}$$

4. Question d)

The inference procedure can be further improved by **assigning a topic to each document**.

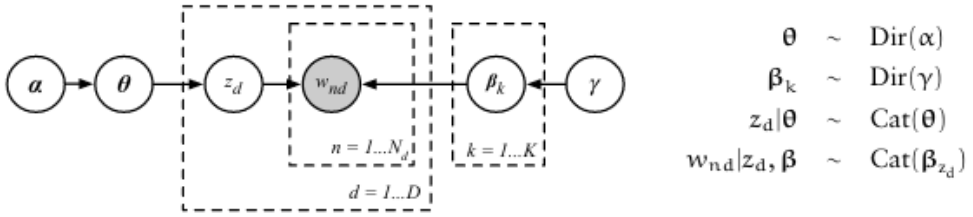

Figure 4.1: Graphical representation of Bayesian mixture of categoricals model. New latent variables are introduced $z_d \in \{1, \ldots, K\}$ to describe the topic assignment as represented in grahic 4.1. Inference is done using **another Dirichlet distribution over all categories/topics** and the probability of a document belonging to group $k \in \{1, \ldots, K\}$ is $\theta_k \in \boldsymbol{\theta} = [\theta_1, \ldots, \theta_K]$

Gibbs sampling is used when calculating these probabilities due to calculating the exact distribution is intractable. At the end of each Gibbs sweep, the parameters $\boldsymbol{\theta}$ can be computed similarly as $\boldsymbol{\beta}$:

$$\theta_k = \frac{c_k + \alpha}{\sum_{l=1}^{K} c_l + \alpha}, \tag{4.1}$$

where c_k is the number documents belonging to topic k.

Figures 4.2 and 4.3 show the evolution of the estimated $\boldsymbol{\beta}$ vs Gibbs iteration for two different initial seeds using $\alpha = 10$ (Dirichlet parameter over mixture components) and $\gamma = 0.1$ (Dirichlet parameter over words).

Each θ_k converges to **different probabilities** in both figures, so different initialized Gibbs samplers move around **different local optimums** and they **do not converge** in this model. However, the **resulting perplexity** of the test set is 2092.3 and 2123.6 respectively, which are quite low compared to previous results. This means Gibbs sampling achieves a decent representation of the **underlying posterior distribution** and, since the underlying distribution is **independent** of the initial condition, Gibbs samples should have **similar behaviours** regardless the random seed.

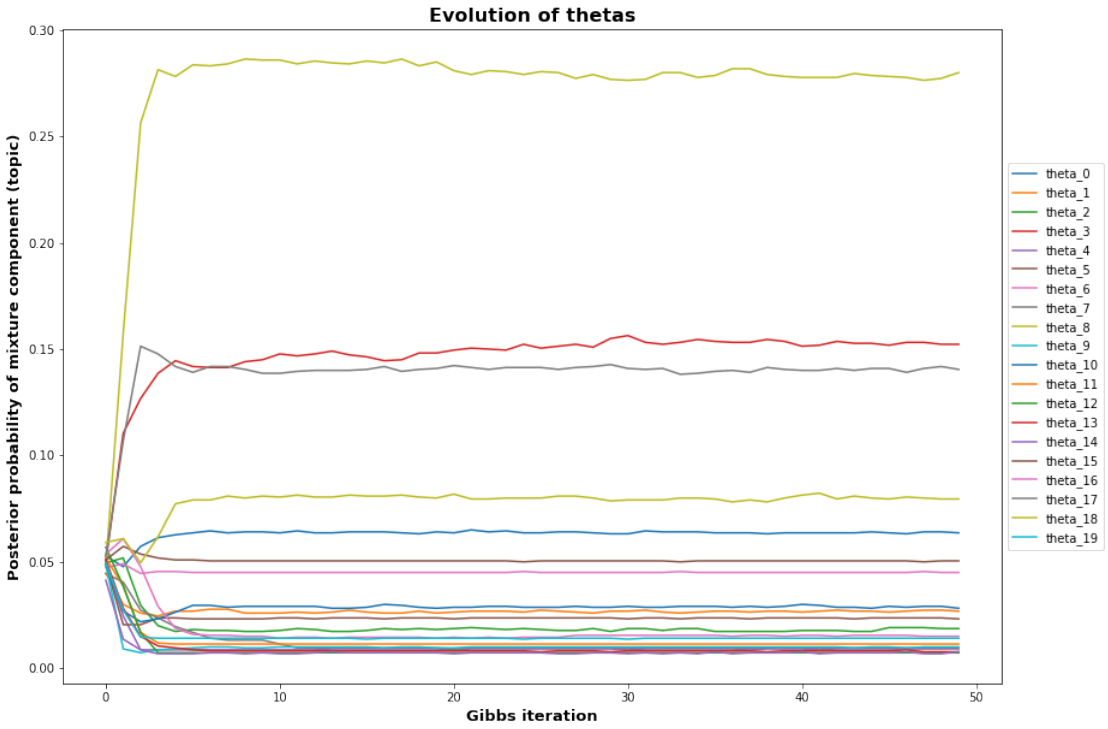

Figure 4.2: Posterior probabilities of components per Gibbs iteration (seed=1)

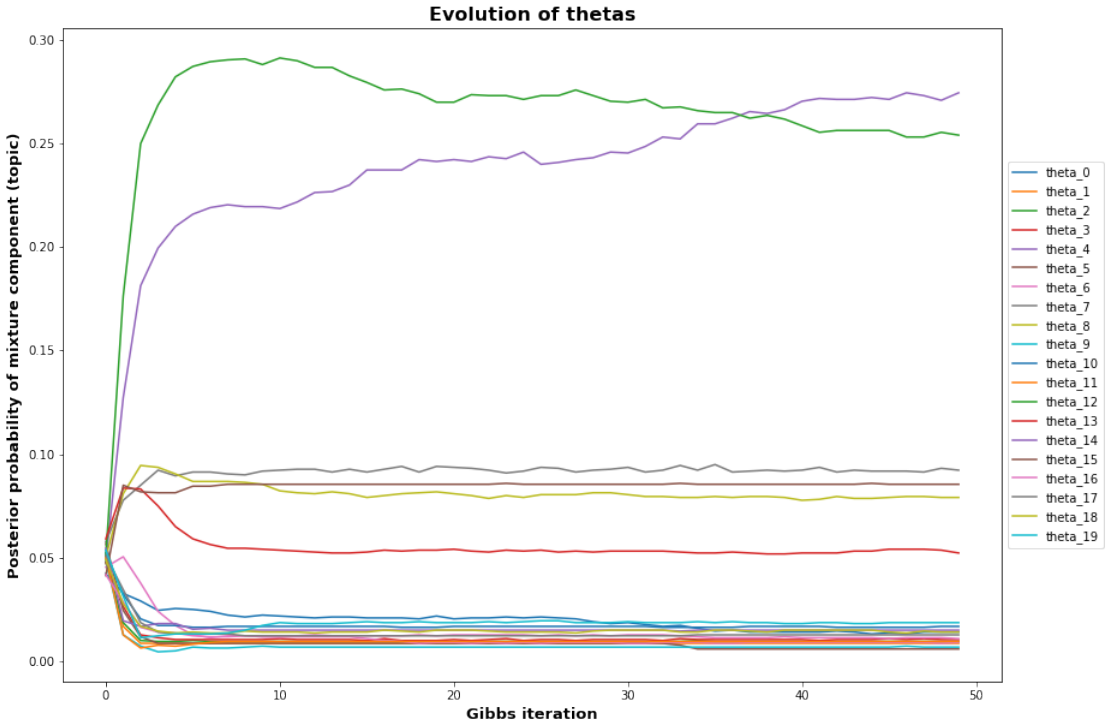

Figure 4.3: Posterior probabilities of components per Gibbs iteration (seed=3)

5. Question e)

Now **each document can have its own topic proportions** $\boldsymbol{\theta}_d = [\theta_{1d}, \ldots, \theta_{Kd}]$, so **a document can belong to multiple topics** $k \in \{1, \ldots, K\}$. This is the **Latent Dirichlet Allocation (LDA)** model, which derives each θ_{kd} from the word counts c_{kd} in document d assigned to topic k:

$$\theta_{kd} = \frac{c_{kd} + \alpha}{\sum_{l=1}^{K} c_{ld} + \alpha} \tag{5.1}$$

If the provided `lda.py` script is modified to calculate $\boldsymbol{\theta}_d$ for each document d and run specifying $\alpha = 0.1$ and $\gamma = 0.1$, a **test set PWPP of** 1641.24 **is obtained (seed=3)**, which is **much lower than any previous PWPPs**. Therefore, this model has the **best performance** at generating the test set.

In figure 5.1 the evolution of the topic proportions for document $d = 1000$ is shown.

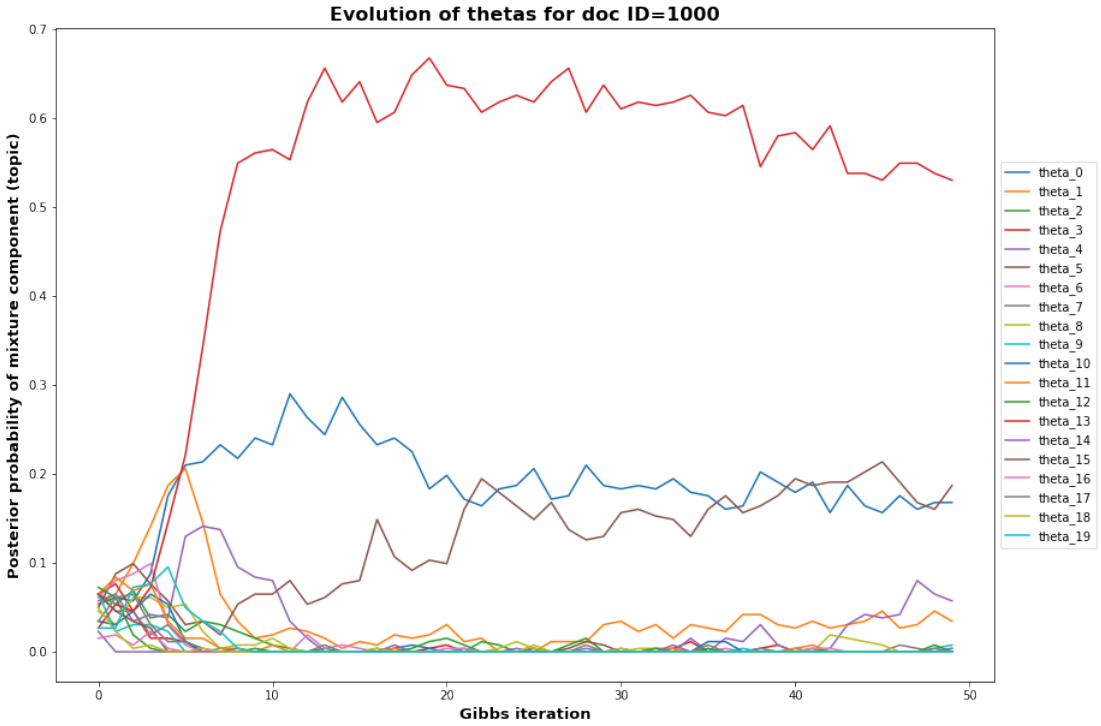

Figure 5.1: Posterior probabilities of components per LDA iteration (seed=1). The fact that **one particular topic has a much larger final proportion** than the others can be explained by the **'rich-get-richer' property**: a topic with large word count is more likely to be assigned to the word being resampled, hence getting even larger word count. Additionally, we can see in figure 5.1 that some θ_{kd} still have a **wiggly** trend, so **50 sweeps might not be enough**.

The word entropy for each of the topics can also be computed by:

$$H_k = -\sum_{m=1}^{M} \beta_{km} \log_2(\beta_{km}) \tag{5.2}$$

Entropy can be interpreted as the **measure of uncertainty associated with each observation**. So zero entropy is achieved when all the probability mass is concentrated in single word. Since the equation 5.2 uses binary logarithm, the entropy unit is 'bits'.

Figure 5.2 plots the evolution of word entropies for each topic per LDA iteration.

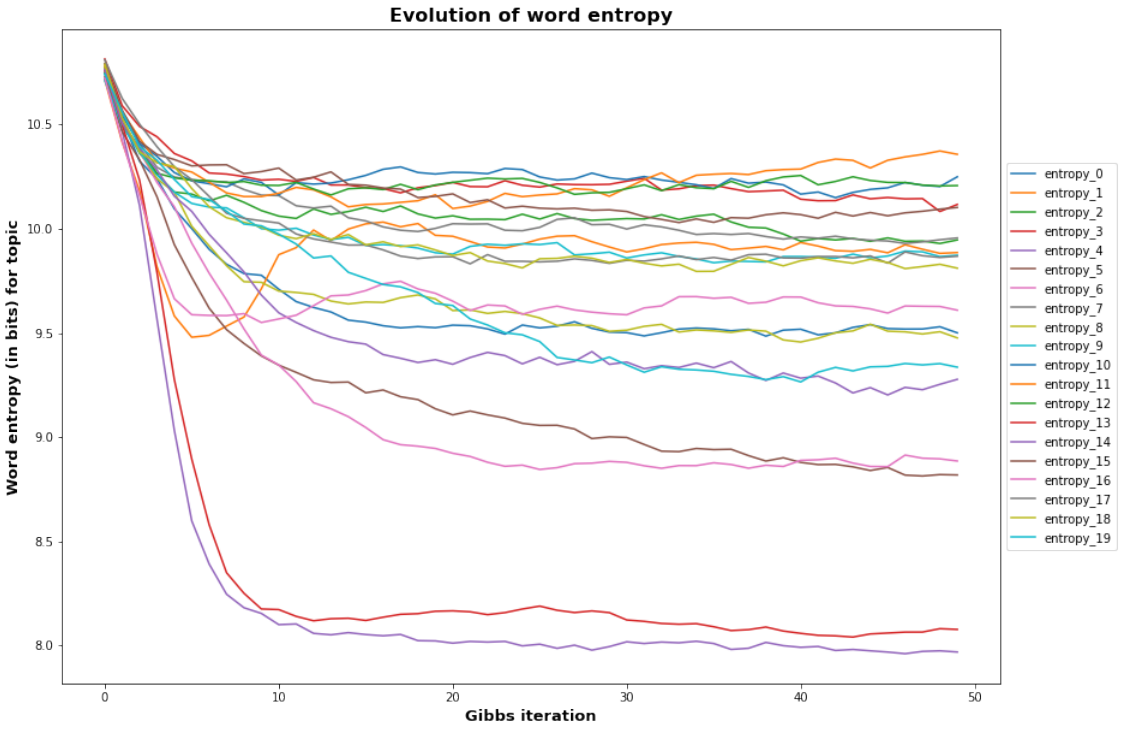

Figure 5.2: Word entropy (bits) for each topic per LDA iteration (seed=3). Initially all entropies have almost the same value. After some epochs, the entropies drop. **A decrease in entropy** can be thought of as **each topic only containing the minimum words required to model it**.

Bibliography

[1] Modelling Document Collections (2021). Carl Edward Rasmussen, David Krueger. https://www.vle.cam.ac.uk/course/view.php?id=69021&sectionid=252511#section-9