CUED - Engineering Tripos Part IIB 2019-2020 **Module Coursework**

Module	4F13	Title of report	Coursework 3 - Latent Dirichlet Allocation
Date submitted: 6/12/2019		Assessment for this module is ☒ 100% / ☐ 25% coursework of which this assignment forms _____ %	

UNDERGRADUATE STUDENTS ONLY		POST GRADUATE STUDENTS ONLY			
Candidate number:	5584F	Name:		College:	

Feedback to the student

☐ **See also comments in the text**

		Very good	**Good**	Needs improvmt
CONTENT	**Completeness, quantity of content:** Has the report covered all aspects of the lab? Has the analysis been carried out thoroughly?			
	Correctness, quality of content Is the data correct? Is the analysis of the data correct? Are the conclusions correct?			
	Depth of understanding, quality of discussion Does the report show a good technical understanding? Have all the relevant conclusions been drawn?			
	Comments:			
PRESENTATION	**Attention to detail, typesetting and typographical errors** Is the report free of typographical errors? Are the figures/tables/references presented professionally?			
	Comments:			

Overall assessment (circle grade)	A*	**A**	B	C	D
Guideline standard	>75%	**65-75%**	55-65%	40-55%	<40%
Penalty for lateness:	*20% of marks per week or part week that the work is late.*				

Marker: Date:

4F13 - Coursework Three - Latent Dirichlet Allocation

Candidate Number: 5584F

Word count: 1000

January 14, 2020

I. a

Using word count data different models for the probability of each word appearing can be created. The simplest model is to perform maximum likelihood optimisation using occurrences seen in training documents. As in equation 1 this requires computing the total count of each word and then renormalising to generate a multinomial distribution. Figure 1 shows the 20 most common words and their probabilities under this model.

$$p_m^{(ML)} = \frac{c_m}{\sum_l c_l} = \frac{c_m}{N} \tag{1}$$

This distribution can be used to calculate the probability of a document given the model. In this simple model each document follows the same distribution, irrespective of their document class. To maximise the log probability the test document should contain only the most probably word from training as independence is assumed between words, a document with only one word would maximise this with log probability = $\log(p_{max})$. The minimum probability occurs when the test document contains at least one new word so has log probability -∞. The implications of this are that new words will break document testing due to a zero probability and atypical sequences of the same word are individually more likely then typical sequences observed in a real document. Considering sequence probabilities and combination factors into the model could solve the latter problem.

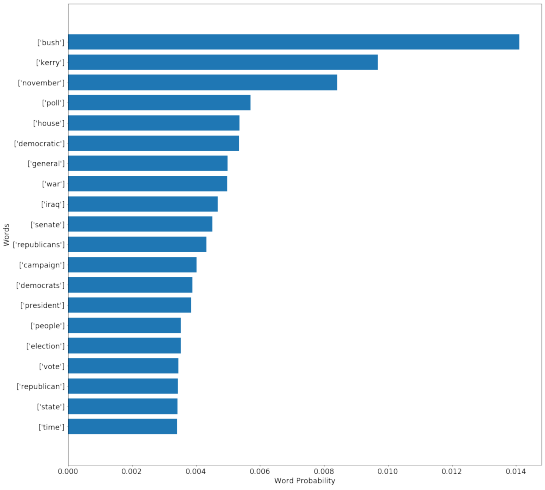

Figure 1: *Most probable words under simple model*

II. b

A better method of model generation uses a prior distribution and inference to update the model after observations. For this a symmetric Dirichlet model with parameter α is used. Equations 2 shows the prior and the posterior update for a K dimensional distribution, renormalisation is essential after the update.

$$Dir(K, \boldsymbol{\alpha}) \rightarrow Dir(K, c_1 + \alpha_1, ..., c_K + \alpha_K) \tag{2}$$

When using this model to predict word probabilities in test data the categorical word probabilities are first drawn from the Dirichlet distribution. After this both models follow equation 3 to calculate the probability of the document

given the model.

$$p(\mathbf{W}_d|\boldsymbol{\beta}) = \prod_{n=1}^{N} p(W_{nd}|\boldsymbol{\beta}) \tag{3}$$

When $\alpha = 1$ the prior is a uniform distribution, with higher values giving more certainty that uniformity is desired as α can be though of as psudo-counts. This prior leads to unseen words in training having a non-zero probability associated with them. Figure 2 shows the probabilities of each word for various α's. Increasing α has the effect of making all probabilities in $\boldsymbol{\beta}$ more equal. Increased by the right amount it can ensure unseen words have a reasonable probability and common word probabilities aren't inflated by their usually high training presence. These both reduce overfitting. To further generalise the model to infinite new words a Dirichlet process could be used.

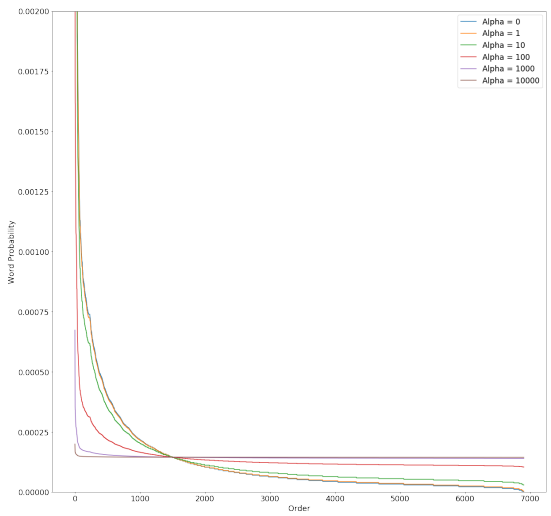

Figure 2: *Effect of changing α in prior, zoomed into lower probabilities*

III. c

$$p(\mathbf{W}_d|\boldsymbol{\beta}) = \frac{N!}{\prod_{u=1}^{U} c_u!} \prod_{n=1}^{N} p(W_{nd}|\boldsymbol{\beta}) \tag{4}$$

The probabilities of a test document under the Baysian model can be calculated as in equation 3. This uses a categorical distribution function. However, if the combinations of word position within a document are considered as in equation 4 you can predict from a multinomial distribution. Table 1 shows some of these probabilities. When using a categorical distribution atypical documents such ones with one word have a high probability compared to typical documents. However, if we assume that all documents being compared are typical with similar word proportions then this effect can be ignored and the word counts alone can be used for simplicity.

α	categorical	multinomial
0	-3691.5.6	-1691.1
1	-3688.6	-1688.2
5	-3640.7	-1640.3
10	-3592.1	-1591.8
100	-3199.1	-1198.6

Table 1: *log probabilities of test document 2001*

The per-word perplexity can be calculated using equation 5. Perplexity is a good way of evaluation language models as a small value relates to high probability under the model. Figure 3 shows the perplexity for document 2001 and test average. A minima is observed in each case which corresponds to the best mix between learning from training and incorporating new words into the model. Per-word perplexities differ between documents as each document has different word counts and probabilities under the trained model.

$$Perplexity = exp(-\frac{\sum_{d=1}^{D} log(p(\mathbf{W}_d|\boldsymbol{\beta}))}{\sum_{l\ in\ test} c_l}) \tag{5}$$

The per-word perplexity under a uniform multinomial distribution for document d is shown in equation 6, for document 2001 this has a value of 6906. Which is higher then the Baysian model as expected.

$$Perplexity = exp(-\frac{log\frac{1}{N}^{N_d}}{N_d}) \tag{6}$$

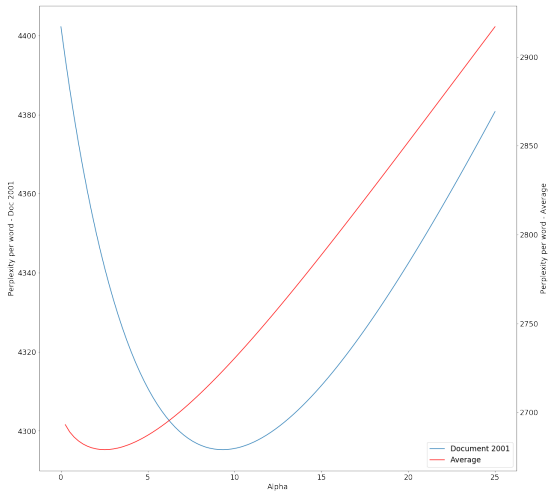

Figure 3: *Test Perplexities for various α values and test documents*

IV. D

The model can be further improved by introducing different topics and having each document a member of one topic. The effect of this when testing is to first assign the document a categorical distribution across words β_k. This is done using another Dirichlet distribution over all categories and the probability a document belongs in group k (θ). The Dirichlet parameters used were 10 for document mixing and 0.1 for word distributions.

Gibbs sampling is required when calculating these extra probabilities. This is due to calculating the exact distribution being intractable. Within the iterations each variable is being sampled to collapse the model to a simpler categorical distribution. In this process documents are progressive updated with a new class until convergence. Figure 4 shows the convergence in a Baysian mixture model with 20 groups. Convergence is achieved as each line plateaus so has constant mean in addition to constant consisted variance as shown in figure 5. The largest group converges the slowest in 25 iterations. To determine the stationary distribution more carefully additional Gibbs iterations could be performed and longer averages taken. Taking account of the auto correlation between samples in this average would further improve certainty convergence as been achieved.

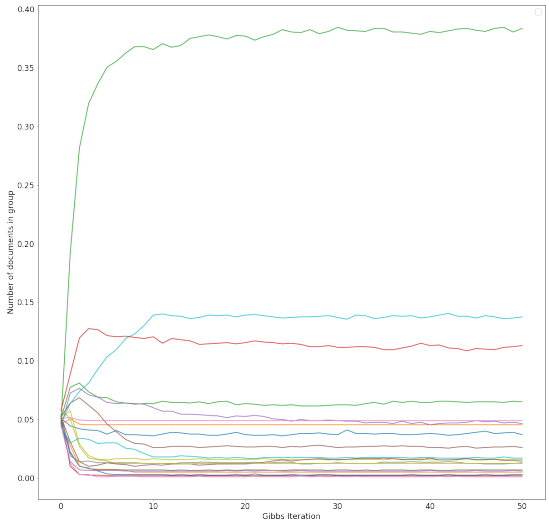

Figure 4: *Document mixing proportions vs Gibbs sample iteration in Baysian mixture model*

V. E

In latent Dirichlet allocation documents individual words within a document can be assigned different topics. This is done by having a mixing proportion across topics for each document, with each word within the document having its topic and categorical word distribution drawn from the document mixture. This change is advantageous as it allows for multi-topic documents to be accurately modelled. Gibbs sampling is used once again to make the problem tractable. Figure 6 shows the convergence for the mixing proportion of words. Convergence is slower then in a BMM which is expected as the number of words is far larger then number of documents. After around 40 iterations the means and standard deviations in figure 7 appear to have settled. However, some groups such as orange seem to still be changing after 50 iterations. For this reason I would say 50 sweeps is inadequate to reach a stationary distribution.

The per-word perplexity can once again be calculated, across all the test docments this has

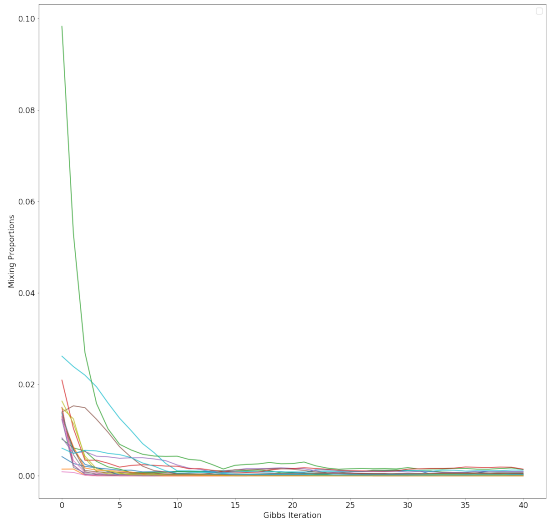

Figure 5: *Standard deviation of mixing proportions for next 10 iterations in BMM,*

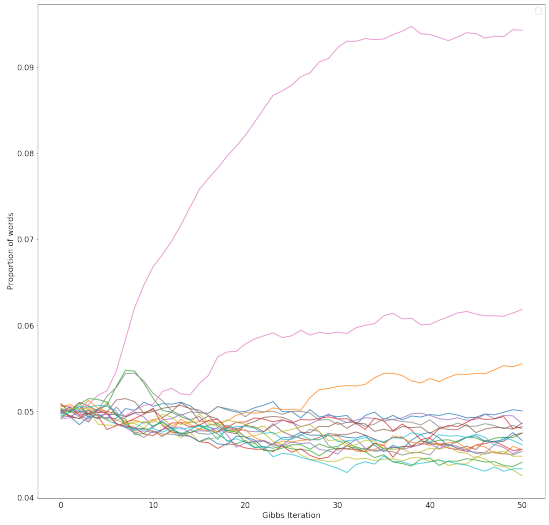

Figure 6: *Word mixing proportions vs Gibbs sample iteration in LDA model*

a value of 2071.7. Comparing this to BMM at 2127.8 and the simpler model in figure 3 with minima at 2675, its clear that the LDA model creates the best generalised model for a variety of test documents.

The word entropies can be calculated for each Gibbs iteration, this uses the number of unique words assigned to each topic. Figure 8 shows this entropy across iterations. The units are nats. Initially all groups have the same entropy as groups start off randomly assigned. As iteration number increases the entropies drop, this is as expected as this means that the groups are approaching a distinct distribution that is not uniform. A decrease in entropy can be thought of as each topic only containing the minimum words required to model that subject.

VI. Appendix

i. a

```
for i in range(N_1):
    counts[A[i,1]-1]+=A[i,2]
mlm = counts/N_w
```

ii. b

```
alpha=10
dp = (counts + alpha)/(N_w+W*alpha)
sorted_word(dp,V,M)
```

iii. c

```
def log_prob(model, document):
    c = document[:,2]
    log_prob = np.dot(c, np.log(model[
        ↪ document[:,1]-1]))
    return log_prob

def get_per(model,document,combi=False)
    ↪ :
    if combi == True:
        lp = combi_log_prob(model,
            ↪ document)
    else:
        lp = log_prob(model,document)
    n = np.sum(document[:,2])
    return np.exp (-lp/n)
```

iv. d

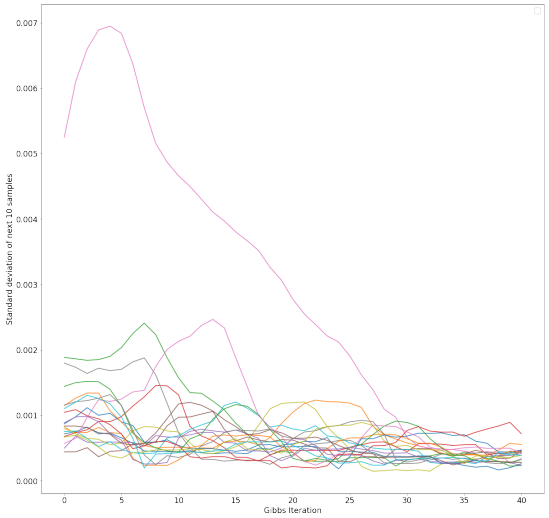

Figure 7: *Standard deviation of mixing proportions for next 10 iterations in LDA*

```
mixing_prop[iter+1, :] = sk_docs[:,0]
```

v. e

```
mixing_prop = np.zeros((num_gibbs_iters
    ↪ + 1, K))
    word_entropy = np.zeros(mixing_prop.
        ↪ shape)
    mixing_prop[0, :] = sk
    for k in range(K):
        word_entropy[0, k] = entropy(
            ↪ swk[:, k])
```

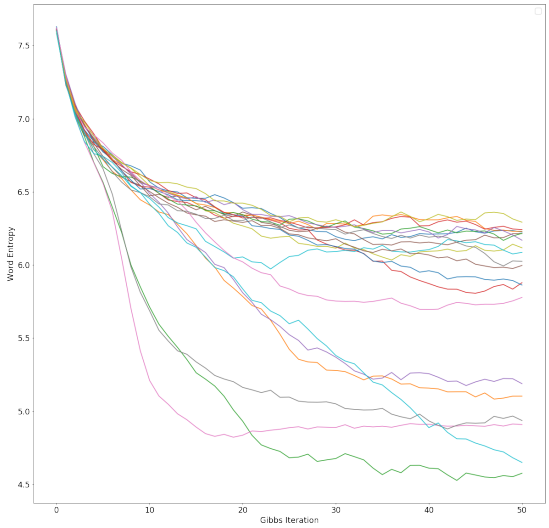

Figure 8: *Word entropies within a topic*