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

Module	4F13	Title of report	Latent Dirichlet Allocation Models
Date submitted: 13/12/2020		Assessment for this module is ☑ 100% / ☐ 25% coursework of which this assignment forms 33.33%	

UNDERGRADUATE STUDENTS ONLY		POST GRADUATE STUDENTS ONLY			
Candidate number:	5673D	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: Probabilistic Machine Learning
Latent Dirichlet Allocation model

Candidate number: 5673D
Words count: approximetly 985

December 13, 2020

a Maximum Likelihood Multinomial over Words

From the lecture notes, we already know that after maximising the likelihood we get the following form for the parameters β:

$$\hat{\beta}_m = \frac{c_m}{N}$$

where c_m is the total count of the world m in A and N is the total number of words in A. In other words, $\hat{\beta}_m$ is the normalised frequency of word m. To find this values, we will use the following code:

```
load kos_doc_data.mat

W = max(A(:,2));                                    % number of unique words in A
D = max(A(:,1));                                    % number of document in A
word_frequency = zeros(W,1);                        % initialise frequency of each word

for i=1:size(A)                                     %iterate over entries
    w = A(i, 2);                                    %get the word in the entry
    c = A(i, 3);                                    % get the counts for the word
    word_frequency(w,1) = word_frequency(w,1) + c;  %add the counts
end

word_frequency = word_frequency/sum(word_frequency); %normalise
```

In this model, the probability of a word is equal to its frequency in the collection A. This means that if a word appears in the test set B and it does not appear in the training set A, its probability is equal to zero. If we calculate the log-probability of a new word, we get minus infinity. This shows us that this model cannot process new words. Moreover, any model that will give finite values for the log-probability will be better than this one.

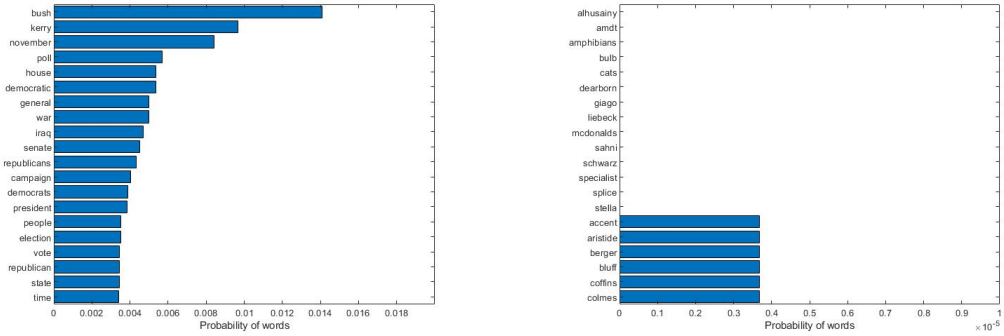

Figure 1: On the left: The most probable 20 words. On the right: The least probable 20 words.

b Bayesian Inference using Symmetric Dirichlet prior

In this model we want to solve the problem of new words with zero probability. To do that we propose using Bayesian inference using a symmetric Direchlet prior with α as a concentration parameter. Our new probability distribution over the words is:

$$p(w_m|\alpha, A) = \beta_m = \frac{c_m + \alpha}{\sum_{l=1}^{M}(c_l + \alpha)}$$

where the parameter α acts as the pseudo-counts of each word and M is the size of the vocabulary. This model increases the probability of words with small frequencies and decreases the probability of the words with high frequency. For instance, if $\alpha = 1$ (small), the probability of words that appear once is increased by almost 100%, while the probability of common words is only slightly decreased. If $\alpha = 10^6$ (very large) our probability distribution tends to a uniform distribution, which is expected as the pseudo-counts become much larger than the actual counts.

```
alpha = 1;                                                  %set the parameter
word_frequency = word_frequency + alpha;                    %add the pseudo-counts
bayes_word_frequency = word_frequency/sum(word_frequency); %normalise
```

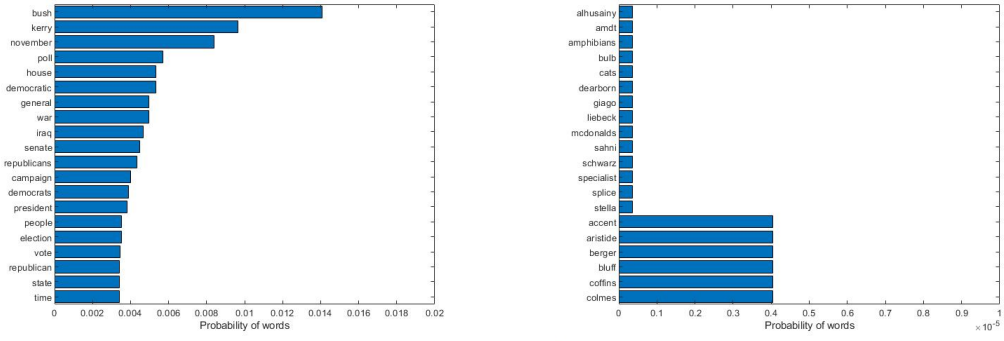

Figure 2: Using Bayesian inference: On the left: The most probable 20 words. On the right: The least probable 20 words.

c Probabilities and perplexities of the test set

In this section, we are interested in finding a measure of performance. To do this, we need will make use of the joint probability of a collection and per word perplexity.

The probability of a given sequence of words(document) is define as:

$$p(\mathbf{w}|\beta) = \prod_{m=1}^{M} \beta_m^{c_m^{(test)}}$$

where β_m is probability of appearance of word m and $c_m^{(test)}$ is the word m count inside the test collection. Note that we are interested in a specific sequence(order) of words and we do not use the combinatorial factor. Therefore, this is a categorical distribution function. Taking the log of the probability, we get:

$$\log(p(\mathbf{w}|\beta)) = \sum_{m=1}^{M} c_m^{(test)} \log(\beta_m)$$

The per word perplexity is given by:

$$Perpelexity = \exp\left(-\frac{p(\mathbf{w}|\beta)}{N}\right)$$

where N is the total number of counts. The result for two documents in the test set B and over all documents in B are presented below. We used βs found in section b with $\alpha = 1$.

Document ID	Words count	Unique words	log probability	Perplexity
2001	440	232	-3688	4373
2060	106	87	-816	2222
All documents	195816	6870	-1.54e-6	2683

Table 1: Log-Probabilities and perplexities

The perplexity is a measure that defines the average uncertainty over the words. Therefore, we expect lower perplexities values for documents that has similar distributions to our training set A (see document 2060) and higher values for documents that have more uncommon words. The perplexity over all documents has a general lower value for perplexity because of the large number of words. The larger the number of words, the closer we are to the true distribution. In the case of uniform distribution over words: $\beta_m = 1/M$. Therefore, $\log(p) = -N_B \log(M)$ and the perplexity $= M = 6906$.

```
for i=1:size_B(1)                                   %iterate over entries
    wb = B(i,2);                                    %get word ID
    cb = B(i,3);                                    %get word count
    frequency_b = bayes_word_frequency(wb);         %get the word's frequency
    log_p = log_p + cb * log(frequency_b);          %recalculate the log-probability
    word_count = word_count + cb;                   %add to the total word count
    if (B(i,1) == D_1)                              %do the same thing for a specific document
        log_p1 = log_p1 + cb * log(frequency_b);
        word_count1 = word_count1 + cb;
    end
end
```

d Mixture of multinomials model

In this section, we are interested in splitting our word in a number of topics K. The mixing proportions are:

$$p(\theta|\mathbf{z}, \alpha) \propto Dir(\mathbf{c} + \alpha)$$

where c_k are the counts for mixture k. Now, if we keep track of this mixing proportions after each Gibbs step and normalise them using the code below:

```
mixing_prop(:,iter+1) = sk_docs + alpha;
mixing_prop(:,iter +1) = mixing_prop(:, iter +1)/ sum(mixing_prop(:, iter +1));
```

We get the results in Fig. 3. We can see that after almost 20 steps, the mixing proportions reach a plateau. This means they converged to a final value. Running the program multiple times, we observed that the final values are not always the same. This behaviour is explained by the initial random generated seeds that converge to a local maximum of the posterior. Once they reach a local minimum they cannot longer explore the whole space for the global maximum. A method to choose between this local maxima is to take the one with minimum perplexity.

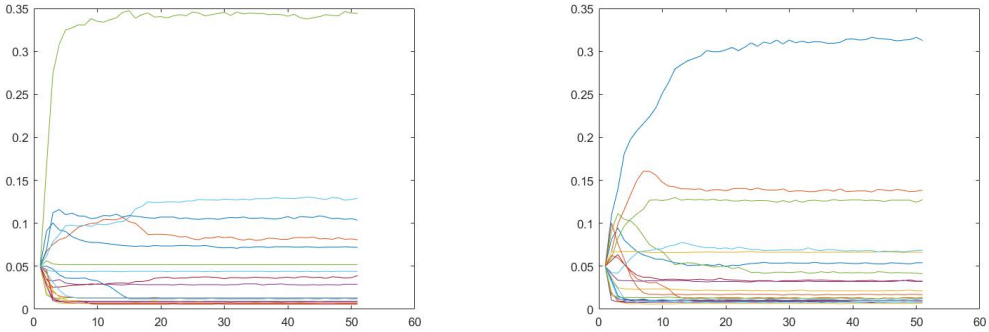

Figure 3: Mixing Proportions vs Gibbs iterations

e Latent Dirichlet Allocation model

Now, let us have a look at how Latent Dirichlet Allocation model performs in comparison with our previous approaches. By looking at the mixing proportion, we can see that we have more relevant topics than we previously had. Moreover, now we have more than one dominant topic. By comparing the perplexities for different models(See Table 2), we can determine that this model has the lowest perplexity, although it does not seem to converge after 50 iterations. Moreover, increasing the number of iterations will result in a marginally better perplexity, but will take considerately more time to compute. Depending on what we want to obtain(accurate prediction or computational speed), we can say that 50 iterations are a fairly good trade off between our targets.

Model	ML	BI	BMM	LDA
Perplexity	-∞	2683	2110	1647

Table 2: Log-Probabilities and perplexities

The world entropy for each topic is given by the following formula:

$$H = -\sum_{m=1}^{M} p(w_m|A,\gamma,z=k)\log_2(p(w_m|A,\gamma,z=k))bits$$

where $p(w_m|A,\gamma,z=k)$ is the probability of the word $m$coming from the topic k. The units used for the word entropy are bits. Again we plot this as entropy for each topic as function of the Gibbs iterations. We can see that the entropy of each topics decreases from the starting value(uniform distribution) as it becomes closer to the true distribution. This behaviour it is highly expected, as the entropy has its maximum when the distribution is uniform. The difference between mixing proportion and entropy is that the entropy converges faster($\tilde{2}5$ iterations) and after that there is little change in values. Also the more probable words and topics has the lowest entropy.

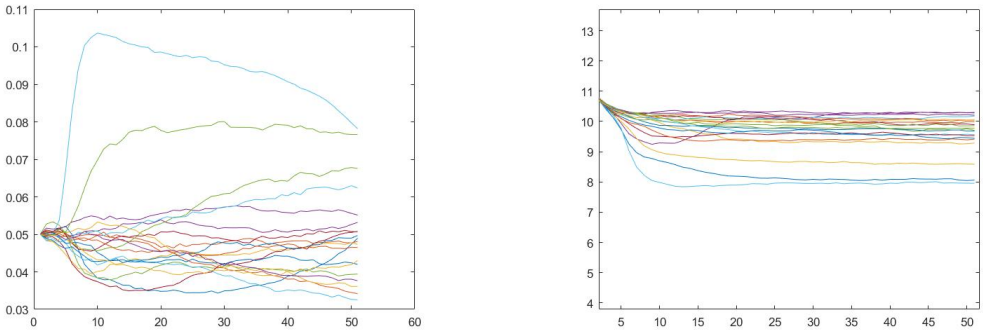

Figure 4: On the left: Mixing proportion vs Gibbs iteration. On the right: Word entropy vs Gibbs iterations