The Pennsylvania State University
The Graduate School
College of Engineering

AN EXTENSION TO THE PARSIMONIOUS TOPIC MODELING

A Thesis in
Electrical Engineering
by
Yezhou Chen

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2015

The thesis of Yezhou Chen was reviewed and approved* by the following:

David J. Miller

Professor of Electrical Engineering

Thesis Advisor

Minghui Zhu

Professor of Electrical Engineering

Committee

Kultegin Aydin

Professor of Electrical Engineering and Department Head

Electrical Engineering Department Head

*Signatures are on file in the Graduate School.

Abstract

In this thesis we develop a new model for estimating topics based on parsimonious topic model and Latent Dirichlet Allocation. In parsimonious models, each word has a topic shared occurring probability or a topic specific occurring probability for each topic and this is controlled by a switch. In our model, we use one more switch set to identify the mentioned switch subset(all switches for one word in all topics) by one of three cases: the word has a topic shared occurring probability for all topics, the word has a topic specific occurring probability for all topics, the word has a topic shared occurring probability for some topics and a topic specific occurring probability for some topics. We use a generalized Expectation-Maximization algorithm as a learning algorithm to optimize the parameters and minimize the objective function. Numerical results are presented to examine the performance of such a model.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background of Topic Modeling

The task of model identification is to choose one among a set of candidate models to describe a given data set. The candidate models include a series of models with different numbers of parameters or structures [2]. We always perform the choice in two general steps: develop a goal function and then find the model that minimizes (or maximizes) the goal function. There are many possible criteria and most of them trade off data fitness and model complexity because these two goodnesses are always conflicted. It is easy to imagine that if a model's parameter size increases, the model may have a better training data fitness but this bring us a heavier parameter complexity also. Another potential problem a lot of models are facing is, their goal functions may not balance the two terms: fitness and complexity well. If the term describing the fitness dominates, the models are prone to over-fitting. In another case, if the term describing the complexity dominates, we will get underfitting models because in this case the "best" model will always be the model with least model order. In this thesis we develop a new model that is different from both Latent Dirichlet Allocation and Parsimonious Topic Models. We will begin by introducing some basic models.

The "Bag of words" model [3] assigns a probability for each word, for each topic and this results in too many free parameters to determine. One important assumption in the "Bag of words" model is, the word order in documents is neglected and we assume words in documents are generated independently. This assumption is used by many topic models like mixtures of unigrams, Latent Dirichlet Alloction(LDA) and parsimonious topic modeling(PTM) [1].In the mixture of unigrams model, each document is generated by a single topic. The procedure of how a documents corpus is generated can be understood through the following algorithm:

Algorithm 1 Mixture of unigram model word generation algorithm [4]

1: **for** each document **do**

Step 1. Choose a topic according to proportions α_d

2: **for** each word slot **do**

Step 2. Choose a word according to the word probabilities in the topic chosen.

3: **end for**

4: **end for**

Latent Dirichlet Allocation [5] is an extension of the mixture of unigrams model. The difference between these two models is, the generation of each word in each document includes an independent choice of topic depending on the topic proportions of that document. The word generation algorithm of LDA is as followings:

Algorithm 2 LDA model word generation algorithm [5]

1: **for** each document **do**

Step 1. Choose topic proportions $\alpha_d, d = 1, 2, 3...dlength$ according to a Dirichlet distribution with parameter η

2: **for** each word slot **do**

Step 2. Choose a topic according to proportions α_d

Step 3. Choose a word according to the word probabilities in the topic chosen.

3: **end for**

4: **end for**

The LDA model introduces free parameters for each word in each topic and it also forces all

topics to be present in every document [5]. This brings us a huge number of free parameter to estimate and is not reasonbale because many words occur with similar frequency in some topics. A revised model based on LDA is PTM [1]. PTM reduces the number of free parameters by allowing each word to have a probability which is shared across all topics. PTM also allow topics to have zero proportions in documents, i.e., each document only has several active topics. These two sparsities are controlled by introducing two new sets of variables and they are u switches and v switches. These two switches repectively indicate if a word uses a topic specific probability or a shared probability and if a topic is active or not in a document. In PTM the documents corpus is generated as followings [1]:

Algorithm 3 PTM model word generation algorithm [1]

1: **for** each document **do**

2: **for** each word slot **do**

Step 2. Choose a topic according to proportions $\{\alpha_{jd}v_{jd}, j = 1 : M\}$

Step 3. Choose a word according to the word probabilities in the topic l chosen$\{\beta_{ln}^{u_{ln}}\beta_{0n}^{1-u_{ln}}, n = 1 : N\}$.

3: **end for**

4: **end for**

In the results of PTM learning, documents always have very few active topics. This brings a problem in the learning algorithm with a model order reduction step. The learning algorithm optimizes parameters and model structures at different model orders, from high to low. As mentioned, this algorithm always make many documents to only have one or two active topics so when the model order step is performed, the algorithm will likely remove topics which are the only active topics some documents have. In our model, we have all topics present in every documents but we reduce the number of free parameters by allowing each word to have a topic shared distribution in each topic.

The following terms are defined

- $D(d, n)$ is the number of word n in document d.
- $dlength$ is the number of documents.

- L_D is the number of total words in all documents .
- M is the number of topics.
- N is the number of unique words.
- α_{jd} is the proportion for topic j in document d.
- β_{jn} is the topic-specific probability of word n under topic j.
- β_{0n} is the shared probability of word n.
- u_{jn} indicates whether or not word n is topic specific in topic j.
- B_{jn} is the occurring probability of word n in topic j: $B_{jn} = \beta_{jn}^{u_{jn}} \beta_{0n}^{1-u_{jn}}$.
- Θ contains all parameters $\{\beta_{jn}, j = 1 : M, n = 1 : N\}$ and $\{\alpha_{jd}, j = 1 : M, d = 1 : dlength\}$
- $\mathbf{H}$ contains the M and $\{u_{jn}, j = 1 : M, n = 1 : N\}$

1.2 Organization of thesis

This thesis contains five chapters. After this introductory chapter, in chapter 2 the BIC cost expression for our topic models is derived.

Chapter 3 talks about the generalized EM learning algorithm used in our model, including the calculation details.

Chapter 4 is numerical results chapter. In this chapter the results are given for the implementation of our topic model to the real data set Ohsumed Corpus data set.

Chapter 5 is the conclusion chapter.

Chapter 2

BIC Cost Calculation

The best performance of a model at a given model order is quantified using the Bayesian Informatin Criterion (BIC) [6] [7], which is a model selection criterion well-known in the machine learning literature. The total BIC cost contains two parts, complexity cost and the data log-likelihood. The first term describes how complex a model is and the second measures the fitness. The total BIC cost is expressed as a difference of two terms:

$$BIC(M) = CC(\mathbf{H}) - loglikehood(\mathbf{H}, \Theta) \tag{2.1}$$

Where M is the model order. We want a good model to have lower complexity and higher fitness but these two goodnesses are always conflicted. The BIC cost expression balances these two. The general procedure of determining the model is determining the model structure and parameters which minimize the BIC cost at each order and pick the order with the least BIC cost. A failure in modeling of data is always caused by one of the two terms in BIC cost dominating. If the complexity term dominates, the learning algorighm will choose the model with least complexity term–the model with least order will be viewed as the best model regardless of its bad fitness. Conversely, if the log-likelihood term dominates, the selected model has a high risk of over-fitting. The following figure shows a case which failed because the complexity term dominates.

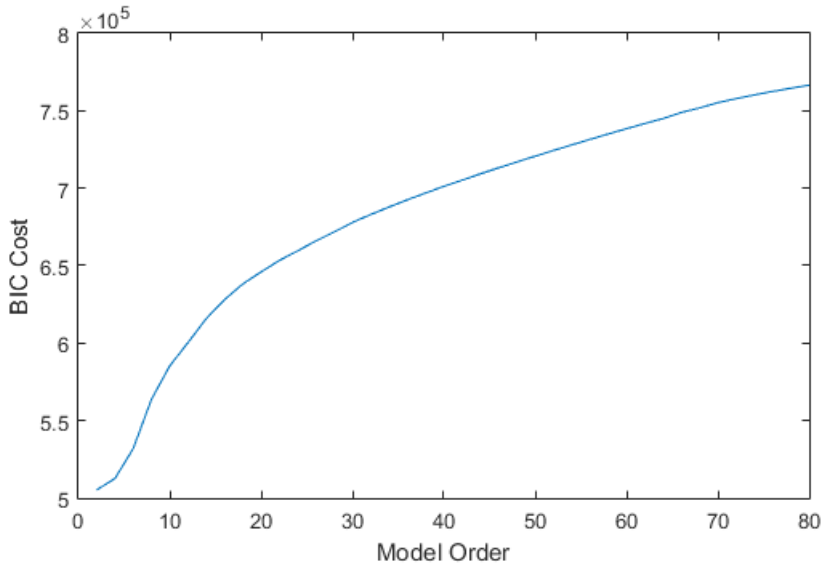

Figure 2.1: BIC curve for a failed modeling applied to Ohsumed data

In this section we derive the objective function for our new model. Based on [1] an approximate negative log-model posterior (BIC cost) can be expressed as [8]:

$$BIC = -\frac{k}{2}log(2\pi) + \frac{1}{2}log(|\tilde{\Sigma}|) - log(p(H)) - log(p(\tilde{\Theta}|H)) - log(p(D|H,\Theta)) \tag{2.2}$$

where $\tilde{\Sigma}$ is the negative of the Hessian matrix.

As an uninformative prior, $log(p(\tilde{\Theta}|H))$ can be neglected and we will derive the expression for other terms. According to standard BIC cost, the cost for each parameter is $\frac{1}{2}log(samplesize)$ [9], thus:

$$log(|\tilde{\Sigma}|) = dlength(M-1)log(L_D) + \sum_{j=1}^{M}\sum_{n=1}^{N} u_{jn} log(L_D) \tag{2.3}$$

This term describes the parameter penalties by transiting from a random description of the parameter $\Theta(\alpha_{jd}, \beta_{jn})$ to treating them as a deterministic unknowns. $log(p(H))$ is the prior probability of the model structure, the u switches $H(u)$. We define the configuration of u switches by figuring out the probability that a given $H(u)$ can be generated. Here we use a simple and

efficient way to code the u switches set [10]. We define all u switches for each unique word as a subset $\{u_{jn}, j = 1, ..., M\}$ and name it as $\mathbf{u_n}$. Then we identify each $\mathbf{u_n}$ by one of the three types:

(1) word n occurs with the shared probability in all topics;

(2) word n occurs with topic specific probabilities in all topics;

(3) word n occurs with the shared probability in some topics and topic specific probabilities in other topics.

To be mathematically precise, we have three functions defined as below:

$$F_1(\mathbf{u_n}) = \begin{cases} 1 & \sum_{j=1}^{M} u_{jn} = 0 \\ 0 & otherwise \end{cases} \tag{2.4}$$

$$F_2(\mathbf{u_n}) = \begin{cases} 1 & \sum_{j=1}^{M} u_{jn} = M \\ 0 & otherwise \end{cases} \tag{2.5}$$

$$F_3(\mathbf{u_n}) = \begin{cases} 1 & 0 < \sum_{j=1}^{M} u_{jn} < M \\ 0 & otherwise \end{cases} \tag{2.6}$$

Assume the numbers of possible active switch distributions for three cases are d_1, d_2 and d_3, then we have the expression of $p(\mathbf{H})$:

$$p(\mathbf{H}) = \prod_{n=1}^{N} ((\frac{1}{d_1})^{F_1(\mathbf{u_n})} (\frac{1}{d_2})^{F_2(\mathbf{u_n})} (\frac{1}{d_3})^{F_3(\mathbf{u_n})}) \tag{2.7}$$

so

$$-log(p(\mathbf{H})) = -\sum_{n=1}^{N} (F_1(\mathbf{u_n}) log(\frac{1}{d_1}) + F_2(\mathbf{u_n}) log(\frac{1}{d_2}) + F_3(\mathbf{u_n}) log(\frac{1}{d_3})) \tag{2.8}$$

For case (1), the $d_1 = 1$ because there is only one possible distribution for u_n (all zeros). Case (2) is similar to case (1) so $d_2 = 1$ also. For case (3), the number of different $\mathbf{u_n}$ distributions is $2^M - 2$. Thus the overall BIC cost expression is:

$$BIC = dlength(M-1)log(\frac{L_D}{2\pi}) + \sum_{j=1}^{M}\sum_{n=1}^{N} u_{jn} log(\frac{L_D}{2\pi}) - \sum_{n=1}^{N}(F_3(\mathbf{u_n}))log(\frac{1}{2^M - 2}) - loglikelihood \tag{2.9}$$

where

$$loglikelihood = \sum_{d=1}^{dlength}\sum_{n=1}^{N} D(d,n) log(\sum_{j=1}^{M} \alpha_j \beta_{jn}^{u_{jn}} \beta_{0n}^{1-u_{jn}}) \tag{2.10}$$

Chapter 3

Learning Algorithm

In this section we will talk about the learning algorithm of this thesis. The goal of the learning is to find the optimal parameter set $\{\beta_{jn}, j = 1...M, n = 1...N\}$ and $\{\alpha_{jd}, j = 1...M, d = 1...M\}$ and the model structure $\{u_{jn}, j = 1...M, n = 1...N\}$ at a range of model orders. Keep in mind that they have interactions because of the constraints:

$$\sum_{n=1}^{N} \beta_{jn}^{u_{jn}} \beta_{0n}^{1-u_{jn}} = 1, j = 1...M \tag{3.1}$$

and

$$\sum_{j=1}^{M} \alpha_{jd} = 1, d = 1...dlength \tag{3.2}$$

Optimizing the parameter set and switch set to get the global minimum value of our goal function is impossible since there are too many parameters with interactions. The basic strategy of our learning is performing a generalized EM algorithm [10] [11] [12] to separately update those parameters and switches by trying to minimize our goal function and get the local minimum. We will preprocess the documents first and get a matrix which describe all of the documents and then we can begin the learning. The general learning procedure includes four main steps. In the first

step we will initialize the parameters and model structure at a starting model order. Then we perform an E-step in which we calculate the expected hidden variables and then we move to an M-step in which we hold the switches unchanged and update the value for $\{\beta_{jn}, j = 1...M, n = 1...N\}$ and $\{\alpha_{jd}, j = 1...M, d = 1...M\}$ and then we hold the parameters unchanged and update the switches. We iteratively perform the E-step and M-step until a convergence condition is reached or we reach the upper bound of the iteration times. Here we get the optimal value for the parameters and switches at the current model order. The fourth step is the model order reduction. In this step we remove several topics and go to the E-step with a new model order. This algorithm will be iteratively performed until the model order reaches the lower bound value we set.

The document preprocessing step including getting the unique words of all documents, performing the symbol removal, standard stopword removal, stemming, coding the documents into a single matrix and removing the unique words that occur too few times. After the preprocessing step we will get a matrix describing the document set which can be understood with an example. Considering we have three documents each containing a single sentence and they are respectively: "Parsimonious topic models", "Topic modeling", "A parsimonious topic model", the unique word set will be "model, parsimonious, topic" and we will get the following matrix with each row representing a sentence.

$$D = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The shared occurring probabilities of all unique words are also determined in the document preprocessing step:

$$\beta_{0n} = \frac{\sum_{d=1}^{dlength} D(d,n)}{L_D} \tag{3.3}$$

3.1 Initialization

There is no need for the initialization to be that reasonable because we will begin with a very high model order. This means we have a lot of space to adjust the parameters. But a bad initialization will also seriously affect our result because our algorithm does not seek the global minimum point of the goal function but tries to reach the local minimum point. So each experimental data will be processed five times with different initial conditions. The initialization procedure is as follows:

Algorithm 4 Initialization Algorithm [1]

1: **for** each topic **do**
Step 1. randomly choose 5 documents.
Step 2. Collect the unique words occurring in these document and count the frequencies.
Step 3. Set the unique words as topic specific words and set their frequencies as the topic specific frequencies.
2: **end for**
3: **for** each document and each topic **do**
Step 4. Calculate the loglikelihood for each document versus each topic
Step 5. Assign each document to the topic with the largest loglikelihood.
4: **end for**
5: **for** each topic **do**
Step 6. For each topic, collect the unique words occurring in the documents assigned and count the frequencies.
Step 7. Set the unique words as topic specific words and set their frequencies as the topic specific frequencies and fill the shared distribution slots with the global shared probabilities.
6: **end for**

3.2 E-step

We define $Z_{dn} \in \{1, 2, 3...M\}$ as the origin of the word n in document d and this will be the hidden variable in our learning algorithm [11]. With the the model parameters $\{\beta_{jn}, j = 1...M, n = 1...N\}$ and $\{\alpha_{jd}, j = 1...M, d = 1...M\}$ and the model structure $\{u_{jn}, j = 1...M, n = 1...N\}$ fixed, we calculate the expected hidden variable set $Hv(j, d, n)$ (Hv will be a three-dimensional matrix) by calculating the probability that a word n in document d is generated by topic j. The expression of the expected hidden variable Hv is

$$Hv(j, d, d) = P(Z_{dn} = j|\Theta, \mathbf{H}) = \frac{\alpha_{jd}B_{jn}}{\sum_{l=1}^{M} \alpha_{ld}B_{ln}} \tag{3.4}$$

3.3 M-step

Using the expected hidden variables calculated, we can derive the expected completed form of the likelihood expression. The theory of EM algorithm framework guarantees that if we make a change to the parameters or switches and this change causes a decreasing on the expected complete form of the likelihood, this change is non-increasing in the incomplete likelihood [13], [12], [14], which is our original likelihood function. The expected complete form of the likelihood expressed with the expected hidden variables is:

$$E[log(p(\mathbf{D}, \mathbf{Z}|\mathbf{\Theta}, \mathbf{H}))] = \sum_{d=1}^{dlength} \sum_{n=1}^{N} \sum_{j=1}^{M} [Hv(j, d, n)(log(\alpha_{jd}) + log(B_{jn})] \tag{3.5}$$

To maximize the expected complete form of the loglikelihood value, because of the constraint mentioned at the very begin of this section we need to build the Lagrangian equation [15]:

$$E[log(p(\mathbf{D}, \mathbf{Z}|\mathbf{\Theta}, \mathbf{H}))] = E\left[CompletedLoglikelihood\right] - L1 - L2 \tag{3.6}$$

where

$$L1 = \sum_{d=1}^{dlength} \lambda_d (\sum_{j=1}^{M} \alpha_{jd} - 1) \tag{3.7}$$

$$L2 = \sum_{j=1}^{M} \mu_j (\sum_{n=1}^{N} (B_{jn}) - 1) \tag{3.8}$$

Setting the partial derivative to be zero we can get the optimal value of α:

$$\alpha_{jd} = \frac{\sum_{n=1}^{N} D(d, n) Hv(j, d, n)}{\sum_{l=1}^{M} \sum_{n=1}^{N} D(d, n) Hv(l, d, n)} \tag{3.9}$$

And the optimization of the topic specific probabilities can be calculated as:

$$x_{jn} = \sum_{d=1}^{dlength} D(d, n) Hv(j, d, n) \tag{3.10}$$

$$\overline{x}_j = \sum_{n=1}^{N} x_{jn} u_{jn} \tag{3.11}$$

$$\mu_j = \frac{\overline{x}_j}{1 - \sum_{n=1}^{N} (1 - u_{jn}) \beta_{0n}} \tag{3.12}$$

And:

$$\beta_{jn} = \frac{x_{jn} u_{jn}}{\mu_j} \tag{3.13}$$

The expression of β_{jn} is calculated here because the expression will be used. Actually in the switches update part we will not use the value of β_{jn} here and it will be updated. So there is no need to calculate it and record its value in M-step.

3.3.1 u_{jn} update

In PTM the u switches are updated by trial-flipping switch for each $\{n = 1, 2, 3...N\}$ for each $\{j = 1, 2, 3...M\}$ one by one. This means, we begin with fixing the value of j as 1, and visit the switch at each value of n, change the value of the switch and calculate the change of the total BIC cost. If the BIC cost decreases, the change will be accepted and not accepted if otherwise. Then we move forward to the next n and this visit is performed cyclically until the switch set $\{u_{1n}, n = 1, 2, 3, ...N\}$ is never changed. Then we repeat the above procedure with $j = 2$ and circularly visit the switch set $\{u_{2n}, n = 1, 2, 3, ...N\}$ until they converge. The update of the switch set $\{u_{jn}, n = 1, 2, 3, ...N\}$ is also repeated circularly until the whole u switch set converges. We can call this update procedure a "circulars in circulars" update and this procedure is timely expensive. Here we derive a revised update method. From the expressions in the parameter estimate section we can know that the values of x_{jn} are independent of the switches and one change on a switch $u_{j'n'}$ will change u'_j and then this will only affect the switches $\{u_{j'n}, n = 1, 2, 3...N\}$. Thus for each value of n', we just need to update all the switches $\{u_{jn'}, j = 1, 2, 3...M\}$ once to reach the convergence. Also we need to cyclically update the switches set for $\{n, n = 1, 2, 3...N\}$. The update algorithm is expressed as following:

Algorithm 5 u update Algorithm [10] [1]

1: **for** each n **do**

2: **for** each j **do**

Step 1. If $u_{j'n'}$ is the only "on" switch among $\{u_{jn'}, j = 1, 2, 3, ...M\}$, set the change on total BIC cost to infinite and skip to **Step 4.**

Step 2. calculate ΔC (complexity cost change) and ΔL (loglikelihood cost change) if we switch the value of $u_{j'n'}$ [1], where

$$\Delta C = \frac{1}{2}(-1)^{u^-_{j'n'}} log(\frac{L_D}{2\pi}) \tag{3.14}$$

$$\Delta L = (-1)^{u^-_{j'n'}} x_{j'n'} log(\frac{x_{j'n'}}{\mu^+_{j'}\beta_{0n'}}) - \overline{x}^-_{j'} log(\frac{\mu^+_{j'}}{\mu^-_{j'}}) \tag{3.15}$$

where

$$\mu^+_{j'} = \frac{\overline{x}^-_{j'} + (-1)^{u^-_{j'n'}} x_{j'n'}}{1 - \sum_{n=1}^{N}(1 - u^-_{j'n})\beta_{0n} + (-1)^{u^-_{j'n'}}\beta_{0n'}} \tag{3.16}$$

Step 3. Calculate the change on total BIC cost ΔBIC where

$$\Delta BIC = \Delta C - \Delta L \tag{3.17}$$

Step 4. Denote the calculated BIC cost change as a entry $E(1, j')$ in the M-element vector ΔE. If $\Delta BIC < 0$, change the value of switch $u_{j'n'}$; Otherwise, keep the switch unchanged.

3: **end for**

Step 5. Calculate three cases cost change, respectively $\Delta C1$, $\Delta C2$ and $\Delta C3$ as following:

$$C1 = |E| u\,[:, n'] \tag{3.18}$$

$$C2 = |E|(1 - u\,[:, n']) \tag{3.19}$$

$$C3 = -log(\frac{1}{2^M - 2}) \tag{3.20}$$

Step 6. Compare $C1$, $C2$ and $C3$. If $C1$ is minimum, change all switches in $\{u_{jn'}, j = 1, 2, 3, ...M\}$ to zeros. If $C2$ is minimum, change all switches in the set above to ones. Otherwise, keep the switches set unchanged.

4: **end for**

Algorithm 6 u update Algorithm (cont')

Step 7. Update $B_{jn} = \beta_{jn}^{u_{jn}} \beta_{0n}^{1-u_{jn}}$. Update μ_j.

Step 8. Repeat the above algorithm until the u switches set converges or a maximum number of iteration is reached.

3.4 Model Order Reduction

We use a reasonable and efficient method to do the model order reduction. The initialization procedure will only be performed once at the very beginning of the learning process. For each model order, once the EM steps are alternately applied until the total BIC cost reaches a local minimum value or the iteration number reaches the upper bound we set, we calculate the sum of $\{\alpha_{jd}, d = 1, 2, 3...dlength\}$ at each j then sort the sums, find the topics with least aggregate masses and remove them [16], [10] i.e., delete rows respectively in α matrix, β matrix and u matrix. After deleting rows from α matrix, the sum of the entries in a column will no longer be one so we need to multiply each column with a multiplier to maintain the constraint.

Chapter 4

Numerical Results

In this section we present the results on Ohsumed data set using Matlab. The machine we use has a Core i5 3.2 GHz processor. For each result we will report the model order at minimum BIC we got, the run time for the learning process from the highest model order to the lowest and the sparsity of our model.

4.1 Ohsumed Corpus

Ohsumed Corpus data set contains 34389 documents. Each document is pre-labled with one or several topics. In our learning we just pick the first 1000 documents. After the applied preprocessing including standard stopword removal, stemming and removing the words occurring for less than 10 times in the whole document set, the documents set has 1689 unique words.

In our learning we start with the model order $M = 80$. After each time the parameters and switches converges, we remove two topics with the least masses. The BIC curve versus model order is shown below.

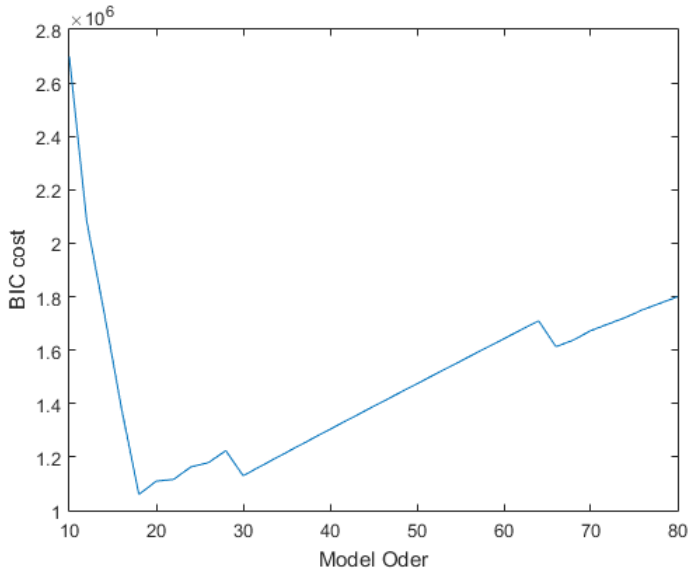

Figure 4.1: BIC cost at different model orders

The sparsity of our model at the lowest, run time and model order with the minimum BIC cost is shown below.

	$dlength$	N	N_{unique}	$\overline{N}$	$T(minute)$	$M_{minimum}$
Our Model	1000	1689	82.6(1.82)	4.59(0.1)	68.1(0.18)	18(0)
PTM	24128	12072	11182(11.96)	863.04(3.2)	2529(97)	105(5.98)

Table 4.1: Results in our model and PTM [1]. N_{unique} is the average number of total unique topic specific words in all topics; $\overline{N}$ is the average number of topic specific words per topic

In this thesis we use Matlab to implement our algorithm. Since Matlab has a low efficiency when running a program containing nested loops, the v switches updating part in our algorithm is very heavy. So in this thesis we only present the results for a small document set size (1000) while in Hossein Soleimani and David J. Miller's work [1] 24128 documents in Ohsumed data are used as the training data set. The results of PTM listed above are from Hossein Soleimani and David J. Miller's work. Directly comparing the results got from these two models is meaningless because of the difference in sample sizes. We will only try to analyze some possible features our results have. The results show that our model has a very large sparsity. Our model's large sparsity is maybe caused by two reasons. The first one is the application of the "three cases method". This method gives an extra cost if the switches for one unique word have different values for all the topics. By

applying this method, the algorithm tends to force a specific switch set $\{u_{jn'}, j = 1, 2, 3, ...M\}$ with few "on" switches to be all "off". Another reason why our model has a large sparsity is we allow every topic to be present in every document. This may significantly increase the sparsity. The model order our learning algorithm gets (18) at the minimum BIC cost point is very closer to the actual value, which is 23. Also the standard deviation value 0 shows that our model has a very stable performance on the model order selection. To further test our model, more work is needed. We need to increase the document set size and make it the same as the size Soleimani and Miller use for PTM. We also need to test our model's fitness as described in [17] [18]. To do this test, we need to divide each test document into two parts: an observed part and a held-out test part and keep no words exist in both parts. Then we will use the observed part to calculate the expected topic proportions (α_{jd} set) and calculate the log-likelihood of words in the held-out part [1].

Chapter 5

Conclusion

In this thesis we develop a model which is different from LDA and PTM. We derive the BIC cost expression and develop the learning algorithm using the framework of EM algorithm theory for our model. Finally we apply our model to a real data set–Ohsumed corpus data set to test our model and analyze the results. Compared to LDA, our model allows many words' occurring probabilities to be shared. This is reasonable because many words are not context-specific and they may be roughly used with almost the same frequency in different topics. Compared to PTM, our model allows every topic to be present in every document. The main reason why we do this change is, this will obviously increase the run speed of the learning algorithm. PTM controls whether a topic j is present in a document d or not by applying a switch set $\{v_{jd}, j = 1, 2, 3...M, d = 1, 2, 3...dlength\}$ and the optimization of this switch set is the most time expensive part in the whole learning algorithm. Another reason is PTM may have a potential weakness in the low order learning. The results got from the PTM show that the in final model we get, the documents have very few average active topics, always less than 2 [1].

By contrast, removing this switch set brings our model a potential weakness at high orders learning. At high orders it is unreasonable to set every topic a non-zero proportion for every topic since each document is expected to have a main theme covered by a modest subset or related topics. Solving this potential weakness will be an improvement to the performance of our model.

Bibliography

[1] Soleimani, H. and D. J. Miller (2015) "Parsimonious Topic Models with Salient Word Discovery," *Knowledge and Data Engineering, IEEE Transactions on*, **27**(3), pp. 824–837.

[2] Chen, S. and P. Gopalakrishnan (1998) "Speaker, environment and channel change detection and clustering via the bayesian information criterion," in *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, vol. 8, Virginia, USA.

[3] Manning, C. D. and H. Schütze (1999) *Foundations of statistical natural language processing*, MIT press.

[4] Boutemedjet, S., N. Bouguila, and D. Ziou (2009) "A hybrid feature extraction selection approach for high-dimensional non-Gaussian data clustering," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **31**(8), pp. 1429–1443.

[5] Blei, D. M., A. Y. Ng, and M. I. Jordan (2003) "Latent dirichlet allocation," *the Journal of machine Learning research*, **3**, pp. 993–1022.

[6] Kass, R. E. and A. E. Raftery (1995) "Bayes factors," *Journal of the american statistical association*, **90**(430), pp. 773–795.

[7] Bishop, C. M. (2006) *Pattern recognition and machine learning*, springer.

[8] Ghosh, J. K., M. Delampady, and T. Samanta (2007) *An introduction to Bayesian analysis: theory and methods*, Springer Science & Business Media.

[9] Schwarz, G. et al. (1978) "Estimating the dimension of a model," *The annals of statistics*, **6**(2), pp. 461–464.

[10] Graham, M. W. and D. J. Miller (2006) "Unsupervised learning of parsimonious mixtures on large spaces with integrated feature and component selection," *Signal Processing, IEEE Transactions on*, **54**(4), pp. 1289–1303.

[11] Dempster, A. P., N. M. Laird, and D. B. Rubin (1977) "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38.

[12] Meng, X.-L. and D. Van Dyk (1997) "The EM algorithm–an old folk-song sung to a fast new tune," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 511–567.

[13] McLachlan, G. and D. Peel (2004) *Finite mixture models*, John Wiley & Sons.

[14] Moon, T. K. (1996) "The expectation-maximization algorithm," *Signal processing magazine, IEEE*, **13**(6), pp. 47–60.

[15] Fan, W., N. Bouguila, and D. Ziou (2013) "Unsupervised hybrid feature extraction selection for high-dimensional non-Gaussian data clustering with variational inference," *Knowledge and Data Engineering, IEEE Transactions on*, **25**(7), pp. 1670–1685.

[16] Figueiredo, M. A. and A. K. Jain (2002) "Unsupervised learning of finite mixture models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **24**(3), pp. 381–396.

[17] Hoffman, M. D., D. M. Blei, C. Wang, and J. Paisley (2013) "Stochastic variational inference," *The Journal of Machine Learning Research*, **14**(1), pp. 1303–1347.

[18] Teh, Y. W., K. Kurihara, and M. Welling (2007) "Collapsed variational inference for HDP," in *Advances in neural information processing systems*, pp. 1481–1488.

Appendix

Matlab Codes

```
clear

% %%%%% Document load process...
filepath = 'D:\PTM\ohsumed\ohsumed_test\'; %<------set the file path
dlength = 1000; % number of documents
[ N, D, beta_shared, beta_shared_d ] = DocProcess( filepath, dlength );
MaxOrder = 40;                              %<------set the maximum model order we ...
    begin with
LD = sum(D(:));
Llcurve = zeros(MaxOrder/2, 1);
BICcurve = zeros(MaxOrder/2, 1);
Discurve = zeros(MaxOrder/2, 1);
Sumofucurve = zeros(MaxOrder/2, 1);
%  N is the number of unique words
%  D is a matrix describing the the documents set with size (dlength * N)
%  beta_shared is a vector describing the shared probabilities of words
%  beta_shared_d is the diagonal matrix of ...
%%
%%%%% Init process...
Dinit = 10; % number of document picked for each topic
M = 80; % number of topics
```

```
[ Beta, Beta_topic, Beta_shared, u, Alpha ] = Init( D, M, Dinit, beta_shared_d );
% Beta is a matrix describing the occur probabilities of unique words with
% size (M * N)
% Beta_topic is a matrix describing the topic specific probability of words
% with size (M * N), for the splot of topic shared word, the value is 0
% Beta_shared is .....with size(M * N), for the ... topic specific ... 0
% Alpha is topic proportions of documents with size (M * dlength)
% u is the switches indicating whether a word is topic specific or shared with size ...
    (M * N)
%%
for c = 1 : MaxOrder
    M = 2 * (MaxOrder+1 - c);
    for iteration = 1 : 4
        u_old = u;
        %%%%% General EM algorithm part...
        % E-step
        [ Hv ] = HiddenVariables_mex( Alpha, Beta, M, N, dlength );% Calculate ...
            hidden variables with size (M * N * d)
        % M-step
        [ Alpha ] = AlphaCal_mex( D, M, N, Hv, dlength ); % Calculate Alpha with ...
            size (M * dlength)
        ur = ones(M, N) - u;
        [ x, xj, uj, ~ ] = BetaNxCalc_mex( Hv, D, M, N, dlength, u, ur, ...
            beta_shared, beta_shared_d ); % Calculate x and Beta both have size (M ...
            , N)
        [ u, Beta ] = UUpdateNew_mex( M, N, uj, xj, x, u, beta_shared, ...
            beta_shared_d, LD);
        if u == u_old
            break
        end
    end
    [ BIC, Loglikelihood, Dis, Sumofu ] = BICcalculate_mex( N, dlength, Alpha, ...
        Beta, D, M, Hv, u, LD ); % Calculate the Bayesian Information Criterion ...
        (BIC) which is a scalar
    Llcurve(c) = Loglikelihood;
    BICcurve(c) = BIC;
    Discurve(c) = Dis;
    Sumofucurve(c) = Sumofu;
    mass = sum(Alpha, 2);
```

```
    [ u, Beta, Alpha ] = TopicRemove( N, dlength, mass, u, Beta, Alpha );
end
plot(BICcurve)
```

```
function [ x, xj, uj, Beta ] = BetaNxCalc( Hv, D, M, N, dlength, u, ur, ...
    beta_shared, beta_shared_d )

x = zeros(M, N);
for j = 1 : M
    Hvj = reshape(Hv(j, :), N, dlength);
    HvjT = Hvj';
    x(j, :) = sum(D .* HvjT);
end

xj = sum(x .* u, 2);
urBeta0j = 1 - ur * beta_shared';
uj = xj ./ urBeta0j;

Uj = repmat(uj, 1, N);
Beta_topic = x .* u ./ Uj;

Beta = Beta_topic + ur * beta_shared_d;
```

```
function [ BIC, Loglikelihood, dis, Sumofu ] = BICcalculate( N, dlength, Alpha, ...
    Beta, D, M, Hv, u, LD )

dis = 0;
for n = 1:N
    if sum(u(:,n)) < M && sum(u(:,n)) > 0
```

```
        dis = dis +1;
    end
end

Loglikelihood = 0;
for d = 1 : dlength
    for n = 1 : N
        for j = 1 : M
            Loglikelihood = Loglikelihood + D(d, n) * Hv(j, (d - 1) * N + n) * ...
                (log(Alpha(j, d)) + log(Beta(j, n)));
        end
    end
end

Sumofu = sum(u(:))
CCost =  dis * M * log(2) + 0.5 * M * sum(log(sum(D,2) / (2 * pi))) + 0.5 * Sumofu ...
    * log(LD / (2 * pi)) * dlength;
BIC  = CCost - Loglikelihood;
end
```

```
function [ N, D, beta_shared, beta_shared_d ] = DocProcess( filepath, dlength  )

filepath = 'D:\PTM\ohsumed\ohsumed_test\';%set filepath
filenamels = ls(filepath);%get the list of all files
filenamels(1:2,:) = [];%remove the first two rows (.&..)
dlength_total = size(filenamels,1)%get the total number of files
UniqueWords = {};%unique words list of a single file
TotalUniqueWords = {};%unique words list of whole file set
UniqueWordsNum = 0;%number of unique words of a single file
TotalUniqueWordsNum = 0;%number of unique words of whole file set

for d = 1:dlength%load the documents and collect the unique words
    f = fopen(strcat(filepath,filenamels(d,:)),'r');
    Words = textscan(f,'%s','delimiter',',. ','MultipleDelimsAsOne',1);
```

```
    fclose(f);
    UniqueWords = unique(lower(Words{1,1}));%get the unique words list of file with ...
        index d
    UniqueWordsNum = size(UniqueWords,1);%get number of unique words of file with ...
        index d
    TotalUniqueWords(TotalUniqueWordsNum + 1:TotalUniqueWordsNum+UniqueWordsNum,1) ...
        = UniqueWords;%add the unique words list of file d to the total unique ...
        words list
    TotalUniqueWords = unique(TotalUniqueWords);%get the unique words of updated ...
        total unique words list
    TotalUniqueWordsNum = size(TotalUniqueWords,1);%get the updated number of ...
        unique words of whole file set
end

UniqueWordsR1 = {};% remove all rows which contain non-letter elements in the ...
    unique words of whole file set
index = 1;
for n = 1:TotalUniqueWordsNum,
    ifletter = isletter(char(TotalUniqueWords(n,:)));
    if ~any(ifletter == 0)
        UniqueWordsR1(index,:) = TotalUniqueWords(n,:);
        index = index + 1;
    end
end
N = length(UniqueWordsR1);

UniqueWordsR2 = {};% remove the standard stopwords from Unique_words_r...updated ...
    list save as Unique_words_R
index = 1;
f = fopen('D:\PTM\stopwords.txt','r');% load the standard stopwords list as "stopwords"
stopwords = textscan(f,'%s','delimiter',',. ','MultipleDelimsAsOne',1);
fclose(f);
stopwords = unique(lower(stopwords{1,1}));
for n = 1:N
    if ~any(ismember(stopwords,UniqueWordsR1(n,:)) == 1)
        UniqueWordsR2(index,:) = UniqueWordsR1(n,:);
        index = index + 1;
    end
end
```

```
N = length(UniqueWordsR2)

D = zeros(dlength,N); %% LOAD THE DOCUMENTS %%
 for d = 1:dlength
     f = fopen(strcat(filepath,filenamels(d,:)),'r');
     Words = textscan(f,'%s','delimiter',',. ','MultipleDelimsAsOne',1);
     fclose(f);
     Words = lower(Words{1,1});
     for l=1:length(Words)
         D(d,:) = D(d,:) + (ismember(UniqueWordsR2,Words(l,:)))';
     end
 end

for n=1:N    %%%%% STEMMING %%%%%
    if n > size(UniqueWordsR2)
        break
    end
    mainword = UniqueWordsR2(n);
    wordgenerator = cell(38,1);              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%   ...
        UNDETERMINED   %%%%%%%%%%%%%%%%%%%%%%%%
    wordgenerator(1,:) = mainword;
    wordgenerator(2,:) = {strcat(char(mainword),'s')};
    wordgenerator(3,:) = {strcat(char(mainword),'es')};
    wordgenerator(4,:) = {strcat(char(mainword),'d')};
    wordgenerator(5,:) = {strcat(char(mainword),'ed')};
    wordgenerator(6,:) = {strcat(char(mainword),'y')};
    wordgenerator(7,:) = {strcat(char(mainword),'ful')};
    wordgenerator(8,:) = {strcat(char(mainword),'fully')};
    wordgenerator(9,:) = {strcat(char(mainword),'ly')};
    wordgenerator(10,:) = {strcat(char(mainword),'ally')};
    wordgenerator(11,:) = {strcat(char(mainword),'ness')};
    wordgenerator(12,:) = {strcat(char(mainword),'nesses')};
    wordgenerator(13,:) = {strcat(char(mainword),'ing')};
    wordgenerator(14,:) = {strcat(char(mainword),'atic')};
    wordgenerator(15,:) = {strcat(char(mainword),'atical')};
    wordgenerator(16,:) = {strcat(char(mainword),'atically')};
    wordgenerator(17,:) = {strcat(char(mainword),'tion')};
    wordgenerator(18,:) = {strcat(char(mainword),'tions')};
    wordgenerator(19,:) = {strcat(char(mainword),'tional')};
```

```
    wordgenerator(20,:) = {strcat(char(mainword),'tionally')};
    wordgenerator(21,:) = {strcat(char(mainword),'er')};
    wordgenerator(22,:) = {strcat(char(mainword),'ers')};
    wordgenerator(23,:) = {strcat(char(mainword),'or')};
    wordgenerator(24,:) = {strcat(char(mainword),'ors')};
    wordgenerator(25,:) = {strcat(char(mainword),'al')};
    wordgenerator(26,:) = {strcat(char(mainword),'ally')};
    wordgenerator(27,:) = {strcat(char(mainword),'ality')};
    wordgenerator(28,:) = {strcat(char(mainword),'alities')};
    wordgenerator(29,:) = {strcat(char(mainword),'ity')};
    wordgenerator(30,:) = {strcat(char(mainword),'ities')};
    wordgenerator(31,:) = {strcat(char(mainword),'ion')};
    wordgenerator(32,:) = {strcat(char(mainword),'ions')};
    wordgenerator(33,:) = {strcat(char(mainword),'ional')};
    wordgenerator(34,:) = {strcat(char(mainword),'ionally')};
    wordgenerator(35,:) = {strcat(char(mainword),'sion')};
    wordgenerator(36,:) = {strcat(char(mainword),'sions')};
    wordgenerator(37,:) = {strcat(char(mainword),'sional')};
    wordgenerator(38,:) = {strcat(char(mainword),'sionally')};

    ifcombine = ismember(UniqueWordsR2(n:length(UniqueWordsR2)),wordgenerator);
    combinelist = find(ifcombine == 1);
    combinelist = combinelist + n - 1; %convert the indexes of words in infcombine ...
        to indexes in Unique_words_R
    D(:,n) = D(:,n:length(UniqueWordsR2)) * ifcombine;
    UniqueWordsR2(combinelist(2:length(combinelist)),:) = [];
    D(:,combinelist(2:length(combinelist))) = [];
end

N = length(UniqueWordsR2)
index = find(sum(D)<10);
UniqueWordsR2(index,:) = [];
D(:,index) = [];
N = length(UniqueWordsR2)

beta_shared = sum(D)/sum(D(:));
beta_shared_d = diag(beta_shared);

end
```

```
function [ Hv ] = HiddenVariables( Alpha, Beta, M, N, dlength )

Hv = zeros(M, N * dlength);
for d = 1:dlength
alpha = Alpha(:,d);
b = alpha' * Beta;
b = 1 ./ b;
alpha_b = alpha * b;
StartIndex = (d - 1) * N + 1;
EndIndex = d * N;
Hv(:,StartIndex : EndIndex) = alpha_b .* Beta;
end
```

```
function [ Beta, Beta_topic, Beta_shared, u, Alpha ] = Init( D, M, Dinit, ...
    beta_shared_d )

DinitT = Dinit * M;                    %total number of documents picked for init
[dlength,N] = size(D);                      %number of documents and unique words
u = zeros(M,N);                      %switches indicating whether topic s or shared
Beta_topic = zeros(M,N);             %topic specific word probabilities
Sequence = randperm(dlength);
Sequence = Sequence(1,1:(DinitT));
D4init = reshape(Sequence,M,Dinit);
for j = 1:M
    for dinit = 1:Dinit
    u(j,:) = u(j,:) + D(D4init(j,dinit),:);
    end
    total = sum(u(j,:));
    Beta_topic(j,:) = u(j,:) / total;
```

```
end
u(u ~= 0) = 1;
ur = ones(M,N) - u;
Beta_shared = ur * beta_shared_d;
sum_of_shared = sum(Beta_shared,2);
for j = 1:M
    Beta_topic(j,:) = Beta_topic(j,:) * (1 - sum_of_shared(j,1));
end
Beta = Beta_topic + Beta_shared;     %Beta first time set


u = zeros(M,N);                     %Reset the switches
dTotal = zeros(DinitT,N);           %(DinitT,N) matrix describing all docs picked
Beta_topic = zeros(M,N);
for dinit = 1:DinitT
    dTotal(dinit,:) = D(Sequence(1,dinit),:);
end
L = dTotal * log(Beta');
for dinit = 1:DinitT
    [~,topic] = max(L(dinit,:));
    u(topic,:) = u(topic,:) + dTotal(dinit,:);
end                                 %here we got a new (M*N) matrix u
for j = 1:M
    total = sum(u(j,:));
    if total ~=0
        Beta_topic(j,:) = u(j,:) / total;%recalculate the top specific proba matrix
    else
        RandomSpecific = randi([1,N]);
        u(j,RandomSpecific) = 1;
        Beta_topic(j,RandomSpecific) = 1;
    end
end
u(u ~= 0) = 1;                          %u initialized
ur = 1 - u;                     %ur initialized
Beta_shared = ur * beta_shared_d;
sum_of_shared = sum(Beta_shared,2);
for j=1:M
    Beta_topic(j,:) = Beta_topic(j,:) * (1 - sum_of_shared(j,1));
```

```
end
Beta = Beta_topic + Beta_shared;    %Beta initialized

Alpha = ones(M, dlength);
Alpha = Alpha/M;
end
```

```
function [ u, Beta, Alpha ] = TopicRemove( N, dlength, mass, u, Beta, Alpha )

for least = 1 : 2
        index = find(mass == min(mass));
        u(index,:) = [];
        Beta(index,:) = [];
        Alpha(index,:) = [];
        mass(index,:) = [];
    end
    for n = 1 : N
        Beta(:, n) = Beta(:, n) / sum(Beta(:, n));
    end
    for d = 1 : dlength
        Alpha(:, d) = Alpha(:, d) / sum(Alpha(:, d));
    end
end
```

```
function [ u, Beta ] = UUpdateNew( M, N, uj, xj, x, u, beta_shared, beta_shared_d, ...
    LD )

Beta = zeros(M, N);
for uiteration = 1:3
```

```
u_old = u;
for n=1:N
    delta = zeros(M,1);
    deltacc = zeros(M,1);
    deltaL = zeros(M,1);

    xj_new = zeros(M,1);
    xj0 = zeros(M,1);
    xj1 = zeros(M,1);
    uj_new = zeros(M,1);
    uj0 = zeros(M,1);
    uj1 = zeros(M,1);
    uc1 = zeros(M,1);
    uc2 = ones(M,1);

    for j = 1:M
        if sum(u(j,:),2) - u(j,n) == 0
            uj1(j,1) = uj(j,1);
            xj1(j,1) = xj(j,1);
            delta(j,1) = inf;
        else
            xj_new(j,1) = xj(j,1) + ((-1) ^ u(j,n)) * x(j,n);
            uj_new(j,1) = xj_new(j,1) / (1 - (1 - u(j,:)) * beta_shared' + ...
                ((-1) ^ u(j,n)) * beta_shared(1,n));

            if u(j,n) == 1
                uj0(j,1) = uj_new(j,1);
                uj1(j,1) = uj(j,1);
                xj0(j,1) = xj_new(j,1);
                xj1(j,1) = xj(j,1);
            else
                uj0(j,1) = uj(j,1);
                uj1(j,1) = uj_new(j,1);
                xj0(j,1) = xj(j,1);
                xj1(j,1) = xj_new(j,1);
            end
            deltacc(j,1) = (1 / 2) * ((- 1) ^ u(j,n)) * log((LD / 1000 * 3)/ (2 ...
                * pi));
            deltaL(j,1) = ((- 1) ^ u(j,n)) * x(j,n) * log(x(j,n) / (uj_new(j,1) ...
```

```
                    * beta_shared(1,n))) - xj(j,1) * log(uj_new(j,1) / uj(j,1));
                delta(j,1) = deltacc(j,1) - deltaL(j,1);
            end
            if delta(j,1) < 0
                u(j,n) = 1 - u(j,n);
            end
        end
        deltac1 = abs(delta') * u(:,n);
        deltac2 = abs(delta') * (1 - u(:,n));
        uc3 = u(:,n);
        deltac3 = - log(1 / (2 ^ M - 2));

        u_new = [uc1,uc2,uc3];

        [~,c] = min([deltac1, deltac2, deltac3]);

        u(:,n) = u_new(:,c);

        uj = u(:,n) .* uj1 + (1 - u(:,n)) .* uj0;
        xj = u(:,n) .* xj1 + (1 - u(:,n)) .* xj0;

        Uj=repmat(uj,1,N);
        Beta_topic = x .* u ./ Uj;

        Beta = Beta_topic + (1 - u) * beta_shared_d; %Beta recalculated;
    end
    ifchange = u - u_old;
    if sum(abs(ifchange)) == 0
        break
    end
end

end
```