Batch and Online Variational LDA for topic modeling on Gutenburg documents

Sahil Tyagi

December 12, 2020

Abstract

This project is part of the course B555-Machine Learning taught by Prof. Roni Khardon at Indiana University, Bloomington, Indiana, USA. The author of this report builds and implements batch and online variational Latent Dirichlet Allocation (LDA) from scratch as postulated in the paper *Online Learning for Latent Dirichlet Allocation* by *Hoffman et al.*. The deviation from the interim report is that the author was able to test and implement generative LDA with Gibbs sampler as well for the sake of batter comparison purposes. The data used for this project is all sets of gutenberg documents under the name of the popular english classic novel writer Jane Austen. The end results to fetch the topics for a given document is done via a web service launched as a serverless cloud function on the Google Cloud Platform (GCP). I would like to thank GCP for providing free credits as part of their university education program.

1 LDA generative model

To understand online variational LDA, we build on top of the LDA generative model that we studied in the course (and as part of our programming assignment 4). The solution is fondly called topic modeling, since we try to infer topics on a set of documents and each topic has an influence on a set of words (vocabulary of the corpus). The solution is a kind of dimensionality reduction where we break down the corpus of documents and their associated words into documents with the associated topics and the topics with their associated words as shown in Fig 1.

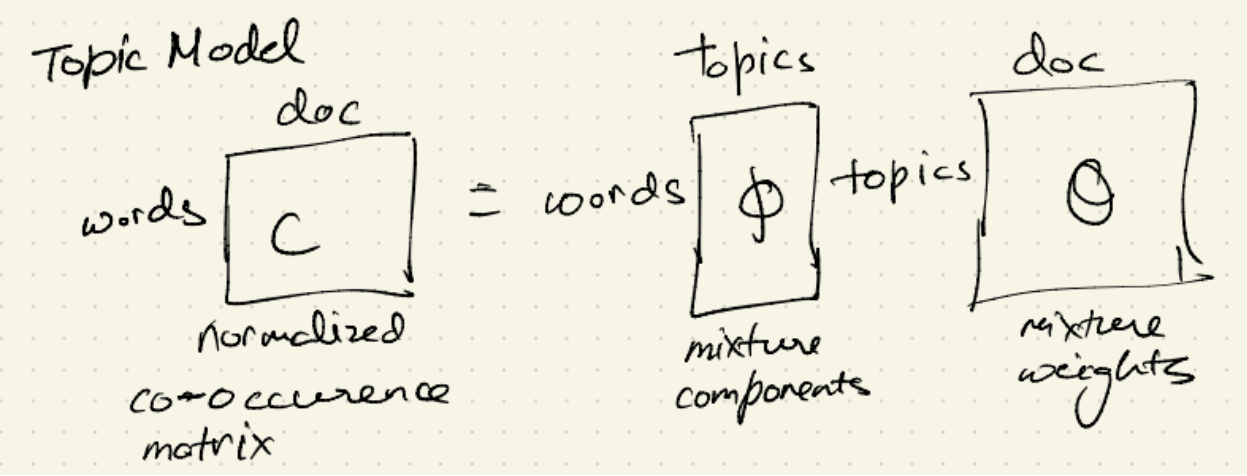

Figure 1: Corpus of documents breakdown

Here, the topic distribution over the documents is a dirichlet parameterized by α and the word distribution over the topics is also a dirichlet parameterized by β:

```
print(tw[0][1])
```

```
perhaps
RB
```

$\forall$ d draw $\theta_{d|k} \sim$ Dirichlet$(\theta_d|\alpha)$ and $\forall$ k draw $\phi i|k \sim$ Dirichlet$(\phi_d|\beta)$

Then, for every document d and every location j in d:

$$draw \quad z^{j,d} \sim \theta_{k|d} \ \ and \ \ draw \quad w^{j,d} \sim \phi k|d = z^{j,d}$$

The illustration of the generative LDA model from lectures is shown in Fig 2.

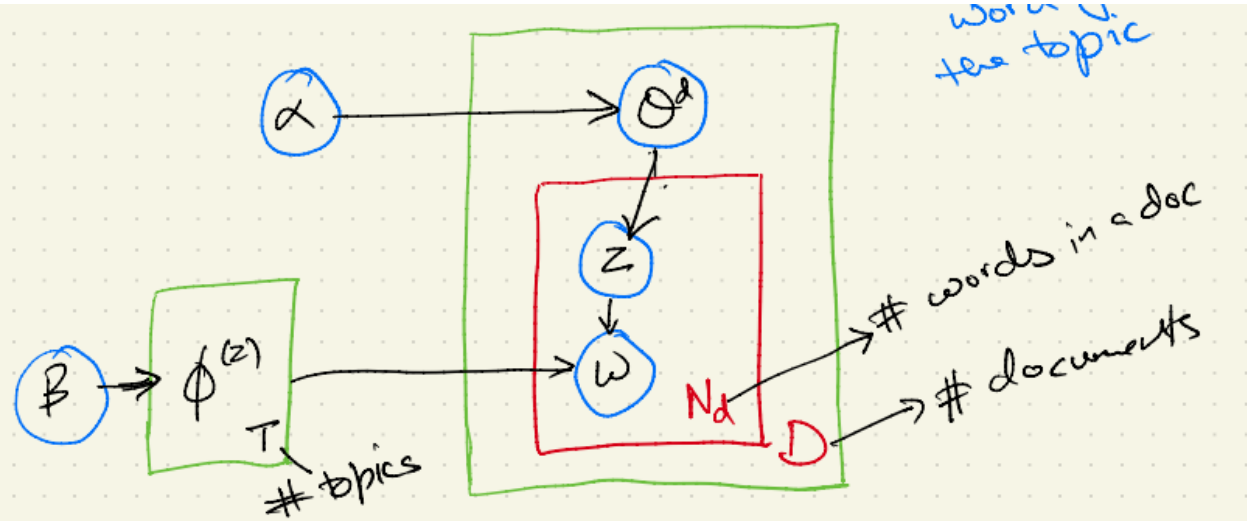

Figure 2: LDA illustration from B555 lectures

Without going over the derivations and calculations, the probability of topics over the documents is given in Fig 3.

$$P(k) = \frac{C_t(k,word)+\beta}{V\beta+\sum_j C_t(k,j)} \frac{C_d(doc,k)+\alpha}{K\alpha+\sum_l C_d(doc,l)}$$

Figure 3: Key compute metric in LDA taken from Programming Problem 4

2 Online Variational LDA

As suggested by Prof. Khardon, I looked into online variational algorithms since the size of the corpus was enormous. To understand how variational LDA works, I referred to *Online Learning for Latent Dirichlet Allocation* by *Hoffman et al.*. Online variational LDA is an extension of batch variational LDA based on online stochastic optimization with a natural gradient step. The advantage of online variational LDA is that it has much better computational efficiency while still having the same (or even better) statistical efficiency as batch variational LDA.

In variational LDA, we try to maximize the Expectation Lower Bound (ELBO) in order to minimize the KL divergence between the distribution $q(z, \theta, \beta)$ and the true posterior $p(z, \theta, \beta|w, \alpha, \eta)$.

Hoffman et al. derive the expectations of topic distribution $log\theta_{dk}$ and word distribution $log\beta_{kw}$ as show. below:

$$E_q[log\theta_{dk}] = \Psi(\gamma_{dk}) - \Psi(\Sigma^K \gamma_{di}) \quad and \quad E_q[log\beta kw] = \Psi(\lambda_{kw}) - \Psi(\Sigma^K \lambda_{ki})$$

where Ψ is the digamma function (the first derivative of the logarithmic of the gamma function), and γ and λ functions are given as:

$$\gamma_{dk} = \alpha + \Sigma_w n_{dw} \phi_{dwk} \quad and \quad \lambda_{kw} = \eta + \Sigma_d n_{dw} \phi_{dwk}$$

The variational parameter ϕ is given as:

$$\phi_{dwk} \propto exp[E_q[log\theta_{dk}] + E_q[log\beta kw]]$$

From the Hoffman paper, the pseudo algorithm for batch variational LDA is shown in Fig 4.

Algorithm 1 Batch variational Bayes for LDA

Initialize $\boldsymbol{\lambda}$ randomly.
while relative improvement in $\mathcal{L}(\boldsymbol{w}, \boldsymbol{\phi}, \boldsymbol{\gamma}, \boldsymbol{\lambda}) > 0.00001$ **do**
  E step:
  for $d = 1$ to D **do**
    Initialize $\gamma_{dk} = 1$. (The constant 1 is arbitrary.)
    repeat
      Set $\phi_{dwk} \propto \exp\{\mathbb{E}_q[\log \theta_{dk}] + \mathbb{E}_q[\log \beta_{kw}]\}$
      Set $\gamma_{dk} = \alpha + \sum_w \phi_{dwk} n_{dw}$
    until $\frac{1}{K} \sum_k |\text{change in} \gamma_{dk}| < 0.00001$
  end for
  M step:
  Set $\lambda_{kw} = \eta + \sum_d n_{dw} \phi_{dwk}$
end while

Figure 4: Batch Variational LDA from Hoffman paper

From the same paper, the algorithm for online variational LDA is shown in Fig 5.

Algorithm 2 Online variational Bayes for LDA

Define $\rho_t \triangleq (\tau_0 + t)^{-\kappa}$
Initialize $\boldsymbol{\lambda}$ randomly.
for $t = 0$ to ∞ **do**
  E step:
  Initialize $\gamma_{tk} = 1$. (The constant 1 is arbitrary.)
  repeat
    Set $\phi_{twk} \propto \exp\{\mathbb{E}_q[\log \theta_{tk}] + \mathbb{E}_q[\log \beta_{kw}]\}$
    Set $\gamma_{tk} = \alpha + \sum_w \phi_{twk} n_{tw}$
  until $\frac{1}{K} \sum_k |\text{change in} \gamma_{tk}| < 0.00001$
  M step:
  Compute $\tilde{\lambda}_{kw} = \eta + D n_{tw} \phi_{twk}$
  Set $\boldsymbol{\lambda} = (1 - \rho_t)\boldsymbol{\lambda} + \rho_t \tilde{\boldsymbol{\lambda}}$.
end for

Figure 5: Online Variational LDA from Hoffman paper

3 Code and implementation

The implementation of this section uses Python3 and available in the jupyter notebook ***scraper_doc_parse.ipynb***. After Prof. Khardon's suggestion, I chose all the books authored by Jane Austen as a starting point to perform topic modeling for a subset of gutenburg documents. We start with the URLs of all books written by Jane Austen in *jane_austen.txt* and use the popular python library ***beautifulsoup*** for parsing and web scraping. Each book is written to a plain text file located in the *books* directory.

After obtaining all the required documents locally, I use the python library ***nltk*** for cleaning and preprocessing. My motivation for writing this segment of the project in Python3 was that nltk is only compatible with this version of python. Using nltk's stopwords corpus (in Fig 6), I iterate through each book and remove common stopwords (like the, a, is, if, etc.). In addition, I got rid of useless and redundant pronouns, prepositions, conjunctions, interjections, verbs and adverbs. After performing all of the above, I finally got a clean relevant set of documents to use for topic modeling. These documents can be found inside the **books** directory.

```
what_to_filter_out = ['CC','EX','IN','PRP','PRP$','UH','WDT','WP','WP$','WRB','VB','VBG','VBD','VBP','VBZ',
                      'DT','EX','.','RB']
```

Figure 6: NLTK stopwords filters used

I had initially intended to implement the conventional batch Latent Dirichlet Allocation (LDA) for topic modeling on gutenberg documents as we did in programming assignment 4. As per Prof. Khardon's suggestion, I moved from typical LDA to online variational LDA for faster performance on larger corpus of documents, which applies well to the gutenberg documents dataset that I use for thing project. The variational LDA algorithm is inspired from the paper *Online Learning for Latent Dirichlet Allocation* by Hoffmann et al. Online variational LDA is implemented in ***Online_Variational_LDA.ipynb*** using popular python2 libraries like numpy and scipy.

The LDA generative model with Gibbs sampler is implemented in **LDA_gibbssampler.ipynb**. Just like programming assignment 4, I compute the matrices **C_d** and **C_t**.

To run both Gibbs sampler as well as online variational LDA, we selected the number of topics to be 100. The vocabulary size is 16128 for the case of Jane Austen. Thus, for 10 documents, the size of the document-word matrix is (10 x 16128). I define θ as the Dirichlet distribution over the topics for each document as a (10 x 100) matrix, i.e., $\theta \sim \text{Dirichlet}(\alpha)$. Similarly, I define a topic distribution over the words as a (100 x 16128) matrix such that $\beta \sim \text{Dirichlet}(\eta)$. Like the paper, I implement the ***E-step*** and the ***M-step***, where the dirichlet expectation for $\log\theta$ and $\log\beta$ are calculated using scipy's digamma function. This is the only function from the library scipy in my project. From the dirichlet expectations, I calculate the ϕ which is the exponential sum of expected $\log\theta$ and $\log\beta$. Finally in the E-step, γ is calculated using dot product of ϕ and the document-word matrix; repeating until changes in γ drop to < 0.00001 (as used in the main paper).

4 Webservice deployment

For each document processed, I build a webservice to return the top-3 topics given the document name. For example, the accesible link is *https://us-central1-cobalt-ion-289207.cloudfunctions.net/function-1?name=Emma*.

Some other names to use are: Emma, Lady Susan, Love and Freindship, Mansfield Park, Northanger Abbey, Persuasion, Pride and Prejudice, Sense and Sensibility, The Letters of Jane Austen, The Watsons By Jane Austen and Concluded by L. Oulton

5 Execution instructions

Using Python3 env (since this piece of code uses nltk which is only compatible with Python3), please run scraper_doc_parser.ipynb with a cell-by-cell order of execution.

LDA Gibbs sampler: Using Python2 env, please run LDA_gibbssampler.ipynb with a cell-by-cell order of execution.

Online variational LDA: Using Python2 env, please run Online_Variational_LDA.ipynb with a cell-by-cell order of execution.

scraper_doc_parser

December 12, 2020

[5]:
```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/sahiltyagi/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/sahiltyagi/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/sahiltyagi/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

[5]:
```
True
```

PARSING JANE AUSTEN BOOKS FROM GUTENBERG DOCUMENTS

[6]:
```
import urllib.request
from bs4 import BeautifulSoup
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import os
from nltk.tokenize import RegexpTokenizer

#removing pronouns, prepositions, conjunctions and interjections, verbs and␣
 ↪adverbs
what_to_filter_out =␣
 ↪['CC','EX','IN','PRP','PRP$','UH','WDT','WP','WP$','WRB','VB','VBG','VBD','VBP','VBZ','DT',
 ↪','RB']

jane_austen_homepage = {}
file = open(os.getcwd() + '/jane_austen.txt', 'r')
for line in file:
    jane_austen_homepage[str(line.split(',')[1])] = str(line.split(',')[0])
```

```
file.close()

for url in jane_austen_homepage.keys():
    stop_words = set(stopwords.words('english'))
    tokenizer = RegexpTokenizer(r'\w+')
    html = urllib.request.urlopen(url)
    f = html.read()
    soup = BeautifulSoup(f, 'html.parser')
    book = []
    tags = soup.find_all('p')
    for tag in tags:
        para = ''
        word_tokens = word_tokenize(str(tag.get_text()))
        for word in word_tokens:
            if word not in stop_words and 'CHAPTER' not in word:
                para = para + str(word) + ' '

        #tokenizer.tokenize(para)
        for w in tokenizer.tokenize(para):
            filter_words = word_tokenize(w)
            tagged_words = nltk.pos_tag(filter_words)
            # to remove all 's' from the contractions like it's, I'm etc...
            if tagged_words[0][1] not in what_to_filter_out and␣
 ↪tagged_words[0][0] != 's':
                book.append(str(tagged_words[0][0]))

    file = open(os.getcwd() + '/books/' + jane_austen_homepage[url] + '.txt',␣
 ↪'w')
    for line in book:
        file.write(line + ' ')

    file.close()
    print('did the book ' + str(jane_austen_homepage[url]) + ' so far..')
```

```
did the book Persuasion so far..
did the book The Letters of Jane Austen so far..
did the book Pride and Prejudice so far..
did the book Emma so far..
did the book Sense and Sensibility so far..
did the book Northanger Abbey so far..
did the book Mansfield Park so far..
did the book The Watsons: By Jane Austen and Concluded by L. Oulton so far..
did the book Lady Susan so far..
did the book Love and Freindship so far..
```

USE WORD FREQUENCY FROM EACH BOOK TO GET SOME IDEAS FOR TOPICS

Online_Variational_LDA

December 12, 2020

Online Variational LDA for topic modeling books from gutenburg documents

[123]:
```
import os
import numpy as np
from scipy import special
import random

seed_val = 1234567
np.random.seed(seed_val)
random.seed(seed_val)
```

[124]:
```
def init_preprocess():

    book_ids = {}
    ctr = 0
    per_book_word_count = {}
    global_unique_words = []

    current_dir = os.getcwd()

    topics_map = {}

    f = open(current_dir + '/frequent_topics.txt', 'r')
    for line in f:
        topics_map[ctr] = line.split()
        ctr += 1

    f.close()
    print('# of topics chosen is ' + str(len(topics_map)))

    ctr = 0
    book_dir = os.listdir(current_dir + '/books/')
    for book in book_dir:
        if '.txt' in book:
            f = open(current_dir + '/books/' + book, 'r')

            book_ids[ctr] = book
            ctr += 1
```

```
            word_count = {}

            for line in f:
                for word in line.split():
                    if word in word_count.keys():
                        word_count[word] = word_count[word] + 1
                    else:
                        word_count[word] = 1

                    if word not in global_unique_words:
                        global_unique_words.append(word)

            f.close()
            per_book_word_count[book] = word_count

    doc_word_matrix = np.empty([1, len(global_unique_words)])
    print('init doc_word_matrix shape ' + str(doc_word_matrix.shape))
    print('length of book_ids map is ' + str(len(book_ids)))
    #print(book_ids)

    for ix in range(0, len(book_ids)):
        book = book_ids[ix]
        doc_word_vector = []
        word_count = per_book_word_count[book]
        for word in global_unique_words:
            if word in word_count.keys():
                doc_word_vector.append(word_count[word])
            else:
                doc_word_vector.append(0)


        doc_word_vector = np.array(doc_word_vector)
        doc_word_vector = np.transpose(doc_word_vector).
↪reshape((len(global_unique_words),1))
        #print('doc_word_vector shape is ' + str(doc_word_vector.shape))
        doc_word_matrix = np.vstack((doc_word_matrix, np.
↪transpose(doc_word_vector)))

    #print(doc_word_matrix.shape)
    doc_word_matrix = np.delete(doc_word_matrix, (0), axis=0)
    print('final doc_word_matrix shape ' + str(doc_word_matrix.shape))

    return book_ids, topics_map, global_unique_words, doc_word_matrix
```

[125]:
```
book_ids, topics_map, global_unique_words, doc_word_matrix = init_preprocess()
```

```
# of topics chosen is 100
init doc_word_matrix shape (1, 16128)
length of book_ids map is 10
final doc_word_matrix shape (10, 16128)
```

[154]:
```
# returns shape as (documents x topics)
# numbers of topics
K = len(topics_map)
# number of documents
D = len(book_ids)
# size of vocabulary
V = len(global_unique_words)
#print(V)
eta = 0.01
# previous_gamma_tk = np.ones((D, K))*0.1
previous_gamma_tk = np.ones((D, K))
print(previous_gamma_tk.shape)

updated_gamma_tk = np.ones((D, K))
print(updated_gamma_tk.shape)

beta_lambda = np.random.gamma(1.0, 1.0, (K, V)) * (D*100)/(K*V)
print('beta_lambda shape is ' + str(beta_lambda.shape))
```

```
(10, 100)
(10, 100)
beta_lambda shape is (100, 16128)
```

Computes the Expected values of logtheta and logbeta

[155]:
```
def expectation_digamma(lda_matrix, v_index, parameter):
    lda_array = lda_matrix[v_index,:]
    if parameter == 'beta':
        lda_array = lda_array.reshape((1,V))
        expected_logtheta_or_logbeta = special.psi(lda_array) - special.psi(np.
 ↪sum(lda_array))
        expected_logtheta_or_logbeta = expected_logtheta_or_logbeta.
 ↪reshape((1,V))

    elif parameter == 'gamma':
        lda_array = lda_array.reshape((1,K))
        expected_logtheta_or_logbeta = special.psi(lda_array) - special.psi(np.
 ↪sum(lda_array))
        expected_logtheta_or_logbeta = expected_logtheta_or_logbeta.
 ↪reshape((1,K))

    return expected_logtheta_or_logbeta
```

Dirichlet parameter for topic distribution

[156]:
```
process_gammas = np.zeros((2,K-1))
process_gammas[0] = 1.0
process_gammas[1] = 0.01

gamma_to_use = process_gammas[0]/(process_gammas[0] + process_gammas[1])
dirichlet_alpha = float(5)/float(K)
print(gamma_to_use.shape)
alpha = np.zeros(K)
multiplier = 1.0
for i in range(0, K-1):
    alpha[i] = gamma_to_use[i]*multiplier
    multiplier = multiplier - alpha[i]

alpha[K-1] = multiplier
alpha = alpha * dirichlet_alpha
print(alpha.shape)
```

```
(99,)
(100,)
```

[157]:
```
# K is the number of topics
#take 2.0
K = len(topics_map)
max_iter = 500
for ix in range(0, D):
    book = book_ids[ix]
    print('current book being processed is ' + str(book))
    #delta = np.sum(np.absolute(updated_gamma_tk - previous_gamma_tk))
    itr = 0

    doc_wordcount_vec = doc_word_matrix[ix,:]
    #print('doc wordcount vect shape ' + str(doc_wordcount_vec.shape))

    expected_logbeta = expectation_digamma(beta_lambda, ix, 'beta')
    expo_logbeta = np.exp(expected_logbeta)
    expo_logbeta = expo_logbeta.reshape((V,1))
    #print('exponent logbeta ' + str(expo_logbeta.shape))
    expected_logtheta = expectation_digamma(updated_gamma_tk, ix, 'gamma')
    expo_logtheta = np.exp(expected_logtheta)
    expo_logtheta = expo_logtheta.reshape((K,1))
    phi_dwk = np.dot(expo_logtheta, np.transpose(expo_logbeta)) + 1e-100
    #print('exponent logtheta ' + str(expo_logtheta.shape))
    #print('exponent logbeta ' + str(expo_logbeta.shape))
    #print('exponent phi_dwk ' + str(phi_dwk.shape))
    alpha = alpha.reshape((K,1))
    #print('alpha shape ' + str(alpha.shape))
```

```
    while itr < max_iter:
        previous_gamma_tk[ix,:] = updated_gamma_tk[ix,:]
        #print('OLD gamma...')
        #print(previous_gamma_tk[ix,:])

        term1 = expo_logtheta * np.dot(doc_wordcount_vec/phi_dwk, expo_logbeta)
        #print('term1 shape is ' + str(term1.shape))
        updated_gamma_tk[ix,:] = np.transpose(alpha + term1)
        #print('updated_gamma_tk shape is ' + str(updated_gamma_tk.shape))
        expected_logtheta = expectation_digamma(updated_gamma_tk, ix, 'gamma')
        expo_logtheta = np.exp(expected_logtheta)
        expo_logtheta = expo_logtheta.reshape((K,1))
        #print('exponent logtheta ' + str(expo_logtheta.shape))
        #print(expo_logtheta)

        phi_dwk = np.dot(expo_logtheta, np.transpose(expo_logbeta)) + 1e-100
        #print('phi_dwk shape ' + str(phi_dwk.shape))

        #prod_phi_wordcount = np.sum(np.dot(doc_wordcount_vec, np.
↪transpose(phi_dwk)))

        #updated_gamma_tk[ix,:] = alpha + prod_phi_wordcount
        #print('updated_gamma_tk shape is ' + str(updated_gamma_tk.shape))
        #print('NEW gamma...')
        #print(updated_gamma_tk[ix,:])

        itr += 1
        errorchange = np.mean(abs(updated_gamma_tk - previous_gamma_tk))
        print(errorchange)
        if itr % 50 == 0:
            print('error is ' + str(errorchange))

        if (errorchange < 0.0001):
            break
```

```
current book being processed is The Watsons: By Jane Austen and Concluded by L.
Oulton.txt
0.4715286515174943
0.032389236720794
0.0003086718318158521
2.7757130396413034e-06
current book being processed is Northanger Abbey.txt
1.945213813000269
```

```
0.02100309941837507
1.1244997754582542e-05
current book being processed is Sense and Sensibility.txt
9.239686517506806
0.9875579220418303
0.0009892959691194907
1.2120810914518998e-05
current book being processed is Mansfield Park.txt
2.6962035629637895
0.001637183055124618
1.21438297538905e-05
current book being processed is Persuasion.txt
1.0931089691303477
0.004035603476229193
1.3090033796666844e-05
current book being processed is Emma.txt
0.20005162126644963
1.7026207724335053e-05
current book being processed is Pride and Prejudice.txt
1.829586504966365
0.3169595010803995
0.0025858709048128363
3.488775637092445e-05
current book being processed is The Letters of Jane Austen.txt
2.091982630479664
0.0036326601579804423
3.508704274090135e-05
current book being processed is Lady Susan.txt
2.0998872653582823
0.0006655528367613285
3.509494542533176e-05
current book being processed is Love and Freindship.txt
0.10008579005263518
3.6894792106477946e-05
```

ANOTHER TRIAL APPROACH

[63]:
```
alpha = float(1)/float(K)
alpha = alpha*np.ones(K)
alpha = alpha.reshape(1,K)
print(alpha.shape)
```

```
(1, 100)
```

[64]:
```
# K is the number of topics
K = len(topics_map)
max_iter = 500
for ix in range(0, D):
```

```
    book = book_ids[ix]
    print(book)
    #delta = np.sum(np.absolute(updated_gamma_tk - previous_gamma_tk))
    itr = 0

    doc_wordcount_vec = doc_word_matrix[ix,:]
    print('doc wordcount vect shape ' + str(doc_wordcount_vec.shape))

    expected_logbeta = expectation_digamma(beta_lambda, ix, 'beta')
    expo_logbeta = np.exp(expected_logbeta)
    expo_logbeta = expo_logbeta.reshape((V,1))
    print('exponent logbeta ' + str(expo_logbeta.shape))

    while itr < max_iter:
        #print('OLD gamma...')
        #print(previous_gamma_tk[ix,:])
        expected_logtheta = expectation_digamma(previous_gamma_tk, ix, 'gamma')

        expo_logtheta = np.exp(expected_logtheta)
        expo_logtheta = expo_logtheta.reshape((K,1))
        #print('exponent logtheta ' + str(expo_logtheta.shape))
        #print(expo_logtheta)

#         phi_dwk = np.dot(expo_logtheta, np.transpose(expo_logbeta))
        phi_dwk = np.dot(expo_logtheta, np.transpose(expo_logbeta)) + 1e-100
        #print('phi_dwk shape ' + str(phi_dwk.shape))

        prod_phi_wordcount = np.sum(np.dot(doc_wordcount_vec, np.
↪transpose(phi_dwk)))
        updated_gamma_tk[ix,:] = alpha + prod_phi_wordcount
        #print('updated_gamma_tk shape is ' + str(updated_gamma_tk.shape))
        #print('NEW gamma...')
        #print(updated_gamma_tk[ix,:])

        #delta = float(np.sum(np.absolute(updated_gamma_tk[ix,:] -␣
↪previous_gamma_tk[ix,:])))/float(K)
        delta = np.sum(np.absolute(updated_gamma_tk[ix,:] -␣
↪previous_gamma_tk[ix,:]))/K

        #test = np.absolute(updated_gamma_tk[ix,:] - previous_gamma_tk[ix,:])
        #print(test)

        itr += 1

        print('delta is ' + str(delta))
        if delta < 0.00001:
            print('reached decent convergence so gonna exit now...')
```

```
            break

        previous_gamma_tk[ix,:] = updated_gamma_tk[ix,:]
```

```
The Watsons: By Jane Austen and Concluded by L. Oulton.txt
doc wordcount vect shape (16128,)
exponent logbeta (16128, 1)
delta is 0.9899999999999999
delta is 0.0
reached decent convergence so gonna exit now…
Northanger Abbey.txt
doc wordcount vect shape (16128,)
exponent logbeta (16128, 1)
delta is 0.9899999999999999
delta is 0.0
reached decent convergence so gonna exit now…
Sense and Sensibility.txt
doc wordcount vect shape (16128,)
exponent logbeta (16128, 1)
delta is 0.9899999999999999
delta is 0.0
reached decent convergence so gonna exit now…
Mansfield Park.txt
doc wordcount vect shape (16128,)
exponent logbeta (16128, 1)
delta is 0.9899999999999999
delta is 0.0
reached decent convergence so gonna exit now…
Persuasion.txt
doc wordcount vect shape (16128,)
exponent logbeta (16128, 1)
delta is 0.9899999999999999
delta is 0.0
reached decent convergence so gonna exit now…
Emma.txt
doc wordcount vect shape (16128,)
exponent logbeta (16128, 1)
delta is 0.9899999999999999
delta is 0.0
reached decent convergence so gonna exit now…
Pride and Prejudice.txt
doc wordcount vect shape (16128,)
exponent logbeta (16128, 1)
delta is 0.9899999999999999
delta is 0.0
reached decent convergence so gonna exit now…
The Letters of Jane Austen.txt
```

```
doc wordcount vect shape (16128,)
exponent logbeta (16128, 1)
delta is 0.9899999999999999
delta is 0.0
reached decent convergence so gonna exit now…
Lady Susan.txt
doc wordcount vect shape (16128,)
exponent logbeta (16128, 1)
delta is 0.9899999999999999
delta is 0.0
reached decent convergence so gonna exit now…
Love and Freindship.txt
doc wordcount vect shape (16128,)
exponent logbeta (16128, 1)
delta is 0.9899999999999999
delta is 0.0
reached decent convergence so gonna exit now…
```

[ ]:

[7]:
```
books_dir = os.getcwd() + '/books/'
files = os.listdir(books_dir)
for file in files:
    if '.txt' in file:
        word_count = {}
        print(books_dir + file)
        f = open(books_dir + '/' + file, 'r')
        for line in f:
            for word in line.split():
                if word in word_count.keys():
                    word_count[word] = word_count[word] + 1
                else:
                    word_count[word] = 1

        f.close()
        all_counts = list(word_count.values())
        all_counts.sort(reverse=True)

        K = 200
        top_K = all_counts[0:K]

        popular_words = []
        for k,v in word_count.items():
            if v in top_K:
                popular_words.append(k)

        topic_file = open(os.getcwd() + '/top_K_topics/' + file, 'w')
        for topic in popular_words:
            topic_file.write(topic + '\n')

        topic_file.close()
```

```
/Users/sahiltyagi/Desktop/gutenberg/books/The Watsons: By Jane Austen and
Concluded by L. Oulton.txt
/Users/sahiltyagi/Desktop/gutenberg/books/Northanger Abbey.txt
/Users/sahiltyagi/Desktop/gutenberg/books/Sense and Sensibility.txt
/Users/sahiltyagi/Desktop/gutenberg/books/Mansfield Park.txt
/Users/sahiltyagi/Desktop/gutenberg/books/Persuasion.txt
/Users/sahiltyagi/Desktop/gutenberg/books/Emma.txt
/Users/sahiltyagi/Desktop/gutenberg/books/Pride and Prejudice.txt
/Users/sahiltyagi/Desktop/gutenberg/books/The Letters of Jane Austen.txt
/Users/sahiltyagi/Desktop/gutenberg/books/Lady Susan.txt
/Users/sahiltyagi/Desktop/gutenberg/books/Love and Freindship.txt
```

[85]:
```
ss = word_tokenize('perhaps')
tw = nltk.pos_tag(ss)
print(tw[0][0])
```