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…
```

[ ]: