Latent Dirichlet Allocation

- Paper: Latent Dirichlet Allocatio
- Author: David M.Blei, Andrew Y.Ng, Michael I.Jordan
- Teammates: Yizi Lin, Siqi Fu
- Github: https://github.com/lyz1206/lda.git (https://github.com/lyz1206/lda.git)

Part 1: Abstract

Latent Dirichlet Allocation (LDA) is a generative probabilitistic model dealing with collections of data such as corpus. Based on the assumption of bag of word and exchangeability, each document in corpus is modeled as random mixture over latent topics and each topic is modeled by a distribution over words. Document is represented in a form of topic probability. In this project, we foucs on the text data, and in this case, each document is represented as a topic probability. We implement *variational inference* and *EM algorithm* to estimate parameters and performed optimazation method to make our algorithm more efficient. We compare LDA with other *topic model* like LSI and HDP.

key words: Topic Model, Latent Dirichlet Allocation, Variational Inference, EM algorithm

Part 2: Background

In our project, we use the the paper "Latent Dirichlet Allocation" by David M. Blei, Andrew Y. Ng and Michael I.Jordan.

Latent Dirichlet allocation (LDA) is a generative probabilistic model of a corpus, it uses a three-level hierarchical Bayesian model to describe the word generative process. Its basic idea is that each document are represented as random mixtures over latent topics, and each topic is characterized by a distribution over words. In general, LDA assumes the following generative process for each document w in a corpus D:

1. Choose $N \sim Possion(\xi)$, which represents the document length.
2. Choose $\theta \sim Dir(\alpha)$, which θ is a column vector representing the topic probability.
3. For each of the N words:
    - Choose $z_n \sim Multinomial(\theta)$, which represents current topic.
    - Choose w_n based on $p(w_n \mid z_n; \beta)$

There are three critical assumption for this model:

- the dimensionality k of the Dirichlet distribution is assumed known and fixed.
- β is a V $\times$ k matrix, where $\beta_{ij} = P(w^j = 1 \mid z^i = 1)$, which means β represents the probability of generating one particular word given the particular topic. β is also assumed to be known and fix.
- words are generated by topics and those topics are infinitely exchangeable within a document.

The generating process is represented as a probabilistic graphical model below:

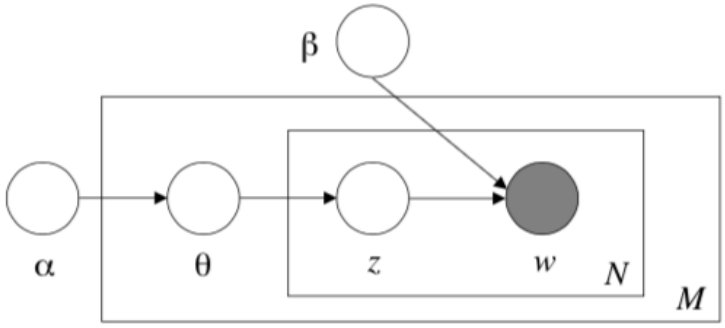

Figure 1: Graphical model representation of LDA. The boxes are "plates" representing replicates. The outer plate represents documents, while the inner plate represents the repeated choice of topics and words within a document.

Based on the model described above, the joint distribution of a topic mixture θ, a set of N topics z, and a set of N words w is given by:

$$p(\theta, z, w|\alpha, \beta) = p(\theta|\alpha) \prod_{n=1}^{N} p(z_n|\theta) p(w_n|z_n, \beta)$$

Integrating over θ and summing over z, we obtain the marginal distribution of a document:

$$p(w|\alpha, \beta) = \int p(\theta|\alpha) (\prod_{n=1}^{N} \sum_{z_n} p(z_n|\theta) p(w_n|z_n, \beta)) d\theta$$

Finally, taking the product of the marginal probabilities of single documents, we obtain the probability of a corpus:

$$P(D \mid \alpha, \beta) = \prod_{d=1}^{M} \int p(\theta_d \mid \alpha) (\prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn} \mid \theta_d) p(w_{dn} \mid z_{dn}, \beta)) d\theta_d$$

Using Bayesian rule, we can get the formula of the posterior distribution of the hidden variables given a document:

$$p(\theta, z|w, \alpha, \beta) = \frac{p(\theta, z, w|\alpha, \beta)}{p(w|\alpha, \beta)}$$

However, this distribution is intractable to capture in general. In this paper, the author use variational EM algorithm to approximate the distribution. We will discuss it in Part 3.

Generally speaking, the main goal of LDA is to find short descriptions of the members of a collection that enable efficient processing of large collections while preserving the essential statistical relationships that are useful for basis tasks. Common applications involve:

- document modeling
- text classification
- collaborative filtering

As a three level hierarchical Bayesian model, LDA is more elaborate than some other latent models such as Mixture of unigrams, and pLSA.

- In the Mixture of unigrams, the word distributions can be viewed as representations od topics under the assumption that each document exihibit only one topic. However, LDA allows documents to exihibit multiple topics to different probabilities.
- The pLSA model does solves the problem of Mixture of unigrams, but it has further problems that it is not well-defined generative model of documents,which means that it cannot be used to assign probability to a previously unused document. Also, since the linear growth in parameter of the pLSA model, it can causes overfitting. However, LDA sufferes neigher of those problems.

From Mixture of unigrams to PLSA TO lDA, the text modeling is improved step by step. LDA introduces the Dirichlet Distribution in the document to topic layer, which is better than PLSA, so that the number of model parameters does not expand with the increase of corpus.

Part 3: Description of algorithm

In part 2, we have mentioned that the posterior distribution of the hidden variables is intractable to capture in general, so the authors in this paper use variational EM algorithm to approximate it. Generally, this algorithm follows such iteration:

1. (E-step) For each document, find the optimizing values of the variational parameters$\{\gamma_d^*$, \Phi_d^\ast, d \in D}.
2. (M-step) Maximize the resulting lower bound on the log likelihood with respect to the model parameters α and β. This correspondings to finding maximum likelihood estimates with expected sufficient statistics for each document under the approximate posterior which is computed in the E-step.

- E-step

The main idea in this step is to find the tightest possible lower bound of the log likelihood and choose variational parameters.

Firstly, we show the procedure of finding the tightest lower bound of the log likelihood.

We begin by applying `Jensen's inequality` to bound the log likehood of document:

$$\begin{aligned} log\ p\ (w \mid \alpha, \beta) &= log \int \sum_z p(\theta, z, w \mid \alpha, \beta) d\theta \\ &= log \int \sum_z \frac{p(\theta, z, w \mid \alpha, \beta)\ q(\theta, z)}{q(\theta, z)} d\theta \\ &\geq \int \sum_z q(\theta, z)\ log\ p(\theta, z, w \mid \alpha, \beta) d\theta - \int \sum_z q(\theta, z)\ log\ q(\theta, z) d\theta \\ &= E_q[log\ p(\theta, z, w \mid \alpha, \beta)] - E_q[log\ q(\theta, z)] \end{aligned}$$

Form the above equation, we get a lower bound of the likelihood for variational distribution $q(\theta, z \mid \gamma, \phi)$.

The difference between the left side and right side of the above qeuation represents the `KL` divergence between variational posterior probability and the true posterior probability. Let $L(\gamma, \phi : \alpha, \beta)$ denote the right-hand side, and we can get:

$$logp(w \mid \alpha, \beta) = L(\gamma, \phi : \alpha, \beta) + D(q(\theta, z \mid \gamma, \phi) \mid\mid p(\theta, z \mid w, \alpha, \beta))$$

This shows the maximize lower bound $L(\gamma, \phi : \alpha, \beta)$ with respect to γ and ϕ is equivalent to minimizing the KL divergence.

So we successfully translate the into the optimization problem as below:

$$(\gamma^*, \phi^*) = argmin_{\gamma, \phi} D(q(\theta, z \mid \gamma, \phi) \mid\mid p(\theta, z \mid w, \alpha, \beta))$$

Secondly, we obtain a tractable family of the $q(\theta, z)$.

In the paper, the authors drop the edges between θ, z and w, as well as the w nodes. This procedure is shown below.

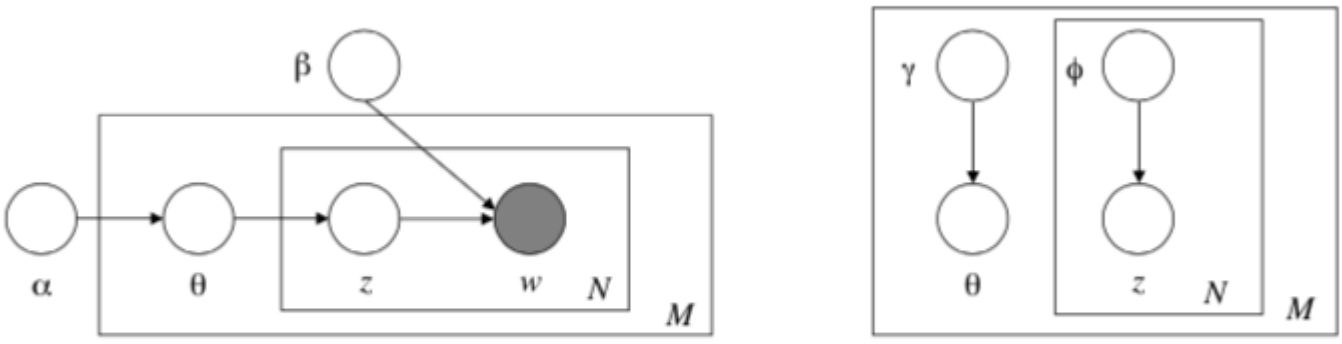

Figure 5: (Left) Graphical model representation of LDA. (Right) Graphical model representation of the variational distribution used to approximate the posterior in LDA.

So $q(\theta, z)$ is characterized by the following variational distribution:

$$q(\theta, z|\gamma, \Phi) = q(\theta|\gamma) \prod_{n=1}^{N} q(z_n|\Phi_n)$$

Thirdly, we expand the lower bound using the factorizations of p and q:

$$L(\gamma, \phi : \alpha, \beta) = E_q[logp(\theta\alpha)] + E_q[logp(z \mid \theta)] + E_q[logp(w \mid z, \beta)] - E_q[logq(\theta)] - E_q[logq(z)]$$

Finally, we use Lagrange method to maximize the lower bound with respect to the variational parameters Φ and γ. The updated equations are:

$$\phi_{ni} \propto \beta_{iw_n} exp[E_q(log(\theta_i) \mid \gamma)]$$

$$\gamma_i = \alpha_i + \sum_{n=1}^{N} \phi_{ni}$$

The pseudocode of E-step is as follow:

(1) initialize $\phi_{ni}^0 := 1/k$ for all i and n
(2) initialize $\gamma_i := \alpha_i + N/k$ for all i
(3) **repeat**
(4) **for** $n = 1$ **to** N
(5) **for** $i = 1$ **to** k
(6) $\phi_{ni}^{t+1} := \beta_{iw_n} \exp(\Psi(\gamma_i^t))$
(7) normalize ϕ_n^{t+1} to sum to 1.
(8) $\gamma^{t+1} := \alpha + \sum_{n=1}^{N} \phi_n^{t+1}$
(9) **until** convergence

- M-step

The main in M-step is to maximize the resulting lower bound on the log likelohood with respect to the model parameters α and β.

We update β through Lagrange method.

$$L_{[\beta]} = \sum_{d=1}^{M} \sum_{n=1}^{N_d} \sum_{i=1}^{k} \sum_{j=1}^{V} \Phi_{dni} w_{dn}^j log\beta_{ij} + \sum_{i=1}^{k} \lambda_i (\sum_{j=1}^{V} \beta_{ij} - 1)$$

So the update equation is:

$$\beta_{ij} \propto \sum_{d=1}^{M} \sum_{n=1}^{N_d} \Phi_{dn_i} w_{dn}^j$$

α is updated by Newton-Raphson Method:

$$\alpha_{new} = \alpha_{old} - H(\alpha_{old})^{-1} g(\alpha_{old})$$

where $H(\alpha)$ is the Hessian matrix and $g(\alpha)$ is the gradient at point α. This algorithm scales as $O(N^3)$ due to the matrix inversion.
Instead, if the Hessian matrix has a special struture $H = diag(h) + \mathbf{1}z\mathbf{1}^T$, we are able to yields a Newton-Raphson algorithm that has linear complexity. This precesure is shown below:

$$H^{-1} = diag(h)^{-1} - \frac{diag(h)^{-1}\mathbf{1}\mathbf{1}^T diag(h)^{-1}}{z^{-1} + \sum_{j=1}^{k} h_j^{-1}}$$

Multiplying by the gradient, we can get the ith component as:

$$(H^{-1}g)_i = \frac{g_i - c}{h_i}$$

where $c = \frac{\sum_{j=1}^{k} g_j/h_j}{z^{-1} + \sum_{j=1}^{k} j_j^{-1}}$

Part 4: Optimization

In this project, we use variational EM algorithm for LDA model to find the value of parameter α and β, to maximize the marginal log likelihood. In general, there are two parts that needed to be optimized.

1. E-step: For each document d, calculate the optimizing values of the variational parameters : γ_d^* and Phi_d^*.
2. M-step: Based on the results of E-step, update α and β.

In the previous part(the plain version), we have optimized the M-step. More specifically, we update α through the Newton-Raphson methods for a Hessian with special structure, which is mentioned in the paper and decrease the time complexity from $O(N^3)$ to linearity.

In this part, our main goal is to optimize the E-step. There are two processes here: optimize γ_d^* and Phi_d^*, then calculate the statistics for M-step.

Method we use:

- Vectorization: in `Estep_singedoc()` function, usd matrix to avoid the use of for loop.
- JIT compilation: for `accumulate_Phi()` and `Estep()` function.

We also tried the cython method, but unfortunately it didn't improve the code performance here.

In [1]:
```
import numpy as np
import pandas as pd
import gensim
import numba
from nltk.stem import WordNetLemmatizer
from nltk import PorterStemmer
from scipy.special import digamma, polygamma
```

Original version

In [2]:
```
def Estep_original(doc, alpha, beta, k, N_d, max_iter = 50):
    '''
    E step for a document, which calculate the posterior parameters.
    beta and alpha is coming from previous iteration.
    Return Phi and gamma  of a document.
    '''
    gamma_old = [alpha[i] + N_d/k  for i in range(k)]
    row_index = list(doc.keys())
    word_count = np.array(list(doc.values()))

    for i in range(max_iter):

        # Update Phi
        Phi = np.zeros((N_d, k))
        for i in range(N_d):
            for j in range(k):
                Phi[i,j] = beta[row_index[i],j]*np.exp(digamma(gamma_old[j]))
            Phi[i,:] = Phi[i,:]/np.sum(Phi[i,:])



        # Update gamma
        Phi_sum = np.zeros(k)
        for j in range(k):
            z = 0
            for i in range(N_d):
                z += Phi[i,j] * word_count[i]
            Phi_sum[j] = z

        gamma_new = alpha + Phi_sum

        # converge or not
        if(i>0) and (convergence(gamma_new, gamma_old)):
            break
        else:
            gamma_old = gamma_new.copy()

    return gamma_new, Phi

def accumulate_Phi_original(beta, Phi, doc):
    '''
    This function accumulates the effect of Phi_new from all documents after e
step.
    beta is V*k matrix.
    Phi is N_d * k matrix.
    Return updated beta.
    '''

    row_index = list(doc.keys())
    word_count = list(doc.values())
    for i in range(len(row_index)):
        beta[row_index[i],:] = word_count[i] * Phi[i,:]

    return beta
```

Opitimized version

In [3]:
```
def Estep_singedoc(doc, alpha, beta, k, N_d, max_iter = 50):
    '''
    E step for a document, which calculate the posterior parameters.
    beta and alpha is coming from previous iteration.
    Return Phi and gamma  of a document.
    '''

    gamma_old = alpha + np.ones(k) * N_d/k
    row_index = list(doc.keys())
    word_count = np.array(list(doc.values()))

    for i in range(max_iter):
        # Update Phi
        Phi_exp = np.exp(digamma(gamma_old))
        Phi = beta[row_index,:] @ np.diag(Phi_exp)
        Phi_new = normalization_row(Phi)

        # Update gamma
        Phi_sum = Phi_new.T @ word_count[:,None] # k-dim
        gamma_new = alpha + Phi_sum.T[0]

        # Converge or not
        if (i>0) & convergence(gamma_new, gamma_old):
            break
        else:
            gamma_old = gamma_new.copy()


    return gamma_new, Phi_new


@numba.jit(cache = True)
def accumulate_Phi(beta, Phi, doc):
    '''
    This function accumulates the effect of Phi_new from all documents after e
step.
    beta is V*k matrix.
    Phi is N_d * k matrix.
    Return updated beta.
    '''
    beta[list(doc.keys()),:] += np.diag(list(doc.values())) @ Phi

    return beta


@numba.jit(cache = True)
def Estep(doc, alpha_old, beta_old, beta_new, gamma_matrix, k, N_d, M):
    '''
    Calculate $\gamma$ and $\Phi$ for all documents.
    '''
    for i in range(M):
        gamma, Phi = Estep_singedoc(doc[i], alpha_old, beta_old, k, N_d[i])
        beta_new = accumulate_Phi(beta_new, Phi, doc[i])
        gamma_matrix[i,:] = gamma
    return beta_new, gamma_matrix
```

In [4]:
```
# Some helpful functions for comparison
def convergence(new, old, epsilon = 1.0e-3):
    '''
    Check convergence.
    '''
    return np.all(np.abs(new - old)) < epsilon

def normalization_row(x):
    '''
    Normaize a matrix by row.
    '''
    return x/np.sum(x,1)[:,None]

def initializaiton(k, V):
    '''
    Initialize alpha and beta.
    alpha is a k-dim vector. beta is V*k matrix.
    '''
    np.random.seed(12345)
    alpha = np.random.uniform(size = k)

    alpha_output = alpha/np.sum(alpha)

    beta_output = np.random.dirichlet(alpha_output, V)

    return alpha_output, beta_output

def lemmatize_stemming(text):
    '''
    Lenmmatize and stem the text.
    '''
    return PorterStemmer().stem(WordNetLemmatizer().lemmatize(text, pos='v'))

def preprocess(text):
    '''
    Preprocess the text.
    '''
    result = []
    for token in gensim.utils.simple_preprocess(text):
        if token not in gensim.parsing.preprocessing.STOPWORDS and len(token)
> 3:
            result.append(lemmatize_stemming(token))
    return result
```

Comparison

In [10]:
```
# Load the data and preprocess it.

data = pd.read_csv('data/articles.csv', error_bad_lines=False)
document = data[['content']].iloc[:50,:].copy()
document['index'] = document.index
processed_docs = document['content'].map(preprocess)
vocabulary = gensim.corpora.Dictionary(processed_docs)
#vocabulary.filter_extremes(no_below=5, no_above=0.1, keep_n=100)
bow_corpus = [vocabulary.doc2bow(doc) for doc in processed_docs]

doc = [dict(bow) for bow in bow_corpus]

N_d = [len(d) for d in doc]
V = len(vocabulary)
M = len(doc)
k = 3
```

- Original Version

In [11]:
```
%%timeit
alpha0, beta0 = initializaiton(k, V)
beta = beta0
gamma_matrix_origin = np.zeros((M, k))
for i in range(M):
    gamma, Phi = Estep_original(doc[i], alpha0, beta0, k, N_d[i])
    beta = accumulate_Phi_original(beta, Phi, doc[i])
    gamma_matrix_origin[i,:] = gamma
```

```
15.2 s ± 2.2 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

- Optimized Version

In [12]:
```
%%timeit
alpha0, beta0 = initializaiton(k, V)
beta = beta0
gamma_matrix_opt = np.zeros((M, k))
beta_new, gamma_matrix_opt = Estep(doc, alpha0, beta0, beta, gamma_matrix_opt,
k, N_d, M)
```

```
221 ms ± 10.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

- Conclusion
  We use 50 documents here, and set up k=3. After opitimizaion, the running time decreased from 15.2 to 221 ms. the increase percentage is nearly 100%.

Part 5: Applications to simulated data sets

In this part, we use our model on a simulated data set. Given α, β, k, M, V, and N_d by ourselves, the data process is described in Part 2. The corresponding code and result is in the file Test-Simulated.ipynb.

The real value of α is [0.15, 0.35, 0.5]. The results of the simulated shows that, , the $\hat{\alpha}$ is [0.49, 0.34, 0.16]. We find that the estiamted value is able to approximate the true value.

We also calculate the average difference between β and $\hat{\beta}$, which is equals to 0.0059 and is acceptable.

In conclusion, the LDA method is able to capture the true value of α and β if we run the code for enough time and start with a "good" point.

Part 6: Real data set

In this part, we implent our algorithm on two real dataset.
We preprocess the dataset before using them. Operation includes: spliting the sentences into words, lemmatizating, removing stop words, creating vocaubulary and establish corpus.

- Dataset in Paper

In the original paper, the authors used 16,000 documents from a subset of the TREC AP corpors(Harman, 1992). It is not easy to get the TREC datast since we need to sign an individual agreement and ask for approval from NIST. Instead, we download the sample data on Blei's webpage (http://www.cs.columbia.edu/~blei/lda-c/). This sample is just a subset of the data that the authors used in the paper, so we cannot get the same result.

	Economy	Life	Politics
0	million	year	presid
1	year	work	unit
2	market	peopl	forc
3	billion	famili	reagan
4	stock	children	defens
5	month	live	talk
6	compani	health	support
7	busi	studi	statement
8	point	life	sourc
9	board	like	troop

The topic words from some of the resulting multinomial distribution $p(w|z)$ are illustrated above. These distribution seem to capture some hidden topics in the corpus.
For example, "millison", "market", "stock", "company" are common words in the topic like "economy", and "president", "reagan", "statement" and "troop" are common words in the topic like "politics".

- Another Dataset

This dataset is named "All the news" and it is coming from kaggle (https://www.kaggle.com/snapcrack/all-the-news). The dataset contains articles from New York Times, Breitbart, CNN, Business Insider, the Atlantic, Fox News and so on. The original dataset has three csv file, but we just use the first 1000 rows in the second file.

	Astronomy	Medical	President Election
0	say	say	trump
1	scientist	patient	presid
2	space	drug	busi
3	year	studi	organ
4	human	medic	hotel
5	planet	doctor	conflict
6	earth	health	properti
7	climat	pain	elect
8	univers	cancer	accord
9	speci	vaccin	tabl

Similar to the previous dataset, the LDA model captures some hidden topics in the corpus. For example, words like "space", "planet", "earth" and "universe" are common in astronomy area.

In conclusion, our package works well. The LDA model is able to capture the hidden topic in the corpus and to provide reasonable insights to us, which is useful for text classification and collaborative filtering.

Part 7 Comparative Analysis with Competing Algorihtms

In this part, we compare the LDA method with two competing algorithm: Latent Semantic Indexing (LSI) and Hierarchical Dirichlet process(HDP). We still use the "All the news" dataset to evaluate the performance of the algorithm.

In [14]:
```
df = pd.read_csv('data/articles.csv')
document = df[['content']].copy()
document['index'] = document.index
processed_docs = document['content'].map(preprocess)
vocabulary = gensim.corpora.Dictionary(processed_docs)
bow_corpus = [vocabulary.doc2bow(doc) for doc in processed_docs]
```

LDA vs LSI

In [15]:
```
from gensim.models import LdaModel
from gensim.models import LsiModel
```

- Compare speed

In [16]:
```
%timeit LsiModel(bow_corpus, 30, id2word = vocabulary)
```

```
914 ms ± 170 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [17]:
```
%timeit LdaModel(bow_corpus, 30, id2word = vocabulary)
```

```
5.33 s ± 144 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

- Compare result

In [18]:
```
lsamodel = LsiModel(bow_corpus, 30, id2word = vocabulary)
ldamodel = LdaModel(bow_corpus, 30, id2word = vocabulary)
```

In [19]:
```
for idx, topic in ldamodel.print_topics(-1):
    if idx<5:
        print('Topic: {} \nWords: {}'.format(idx, topic))\
```

```
Topic: 0
Words: 0.017*"trump" + 0.008*"say" + 0.007*"presid" + 0.006*"like" + 0.005*"p
eopl" + 0.004*"state" + 0.004*"american" + 0.004*"time" + 0.004*"republican"
+ 0.003*"year"
Topic: 1
Words: 0.017*"trump" + 0.009*"presid" + 0.006*"say" + 0.004*"like" + 0.004*"a
merican" + 0.004*"work" + 0.004*"peopl" + 0.004*"time" + 0.004*"year" + 0.003
*"go"
Topic: 2
Words: 0.015*"trump" + 0.008*"school" + 0.007*"say" + 0.006*"like" + 0.005*"s
tate" + 0.005*"presid" + 0.005*"year" + 0.005*"peopl" + 0.004*"work" + 0.003
*"countri"
Topic: 3
Words: 0.011*"trump" + 0.008*"say" + 0.007*"peopl" + 0.006*"like" + 0.005*"ye
ar" + 0.004*"presid" + 0.004*"think" + 0.004*"work" + 0.003*"know" + 0.003*"s
tate"
Topic: 4
Words: 0.006*"say" + 0.005*"peopl" + 0.005*"work" + 0.005*"presid" + 0.004*"t
rump" + 0.004*"like" + 0.004*"think" + 0.004*"time" + 0.003*"go" + 0.003*"com
e"
```

In [20]:
```
for idx, topic in lsamodel.print_topics(-1):
    if idx<5:
        print('Topic: {} \nWords: {}'.format(idx, topic))
```

```
Topic: 0
Words: 0.767*"trump" + 0.307*"presid" + 0.128*"busi" + 0.116*"organ" + 0.103
*"compani" + 0.086*"hotel" + 0.084*"properti" + 0.079*"conflict" + 0.074*"acc
ord" + 0.073*"elect"
Topic: 1
Words: 0.367*"say" + 0.270*"peopl" + -0.213*"trump" + 0.206*"like" + 0.153*"y
ear" + 0.152*"state" + 0.150*"work" + 0.136*"think" + 0.128*"school" + 0.127
*"time"
Topic: 2
Words: -0.293*"senat" + -0.232*"republican" + -0.190*"democrat" + -0.171*"sta
te" + -0.166*"trump" + -0.162*"confirm" + 0.161*"peopl" + -0.160*"vote" + -0.
136*"hous" + -0.135*"nomin"
Topic: 3
Words: 0.667*"school" + 0.409*"student" + 0.253*"educ" + 0.152*"public" + 0.1
22*"univers" + -0.118*"peopl" + 0.111*"devo" + 0.102*"state" + 0.098*"teache
r" + 0.092*"charter"
Topic: 4
Words: -0.253*"patient" + 0.246*"state" + -0.208*"senat" + -0.193*"drug" + 0.
169*"countri" + -0.143*"confirm" + 0.142*"american" + -0.141*"studi" + -0.140
*"say" + 0.140*"unit"
```

The result above shows that the LSI algorithm is even faster, since it implement a SVD decompisition to reduce the dimension of the input. Howvever, the LDA method implement a variational EM algorithm or gibbs sampling, which require a lot of iteration to make the estimated value of every document converge, and thus is time consuming. As for the result, we find that all the coefficients in LDA is positive, but some of the coefficients in LSI is negative. In conclusion, the speed of LSI algotithm is significantly faster than the LDA algorithm. In the case that we need to use cross-validation to choose the topic number, LSI is more effective. However, the components of the topic in LSI method are arbitrarily positive/negative, which it is difficult to interpret. Moreover, LSI is unable to capture the multiple meanings of words.

LDA vs HDP

One character of the LDA algorithm is that we need to specify the number of topic. However, in most cases, we do not know the exactly topic numbers. Cross validatio is a method to deal with this problem. However, in the previous part, we have shown that the LDA algorithm is less effective, using cross validation is time consuming.
HDP algorithm is a natural nonparametric generalization of Latent Dirichlet allocation, where the number of topics can be unbounded and learnt from data, so we don't need to select the topic numbers.

In [21]:
```
from gensim.models import HdpModel
```

- Compare speed

In [22]:
```
%timeit HdpModel(bow_corpus, vocabulary)
```

```
7.97 s ± 897 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [23]:
```
%timeit LdaModel(bow_corpus, 30, id2word = vocabulary)
```

```
5.23 s ± 370 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

- Compare results

In [24]:
```
hdp = HdpModel(bow_corpus, vocabulary)
```

In [25]:
```
for idx, topic in hdp.print_topics(-1):
    if idx<5:
        print('Topic: {} \nWords: {}'.format(idx, topic))
```

```
Topic: 0
Words: 0.022*trump + 0.009*presid + 0.007*say + 0.005*like + 0.004*state + 0.
004*peopl + 0.004*time + 0.003*year + 0.003*countri + 0.003*busi
Topic: 1
Words: 0.010*trump + 0.007*say + 0.006*peopl + 0.006*like + 0.005*state + 0.0
04*year + 0.004*presid + 0.004*time + 0.004*work + 0.003*american
Topic: 2
Words: 0.006*say + 0.004*like + 0.004*patient + 0.004*peopl + 0.003*trump +
0.003*studi + 0.003*work + 0.003*year + 0.003*time + 0.003*doctor
Topic: 3
Words: 0.010*trump + 0.005*say + 0.004*presid + 0.003*like + 0.003*state + 0.
003*polici + 0.002*year + 0.002*american + 0.002*school + 0.002*order
Topic: 4
Words: 0.017*trump + 0.003*busi + 0.003*presid + 0.003*countri + 0.002*financ
i + 0.002*like + 0.002*say + 0.002*organ + 0.002*elect + 0.002*state
```

Although the speed the LDA algorithm is faster, if we want to implement cross validaiton to ensure the topic numbers, we have to run the model serveal time, and thus the total time is longer than the HDP algorithm. In this case, HDP is better.
However, compared the results from two algorithm, it is obvious that they are different. Sometimes it is difficult to interpret the result of HDP algorithm. What's more, if the previous experience tells us what the topic number is, the LDA model is more effective.

Part 8 Discussion

Based on the performance of our algorithm on the real dataset and the dataset in paper, our algorithm does fulfill the need that divide the document into different topics and explore the words occurring in that topic of different weight.

LDA can be used in a variety of purposes:

- Clustering: The topic in the clustering center, and the document associate with multer clusters. Clustering is very helpful in organizing and summarizing article collections.
- Feature generation: LDA can generate features for other machine learning algorithm. As mentioned before, LDA generates topics for each document, and these K topics can be treated as K features, and these features can be used to predict as in logistic regression and decision tree.
- Dimension Reduction: LDA provides a topic distribution which can be seen as a concise summary of the article.Comparing articles in this dimensionally reduced feature space is more meaningful than in the feature space of the original vocabulary.

Even though the LDA performs well in our dataset, it does have some limitations. For example, the number of topics k is fixed and must be known ahead of time. Also, Since the Dirichlet topic distribution cannot capture correlations, the LDA can only capture uncorrelated topics. Finally, the LDA algorithm is based on the assumption of BoW, which assumes words are exchangeable, and does not consider sentence sequence.

To overcome the limitation, extend LDA to the distributions on the topic variables are elaborated. For example, arranging the topics in a time series, so that it relax the full exchangeability assumption to one of partial exchangeability.

Part 9 References/bibliography

[1] Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." the Journal of machine Learning research 3 (2003): 993-1022.