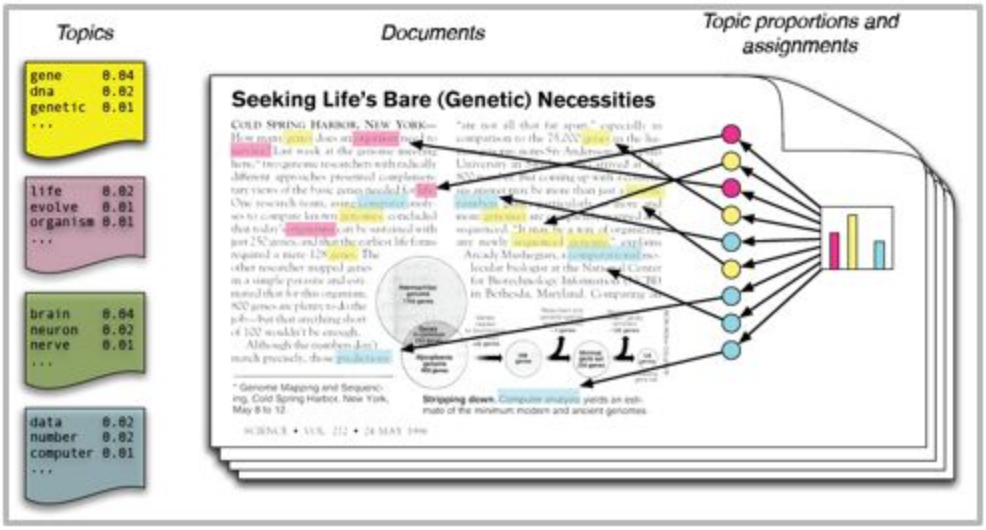
Topic Modelling using Latent Dirichlet Allocation

Topic Modelling is a process to automatically identify topics present in a text object and to derive hidden patterns exhibited by a text corpus. Topic Models are very useful for the purpose for document clustering, organizing large blocks of textual data, information retrieval from unstructured text and feature selection.



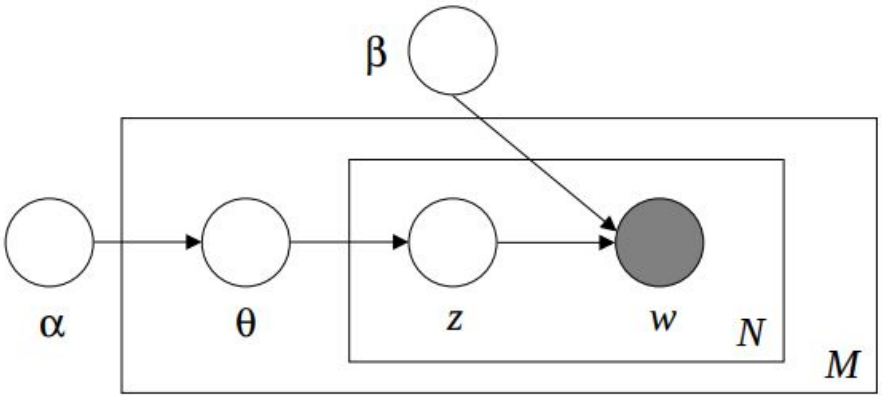
Latent Dirichlet allocation:

Latent Dirichlet allocation is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar.

Intuition behind the algorithm:

LDA assumes the following generative process for each document w in a corpus D:

1. Choose N ~ Poisson(ξ).
2. Choose θ ~ Dir(α).
3. For each of the N words wn:
    (a) Choose a topic zn ~ Multinomial(θ).
    (b) Choose a word wn from p(wn |zn,β), a multinomial probability conditioned on the topic zn.



Algorithm:

1) Randomly assign each word in all the documents one of the K topics. This is the initial word-topic assignment, and will be updated later.

2) Iterate over each document d

3) For every word w in document d we assign it a topic according to the initial word-topic assignment

4) Now we reassign word w in document d to topic k according to the probability above

$$Pr(z_{(d,n)} = k \mid z_{-(d,n)}, V, \alpha, \beta)$$

5) Use the word-topic assignment in 4) as the new initial word-topic assignment (updating step 1)

6) Repeat steps 2-5 several times (say 100 times).

Gibbs Sampling:

Gibbs sampling works by sampling each of those variables given the other variables (full conditional distribution).
It is given by:

$$p(z_{a,b} \mid z_{-(a,b)}, w, \alpha, \beta) = \frac{p(z_{a,b}, z_{-(a,b)}, w \mid \alpha, \beta)}{p(z_{-(a,b)}, w \mid \alpha, \beta)}$$

After integrating over latent parameters θ *and* ϕ,

$$p(z_{a,b} \mid z_{-(a,b)}, w, \alpha, \beta) = \frac{n_{d,k} + \alpha_k}{\sum_{i=1}^{k} (n_{d,i} + \alpha_i)} \times \frac{n_{k,w_{d,n}} + \beta_k}{\sum_{i=1}^{k} (n_{k,i} + \beta_i)}$$

Applications of LDA :

- Collaborative filtering
- Spam detection
- Music industry.
- Image.

LDA limitations:

- Fixed K (the number of topics is fixed and must be known ahead of time)
- Uncorrelated topics (Dirichlet topic distribution cannot capture correlations)
- Non-hierarchical (in data-limited regimes hierarchical models allow sharing of data)
- Static (no evolution of topics over time)
- Bag of words (assumes words are exchangeable, sentence structure is not modeled)

- Unsupervised (sometimes weak supervision is desirable, e.g. in sentiment analysis)
-

IMPROVEMENTS :

- We have tagged parts of speech to model.We should stop common stop words (the, a, it, etc),should also make sure that you don't allow very high frequency words to overpower the rest of the corpus & very infrequent words either.
- We used techniques like spell correction and lemmatization to further aggregate words and reduce the dimension of the data.
- LDA Limitation : Fixed K (the number of topics is fixed and must be known ahead of time) ; Solution : HDP-LDA

HDP also uses a Dirichlet process to capture the uncertainty in the number of topics. So a common base distribution is selected which represents the countably-infinite set of possible topics for the corpus, and then the finite distribution of topics for each document is sampled from this base distribution.

- LDA Limitation : Uncorrelated topics ; Solution : (CTM)

A topic model for text or other discrete data that models correlation between the occurrence of different topics in a document. Rather than using a Dirichlet, the CTM draws a real valued random vector from a multivariate Gaussian and then maps it to the simplex to obtain a multinomial parameter. This is the defining characteristic of the logistic Normal distribution.The covariance of the Gaussian induces dependencies between the components of the transformed random simplicial vector, allowing for a general pattern of variability between its components.

- LDA Limitation : Bag of words (assumes words are exchangeable, sentence structure is not modeled) ; Solution : TF-IDF

In TF-IDF, it is the term weight which is represented in Vector space model.Thus entire document is a feature vector. which points to a point in vector space such that there is an axis for every term in our bag.

Code:

```
from __future__
import division
                from nltk.tokenize import RegexpTokenizer
                from nltk.corpus import stopwords
                from nltk.stem.porter import PorterStemmer
                from tqdm import tqdm
                import numpy as np
                import pandas as pd
                import re



                #
                ################################################################################
                ############### Generating vocab


                comments = pd.read_csv("YoutubeCommentsSpam.csv")
                tokenizer = RegexpTokenizer(r'\w+')  # remove punctuation
                stop_words = stopwords.words('english')
                text_data = [line for line in comments["commentText"] if line != '']
                for line in range(len(comments)):
                   line_token = tokenizer.tokenize(text_data[line].lower())
                   clean_token = [re.sub(r'[^a-zA-Z]', '', word) for word in
line_token]  # only alphabets
                   stop_token = [word for word in clean_token if word not in
stop_words if word != '']  # remove stop words
                   stem_token = [str(PorterStemmer().stem(word)) for word in
stop_token]  # stemmed
                   text_data[line] = stem_token
                vocab_total = set([words for sublist in text_data for words in
sublist])
                # print(vocab_total)
                text_ID = []
```

```
for line in range(len(text_data)):
   comment_vector = [list(vocab_total).index(words) for words in
text_data[line]]
   text_ID.append(comment_vector)
# print(text_ID)


#
##########################################################################
################## Random init and other vectors


K = 2  # no of topics
V = len(vocab_total)
D = len(text_ID)
word_topic_count = np.zeros((K, V))
doc_word_topic_assignd = [np.zeros(len(sublist)) for sublist in
text_ID]
doc_topic_count = np.zeros((D, K))
for doc in range(D):
   for word in range(len(text_ID[doc])):
       doc_word_topic_assignd[doc][word] = np.random.choice(K, 1)
       word_topic = int(doc_word_topic_assignd[doc][word])
       word_doc_ID = text_ID[doc][word]
       word_topic_count[word_topic][word_doc_ID] += 1
for doc in range(D):
   for topic in range(K):
       topic_doc_vector = doc_word_topic_assignd[doc]
       doc_topic_count[doc][topic] = sum(topic_doc_vector == topic)


#
##########################################################################
################
```

```
alpha = 0.2

beta = 0.001

corpus_itter = 20

for itter in tqdm(range(corpus_itter)):

   for doc in range(D):

       for word in range(len(text_ID[doc])):

           init_topic_assign = int(doc_word_topic_assignd[doc][word])

           word_id = text_ID[doc][word]

           doc_topic_count[doc][init_topic_assign] -= 1

           word_topic_count[init_topic_assign][word_id] -= 1

           # ################ Gibb's sampling begin

           denom1 = sum(doc_topic_count[doc]) + K * alpha

           denom2 = np.sum(word_topic_count, axis=1) + V * beta

           numerator1 = [doc_topic_count[doc][col] for col in
range(K)]

           numerator1 = np.array(numerator1) + alpha

           numerator2 = [word_topic_count[row][word_id] for row in
range(K)]

           numerator2 = np.array(numerator2) + beta

           prob_topics = (numerator1 / denom1) * (numerator2 / denom2)

           prob_topics = prob_topics / sum(prob_topics)

           # ################ Gibb's sampling end

           update_topic_assign = np.random.choice(K, 1,
list(prob_topics))

           doc_word_topic_assignd[doc][word] = update_topic_assign

           doc_topic_count[doc][init_topic_assign] += 1

           word_topic_count[init_topic_assign][word_id] += 1

theta = (doc_topic_count+alpha)

theta_row_sum = np.sum(theta, axis=1)

theta = theta/theta_row_sum.reshape((D, 1))  # probab of a doc falling
in a given topic after running lda
```

```
phi = (word_topic_count + beta)

phi_row_sum = np.sum(phi, axis=1)

phi = phi/phi_row_sum.reshape((K, 1))  # probab of a word falling in a
given topic after running lda
list_dict_topics = []

for topic in range(K):

   mydict = {}

   for word in range(V):

       mydict[list(vocab_total)[word]] = phi[topic][word]

   list_dict_topics.append(mydict)

# print(sorted([(value, key) for (key, value) in
list_dict_topics[0].items()])[::-1][10:20])
# print(sorted([(value, key) for (key, value) in
list_dict_topics[1].items()])[::-1][10:20])
```

References:

- Research Paper : http://ai.stanford.edu/~ang/papers/jair03-lda.pdf
- Understanding of the Paper : https://www.youtube.com/watch?v=DWJYZq_fQ2A
- Paper: http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf
- Generative process: https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation
- Gibbs sampling: https://lingpipe.files.wordpress.com/2010/07/lda3.pdf
- https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/