Gaussian LDA

Reference

- Gaussian LDA for Topic Models withWord Embeddings, Rajarshi Das et al.
- bab2min.tistory.com

Introduction

{토픽 모델링, 자연어 처리, 삼겹살, 소고기}라는 단어 세트가 주어졌을 때, 컴퓨터는 이를 어떻게 인식할까? 컴퓨터에게 이 단어 세트는 {0, 1, 2, 3}를 의미할 뿐이다. 즉, {가, 나, 다, 라}와 동일한 의미를 가진다는 것이다. 사람 입장에서는 이전의 경험이나 언어 능력으로 인해서 명백히 두 단어 세트는 다름에도, 컴퓨터 입장에서는 어쩔 수 없다. 하지만 충분히 많은 단어 세트가 주어지고, 문서라는 상위 포함 관계까지 주어지면 학습을 통해 컴퓨터는 {토픽 모델링, 자연어 처리}, {삼겹살, 소고기}를 동일한 주제에 속한다고 인식하는 것이다.
LDA에서도 단어와 단어 사이의 유사도를 따로 고려하지 않는다. 이는 LDA의 문서 생성과정을 살펴보면 알 수 있다. 먼저 간략하게 LDA의 아키텍쳐를 복습해보자.

1. $\theta_d \sim Dir(\alpha)$ (문서의 주제 분포)
2. $z_{d,n} \sim Multi(\theta_d)$ (단어의 주제 분포)
3. $\beta_k \sim Dir(\beta)$ (주제별 단어 분포)
4. $w_{d,n} \sim Multi(\beta_{z_{d,n}})$ (단어 생성)

d번째 문서의 n번째 단어 $w_{d,n}$를 생성하는데 여러 generative process를 거쳐서 결국은 다항분포에서 생성한다. 예를 들어, {토픽 모델링, 자연어 처리, 삼겹살, 소고기}의 단어 세트로 이루어진 corpus를 생각해보자. $\beta_{z_{d,n}}$은 4×1 크기의 확률을 담고 있는 벡터이고, 이 확률 벡터를 모수로 하는 다항분포에서 {토픽 모델링, 자연어 처리, 삼겹살, 소고기} 중 하나의 단어를 뽑을 것이다. 그런데, 컴퓨터 입장에서는 이것이 {가, 나, 다, 라}와 다르지 않다는 것이다. 물론 충분한 학습 후에는 semantically 비슷해지겠지만, 만약에

- 충분한 데이터가 주어지지 않거나 질이 나쁘다면?
- 충분한 데이터가 있더라도 학습 이전에 단어들 사이의 semantic 유사성을 모델에 알려주면 더 잘 학습할 수 있지 않을까?

라는 의문점이 들곤 한다. Gaussian LDA는 이러한 문제를 해결해주는 topic modeling이다. 즉, 단어를 discrete하게 {0, 1, 2, 3}으로 나타내는 것이 아니라 마치 word2vec의 단어 임베딩처럼 (0.1, 0.5), (-0.1,-0.7)로 나타내서 단어들 사이의 semantic similarity을 반영할 수 있게 해준다.
기존의 LDA는 또 다른 문제를 가지고 있다. 바로 훈련 데이터에 등장하지 않은 단어에 대한 처리 문제이다. (Out Of Vocabulary) 예를 들어, D개의 문서로 구성된 corpus에 LDA를 수행하였다고 하자. 이 corpus는 총 V개의 단어를 가지고 있으므로 β_k의 차원은 $V \times 1$일 것이다. 하지만, 만약 학습한 corpus에서 등장하지 않은 단어가 test 데이터에 등장한다면 기존에 추정한 β_k에는 이 단어를 포함하지 않으므로 곤란한 상황이 발생한다. 이러한 문제는 결국, 주제 분포를 corpus에 등장하는 단어들의 discrete 집합으로 보기 때문에 발생하므로, G-LDA와 같이 주제 분포를 continuous하게 뽑는다면 해결할 수 있는 문제이다. G-LDA에서 각 단어는 {0, 1, 2, 3}과 같이 discrete한 것이 아니라, continous하게 표현이 된다. 따라서 OOV 단어가 등장한다고 하더라도, 그 단어의 임베딩 벡터와 원래 학습된 단어들의 임베딩 벡터를 비교하여 OOV 단어의 주제를 더 정확하게 파악할 수 있기 때문이다.

Gaussian LDA

가장 먼저, 기존의 LDA와 G-LDA는 단어를 보는 시각이다르다. 기존의 LDA는 문서를 단어의 집합이라고 보는 반면에, G-LDA는 문서를 단어 임베딩의 집합으로 본다. 문서를 구성하는 기본 단위를 discrete하게 보느냐, continuous하게 보느냐에 대한 차이이다. notation으로는 단어 $w_{d,n}$의 임베딩 벡터를 $\mathbf{v}(w_{d,n}) \in \mathbb{R}^M$, 즉 M차원 벡터라고 가정한다.
그러면 G-LDA는 주제에 대한 continuous 벡터를 어떻게 뽑는 것인가? 이를 생각하기 이전에 LDA에서 주제에 대한 discrete 벡터를 어떻게 만들었는지 생각해보자. discrete 벡터에 대한 conjugate prior은 dirichlet 분포이다. 따라서 기존의 LDA도 이러한 틀을 따랐다. 이제는 continous 벡터이다. 1개의 continuous 값을 뽑는 경우를 생각해보자. 이를 우리는 정규분포에서 뽑을 수 있다. R의 내장함수를 이용하면, rnorm을 통해서 정규분포로부터 임의의 난수를 추출할 수 있다. 그러면 p차원의 벡터는 다변량 정규분포에서 뽑을 수 있다. 여기서 p는 단어 세트의 크기가 아니라, 뽑으려는 주제 벡터의 차원이다. 아마도 사용자가 미리 지정하는 hyperparameter인 듯 하다. 이제 모델 아키텍쳐를 살펴보자.

1. for $k = 1$to K
   (a) Draw topic covariance $\Sigma_k \sim \mathcal{W}^{-1}(\mathbf{\Psi}, v)$
   (b) Draw topic mean $\boldsymbol{\mu}_k \sim \mathcal{N}(\boldsymbol{\mu}, \frac{1}{\kappa}\Sigma_k)$

   여기서 k는 topic index임에 주의하자. 즉, $\boldsymbol{\mu}_k, \Sigma_k$는 k번째 토픽에서 단어를 MVN에서 뽑을 때 사용하는 모수이다.

2. for each document d in corpus D (여기서부터는 기존의 LDA의 틀과 유사하다)
   (a) Draw topic distribution $\boldsymbol{\theta}_d \sim Dir(\alpha)$

(b) For each word index n from 1 to N_d

i. Draw a topic $z_n \sim Categorical(\boldsymbol{\theta}_d)$

ii. Draw $\mathbf{v}_{d,n} \sim \mathcal{N}(\boldsymbol{\mu}_{z_n}, \Sigma_{z_n})$

3-(b)-ii.을 살펴보자. 단어 벡터 $\mathbf{v}_{d,n}$이 MVN에서 뽑힌다. 기존의 LDA는 discrete 값을 가지는 단어 $w_{d,n}$가 다항분포에서 뽑혔음을 상기해보자. 바로 이 부분에서 기존의 LDA와 G-LDA의 단어를 보는 시각의 차이를 엿볼 수 있고 그로 인해서 주제 벡터의 generation process도 조금씩 달라지는 것이다.

Posterior Inference

단어 벡터 $\mathbf{v}_{d,n}$의 주제 $z_{d,n}$은 collapsed gibbs sapling을 통해 아래와 같이 유도된다.

$$P(z_{d,n} = k \mid \mathbf{z}_{-(d,n)}, \mathbf{V}_d, \boldsymbol{\zeta}, \boldsymbol{\alpha}) \propto (n_{k,d} + \alpha_k) \times t_{v_k - M+1}\left(\mathbf{v}_{d,n} \mid \boldsymbol{\mu}_k, \frac{\kappa_k + 1}{\kappa_k}\Sigma_k\right)$$

먼저 first term을 살펴보자. 어디서 많이 본 듯 했는데, 바로 original LDA의 first term과 동일하다. 이는 G-LDA와 original LDA의 문서 생성 과정은 동일하기 때문이다.
second term을 살펴보자. multivariate t 분포는 $t_v(x \mid \mu, \Sigma)$에서 $|\Sigma|$와 $(x-\mu)^T\Sigma^{-1}(x-\mu)$을 포함하는데, 이는 어마무시한 계산량을 자랑한다. complexity 측면에서 이는 $O(p^3)$인데, 이는 현실적으로 불가능하기 때문에 논문에서는 다른 방법을 제안한다.

Cholesky Decomposition

숄레스키 분해는 어떠한 대칭 행렬 X에 대해서 $X = LL^T$가 성립한다고 말한다. 여기서 L은 lower triangular matrix이다. 그러면 대칭행렬 Σ의 determinant는 $|\Sigma| = |LL^T| = |L| \times |L^T| = \left(\prod_{i=1}^{p} l_{i,i}\right)^2$로 간단하게 구할 수 있다. 또한 $\mathbf{b}^T\Sigma^{-1}\mathbf{b}$를 살펴보자. Σ가 대칭행렬이므로 Σ^{-1}도 대칭행ㄹ ㄹ이다. 따라서

$$\begin{aligned}
\mathbf{b}^T\Sigma^{-1}\mathbf{b} &= \mathbf{b}^T\left(LL^T\right)^{-1}\mathbf{b} \\
&= \mathbf{b}^T\left(L^{-1}\right)^T L^{-1}\mathbf{b} \\
&= \left(L^{-1}\mathbf{b}\right)^T L^{-1}\mathbf{b}
\end{aligned}$$

여기서 $L^{-1}\mathbf{b}$는 $L\mathbf{x} = \mathbf{b}$의 솔루션이고 L은 lower triangular matrix이기 때문에 쉽게 해를 구할 수 있다.
이렇게 숄레스키 분해를 이용하여, 계산량을 크게 줄여서 complexity를 $O(KM^2)$까지 달성했다고 한다.

Further Improvement

기존의 LDA는 관찰된 데이터가 discrete한 단어라고 생각하는 반면, G-LDA에서는 관찰된 데이터가 continuous한 단어, 즉 단어 벡터라고 생각한다. 다시 말하면, G-LDA의 input으로 사전에 훈련된

단어 임베딩 벡터가 들어가야한다는 것이다. 이는 word2vec, glove, fasttext 등을 이용하면 된다. 어떻게 보면 미리 학습한 단어 벡터를 넣는 것이 번거롭겠지만 단어간에 semantic 관계를 잘 반영한다는 점에서, 어떤 임베딩 벡터를 넣느냐에 따라서 성능도 꽤 달라질 것 같다. 이에 대해서는, 미리 훈련된 단어 벡터를 넣거나 현재 corpus를 훈련된 단어 벡터에 얹어서 추가 훈련하는 방식으로 정교한 단어 벡터를 가져오는 것이 point일 것이다.