Probabilistic topic modeling using the Latent Dirichlet Allocation

Théo Delobel, Louis Duhem, Saimourya Surabhi, Richard Trezeux

2019-2020

Contents

Chapter 1

Introduction

Latent Dirichlet Allocation (LDA) is a generative statistical model that is used in analysis of sets of observations. The key of LDA is to consider unobserved groups that contains similar data. LDA is especially used in data mining and natural language processing. For example, this methods has been used to analyse tweets linked to the presidential election of 2016 in the US and try to predict who will win.

To understand LDA and try to implement our first algorithms using the LDA methods, some mathematics background is needed. Monte Carlo methods, Markov chain and some knowledge in bayesian computing are needed. That's why next chapter will briefly summarize some basic knowledge about these subjects. A few examples will be developed to illustrate concepts.

Chapter 2

Mathematics prerequisite

2.1 Markov Chain

Markov chains are among the most important stochastic processes. They are stochastic processes for which the description of the present state fully captures all the information that could influence the future evolution of the process. Predicting traffic flows, communications networks, genetic issues, and queues are examples where Markov chains can be used to model performance. Devising a physical model for these chaotic systems would be impossibly complicated but doing so using Markov chains is quite simple. Modeling the to-and-fro communications traffic over the Internet is a prime example of what Markov chains can do. Nowadays, the Internet has become exceedingly complex, and traffic moves in a random way between network nodes. Markov chains are commonly used to describe not only the traffic, which is inherently unpredictable, but also how the network will perform, despite the complexity of its structure. The PageRank of a webpage as used by Google is defined by a Markov chain.Now lets look at a example.

2.1.1 Gambler's Ruin

Consider a gambling game in which on any turn you win \$1 with probability $p = 0.4$ or lose \$1 with probability $1-p = 0.6$.Suppose further that you adopt the rule that you quit playing if your fortune reaches \$N. Of course, if your fortune reaches \$0 the casino makes you stop.
Solution:
Let X_n be the amount of money you have after n plays. then the fortune, X_n has the **Markov property**. In other words, this means that given the current state,X_n, other information about the past is irrelevant for predicting the next state X_{n+1}. To check this for the gambler's ruin chain, we note that if we are still playing at time n, i.e., the fortune $X_n = i$ with $0 < i < N$, then for any possible history of the wealth $i_{n-1}, ..., i_1, i_0$

$$P(X_{n+1} = i + 1 | X_n = i, X_{n-1} = i_{n-1}, ...X_0 = i_0) = 0.4$$

since to increase the amount by one unit we have to win the next bet. Here we have used $P(B|A)$ for the conditional probability of the event B given that A occurs. It is defined by

$$P(B|A) = \frac{P(B \cap A)}{P(A)}$$

Turning now to the formal definition, we say that X_n is a discrete time **Markov chain** with **transition matrix** *p(i, j)* if for any $j, i, i_{n1}, ...i_0$

$$P(X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, ...X_0 = i_0) = p(i,j)(1.1)$$

Equation (1.1) explains what we mean when we say that "given the current state X_n, any other information about the past is irrelevant for predicting X_{n+1}." In formulating (1.1) we have restricted our attention to the temporally homogeneous case i.e. the transition probability does not depend on the time n

$$p(i,j) = P(X_{n+1} = j | X_n = i)$$

Intuitively, the transition probability gives the rules of the game. It is the basic information needed to describe a Markov chain. In the case of the gambler's ruin chain, the transition probability has

$$p(i, i+1) = 0.4 \qquad p(i, i1) = 0.6 \qquad if 0 < i < N$$
$$p(0, 0) = 1 \qquad p(N, N) = 1$$

When **N=5** the transtion matrix is:

$$\begin{array}{c} \begin{array}{cccccc} \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{array} \\ \begin{pmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 \\ 0.6 & 0 & 0.4 & 0 & 0 & 0 \\ 0 & 0.6 & 0 & 0.4 & 0 & 0 \\ 0 & 0 & 0.6 & 0 & 0.4 & 0 \\ 0 & 0 & 0 & 0.6 & 0 & 0.4 \\ 0 & 0 & 0 & 0 & 0 & 1.0 \end{pmatrix} \end{array} \begin{array}{c} \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \end{array}$$

One of the well know application of Markov chains is **Google Page Rank Algorithm**

2.1.2 Page Rank Algorithm

The PageRank algorithm gives each page a rating of its importance, which is a recursively defined measure whereby a page becomes important if important pages link to it. This definition is recursive because the importance of a page refers back to the importance of other pages that link to it.

One way to think about PageRank is to imagine a random surfer on the web, following links from page to page. The page rank of any page is roughly the probability that the random surfer will land on a particular page. Since more links go to the important pages, the surfer is more likely to end up there.The behavior of the random surfer is an example of a Markov process, which is any random evolutionary process that depends only of the current state of a system and not on its history.
Suppose that we have N web pages and wish to rank them in terms of importance. For example, the N pages might all contain a string match to "statistical learning" and we might wish to rank the pages in terms of their likely relevance to a websurfer. The *PageRank* algorithm considers a webpage to be important if many other webpages point to it. However the linking webpages that point to a given page are not treated equally: the algorithm also takes into account both the importance (PageRank) of the linking pages and the number of outgoing links that they have. Linking pages with higher PageRank are given more weight, while pages with more outgoing links are given less weight. These ideas lead to a recursive definition for PageRank. Let $L_{ij} = 1$ if page j points to page i, and zero otherwise. Let $c_j = \sum_{i=1}^{N} L_{ij}$ equal the number of pages pointed to by page j (number of out-links). Then the Google PageRanks p_i are defined by the recursive relationship

$$p_i = (1-d) + d\sum_{j=1}^{N}(\frac{L_{ij}}{c_j})p_j$$

where d is a positive constant (generally set to 0.85). The idea is that the importance of page i is the sum of the importances of pages that point to that page. The sums are weighted by $\frac{1}{c_j}$ i.e, each page distributes a total vote of 1 to other pages. The constant d ensures that each page gets a PageRank of at least $1-d$. In matrix notation

$$\mathbf{p} = (1-d)e + d.\mathbf{L}\mathbf{D}_c^{-1}p$$

where $\mathbf{e}$ is a vector of $\mathbf{N}$ ones and D_c = diag(c) is a diagonal matrix with diagonal elements c_j . Introducing the normalization $e.T_p = N$ (i.e., the average PageRank is 1), we can write the above equation as following.

$$\mathbf{p} = [(1-d)ee^T/N + d.\mathbf{L}\mathbf{D}_c^{-1}]p = \mathbf{A}p$$

where the matrix A is the expression in square braces. Exploiting a connection with Markov chains, it can be shown that the matrix A has a real eigenvalue equal to one, and one is its largest eigenvalue. This means that we can find ˆp by the power method: starting with some $p = p_0$ we iterate

$$p_k \longleftarrow \mathbf{A}p_{k-1}, p_k \longleftarrow N\frac{p_k}{e^T.p_k}$$

The fixed points p̂ are the desired PageRanks.
Viewing PageRank as a Markov chain makes clear why the matrix $\mathbf{A}$ has a

maximal real eigenvalue of 1. Since A has positive entries with each column summing to one, Markov chain theory tells us that it has a unique eigenvector with eigenvalue one, corresponding to the stationary distribution of the chain A small network is shown for illustration is shown below :

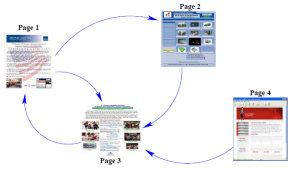


Figure 2.1: Example of a small network

The link matrix is

$$\mathbf{L} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

and the number of outlinks is $c = (2, 1, 1, 1)$ The PageRank solution is $\hat{p} = (1.49, 0.78, 1.58, 0.15)$. Notice that page 4 has no incoming links, and hence gets the minimum PageRank of 0.15.

2.2 Markov Chain Monte Carlo

Markov chain Monte Carlo (MCMC) methods use computer simulation of Markov chains in the parameter space. The Markov chains are defined in such a way that the posterior distribution in the given statistical inference problem is the asymptotic distribution. This allows to use ergodic averages to approximate the desired posterior expectations. Several standard approaches to define such Markov chains exist, including Gibbs sampling, Metropolis–Hastings, and reversible jump. Using these algorithms it is possible to implement posterior simulation in essentially any problem which allows point wise evaluation of the prior distribution and likelihood function

2.2.1 Metropolis-Hastings

This a very useful tool for computing posterior distributions in Bayesian statistics , reconstructing images , and investigating complicated models in statistical physics e.t.c. To describe the simple idea that is the important to the method. We begin with a Markov chain q(x, y) that is the proposed jump distribution. A move is accepted with probability

$$r(x, y) = min\left\{ \frac{\pi(y)q(y, x)}{\pi(x)q(x, y)}, 1 \right\}$$

so the transition probability will be

$$p(x, y) = q(x, y)r(x, y)$$

To check that $\pi(x)$ satisfies the detailed balance condition we can suppose that

$$\pi(y)q(y, x) > \pi(x)q(x, y)$$

In this case

$$\begin{aligned}
\pi(x)p(x, y) &= \pi(x)q(x, y)1 \\
\pi(y)p(y, x) &= \pi(y)q(y, x)\pi(x)q(x, y) \\
\pi(y)q(y, x) &= \pi(x)q(x, y)
\end{aligned}$$

To generate one sample from $\pi(x)$ we run the chain for a long time so that it reaches equilibrium. To obtain many samples, we output the state at widely separated times. We will use an example using discrete distributions to illustrate the method using geometric distribution

2.2.2 Geometric Distribution

Suppose $\pi(x) = \theta^x(1 - \theta)$ for $x = 0, 1, 2, ...$ We use a symmetric random walk $q(x, x+1) = q(x, x-1) = 1/2$. As q is symmetric r(x, y) = $\min\{1, \pi(y)/\pi(x)\}$. In this case if $x > 0$, $\pi(x–1) > \pi(x)$ and $\pi(x + 1)/\pi(x) = \theta$ so

$$p(x, x–1) = 1/2 \qquad\qquad p(x, x + 1) = \theta/2$$
$$p(x, x) = \frac{1}{2}–\frac{\theta}{2}$$

When x= 0, $\pi(1) = 0$ so

$$p(0, 1) = 0 \qquad\qquad p(0, 1) = \theta/2$$
$$p(0, 0) = 1–(\theta/2)$$

To check reversibility we consider that if $x \geq 0$ then

$$\pi(x)p(x, x + 1) = \theta^x(1–\theta)(\frac{\theta}{2}) = \pi(x + 1)p(x + 1, x)$$

the choice of q is important. If θ is close to 1 then we would want to choose

$$q(x, x + i) = \frac{1}{2L + 1}$$

for $–L \leq i \leq L$ where $L = O(1/(1–\theta))$ to make the chain move around the state space faster while not having too many steps rejected

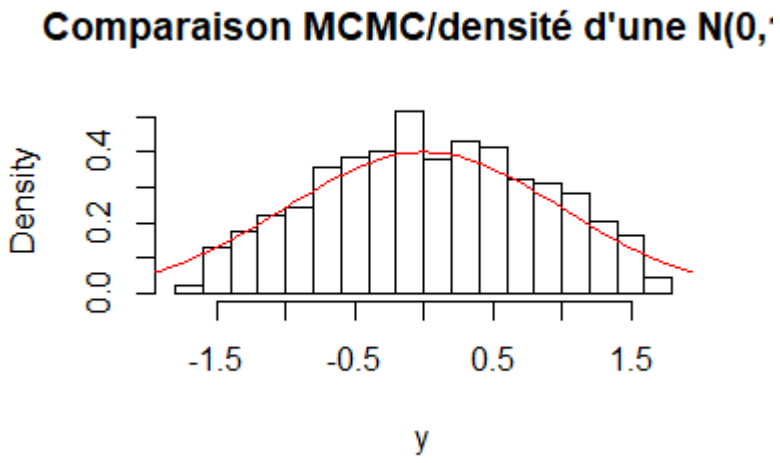
2.2.3 Simulation of the Algorithm MCMC

Algorithm

1. Simulate a candidate value θ^* from a proposal density $p(\theta^*|\theta^{t-1})$.
2. Compute the ratio
$$R = \frac{g(\theta^*)p(\theta^{t-1}|\theta^*)}{g(\theta^{t-1})p(\theta^*|\theta^{t-1})}$$
3. Compute the acceptance probability P = min$\{$R, 1$\}$Sample a value t such that $\theta^t = \theta^*$ with probability P; otherwise θ^t=θ^{t-1}

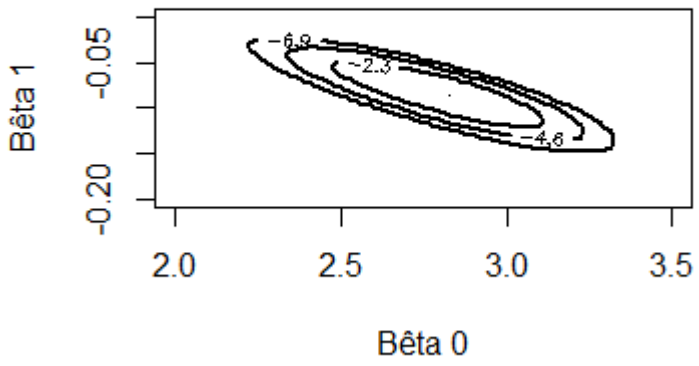
```
######MCMC : Metropolis-Hastings algorithm#####

##First step : simulate a gaussian by hand##
##Hypothesis : Independance algorithm will be used

mcmcnorm <- function(mu,sigma2,ndraws){
  theta <-c(mu) #mu is chosen as first value
  z1 <- qnorm(0.05,mu,sigma2)
  z2 <- qnorm(0.95,mu,sigma2)
  for (i in 1:(ndraws-1)){
    y <- runif(1,z1,z2) #The uniform(z1,z2) is chosen as proposal density
    u <- runif(1,0,1)
    if (u<=min(exp(((theta[i]-mu)**2-(y-mu)**2)/(2*sigma2)),1)){
      theta <-c(theta,y)
    }
    else {
      theta <- c(theta,theta[i])
    }
  }
  return(theta)
}

y <- mcmcnorm(0,1,2000)
hist(y,breaks=20,freq=FALSE,main="Comparaison MCMC/densit d'une N(0,1)")
#Let's compare with the real density
plot(function(x) dnorm(x,0,1),xlim=c(-4,4),col='red',add=TRUE)
```

Here comes the output : it seems to work !



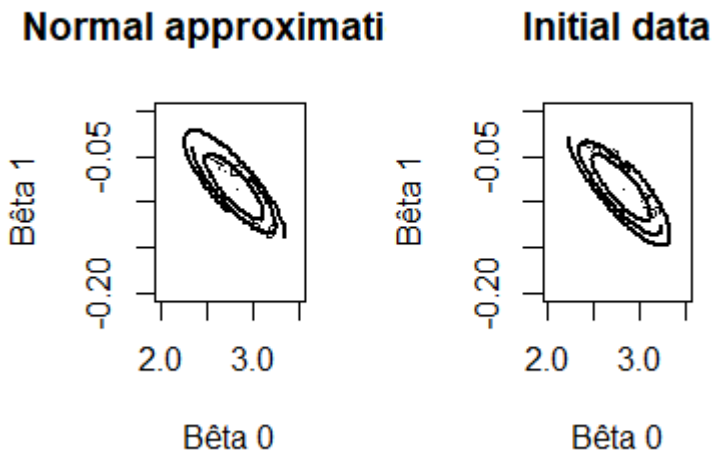

```
#Some function are already in LearnBayes package to compute such
    algorithm
library(LearnBayes)
#Needs the log posterior density and estimated mean and variance
#Here, we exactly know the mean and variance, but in reality, we need to
    estimate them
#In praxis : we will use approximation based on posterior modes
#cf following example

######MCMC output analysis######
##We will test the function of LearnBayes package and analyse the output
##To do that, we will use the data of exercise 4 chapter 5.
#It corresponds to the number of subjects recalling one stressful event
    in function of the number of months

Y<-c(15,11,14,17,5,11,10,4,8,10,7,9,11,3,6,1,1,4)
#To use functions of LearnBayes, we need an approximation of the
    variance/covariance matrix
#We will use the Laplace function :
#Compute the log poserior density :
logposterior = function(theta,data)
{
  n<-length(data)
  B0<-theta[1]; B1<-theta[2]
  s<-0
  for (i in 1:n){
    s<-s+data[i]*(B0+B1*i)-exp(B0+B1*i)
  }
  return(s)}
```

```
#Estimation of mean and standard deviation for B1
#To do that, we need an initial guess of B0,B1.
mycontour(logposterior,c(2,3.5,-0.2,0),Y,xlab="Bta 0",ylab="Bta 1")
```



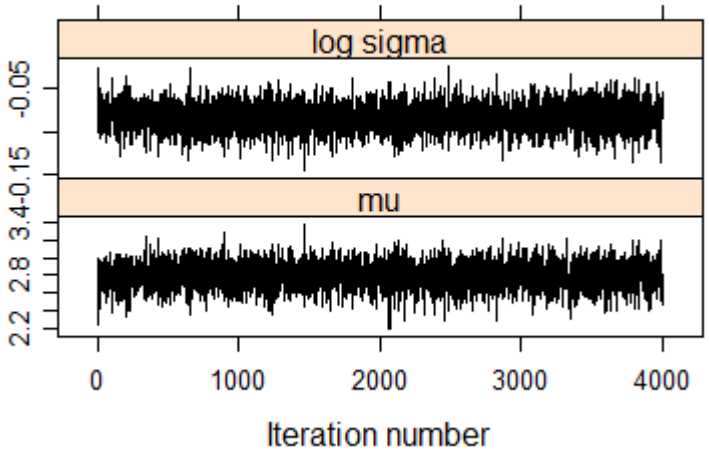
We'll take $\beta_o = 2.5$ and $\beta_1 = -0.05$ to be on the inner contour and so have the best approximation.

```
fit<-laplace(logposterior,c(2.5,-0.05),Y)
fit
#Gives a normal approximation for (B0,B1) with mean $mode and var $var
#Let's compare with the contour of this normal
par<-list(m=fit$mode,v=fit$var)
par(mfrow=c(1,2))
mycontour(lbinorm,c(2,3.5,-0.2,0),par,xlab="Bta 0",ylab="Bta
    1",main="Normal approximation")
mycontour(logposterior,c(2,3.5,-0.2,0),Y,xlab="Bta 0",ylab="Bta
    1",main="Initial data")
```



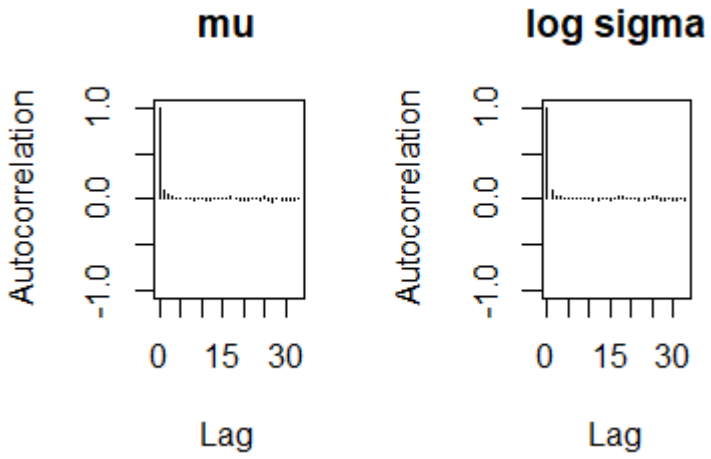
Both contour are nearly the same, so the approximation by the multivariate (Laplace method) is pretty good. We now have an approxmation for the mean and standard deviation of β_0 and β_1.

```
#We can now use the random walk and independant Metropolis-Hastinger :
prop <- list(mu=t(fit$mode),var=fit$var)
bayesfit <-
    indepmetrop(logposterior,proposal=prop,start=c(2.5,-0.05),5000,Y)
bayesfit
bayesfit$accept #High rate of acceptance of the algorithm (0.95)

#Let's analyse the output
library(coda)
dimnames(bayesfit$par)[[2]]=c("mu","log sigma")
#Without the first 1000 samples (we need n to be big enough)
lattice::xyplot(mcmc(bayesfit$par[-c(1:1000),]),col="black")
```



We can see that both values are nearly constant in function of the number of iteration number, because we did not plot the first thousand iteration

```
#What about autocorrelation ?
par(mfrow=c(1,2))
autocorr.plot(mcmc(bayesfit$par),auto.layout = FALSE)
```



There is no autocorrelation, it illustrates the independence hypothesis.

2.3 Hierarchical modeling

Hierarchical modeling can be used for a problem in which we are interested in learning about parameters that are connected in some way.
In Bayesian modeling observations are given a distribution conditional on parameters, and the parameters are supposed to have a distribution called prior distribution conditional to parameters called hyperparameters (either fixed or defined). Hierarchical modeling suppose that these hyperparameters in turn have distributions called hyperpriors.

Hierarchical modeling is also based on the prior belief of *exchangeability* defined as :
For a fixed number of parameters $\theta = (\theta_1, \theta_2, ..., \theta_k)$, we say that θ is exchangeable if the joint probability $P(\theta_1, \theta_2, ..., \theta_k)$ is invariant under permutations of the indices. We can create an exchangeable prior by assuming that the parameters from θ are a random sample from the same prior distribution $\pi(\theta|\lambda)$ and that the vector λ has a hyperprior $\pi_2(\lambda)$

In this section, the author illustrates this modeling with the hearttransplants dataset from the LearnBayes R library. The hierarchical model is made to estimate the true death rates $\lambda_1, ..., \lambda_{94}$ of 94 hospitals.
The true death rates are assumed to be a random sample of a $gamma(\alpha, \frac{\alpha}{\mu})$ distribution. At the second stage, μ is asigned an $inversegamma(a, b)$ distribution (hyperprior) and α a $g(\alpha)$ distribution.

We can observe that this prior distribution induces positive correlation between the true death rates λ_i which is not absurd as we can assume that if one is told about a hospital's true death rate, it would likely affect one's prior on the true rate of another hospital (true rates are most likely similar in size).

In order to simulate the posterior density of the parameters :

- we simulate α, μ from the marginal posterior distribution.
- we simulate $(\lambda_1, ..., \lambda_{94})$ from their distribution (gamma) conditional on the values simulated of (α, μ).

Then with the simulation of the λ_i one can make various inferences like comparing hospitals' death rates : Approximating the probability that ith hospital's death rate is lower than jth hospital one's by sampling λ_i and λ_j.

Chapter 3

Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a generative statistic model used in modeling text corpora and, generally, in collections of discrete data. It uses unobserved groups to explain sets of observations and why some parts of the data are similar. It tends to preserve the main statistical relationships for tasks like classifications, summarization or similarity.

The underlying idea of the latent Dirichlet allocation (LDA) model is that the exchangeability of random variables is essentially interpreted as "conditionally independent and identically distributed," where the conditioning is with respect to an underlying latent parameter of a probability distribution.

To study a text corpora, the first step is to reduce each document in the corpus to a vector of real numbers, a count is then formed of the number of occurences of each word. The next step is to to normalize it, and compare the term frequency to an inverse document frequency count, by measuring the number of occurences of a word in the entire corpus. We finally obtain a matrix X, each colum contains the *tf-idf* (term frequency–inverse document frequency, reflects how important a word is to a document in a collection or corpus) values for each documents in the corpus.

3.1 Notation

In order to introduce the Latent Dirichlet Allocation, we will use the following terminology:

- A *word*: item from a set, which is a *vocabulary* indexed by {1,...,V}. They will be represented as vectors of reals numbers.
- A *document*: sequence of N words, such as $\mathbf{w} = (w_1, ..., w_N)$.
- A *corpus*: collection of N documents, such as $D = \{\mathbf{w}_1, ..., \mathbf{w}_n\}$.

3.2 Definition and algorithm

LDA generative probabilistic model of a corpus, it uses the following algorithm:

1. Choose N $|\xi \sim$ Poisson(ξ)
2. Choose proportions $\theta|\alpha \sim$ Dir(α)
3. For $n \in \{1, ..., N\}$:
   - Choose topic $Z_n|\theta \sim Mult(\theta)$
   - Choose word $W_n|\{z_n, \beta_{1:K}\} \sim Mult(\beta_{z_n})$

This model is a simplified one, K (the Dirichlet distribution dimensionality) is supposed to be fixed, as the word probabilities are. We will discuss this hypothesis later.

Given the parameters α and β, the joint distribution of a topic mixture θ, a set of N topics z, and a set of N words w is given by

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

where p($z_n \mid \theta$) is simply θ_i for the unique i such that $z_n^i = 1$. Integrating over θ and summing over z, we can obtain the marginal distribution of a document:

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

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

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

The LDA model can be represented by the following model (Figure 1) in plate notation (method of representing variables that repeat in a graphical model) where:

- M is the number of documents
- N is the number of words in a document
- α is the parameter of the Dirichlet prior on the document topic distribution (step 2.)
- β is the number of the Dirichlet prior on the topic word distribution
- θ is a vector which contains the topic distribution of each document i
- ϕ is a vector which contains the word distribution of each topic k
- z_{ij} is the topic of the j-th word in document i
- w_{ij} is a word

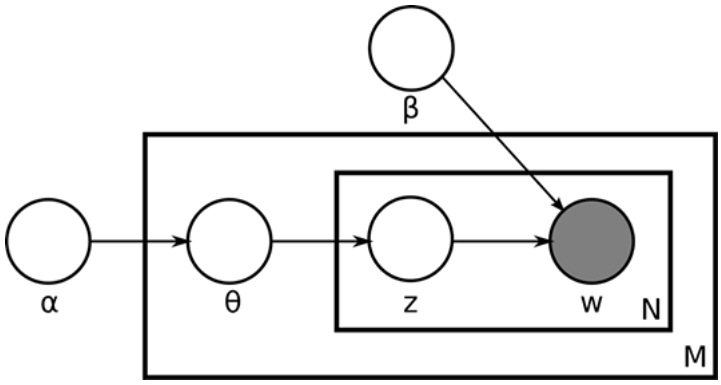


Figure 3.1: Plate notation of the LDA model

Dirichlet distribution

The Dirichlet is a convenient distribution on the simplex . it is in the exponential family and is conjugate to the multinomial distribution.

The Dirichlet distribution of order $K \geq 0$ with parameters $\alpha_1, ..., \alpha_K > 0$ has the following probability density function:

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{i=1}^{K} \alpha_i)}{\prod_{i=1}^{K}(\alpha_i)} \prod_{i=1}^{K} \theta_i^{\alpha_i - 1}$$

3.2.1 Exchangeability

Given a finite set of random variable $\{Z_1, ..., Z_n\}$, this set is exchangeable if the joint distribution is invariant to permutation. For example if π if a permutation of $[|1, N|]$:

$$p(z_1, ..., z_N) = p(z_{\pi(1)}, ..., z_{\pi(N)})$$

If every finite subsequence of an infinite sequence of random variables is exchangeable then it is said to be *infinitely exchangeable*.

By using De Finetti's representation theorem (1990), we can aim that "the joint distribution of an infinitely exchangeable sequence of random variables is as if a random parameter were drawn from some distribution and then the random variables in question were *independant* and *identically distributed*, conditioned on that parameter." []

Latent Dirichlet Allocation considers words generated by topics, those topics being infinitely exchangeable within a document.

De Finetti's theorem gives us the following probability of a sequence of words and topics:

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

where θ is the random parameter of a multinomial over topics.

3.2.2 A continuous mixture of unigrams

LDA be understood as a two-level model, which often studied in the classical hierarchical Bayesian literature. In order to do that, we will let us use the word distribution $p(w|\theta, \beta$:

$$p(w|\theta, \beta_{1:K}) = \sum_z p(w|z, \beta_{1;K}) p(z|\theta)$$

, depending on the random quantity θ.

The generative process for a document **w** will be now:

1. Choose $\theta|\alpha \sim Dir(\alpha)$
2. For n $\in \{1, ..., N\}$:
    3. Choose word $W_n|\theta, \beta_{1:K} \sim Mult(\sum_{i=1}^{K} \theta_{ii}$

Then we define the marginal distribution of a document as a continuous mixture distribution:

$$p(\mathbf{w}|\alpha, \beta_{1:K}) = \int p(\theta|\alpha)(\prod_{n=1}^{N} p(w_n|\theta, \beta_{1:K}))d\theta$$

LDA can be visualized by the Figure 2:

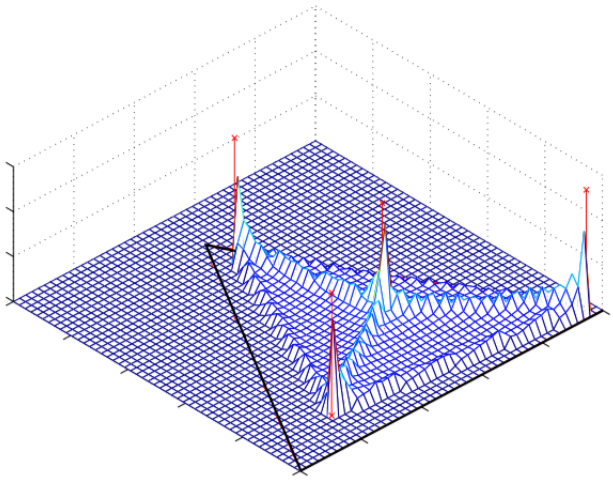


Figure 3.2: Example of a density on unigram distribution $p(w|\theta, \beta_{1:K})$ with LDA for three words and four topics.

3.3 Posterior inference

Now that the idea and the use of LDA have been presented, we can treat the subject of the procedures for posterior inference under LDA.

If we fix the topic distribution $\beta_{1:K}$ and the Dirichlet parameter α, the posterior distribution of the document-specific hidden variables given a document is:

$$p(\theta, \mathbf{z}|\mathbf{w}, \alpha, \beta_{1:K}) = \frac{p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta_{1:K})}{p(\mathbf{w}|\alpha, \beta_{1:K})}$$

Nevertheless we cannot compute this distribution in general, mostly because of the normalization.

Consequently we have to consider approximate inference algorithms for LDA, for example Laplace approximation, variational approximation and Markov chain Monte Carlo. In this work, we will focus on Markov chain Monte Carlo (MCMC).

Chapter 4

Collapsed Gibbs sampling

4.1 Derivation

ϕ^k is a discrete probability distribution over a fixed vocabulary that represents the kth topic distribution, θ_d is a document-specific distribution over the available topics, z_i is the topic index for word w_i, α and β and are hyperparameters for the symmetric.For LDA, we are interested in the latent document-topic portions θ_d, the topic-word distributions ϕ^z, and the topic index assignments for each word z_i. While conditional distributions – and therefore an LDA Gibbs Sampling algorithm – can be derived for each of these latent variables, we can say that both θ_d and ϕ^z can be calculated using just the topic index assignments z_i (i.e. z is a sucient statistic for both these distributions). Therefore, a simpler algorithm can be used if we integrate out the multinomial parameters and simply sample z_i. This is the collapsed Gibbs sampler .Using the collapsed Gibbs sampler for LDA we compute the probability of a topic z being assigned to a word w_i, given all other topic assignments to all other words. Somewhat more formally, we are interested in computing the following posterior up to a constant:

$$p(z_i|z_{i-1}, \alpha, \beta, \mathbf{w})$$

where z_{i-1} means all topic allocations except for z_i. To begin, the rules of conditional probability tell us that:

$$p(z_i|z_{i-1}, \alpha, \beta, \mathbf{w}) = \frac{p(z_i, z_{i-1}, \mathbf{w}|\alpha, \beta)}{p(z_{i-1}, \mathbf{w}|\alpha, \beta)} \propto p(z_i, z_{i-1}, \mathbf{w}|\alpha, \beta) = p(\mathbf{z}, \mathbf{w}|\alpha, \beta)$$

We then have:

$$p(\mathbf{w}, \mathbf{z}|\alpha, \beta) = \int\int p(\mathbf{z}, \mathbf{w}, \theta, \phi|\alpha, \beta) d\theta d\phi$$

The generative process can be described as the following joint distribution (based on the LDA algorithm):

$$p(w, z, \theta, \phi|\alpha, \beta) = p(\phi|\beta)p(\theta|\alpha)p(\mathbf{z}|\theta)p(\mathbf{w}|\phi_z)$$

$$p(\mathbf{w}, \mathbf{z}|\alpha, \beta) = \int\int p(\phi|\beta)p(\theta|\alpha)p(\mathbf{z}|\theta)p(\mathbf{w}|\phi_z)d\theta d\phi$$

Grouping the terms that have dependent variables:

$$p(\mathbf{w}, \mathbf{z}|\alpha, \beta) = \int p(\theta|\alpha)p(\mathbf{z}|\theta)d\theta \int p(\mathbf{w}|\phi_z)p(\phi|\beta)d\phi$$

Both terms are multinomials with Dirichlet priors. As the Dirichlet distribution is conjugate to the multinomial distribution, multiplying the two results in a Dirichlet distribution with an adjusted parameter. Beginning with the first term, we have:

$$\begin{aligned}\int p(\theta|\alpha)p(\mathbf{z}|\theta)d\theta &= \int \prod_i \theta_{z,d_i} \frac{1}{B(\alpha)} \prod_k \theta_{d,k}^{\alpha_k} d\theta_d \\ &= \frac{1}{B(\alpha)} \int \prod_k \theta_{k,w}^{\alpha_k + n_{d,k}} d\theta_d \\ &= \prod_d \frac{B(n_{d,.} + \alpha)}{B(\alpha)}\end{aligned}$$

where $n_{d,k}$ is the number of times words in document d are assigned to topic k, a Δ indicates summing over that index, and $B(\alpha)$ is the multinomial beta function, $B(\alpha) = \frac{\prod_k \Gamma(\alpha_k)}{\Gamma(\sum_k \alpha_k)}$. Similarly, for the second term (calculating the likelihood of words given certain topic assignments):

$$\begin{aligned}\int p(\mathbf{w}|\phi_z)p(\phi|\beta)d\phi &= \int \prod_d \prod_i \phi_{z_{d,i},w_{d,i}} \prod_k \frac{1}{B(\alpha)} \prod_w \phi_{k,w}^{\beta_w} d\phi_k \\ &= \frac{1}{B(\alpha)} \prod_w \phi_{k,w}^{\beta_w + n_{k,w}} d\phi_k \\ &= \prod_k \frac{B(n_{k,.} + \beta)}{B(\beta)}\end{aligned}$$

Combining both the equations, the expanded joint distribution is then:

$$p(\mathbf{w}, \mathbf{z}|\alpha, \beta) = \prod_d \frac{B(n_{d,.} + \alpha)}{B(\alpha)} \prod_k \frac{B(n_{k,.} + \beta)}{B(\beta)}$$

The Gibbs sampling equation for LDA is then derived using the chain rule (where we leave the hyperparameters α and β out for clarity) , the superscript

i signifies leaving the i^{th} token out of the calculation:

$$
\begin{aligned}
p(z_i|\mathbf{z}^{-i}, \mathbf{w}) = \frac{p(\mathbf{w}, \mathbf{z})}{p(\mathbf{w}, \mathbf{z}^{-i})} = \frac{p(\mathbf{z})}{\mathbf{z}^{-i}} \frac{p(\mathbf{w}|\mathbf{z})}{p(\mathbf{w}^{-i}|\mathbf{z}^{-i})p(w_i)} \\
\propto \prod_d \frac{B(n_{d,.} + \alpha)}{B(n_{d,.}^{-i} + \alpha)} \prod_k \frac{B(n_{k,.} + \beta)}{B(n_{k,.}^{-i} + \beta)} \\
\propto (n_{d,k}^{-i} + \alpha_k) \frac{n_{k,w}^{-i} + \beta_w}{\sum_{w'} n_{k,w'}^{-i} + \beta_{w'}}
\end{aligned}
$$

4.2 Implementation

Implementing an LDA collapsed Gibbs sampler is straightforward. It involves setting up the requisite count variables, randomly initializing them, and then running a loop over the desired number of iterations where on each loop a topic is sampled for each word instance in the corpus. Following the Gibbs iterations, the counts can be used to compute the latent distributions θ_d and ϕ_k. The only required count variables include $n_{d,k}$, the number of words assigned to topic k in document d; and $n_{k,w}$, the number of times word w is assigned to topic k. However, for simplicity and efficiency, we count n_k, the total number of times any word is assigned to topic k. Finally, in addition to the obvious variables such as a representation of the corpus $\mathbf{w}$, we need an array z which will contain the current topic assignment for each of the N words in the corpus. Because the Gibbs sampling procedure involves sampling from distributions conditioned on all other variables , before building a distribution from equation fro final equation, we must remove the current assignment from the equation. We do this by decrementing the counts associated with the current assignment because the topic assignments in LDA are exchangeable. We then calculate the (unnormalized) probability of each topic assignment using final equation . The distribution is then sampled from and the chosen topic is set in the $\mathbf{z}$ array and the appropriate counts are then incremented.

Algorithm 1 LDA Gibbs sampler

Input: words $\mathbf{w} \in$ documents $\mathbf{d}$

Output: topic assignments $\mathbf{z}$ and counts $n_{d,k}$, $n_{k,w}$, and n_k

randomly initialize $\mathbf{z}$ and increment counters
for each iteration **do**
    for $i = 0 \rightarrow N$-1 **do**
        $word \leftarrow w[i]$
        $topic \leftarrow z[i]$
        $n_{d,topic} - = 1$; $n_{word,topic} - = 1$; $n_{topic} - = 1$
        for $K = 0 \rightarrow K$-1 **do**
            $p(z = k|.) = (n_{d,k} + \alpha_k) \frac{n_{k,w} + \beta_k}{n_k + \beta * W}$
        end for
        topic$\leftarrow$ sample from $p(z|\cdot)$
        $z[i] \leftarrow$ topic
        $n_{d,topic}$+=1; $n_{word,topic}$+=1; n_{topic}+=1
    end for
end for
return $\mathbf{z}$, $n_{d,k}$, $n_{k,w}$, n_k

Chapter 5

Application of LDA

5.1 Description and Code

To test the algorithm, we used Newyork times data set contains different headlines of the news paper grouped in a text file. Using this data, we want to sample using the LDA Gibbs Sampler, and we will keep the top 10 words for different topics, that means the words with the highest probability for a given topic.

```
# preprocessing the data
import scipy.io
import numpy as np
import argparse

def nytdata_generator(outdir="C:/Users/91970/Documents/MASTERS/YEAR
    2/PROJECT/LDA/data/"):
  mat = scipy.io.loadmat('C:/Users/91970/Documents/MASTERS/YEAR
      2/PROJECT/LDA/data/nyt_data.mat')

  word_id = np.array([i[0].ravel()-1 for i in mat['Xid'].ravel()]) #
      matlab index starting from 1
  word_count = np.array([i[0].ravel() for i in mat['Xcnt'].ravel()])
  vocabulary = np.array([i[0][0] for i in mat['nyt_vocab']])

  D = len(word_count)
  vocab_size = len(vocabulary)
  print('number of documents:',D)
  print('vocabulary size:',vocab_size)

  # generate word count matrix of corpus, doc_cnt.shape = vocab_size, D
  wordcnt_mat = np.zeros((vocab_size,D))
  for d in range(D):
```

```
      wordcnt_mat[word_id[d],d] = word_count[d]

   np.save(outdir+'nytdata_mat.npy', wordcnt_mat)
   np.save(outdir+'nytdata_voc.npy', vocabulary)
   print('The generated nyt data is sucessfully saved in
       {}'.format(outdir))

# LDA Gibbs sampling
# -*- coding: utf-8 -*-


import random
import sys
from scipy.special import gammaln, psi
import numpy as np


class LDAGibbs(object):
    def __init__(self, Datapath, ntopics):
        self.TOPICS = ntopics
        # NUMBER OF TOPICS
        self.documents = {}
        # TRAINING DATA: {DocID: [WordID1, WordID1, WordID2, ...]}
        self.indD = {}                              # MAP DOCUMENT INTO
            INDEX:
        self.indD = {DocID: INDEX}
        self.indV = {}                             # MAP WORD INTO INDEX:
        self.indV = {VocabID: INDEX}
        self.DOCS = 0
        # NUMBER OF DOCUMENTS
        self.VOCABS = 0
        # NUMBER OF VOCABULARIES
        self.alpha = np.ones(self.TOPICS)
        for i in range(self.TOPICS):
            self.alpha[i] *= 0.01
            # np.random.gamma(0.1, 1)
        self.beta = 0.01                            # np.random.gamma(0.1, 1)
        data = np.load(Datapath).T
        self.DOCS = data.shape[0]
        self.VOCABS = data.shape[1]
        self.documents = {}
        for i, doc in enumerate(data):
            tmp_doc = []
            for j, word in enumerate(doc):
                if(word==0):
                    continue
                while(word!=0):
                    tmp_doc.append(j)
                    word -=1
            random.shuffle(tmp_doc)
```

```
            self.documents[i] = tmp_doc

        self.theta = np.zeros([self.DOCS, self.TOPICS])
        self.phi = np.zeros([self.TOPICS, self.VOCABS])

        self.sample_theta = np.zeros([self.DOCS, self.TOPICS])
        self.sample_phi = np.zeros([self.TOPICS, self.VOCABS])
                                # SPACE FOR PHI MATRIX WITH 0s




def LogLikelihood(self):                    # FIND (JOINT)
    LOG-LIKELIHOOD VALUE
    ll = 0
    for z in range(self.TOPICS):
    # Symmetric Dirichlet Distribution (beta distribution in high
        dimension) Words | Topics, beta
        ll += gammaln(self.VOCABS*self.beta) # gamma distribution
        ll -= self.VOCABS * gammaln(self.beta)
        ll += np.sum(gammaln(self.cntTW[z] + self.beta))
        ll -= gammaln(np.sum(self.cntTW[z] + self.beta))
    for doc_num, doc in enumerate(self.documents): # Dirichlet
        Distribution: Topics | Docs, alpha
        ll += gammaln(np.sum(self.alpha)) # Beta(alpha)
        ll -= np.sum(gammaln(self.alpha))
        ll += np.sum(gammaln(self.cntDT[doc_num] + self.alpha))
        ll -= gammaln(np.sum(self.cntDT[doc_num] + self.alpha))
    return ll

def gibbs_update(self, d, w, pos):
    z = self.topicAssignments[d][pos] # old theme
    self.cntTW[z,w] -= 1
    self.cntDT[d,z] -= 1
    self.cntT[z] -= 1

    prL = (self.cntDT[d] + self.alpha) / (self.lenD[d] -1 +
        np.sum(self.alpha))
    prR = (self.cntTW[:,w] + self.beta) / (self.cntT + self.beta *
        self.VOCABS)
    prFullCond = prL * prR
    prFullCond /= np.sum(prFullCond)
    new_z = np.random.multinomial(1, prFullCond).argmax()
    self.topicAssignments[d][pos] = new_z
    self.cntTW[new_z, w] += 1
    self.cntDT[d, new_z] += 1
    self.cntT[new_z] += 1

def update_alpha_beta(self):
```

```
    # Update Beta
    x = 0
    y = 0
    for z in range(self.TOPICS):
        x += np.sum(psi(self.cntTW[z] + self.beta) - psi(self.beta))
        y += psi(np.sum(self.cntTW[z] + self.beta)) - psi(self.VOCABS
            * self.beta)
    self.beta = (self.beta * x) / (self.VOCABS * y)  # UPDATE BETA

    # Update Alpha
    x = 0
    y = 0
    for d in range(self.DOCS):
        y += psi(np.sum(self.cntDT[d] + self.alpha)) -
            psi(np.sum(self.alpha))
        x += psi(self.cntDT[d] + self.alpha) - psi(self.alpha)
    self.alpha *= x / y                    # UPDATE ALPHA

def update_phi_theta(self):
    for d in range(self.DOCS):
        for z in range(self.TOPICS):
            self.sample_theta[d][z] = (self.cntDT[d][z] +
                self.alpha[z])/ (self.lenD[d] + np.sum(self.alpha))
    for z in range(self.TOPICS):
        for w in range(self.VOCABS):
            self.sample_phi[z][w] = (self.cntTW[z][w] + self.beta) /
                (self.cntT[z] + self.beta * self.VOCABS)

def print_alpha_beta(self):
    print('Alpha')
    for i in range(self.TOPICS):
        print(self.alpha[i])
    print('Beta: {}'.format(self.beta))


def run(self,max_iter = 50):
    burnin = max_iter*0.8               # GIBBS SAMPLER
    self.topicAssignments = {}
    self.cntTW = np.zeros([self.TOPICS, self.VOCABS]) # count topic
        to words
    self.cntDT = np.zeros([self.DOCS, self.TOPICS]) # count docs to
        topics
    self.cntT = np.zeros(self.TOPICS)
    self.lenD = np.zeros(self.DOCS)

    # Iterate All the Documents, Initilaze the probibability matrix
    for doc_num,doc in enumerate(self.documents):
        doc_size = len(self.documents[doc])
        tmp = np.random.randint(0,self.TOPICS, size = doc_size)
```

```
            self.topicAssignments[doc_num] = tmp
            for i, word in enumerate(self.documents[doc]):
                self.cntTW[tmp[i],word] += 1
                self.cntDT[doc_num, tmp[i]] += 1
                self.cntT[tmp[i]] += 1
                self.lenD[word] +=1


        print('LIKELIHOOD:\n', self.LogLikelihood())
        self.print_alpha_beta()
        SAMPLES = 0
        for s in range(max_iter):
            print('Iter: {}'.format(s))
            for doc_num, doc in enumerate(self.documents):
                for i, word in enumerate(self.documents[doc]):
                    self.gibbs_update(doc_num, word, i) # word itself is
                        its numerate.
            self.update_alpha_beta()
            print('Likelihood{}'.format(self.LogLikelihood()))
            self.print_alpha_beta()

            if(s>burnin):
                self.update_phi_theta()
                self.theta +=self.sample_theta
                self.phi += self.sample_phi

        self.theta /= (max_iter - burnin-1)
        self.phi /= (max_iter - burnin-1)

# visualization

import numpy as np

def viz(gamma, vocab, topk=20):
    sorted_idx = np.argsort(-gamma, axis=0)
    sorted_gamma = -np.sort(-gamma, axis=0)
    sorted_gamma /= np.sum(sorted_gamma, axis=0)
    _,topic_num = gamma.shape
    for i in range(topic_num):
        print('topic %i:'%(i+1))
        print('top-%i key words:\n'%topk, vocab[sorted_idx[:topk,i]])
        print('distribution of top-%i key
            words:\n'%topk,sorted_gamma[:topk,i])
```

The results are presented in the next section.

5.2 Results

We want 10 words for 4 topics. We represent for each word of the top 10 the count, which is the probability multiplying by 100. The results for 4 topics are on the new page. As expected, some words are far more quoted than others. The algorithm seems to work, especially because clear topics are highlighted. For example for the first topic, top words are *beaker,telescope,space* and *goggles*, clearly referring to the topic of science.

Top words of second topic are *athlete,touchdown* and *running*, referring to the topic of sport. The third topic seems to be science material, while the fourth topic is not so clear. It appears, that we can know quiet fast the topics of these headlines : sport and science.

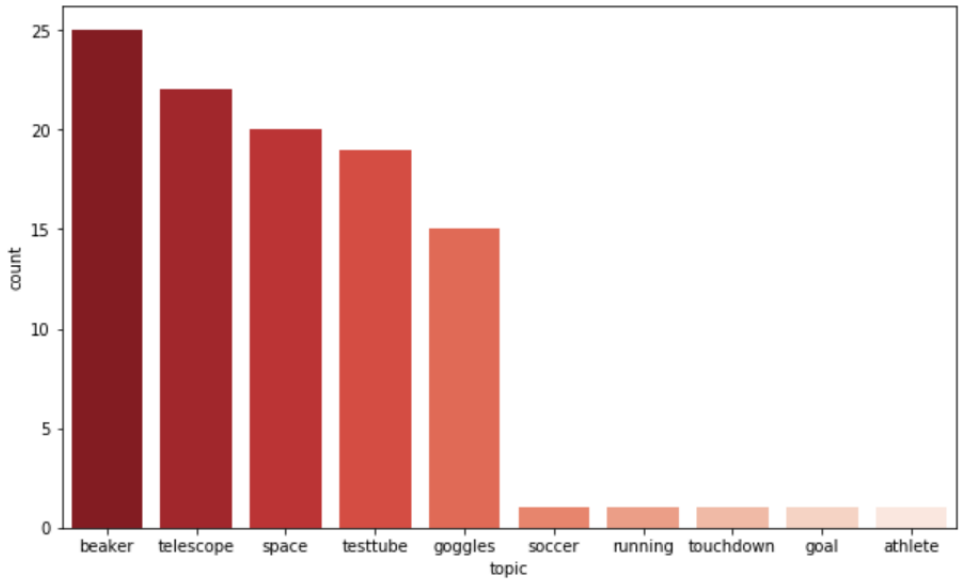


Figure 5.1: Results for topic 1

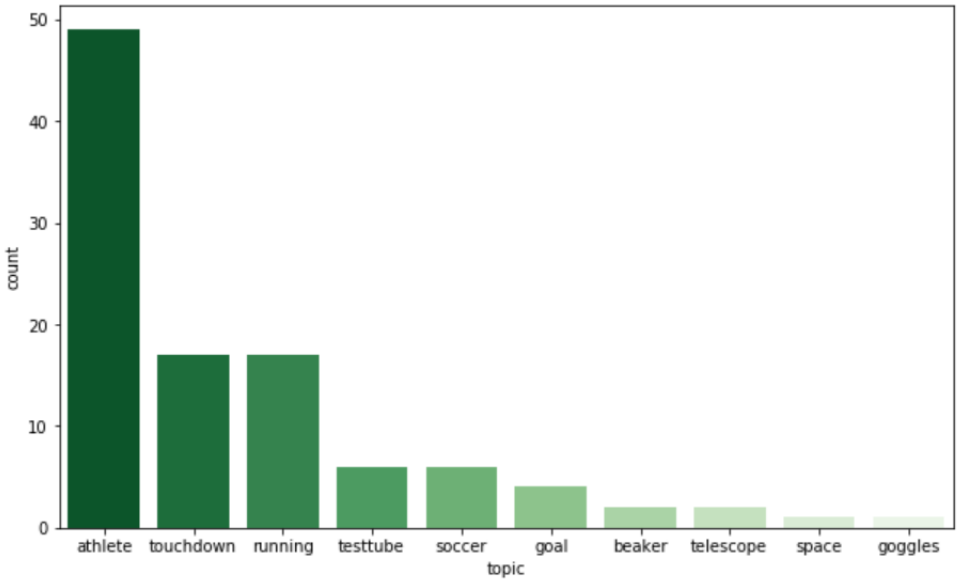


Figure 5.2: Results for topic 2

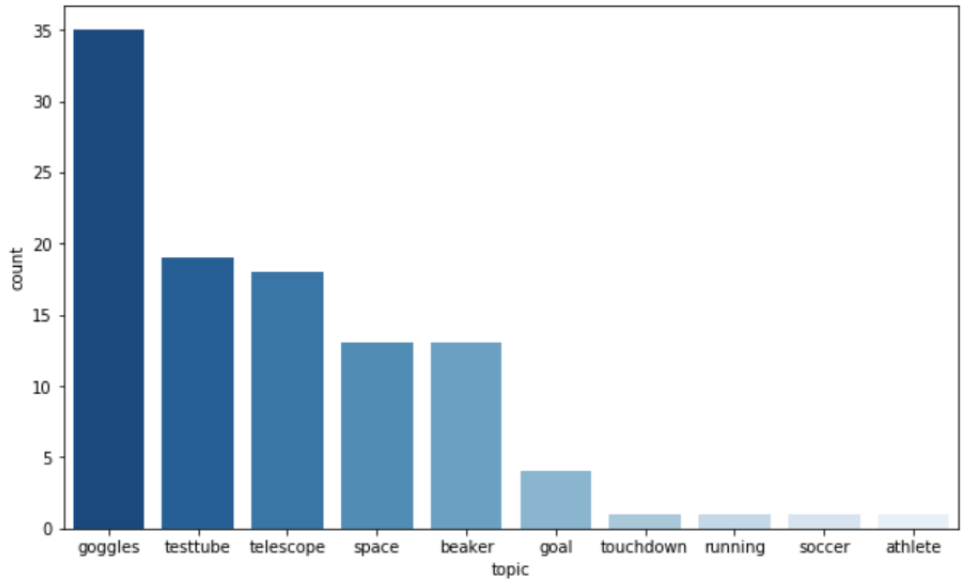


Figure 5.3: Results for topic 3

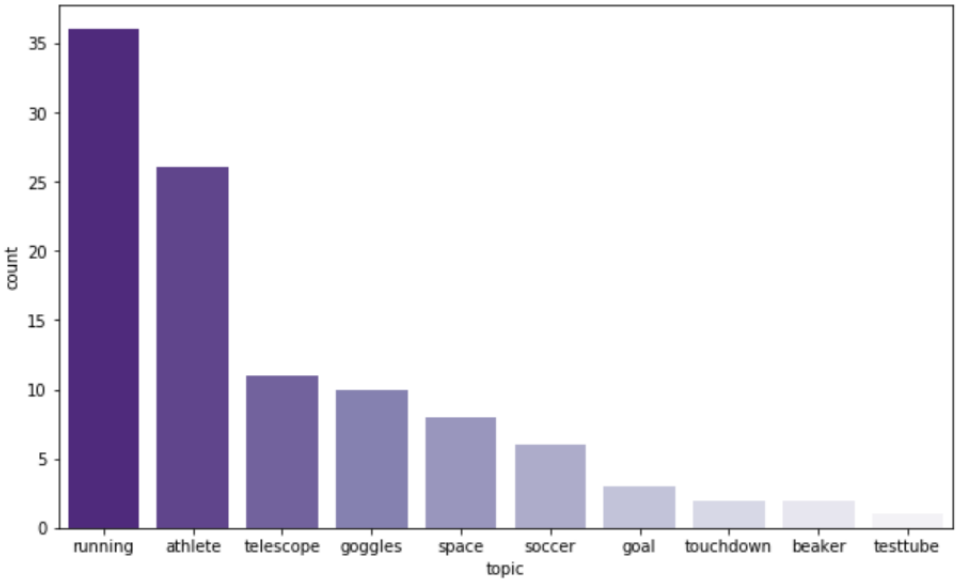


Figure 5.4: Results for topic 4

Chapter 6

Latent Dirichlet Allocation for Online Content Generation

We can now interest ourselves in bigger applications of the LDA, we chose to develop one of them: the Online Content Generation, based on the thesis of Yajia Yang made in UCLA.

6.1 Online Content Generation

Our daily use of internet and online contents like videos, photos or articles is a major field of study for the online publishers. Indeed they have to be aware of the trends, and what people want. It was quite easy in the past, because they just had to follow the evolution of keywords, and make title which matched with them.

Nevertheless, it brought search engine providers to change their algorithms, into a better one, which only pointed out high quality content. As a consequence the task was harder for the online publishers, they had to review their business strategies. But there were too many factors to consider: social popularity, domain authority, competitiveness,... Plus they had to consider than the history would play a great part in the research process.

The idea is to reorganize the database into highly related content group, in order to have higher chances of showing up in people's research when they use the associated keywords. In his thesis, Yajia Yang applies it to livestron.com data, we will not focus on this study or even the theory, but we will point oui the examples he gives in his last chapter.

6.2 Examples of Smart Online Content Generation System

The problematic has already been introduced, our goal is to organize the contents the most efficient way possible.

At first we have to build the topic universe, which must be related to a business environment. It is not enough to just collect the top search queries that people use to go to our site, because our goal is to increase our number of targets. So we use LDA to expand our topic universe, by using the topic representations. Let us see how this works.

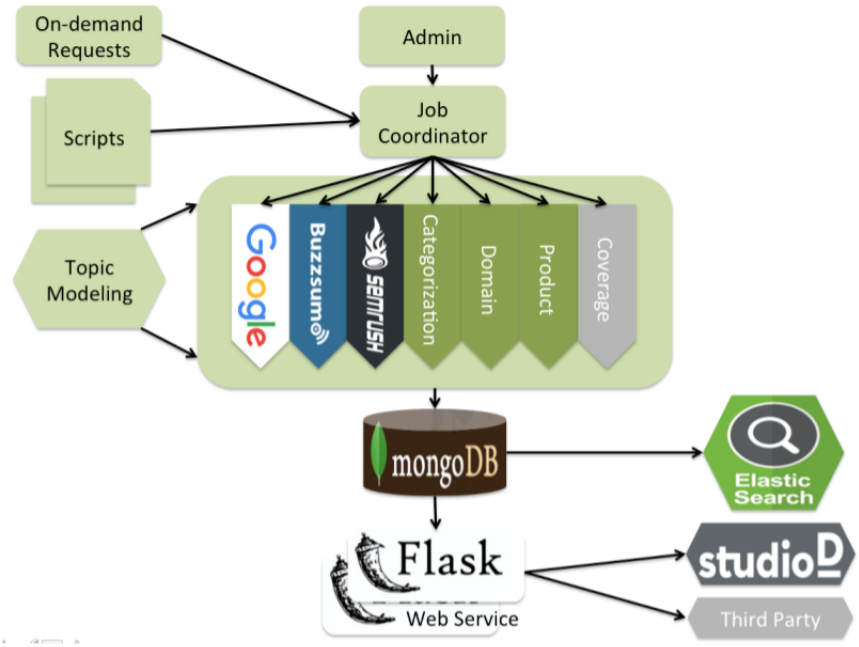


Figure 6.1: Example of an automated online content generation system

The figure 5.1 is a good illustration of the operation. We input in the system a domain name, a list of its competitors' domain name and a list of popular phrases. The system goes online, finds related contents and clusters them into topics. Then its goal is to expand the topic representation, like we presented it. To do so, he uses google auto completes. The next step is to query data and establish a popularity score. For each phrase we search, the system finds associated contents and uses pre-trained LDA model to create category labels. Finally all the reports go through a category filter. This is used for engine optimization process and content generation process.

We can now go in the details of massive applications of the LDA.

6.2.1 Cascaded LDA

Cascaded LDA is used when we have a lot of contents to organize and manage, so we need to create a taxonomy, i.e a classification of things or concepts, including the principles that underlie such classification.

Our goal is still to have a well organization of our contents, we could be penalised by the search engine if it is not the case. The taxonomy used to be made "by hand", by many people, but it had obviously two major drawbacks: the time and the different versions of it, with all the disagreements that could be brought. A good way to do it is to use Cascaded LDA.

The process of cascaded LDA is the following: obtain the second level clusters by running the same process on the first clusters, and continue like this to have the next levels. In this way we build a hierarchical taxonomy, which can go any deep you want. The hierarchically supervised latent Dirichlet allocation allows us to have more information available, and consequently models with a better performance.

6.2.2 In-cluster Similarity

We use article similarity for de-duplication and article recommendations. It is possible that people publish similar or related contents at the same time. We can think of Twitter or Facebook, often stricken by trends, but it is not a problem for them because it allows them to create trending topics. But some social media like Reddit this is a problem, since they prefer their users to refer to a big post instead of separated posts. It is also used for online shopping websites, where people do not systematically create accounts and use them as guests.

We cannot just use one-vs-all similarity, because the amount of data is too big. Actually the solution is to use LDA, at first we cluster the articles, then we can compute the similarity in the cluster. This allows us to obtain a good similarity score while having a reasonable computation time.

6.2.3 Auto Categorization

Finally, it is possible to combine the output of LDA with human knowledge. It is possible to come up with a incorrectly labeled corpus, however it is still impossible to do this "by hand", it would be a huge waste of time, and people would not have the same categorization.

So we need a reasonable model with a reasonable categorization. The idea

is the label the topic manually, and use them for supervised tasks like classification. Then we can categorize all phrases with a trained LDA model and label the phrases with topics. It also allows us to expand our training dataset, each time the algorithm find a related content to a topic.

Chapter 7

Conclusion

Some mathematics prerequisites allowed us to present the Latent Dirichlet Allocation, a generative statistical model, based on unobserved data that contains similar data.

We presented the theory, and implemented an LDA algorithm, using Markov chain Monte Carlo sampling methods (collapsed Gibbs sampling method) from scratch without using any pre-built libraries. We focused on both the calculations and the application of the LDA. We tested our algorithm on the Newyork times database, and obtained quite good results.

Finally, we extended our study to three real world uses cases of LDA model, focusing on the Online Content Generation. It has to be seen that although LDA is a nature language processing model, it is still a powerful tool of machine learning. We could also give other examples such as computer vision machine translation, and many other.

7.1 Continuation

This model can be further extended to Twitter data base where, we can group together tweets occurring in the same user-to-user conversation. Under the scheme,tweets and their replies are aggregated into a single document and the users who posted them are considered as co-authors.

Bibliography

[1] Philippe Carmona. *Notes du cours processus stochastiques.* (French) [*Study notes on stochastic process*] Laboratory Jean Leray, University of Nantes, 2019.

[2] Jim Albert. *Bayesian Computation with R.* Second Edition, Springer, 2009.

[3] Mathieu Ribatet. *Advanced Bayesian Inference.* Ecole Centrale de Nantes, 2020.

[4] David Meir Blei. *Probabilistic models of text and images.* University of California, Berkeley, 2004.

[5] Yajia Yang. *Application of Latent Dirichlet Allocation in Online Content Generation.* University of California, Los Angeles, 2015.

[6] Latent Dirichlet Allocation,
`https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation`

[7] Natural language processing,
`https://en.wikipedia.org/wiki/Natural_language_processing`

[8] Taxonomy for search engines,
`https://en.wikipedia.org/wiki/Taxonomy_for_search_engines`

[9] Efficient Collapsed Gibbs Sampling For Latent Dirichlet Allocation,Han Xiao Thomas Stibor

[10] Fast Collapsed Gibbs Sampling For Latent DirichletAllocation ,Ian PorteousDept ,David Newman, Alexander Ihler,Arthur Asuncion , Padhraic SmythDept, Max Welling