DATA SCIENCE • LATENT DIRICHLET ALLOCATION • TOPIC MODELING • TEXT MINING • R

Latent Dirichlet Allocation - under the hood

BY ANDREW BROOKS JANUARY 17, 2015 4 COMMENTS TWEET LIKE +1

LDA for mortals

I've been intrigued by LDA topic models for a few weeks now. I actually built one before I really knew what I was doing. That kind of frightens and excites me. On one hand, LDA provides rich output which is easy for the humanist researcher to interpret. On the other hand, shouldn't I know what kind of machinery I'm operating that's spitting out results? I would certainly like to know more than what's provided in documentation of whatever software implementation I'm using (currently gensim), but I don't necessarily need to be able to prove every last algorithm before I kick the tires and start building models.

I echo Ted Underwood's observation that it can be difficult to develop a basic intuition of LDA from the hardcore computer science literature. Proving that it works, understanding how it works, and making it work, are very different things. My goal for this exercise was to bridge the gaps in my brain between the technical theoretical papers and the high level LDA articles out there by developing a rudimentary LDA model from scratch.

Generative, so what?

Most academic papers I encountered begin by describing LDA as a generative model. That is, a model that can randomly generate observable data (documents). Blei, Ng, & Jordan, 2003 outline this process in their seminal paper on the topic:

> *LDA assumes the following generative process for each document w in a corpus D:*
> *1. Choose $N \sim$ Poisson(ξ).*
> *2. Choose $\theta \sim$ Dir(α).*
> *3. For each of the N words w_n:*
> *(a) Choose a topic $z_n \sim$ Multinomial(θ).*
> *(b) Choose a word w_n from $p(w_n \mid z_n, \beta)$, a multinomial probability conditioned on the topic z_n.*

I found this confusing at first. I was initially mixing up the generative and inference/training algorithms. They are different, very different. I already have my documents…why would I want to probabilistically generate new documents with my LDA model? Short answer, you probably don't. However, the generative approach lets you do some cool things:

- Assign probabilities to just about everything: words, documents and topics. Probabilistic Latent Semantic Indexing (pLSI), the precursor to LDA, pushed some boundaries in information retrieval, but fell short in providing a probabilistic model at the document level.
- The ability to calculate probabilities (topic assignments) for previously unseen documents, which is not possible with pLSI.
- Model updates. That is, you can continue training your model with new documents when they come in without repeatedly starting from scratch and re-training on the full corpus. Online learning should be supported in most LDA implementations.

There is even some new research (Zhai and Boyd-Graber, 2013) exploring breeds of LDA that allow for infinite vocabularies – assigning probabilities to new documents with words outside of the training vocabulary.

Statistical inference… how we actually train an LDA model

So the goal of LDA is to compute the 2 hidden parameters often commonly denoted as θ and φ, where θ represents topic-document distribution and φ represents the word-topic distribution.

However, when you do some fancy math, it becomes clear that the posterior distribution for these parameters is intractable. That is, we're left with integrals that we cannot compute analytically in high dimensional space where numerical approximation gets hairy. How to handle this issue is the topic of a large swath of the LDA literature. It seems that most advocate one of two methods:

- **Variational inference** seems to be the preferred method for getting fast and accurate results in most software implementations.
- **Markov chain Monte Carlo (sampling)** methods have the benefit of being unbiased and easy understand. However, it seems MCMCs are handicapped by being computationally inefficient when working with large corpora.

Building LDA from scratch

Purpose

I have no intention for anyone (including myself) to use this code for anything beyond pedagogical purposes. There are countless ways to optimize the following approach. I simply wanted to put the theory into action as clearly as possible and dispel any myths in my head that LDA is black magic. I leave the implementation and optimization to the experts.

Getting started

Of all the academic papers I read trying to wrap my head around LDA, I found Streyver & Griffith's Probabilistic Topic Models paper to be the most helpful for understanding implementation. Their 2004 paper, Finding Scientific Topics that everyone seems to cite provides a more thorough derivation of the Baysesian magic, but I found their subsequent paper most useful. They clearly walk through how to conduct inference using Collapsed Gibbs Sampling which I translated into R code.

Edwin Chen's Introduction to Latent Dirichlet Allocation post provides an example of this process using Collapsed Gibbs Sampling in plain english which is a good place to start.

I did find some other homegrown R and Python implementations from Shuyo and Matt Hoffman – also great resources. Especially Shuyo's code which I modeled my implementation after.

So let's code it

I generated a very trivial corpus of 8 documents. To focus just on the LDA mechanics, I opted for the simplest of R objects: lists and matrices.

```
## Generate a corpus
rawdocs <- c('eat turkey on turkey day holiday',
          'i like to eat cake on holiday',
          'turkey trot race on thanksgiving holiday',
          'snail race the turtle',
          'time travel space race',
          'movie on thanksgiving',
          'movie at air and space museum is cool movie',
          'aspiring movie star')
docs <- strsplit(rawdocs, split=' ', perl=T) # generate a list of documents

## PARAMETERS
K <- 2 # number of topics
alpha <- 1 # hyperparameter. single value indicates symmetric dirichlet prior. h
eta <- .001 # hyperparameter
iterations <- 3 # iterations for collapsed gibbs sampling.  This should be a lot
```

```
print(docs)
```

```
## [[1]]
## [1] "eat"     "turkey"  "on"      "turkey"  "day"     "holiday"
##
## [[2]]
## [1] "i"       "like"    "to"      "eat"     "cake"    "on"      "holiday"
##
## [[3]]
## [1] "turkey"       "trot"         "race"         "on"
## [5] "thanksgiving" "holiday"
##
## [[4]]
## [1] "snail"  "race"   "the"    "turtle"
##
## [[5]]
## [1] "time"   "travel" "space"  "race"
##
## [[6]]
## [1] "movie"        "on"           "thanksgiving"
##
## [[7]]
## [1] "movie"  "at"     "air"    "and"    "space"  "museum" "is"     "cool"
## [9] "movie"
##
## [[8]]
## [1] "aspiring" "movie"    "star"
```

Abstracting the words away - now we have a numbers problem.

```
## Assign WordIDs to each unique word
vocab <- unique(unlist(docs))

## Replace words in documents with wordIDs
for(i in 1:length(docs)) docs[[i]] <- match(docs[[i]], vocab)
```

```r
print(docs)
```

```
## [[1]]
## [1] 1 2 3 2 4 5
##
## [[2]]
## [1] 6 7 8 1 9 3 5
##
## [[3]]
## [1]  2 10 11  3 12  5
##
## [[4]]
## [1] 13 11 14 15
##
## [[5]]
## [1] 16 17 18 11
##
## [[6]]
## [1] 19  3 12
##
## [[7]]
## [1] 19 20 21 22 18 23 24 25 19
##
## [[8]]
## [1] 26 19 27
```

Now we need to generate some basic count matrices. First we randomly assign topics to each word in each document. Then we create a word-topic count matrix.

```r
## 1. Randomly assign topics to words in each doc.  2. Generate word-topic count
wt <- matrix(0, K, length(vocab)) # initialize word-topic count matrix
ta <- sapply(docs, function(x) rep(0, length(x))) # initialize topic assignment
for(d in 1:length(docs)){ # for each document
  for(w in 1:length(docs[[d]])){ # for each token in document d
```

```
    ta[[d]][w] <- sample(1:K, 1) # randomly assign topic to token w.
    ti <- ta[[d]][w] # topic index
    wi <- docs[[d]][w] # wordID for token w
    wt[ti,wi] <- wt[ti,wi]+1 # update word-topic count matrix
  }
}
```

```
print(wt) # Word-topic count matrix
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    2    2    0    0    1    1    0    0    1     1     2     1     0
## [2,]    0    1    4    1    2    0    1    1    0     0     1     1     1
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
## [1,]     0     0     0     0     2     1     0     1     1     1     1
## [2,]     1     1     1     1     0     3     1     0     0     0     0
##      [,25] [,26] [,27]
## [1,]     1     0     0
## [2,]     0     1     1
```

```
print(ta) # Token-topic assignment list
```

```
## [[1]]
## [1] 1 1 2 1 2 2
##
## [[2]]
## [1] 1 2 2 1 1 2 1
##
## [[3]]
## [1] 2 1 1 2 2 2
##
## [[4]]
## [1] 2 2 2 2
```

```
##
## [[5]]
## [1] 2 2 1 1
##
## [[6]]
## [1] 2 2 1
##
## [[7]]
## [1] 1 2 1 1 1 1 1 1 2
##
## [[8]]
## [1] 2 2 2
```

Now we generate a document-topic count matrix where the counts correspond to the number of tokens assigned to each topic for each document.

```
dt <- matrix(0, length(docs), K)
for(d in 1:length(docs)){ # for each document d
  for(t in 1:K){ # for each topic t
    dt[d,t] <- sum(ta[[d]]==t) # count tokens in document d assigned to topic t
  }
}
```

```
print(dt) # document-topic count matrix
```

```
##      [,1] [,2]
## [1,]    3    3
## [2,]    4    3
## [3,]    2    4
## [4,]    0    4
## [5,]    2    2
## [6,]    1    2
## [7,]    7    2
## [8,]    0    3
```

I'm beginning to mix notation from the code and theory. Oops. Anything in *italics* refers to theory. Anything in an `inline code block` refers to my iterators and objects in R code.

If following along with Streyvers and Griffiths, `wt` corresponds to the C^{WT} matrix and `dt` corresponds to the C^{DT} matrix.

Now we get to the fun part where LDA starts learning topics. We will iteratively update the random (bad) topic assignments we populated into `ta` one token at a time. This is where I had to brush up on my Bayesian statistics.

Rather than directly estimating our parameters of interest (φ and θ), we can directly esimate the posterior over **z** which represents topic assignments for each token. This is `p_z`, the full-conditional distribution for **z** which represents the probability that token `w` belongs to topic `t`.

The left side of `p_z` (`(wt[,wid] + eta) / denom_b`) represents the probability of word `w` under topic `t`. After many tokens of a word have been assigned to a particular topic, LDA increasingly wants to assign new occurrences of this token to that topic.

The right side of `p_z` (`(dt[d,] + alpha) / denom_a`) represents the probablity of topic `t` in document `d`. If many tokens in a document have been assigned to a particular topic, LDA wants to assign tokens to to that topic at each iteration of Gibbs Sampling.

Example: So if 90% of the tokens in a document belong to topic 1 at the start of an iteration of Gibbs Sampling AND those same tokens in other documents are assigned to topic 1, it's a no-brainer for LDA. Topic 1 it is.

However, if those 90% of tokens that belonged to topic 1 in our document all belong to topic 2 in the other documents, then it's a toss-up and LDA is forced to spread the wealth across the two topics.

Defining some notation…

> `p_z` = *$P(z_i = j \mid z_{-i}, w_i, d_i)$*
> *where*
> *$P(z_i = j)$ is the probability that token i is assigned to topic j*
> *z_{-i} represents topic assignments of all other tokens*
> *w_i is the word index of token i (1…* `length(vocab)` *)*
> *d_i is the document index of token i (1…* `length(docs)` *)*

This is really the only "magic" part of the whole LDA learning process far. To dispel the magic and truly understand why this works read Griffiths and Streyvers – they walk through the derivation of the full-conditional distribution (`p_z`) much more elegantly than I could.

```
for(i in 1:iterations){ # for each pass through the corpus
  for(d in 1:length(docs)){ # for each document
    for(w in 1:length(docs[[d]])){ # for each token

      t0 <- ta[[d]][w] # initial topic assignment to token w
      wid <- docs[[d]][w] # wordID of token w

      dt[d,t0] <- dt[d,t0]-1 # we don't want to include token w in our document-
      wt[t0,wid] <- wt[t0,wid]-1 # we don't want to include token w in our word-

      ## UPDATE TOPIC ASSIGNMENT FOR EACH WORD -- COLLAPSED GIBBS SAMPLING MAGIC
      denom_a <- sum(dt[d,]) + K * alpha # number of tokens in document + number
      denom_b <- rowSums(wt) + length(vocab) * eta # number of tokens in each to
      p_z <- (wt[,wid] + eta) / denom_b * (dt[d,] + alpha) / denom_a # calculati
      t1 <- sample(1:K, 1, prob=p_z/sum(p_z)) # draw topic for word n from multi

      ta[[d]][w] <- t1 # update topic assignment list with newly sampled topic f
      dt[d,t1] <- dt[d,t1]+1 # re-increment document-topic matrix with new topic
      wt[t1,wid] <- wt[t1,wid]+1 #re-increment word-topic matrix with new topic

      if(t0!=t1) print(paste0('doc:', d, ' token:' ,w, ' topic:',t0,'=>',t1)) #
    }
  }
}
```

```
## [1] "doc:1 token:2 topic:1=>2"
## [1] "doc:1 token:4 topic:1=>2"
## [1] "doc:1 token:5 topic:2=>1"
## [1] "doc:1 token:6 topic:2=>1"
## [1] "doc:2 token:1 topic:1=>2"
## [1] "doc:2 token:3 topic:2=>1"
## [1] "doc:3 token:2 topic:1=>2"
## [1] "doc:3 token:3 topic:1=>2"
## [1] "doc:3 token:5 topic:2=>1"
## [1] "doc:3 token:6 topic:2=>1"
## [1] "doc:5 token:1 topic:2=>1"
## [1] "doc:5 token:2 topic:2=>1"
## [1] "doc:5 token:4 topic:1=>2"
## [1] "doc:7 token:1 topic:1=>2"
## [1] "doc:7 token:2 topic:2=>1"
## [1] "doc:7 token:6 topic:1=>2"
## [1] "doc:7 token:7 topic:1=>2"
## [1] "doc:1 token:5 topic:1=>2"
## [1] "doc:2 token:2 topic:2=>1"
## [1] "doc:2 token:3 topic:1=>2"
## [1] "doc:2 token:5 topic:1=>2"
## [1] "doc:7 token:3 topic:1=>2"
## [1] "doc:7 token:4 topic:1=>2"
## [1] "doc:7 token:8 topic:1=>2"
## [1] "doc:8 token:1 topic:2=>1"
## [1] "doc:8 token:3 topic:2=>1"
## [1] "doc:1 token:5 topic:2=>1"
## [1] "doc:2 token:2 topic:1=>2"
## [1] "doc:2 token:3 topic:2=>1"
## [1] "doc:2 token:5 topic:2=>1"
## [1] "doc:4 token:4 topic:2=>1"
## [1] "doc:5 token:1 topic:1=>2"
## [1] "doc:5 token:2 topic:1=>2"
## [1] "doc:7 token:2 topic:1=>2"
```

The printed output above is a window into the learning process. We printed a line each time a token got reassigned to a new topic. Remember we only made 3 passes (`iterations <- 3`) through the corpus, so our topic assignments are likely still pretty terrible. In practice, with many more iterations, these re-assignments would eventually stabilize.

So now we've finished learning topics. We just have to analyze them. `theta` normalizes the counts in the document-topic count matrix `dt` to compute the probability that a document belongs to each topic. `theta` is technically the posterior mean of the parameter θ.

```
theta <- (dt+alpha) / rowSums(dt+alpha) # topic probabilities per document
print(theta)
```

```
##        [,1]   [,2]
## [1,] 0.5000 0.5000
## [2,] 0.5556 0.4444
## [3,] 0.3750 0.6250
## [4,] 0.3333 0.6667
## [5,] 0.3333 0.6667
## [6,] 0.4000 0.6000
## [7,] 0.1818 0.8182
## [8,] 0.6000 0.4000
```

`phi` normalizes the word-topic count matrix `wt` to compute the probabilities of words given topics. `phi` is technically the posterior mean of the parameter φ.

```
phi <- (wt + eta) / (rowSums(wt+eta)) # topic probabilities per word
colnames(phi) <- vocab
print(phi)
```

```
##           eat    turkey         on       day  holiday         i      like
## [1,] 0.133160 6.655e-05 6.655e-05 0.066613 0.199707 6.655e-05 6.655e-05
## [2,] 0.000037 1.110e-01 1.480e-01 0.000037 0.000037 3.704e-02 3.704e-02
##            to      cake       trot      race thanksgiving     snail
## [1,] 0.066613 0.066613 6.655e-05 6.655e-05     0.133160 6.655e-05
## [2,] 0.000037 0.000037 3.704e-02 1.110e-01     0.000037 3.704e-02
##            the    turtle      time     travel     space     movie        at
## [1,] 6.655e-05 0.066613 6.655e-05 6.655e-05 0.133160 6.655e-05 6.655e-05
## [2,] 3.704e-02 0.000037 3.704e-02 3.704e-02 0.000037 1.480e-01 3.704e-02
##            air       and    museum        is      cool aspiring      star
## [1,] 6.655e-05 6.655e-05 6.655e-05 6.655e-05 6.655e-05 0.066613 0.066613
## [2,] 3.704e-02 3.704e-02 3.704e-02 3.704e-02 3.704e-02 0.000037 0.000037
```

4 Comments simpleblog 1 Login

Recommend 3 Share Sort by Best

Join the discussion…

Roberta Coeli • 5 months ago

Thanks a bunch for the explanation! Nice article, congrats!

Reply • Share ›

ethen • a year ago

Thank you for coding this up andrew! I reimplemented it a bit at the following link (to make the notations a bit clearer, i hope...).

http://ethen8181.github.io/mac...

Another quick question, where can your gibbs sampling code be optimized in R?

Reply • Share ›

Andrew Brooks Mod → ethen • a year ago

Glad you found it helpful, and awesome extension on my original post! As I'm looking back on it, wrapping it up in a function as you did, is a great idea and makes it easier to jump back in.

I wrote this post to understand LDA in R, but I actually used gensim in Python to do the topic modeling for the side project I was experimenting on. I did some Bayesian Statistics in grad school using R, but we coded gibbs sampling mostly from scratch as I recall. I've seen gibbs.met package, but haven't personally used it. Have you seen this article? this https://darrenjw.wordpress.com...

Reply • Share ›

ethen → Andrew Brooks • a year ago

Thanks for providing the link, I had a look at it!~ Though I must admit I'm not that familiar with C++. Thus, I'll probably come back to it later if I really need the speed boost of using things other than pure R ....

Reply • Share ›

ALSO ON **SIMPLEBLOG**

Interactive association rules exploration app

15 comments • a year ago•

Ahdieh — Hi Raj, I ran into the same error. How did you solve it? Thanks!

Render reports directly from R scripts

7 comments • a year ago•

Andrew Brooks — Hi Luis Zertuche thanks for the feedback! To my knowledge, if you want to continuously tweak small things like text or

New York Times Article Search API to MongoDB

1 comment • a year ago•

prasika — very useful...thank u

Blogging with Jekyll and R Markdown using knitr

16 comments • a year ago•

Andrew Brooks — Hi Eduardo - Great feedback, Thanks for posting as others may find this useful as well! ... and as I haven't

Subscribe Add Disqus to your site Add Disqus Add Privacy DISQUS

Previous Next