LDA

Aidan Boland

2015-10-15

Latent Dirichlet Allocation

This documents is a summary of Latent Dirichlet Allocation, a method first presented in Blei et al., 2002.

Notation

The data is composed as follows,

- A vocabulary of V possible words $(w^1, ..., w^V)$.
- A corpus of M individual documents, $\mathbf{D} = (\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_M)$.
- Document d is composed of N_d words, $\mathbf{w}_d = (w_{d,1}, w_{d,2}, ..., w_{d,N_d})$.
- There are k topics $(1, ..., k)$.

For consitency the following letters will be used as sub and super scripts,

- i for topic $(1, ..., k)$.
- j for word in vocabulary $(1, ..., V)$.
- n for word in document $(1, ..., N_d)$.
- d for document $(1, ..., M)$.

Generating a document

The following steps show how document $\mathbf{w}_d$ is generated.

1. Choose the number of words in the document, $N_d \sim Po(\xi)$.
2. Choose the probabilities of the topics for the document, $\theta_d \sim Dir(\alpha)$. ($\theta$ is a $1 \times k$ vector)
3. For each of the N_d words $w_{d,n}$,
    - Choose a topic $z_{d,n} \sim Mult(\theta_d)$, ($z_{d,n}$ is a single value from 1 to k, the topic of word n in doc d)
    - Choose a word $w_{d,n} \sim Mult(\phi_{z_{d,n}})$,

where $\phi \sim Dir(\beta)$. The choices of the hyperparameters α and β will be discussed further in the next section. Figure 3 shows a graphical representation of LDA.

Parameters

The word probabilities are paramterised by the $k \times V$ matrix ϕ,

$$P(w_{d,n} = j | z_{d,n} = i) = \phi_{j,i}$$

The topic probabilities for each document are paramterised by the vector θ_d

$$P(z_{d,n} = j) = \theta_{d,j}$$

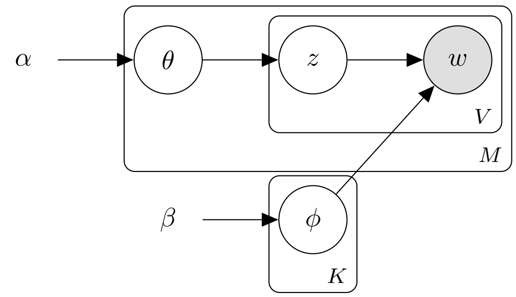

Figure 1: Graphical model for LDA.

Latent dirichlet as presented in Blei et al., 2002 is a problem of maximising the following,

$$P(D|\phi, \alpha) = \int P(D|\phi, \theta)P(\theta|\alpha)d\theta,$$

this can be achieved using variational inference an the EM algorithm. (note, in the Blei paper they do not put a prior on ϕ)

Gibbs sampling

An alternative efficient procedure is collapsed Gibbs sampling (alternative paper) where θ and ϕ are marginalised out and only the latent variables $\mathbf{z}$ are sampled. After the sampler has burned-in we can calculate an estimate of θ and ϕ given $\mathbf{z}$.

The posterior distribution of interest in this case is,

$$P(\mathbf{z}|D) = \frac{P(D, \mathbf{z})}{\sum_{\mathbf{z}} P(D, \mathbf{z})} \propto P(D|\mathbf{z})P(\mathbf{z})$$

The joint distribution, $P(D, \mathbf{z})$, can be found through integrating

$$P(\theta, \phi, \mathbf{z}, D|\alpha, \beta) = P(D|\mathbf{z}, \phi)P(\phi|\beta)P(\mathbf{z}|\theta)P(\theta|\alpha)$$

with respect to ϕ and θ.
Integrating with repsect to ϕ gives the following,

$$p(D|\mathbf{z}) = \prod_{i=1}^{K} \left[\frac{\Gamma\left(\sum_{j=1}^{V} \beta_{i,j}\right)}{\prod_{j=1}^{V} \Gamma(\beta_{i,j})} \times \frac{\prod_{j=1}^{V} \Gamma\left(C_{j,\cdot}^{i} + \beta_{i,j}\right)}{\Gamma\left(\sum_{j=1}^{V} C_{j,\cdot}^{i} + \beta_{i,j}\right)} \right] \quad (1)$$

while integrating with respect to θ gives,

$$p(\mathbf{z}) = \prod_{d=1}^{M} \left[\frac{\Gamma\left(\sum_{j=1}^{k} \alpha_j\right)}{\prod_{j=1}^{k} \Gamma(\alpha_j)} \times \frac{\prod_{j=1}^{k} \Gamma\left(C_{j,d}^{\cdot} + \alpha_j\right)}{\Gamma\left(\sum_{j=1}^{k} (C_{j,d}^{\cdot} + \alpha_j)\right)} \right] \quad (2)$$

Where $C_{j,d}^{i,-(d,n)}$ is the count of words w^j that have topic i in document $\mathbf{w}_d$ not including the word $w_{d,n}$. The notation $\cdot$ is a sum over that subscript,
e.g. $C_{j,\cdot}^{i,-(d,n)}$ is the number of words w^j that are assigned to topc j in the full corpus not including the word $w_{d,n}$.

The full conditional distribution $P(z_{d,n} = i|\mathbf{z}_{-(\mathbf{d},\mathbf{n})}, w_{d,n} = j, D, \alpha, \beta)$ can be found using equations (1) and (2),

$$P(z_{d,n} = i|\mathbf{z}_{-(\mathbf{d},\mathbf{n})}, w_{d,n} = j, D, \alpha, \beta) \propto \frac{\beta_{i,j} + C_{j,\cdot}^{i,-(d,n)}}{\beta_{i,\cdot} + C_{\cdot,\cdot}^{i,-(d,n)}} \frac{\alpha_i + C_{\cdot,d}^{i,-(d,n)}}{\alpha_\cdot + C_{\cdot,d}^{\cdot,-(d,n)}}$$

The labels in the corpus can be updated with Gibbs sampling using full conditional distribution.

Implementation

The following code can be used to update the labels using the collapsed Gibbs sampler as described above. The first function creates documents given a set of priors.

```r
library(MCMCpack)  # for dirichlet dist, alternative is library(gtools)

GenerateLDA <- function(alpha, beta, xi, Vocab, Ndoc){
   # function to generate a single document
   # arguments:
   #   alpha, k x 1 concentration paramter for topics,(take input as single value for now)
   #   beta, k x V each row represents the concentration parameter for words in topic k
   #   xi, parameter to simulate length of documents
   #   Vocab, vector of size V containing all possible words
   #   ndoc, number of documents to simulate
   #
   # todo:
   #   change output to document term matrix rather than list....

   alpha <- rep(alpha, nrow(beta))

   if(dim(beta)[2] != length(Vocab))
      stop("Incorrect dimension for Beta!!")

   docs <- list()  # blank list to store words, documents diff size so array
                   # is not suitable, could use DTM alternatively...
   z_out <- NULL
   for(i in 1:Ndoc){
      N <- rpois(1, lambda = xi)  # Choose the length of the ith doc
      theta <- rdirichlet(n = 1, alpha = alpha)

      z <- rmultinom(1, 1, prob = theta)  # Choose a topic for 1st word in doc i
      z_out <- c(z_out, which(z == 1))
      phi <- rdirichlet(n = 1, alpha = beta[which(z == 1),])  # generate probs
      docs[[i]] <- Vocab[which(rmultinom(1, 1, prob = phi) == 1)]  # Initialise ith doc

      for(j in 2:N){
         z <- rmultinom(1, 1, prob = theta)  # Choose a topic for jth word in doc i
         z_out <- c(z_out, which(z == 1))
         phi <- rdirichlet(n = 1, alpha = beta[which(z == 1),])
         docs[[i]] <- c(docs[[i]], Vocab[which(rmultinom(1, 1, prob = phi) == 1)])
      }
   }
   list(docs = docs, z = z_out)
}
```

```r
my_topic_counts <- function(dtm, z, n_topics){
  # Function to calculate topic counts for words and documents
  # arguments:
  #   dtm is a document term matrix in triplet form
  #     dtm$i denotes the document
  #     dtm$j denotes the word
  #     dtm$v is the number of occurences of word j in doc i
  #   z is a set of topic labels for each word

  word <- rep(dtm$j,dtm$v)
  doc <- rep(dtm$i,dtm$v)

  word_count <- array(0,c(n_topics,ncol(dtm)))
  doc_count <- array(0,c(n_topics,nrow(dtm)))

  for(w in 1:length(doc)){
      word_count[z[w],word[w]] <- word_count[z[w],word[w]] + 1
      doc_count[z[w],doc[w]] <- doc_count[z[w],doc[w]] + 1
  }
  list(word_count, doc_count)
}

my_lda_gibbs <- function(dtm, n_topic = 2, iterations = 10){

   n_vocab <- ncol(dtm)  # Number of unique words
   alpha <- rep(1, n_topic)  # prior on theta
   beta <- array(1,c(n_topic, n_vocab))  # prior on phi
   theta <- array(0,c(nrow(dtm),n_topic))
   phi <- array(0,c(n_topic, n_vocab))

   doc <- rep(dtm$i,dtm$v)
   word <- rep(dtm$j,dtm$v)

   z <- array(0, c(iterations + 1, length(doc)))
   z[1, ] <- sample(1:n_topic, length(doc), replace = TRUE)  # choose initial labels
   if(length(unique(z[1, ])) != n_topic)
      stop("Number of topics in z vector do not match argument!\n")

   initial_counts <- my_topic_counts(dtm, z[1, ], n_topics =n_topic)
   word_count <- initial_counts[[1]]
   doc_count <- initial_counts[[2]]
   topic_prob <- array(0, n_topic)

   for(iter in 1:iterations){
      for(w in 1:length(word)){
         # Loop over each word
         for(j in 1:n_topic){
            if(z[iter, w] == j){
                topic_prob[j] <- prod(sum(beta[j, word[w]], word_count[j, word[w]], -1),
                                      sum(alpha[j], doc_count[j, doc[w]], -1)) /
                  prod(sum(beta[j, ], word_count[j, ], -1),
                     (sum(alpha, doc_count[, doc[w]], -1)))
            }else{
```

```
                topic_prob[j] <- prod(sum(beta[j, word[w]], word_count[j, word[w]]),
                             sum(alpha[j], doc_count[j, doc[w]])) /
                  prod(sum(beta[j, ], word_count[j, ]),
                     sum(alpha, doc_count[, doc[w]]))
            }
         }
         z[iter + 1, w] <- sample.int (n_topic, size = 1, prob = topic_prob)

         if(z[iter + 1, w] != z[iter, w]){
            word_count[z[iter, w], word[w]] <- word_count[z[iter, w], word[w]] - 1
            doc_count[z[iter, w], doc[w]] <- doc_count[z[iter, w], doc[w]] - 1
            word_count[z[iter + 1, w], word[w]] <- word_count[z[iter + 1, w], word[w]] + 1
            doc_count[z[iter + 1, w], doc[w]] <- doc_count[z[iter + 1, w], doc[w]] + 1
         }
      }
      # Old code when counts updated after each full sweep of words
      #update_counts <- my_topic_counts(dtm, z[iter + 1, ], n_topic)
      #word_count <- update_counts[[1]]
      #doc_count <- update_counts[[2]]
  }
  for(j in 1:n_topic)
     phi[j, ] <- (beta[j, ] + word_count[j, ]) / (sum(beta[j, ]) + sum(word_count[j, ]))
  for(d in 1:nrow(dtm))
     theta[d, ] <- (alpha + doc_count[, d]) / (sum(alpha) + sum(doc_count[, d]))

  list(z, t(phi), theta)
}
```

Testing the functions

We can now implement these functions using a sample set of words. The corpus was created using a specific set of β prior values.

```
my_vocab <- c("car", "engine", "exhaust", "wheel",
              "milk", "cream", "dairy", "yogurt",
              "coke", "water", "juice", "coffee", "drink", "bottle", "can")
n_topics <- 3
# my_beta <- matrix(1, nrow = n_topics, ncol = length(my_vocab))
my_beta <- matrix(c(10, 10, 10, 10, 01, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
                    0.1, 0.1, 0.1, 0.1, 10, 10, 10, 10, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
                    0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 10, 10, 10, 10, 10, 10, 10),
                  byrow = T, nrow = n_topics, ncol = length(my_vocab))
my_corpus <- GenerateLDA(alpha = 1, beta = my_beta, xi = 10, Vocab = my_vocab, Ndoc = 100)
library(tm)  # Library to create DTM
my_dtm <- DocumentTermMatrix(Corpus(VectorSource(my_corpus$docs)))
```

The corpus was then analysed using a pre-built function from the ***topicmodels*** package (`LDA`) and the function created above (`my_lda_gibbs`). The priors for the Gibbs steps were left vague.

```
library(topicmodels)
time_pacakge <- system.time(
  control_gibbs <- LDA(my_dtm, k = 3, method="Gibbs", iterations = 3000, burn.in=1000)
```

```
)

time_mine <- system.time(
  my_gibbs <- my_lda_gibbs(my_dtm, n_topic = 3, iterations = 3000)
)
```

The `LDA` function took 0.11 seconds to run while the `my_lda_gibbs` function took 89.51 seconds. The `LDA` function is optimised to run in `c`, this is where it gains it's time advantage. One problem is that neither of the functions seem to converge well. The tables below compare the 2 algorithms with the true values for each word in the corpus.

```
my_gibbs_z <- apply(my_gibbs[[1]][-1000, ], 2, median)
table(z = my_corpus$z, my_gibbs = my_gibbs_z)
```

```
   my_gibbs
z     1   2   3
  1  65  74 161
  2  75 204  63
  3 191  83  67
```

```
table(z = my_corpus$z, control_gibbs@z)
```

```

z     1   2   3
  1 170  68  62
  2  68 195  79
  3  83  81 177
```

```
table(package_gibbs = control_gibbs@z, my_gibbs = my_gibbs_z)
```

```
             my_gibbs
package_gibbs   1   2   3
            1  29   4 288
            2   1 342   1
            3 301  15   2
```

Seems to do ok, not fantastic. There may be a label matching problem but this not too big of a deal, it is a standard problem in clustering. What is nice is that both algorithms seem to get the same results (last table). This indicates that the functions created in this document are correct.

Seeded topics

One simple solution to incorporating seed words into the LDA model is presented in Jagarlamundi et. al ?. A set of seed words can be provded by the user, these are used to help the model learn about the topics. Jagarlamundi et. al build a model which uses the seed words to improve both the topic-word and document-topic probability distributions. Following the paper both of these models are presented separately first (Model 1 and Model 2) and then combined.

Word-Topic Distributions (Model 1)

This model chooses words from two Multinomial distributions: a `seed topic' distribution and a`regular topic' distribution. The seed topic distribution is constrained to only generate words from a corresponding seed set. The regular topic distribution may generate any word including seed words.

1. Choose regular topic $\phi_k^r \sim Dir(\beta_r)$
2. Choose seed topic $\phi_k^s \sim Dir(\beta_s)$
3. Choose $\pi_k \sim Beta(1, 1)$

Document $\mathbf{w}_d$ is then generated as follows.

1. Choose the number of words in the document, $N_d \sim Po(\xi)$.
2. Choose the probabilities of the topics for the document, $\theta_d \sim Dir(\alpha)$. ($\theta$ is a $1 \times k$ vector)
3. For each of the N_d words $w_{d,n}$,
   - Choose a topic $z_{d,n} \sim Mult(\theta_d)$, ($z_{d,n}$ is a single value from 1 to k, the topic of word n in doc d)
   - Select an indicator $x_{d,n} \sim Bern(\pi_{z_{d,n}})$
   - If $x_{d,n} = 0$ choose a word $w_{d,n} \sim Mult(\phi^s_{z_{d,n}})$,
   - If $x_{d,n} = 1$ choose a word $w_{d,n} \sim Mult(\phi^r_{z_{d,n}})$,

where $\phi \sim Dir(\beta)$. The choices of the hyperparameters α and β will be discussed further in the next section. Figure 3 shows a graphical representation of LDA.

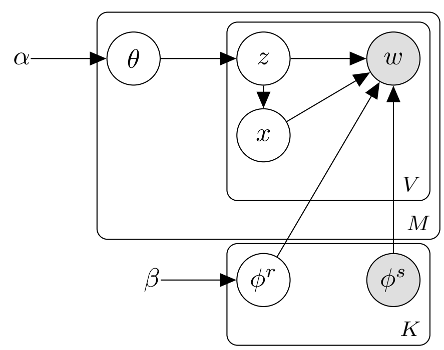

Figure 2: Graphical model for Model 1.

Document-Topic Distributions (Model 2)

Very high level idea is that documents which contain a word from one of the seed topics have a higher probability of coming from this topic.....sounds a bit simple

1. For each topic choose $\phi_j \sim Dir(\beta)$
2. For each seed topic choose a group-topic distribtuion $\psi_j \sim Dir(\alpha)$

Document $\mathbf{w}_d$ is then generated as follows.

1. Choose binary vector b (vector of length K indicating whether seed topic j is in document)
   - when generating documents this can be choosen randomly

2. Choose document-group distribution $\xi_d \sim Dir(\tau b)$
3. Choose a group varaible $g \sim Mult(\xi_d)$
4. Choose $\theta_d \sim Dir(\psi_g)$
5. For each of the N_d words $w_{d,n}$,
    - Choose a topic $z_{d,n} \sim Mult(\theta_d)$,
    - Choose a word $w_{d,n} \sim Mult(\phi_{z_{d,n}})$.

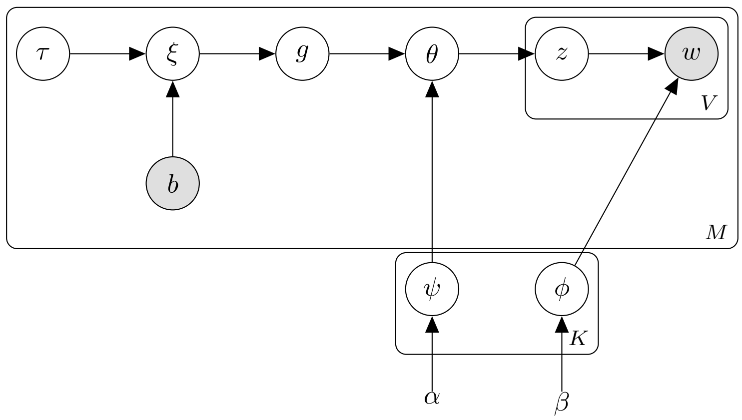

Figure 3: Graphical model for Model 2.

Hierarchical LDA

Hierarchically Supervised Latent Dirichlet Allocation Perotte et. al 11.