Algorithms for Fitting Mean-Field Latent Dirichlet Allocation

Sebastian Ibanez (sci2104) and Wicaksono Wijono (ww2522)

Introduction

Topic modeling is a statistical method for uncovering the underlying topics in a body of text. The idea behind topic modeling is that analyzing the words that appear in a document should allow one to learn about the topics that the document is discussing. For example, one expects words like "stocks" and "bonds" to appear in articles about business and finance, while words like "government" and "polls" should be more common in texts about politics. Since documents can discuss a myriad number of topics in different proportions (e.g. an article that talks about how a new government regulation affects the stock market covers both "business" and "politics"), a topic model allows us to more formally represent the relationship between a document and various topics.

For this project, we use Latent Dirichlet Allocation (LDA) as our topic model. In their paper, Blei, Ng, and Jordan [1] describe LDA as a generative probabilistic model. Specifically, it is a three-level hierarchical Bayesian model "in which each item of a collection is modeled as a finite mixture over an underlying set of topics". In this case, we use vectors of topic probabilities to characterize a corpora of news articles. The implementation is done using the R programming language.

Dataset

In this project, we use the All the news dataset from Kaggle [2]. The dataset consists of online news articles that have been scraped from the websites of various news publications. The publications include: the New York Times, Breitbart, CNN, Business Insider, the Atlantic, Fox News, Talking Points Memo, Buzzfeed News, National Review, New York Post, the Guardian, NPR, Reuters, Vox, and the Washington Post.

In particular, we use the articles1.csv table which contains 50,000 news articles (approximately one-third of the total dataset) that were mostly published between the years of 2015 and 2017. The table contains various metadata and related information attached to each article such as its title, publication name, author name, date of publication, and the article's URL. Given the nature of this project, however, we focus our attention on the contents of the article.

Preprocessing

To prepare our data, we use the tm package. The package provides a text mining framework that grants users a variety of methods for data import, corpus handling, text preprocessing, metadata management, etc.

We preprocess our dataset by going over every article and performing the following actions: (1) removing white spaces, (2) setting all words to lower case, (3) removing all numeric characters, and (4) removing common English stop words prescribed by the package (e.g. the, a, an). This process is done by the preprocess.r script.

We then construct a word-frequency table by iterating over every document and counting the occurrences of each word within that document. The final aggregated table has the following variables: article_id, word, and frequency. This process is done by the tableizer.r script.

Algorithms

NUTS

NUTS sounds appealing because it is so few lines of code to implement LDA from scratch. However, after 90 minutes, NUTS still has not finished its warmup phase. This is unsurprising, as the gradient computations must be extraordinarily expensive, and we expect the posterior to be multimodal so there might not be any good hyperparameters, not to mention the identifiability issues with Bayesian clustering algorithms. Gibbs sampling should work better since the mean-field LDA is conditionally conjugate, but even that will be extraordinarily slow to generate good posterior distributions.

Coordinate Ascent Variational Inference

The basic idea of variational inference is: the posterior p is intractable, and we want to approximate it by positing a family of distributions q to approximate the exact posterior. The measure of closeness is typically the KL divergence:

$$KL(q||p) = \mathbb{E}_q[log(\frac{q(\nu)}{p(\theta|X)})]$$

But this requires us to know p, so instead we use evidence lower bound (ELBO) as the objective function:

$$ELBO = log(p(X)) - KL(q||p)$$

Note that if $q = p$ then $KL(q||p) = 0$ and hence the ELBO attains its maximum value, the log evidence.

Traditionally, VI is fit by an EM-style approach where we update parameters one at a time, increasing the ELBO with each update. The updating equations are model-specific and need to be derived by hand. To simplify the algorithm, we restrict q to the mean-field family, i.e. we assume local parameters are conditionally independent given a cluster. The LDA algorithm used here uses the mean-field assumption, though more complex variational algorithms for the correlated topic model does exist.

While implementations for CAVI for LDA exist, we want to implement it by hand to better understand the algorithm. The algorithm is given in the original LDA paper [1] but it is much more readable in a future paper [4]. We will not copy the equations to this pdf.

While CAVI works, a single pass through the corpus requires so much time that we don't see it converging in any reasonable time. In a more efficient language with parallelization, it should be doable on a large dataset. However, we want to jump to an even more scalable algorithm.

Stochastic Variational Inference

SVI [5] approximates the natural gradient of the ELBO by framing it as an expectation and using Monte Carlo techniques to approximate it, allowing us to do stochastic gradient ascent. Convergence is guaranteed under two conditions:

First, the estimator of the gradient has to be unbiased, i.e. the expectation of the sample gradient is the exact gradient. We sample a single document for each update to achieve this.

Second, we must satisfy the Robbins-Monro condition [6]. That is, the sequence of step sizes ϵ_t have

$$\sum_{t=1}^{\infty} \epsilon_t = \infty$$

$$\sum_{t=1}^{\infty} \epsilon_t^2 < \infty$$

A common step size schedule is:

$$\epsilon_t = (\tau + t)^{-\kappa}, \tau > 0, \kappa \in (0.5, 1)$$

We went with $\tau = 1$ and $\kappa = 0.75$.

We sample a single document at each time step and run VI on the document, then use the results to update the global parameters $\hat{\beta}$ using

$$\hat{\beta}^{(t+1)} = (1 - \epsilon_t)\hat{\beta}^{(t)} + \epsilon_t \hat{\lambda}^{(t)}$$

Ideally, we use ELBO to check convergence. I don't think my calculation of ELBO is correct, in particular the last part, as I store local thetas only for the current document so I approximate that part. Regardless, it is not used because it is too computationally complex. In each iteration, we only need to evaluate a single document to perform the update, but we need to pass through the entire corpus to compute the ELBO.

Results

The main diagnostic is to check whether or not ELBO has converged. However, as mentioned, we skip it to have reasonable running time.

Ideally, we use the results for something else, such as document clustering and similarity. And if time permits, we wished to use a metric such as held-out perplexity to choose the number of topics. However, the goal of this project is to deeply understand the nuts and bolts of the LDA algorithm and how to fit it to large datasets.

The main illuminating result is the top words per document:

```r
library(data.table)
load('SVI_results.RData')
print(counter)                                    # number of iterations the SVI was run
```

```
## [1] 5014
```

```r
top_words <- list()
for(k in 1:K){
  top_words[[k]] <- vocab[order(-glo_beta[k,])][1:20]
}
data.table(                                       # top 20 words per topic
  matrix(unlist(top_words),
         byrow = FALSE, ncol = K)
)
```

```
##          V1        V2         V3        V4          V5
##  1:     new      said         mr   countri       trump
##  2:    will    report      state       war     clinton
##  3:    like     polic       said   migrant      presid
##  4:   peopl     offic       will     islam      donald
##  5:    said       cnn        law    muslim  republican
##  6:      go      told     govern  militari    campaign
##  7:    year    attack       year     group    democrat
##  8:     get       new    univers      forc       obama
##  9:    time       man   american      unit     hillari
## 10:     sai    accord       unit    nation       elect
## 11:     can     peopl     immigr     syria  presidenti
## 12:    just       two       case     europ        vote
## 13:    make  investig       plan     world        hous
## 14:  follow    fridai    student    region       parti
## 15:     now      citi       also    terror       senat
## 16: twitter      call administr terrorist      candid
## 17:   first statement     includ       isi     support
## 18:   think    releas        new    govern      former
## 19:    even       sai     polici      citi       voter
## 20:    want     video     public      navi       polit
```

Humans need to give interpretation to the topics. Good labels might be: social media, crime, public policy, international affairs, and domestic politics. The interpretability of these topics suggests that the SVI has been implemented correctly, and we obtained good results with only 5000 samples, i.e. we did not even need to go through the entire corpus of around 50000 documents.

Preprocessing the data

```r
library(data.table)
library(tm)
library(dplyr)

# Load raw data
dt_articles = fread('articles1.csv', sep = ',', data.table = TRUE)[, !'V1']

# Convert DF of articles into a corpus
documents = VCorpus(VectorSource(dt_articles$content))

# Remove white spaces
documents = tm_map(documents, stripWhitespace)

# Set all words to lower case
documents = tm_map(documents, content_transformer(tolower))

# Remove all non-alphanumeric characters
removeSpecialChars <- function(x) gsub("[^a-zA-Z0-9 ]", "", x)
documents = tm_map(documents, content_transformer(removeSpecialChars))
#documents = tm_map(documents, removePunctuation)

# Remove all numbers
documents = tm_map(documents, removeNumbers)

# Remove stopwords
documents = tm_map(documents, removeWords, stopwords("english"))

# Re-remove white spaces
documents = tm_map(documents, stripWhitespace)

# Convert corpus into DF
df_articles = data.frame(text = sapply(documents, as.character))

# Construct dictionary
bag_of_words = data.frame(
  words = unique(unlist(strsplit(paste(df_articles$text, collapse = " "),
                                 split = "\\s+"))))

df_articles = df_articles[1:50000,]

list_wordcount = lapply(df_articles, function(x) {
    as.data.frame(table(strsplit(x, split = "\\s+")))
})

df_wordcount = bind_rows(list_wordcount, .id = "article_id")

fwrite(df_wordcount, file = 'df_wordcount_fast.csv', row.names = FALSE)
```

Fitting the model

```r
library(data.table)
library(rethinking)
library(SnowballC)
library(tm)
```

A little further preprocessing is needed, as it turns out Stan doesn't work with wordcounts.

```r
dat <- fread('df_wordcount_fast.csv')
dat$Word <- wordStem(dat$Word)
dat <- dat[!Word %in% stopwords('english')]
dat <- dat[nchar(Word) > 1]
dat <- dat[,list(Freq = sum(Freq)), by = list(article_id, Word)]

wordcounts <- dat[,list(wc = sum(Freq)), by = Word]
wordcounts <- wordcounts[order(-wc)]
vocab <- wordcounts$Word[1:2000]

dat <- dat[Word %in% vocab,]
dat$word_id <- match(dat$Word, vocab)
```

Afterwards, we put everything into the correct variables.

```r
K <- 5L                                 # number of topics
V <- length(unique(dat$word_id))        # number of words in vocabulary
M <- length(unique(dat$article_id))     # number of documents
N <- sum(dat$Freq)                      # number of words in the corpus
words <- as.integer(unlist(mapply(      # vector of words in corpus
  function(x, y) rep(x, y),
  dat$word_id,
  dat$Freq
)))
doc <- as.integer(unlist(mapply(        # vector of document_id of the words
  function(x, y) rep(x, y),
  dat$article_id,
  dat$Freq
)))
alpha <- runif(K) * 10 + 1              # prior on topic proportions
beta <- runif(V) * 10 + 1               # prior on distribution of words per topic
```

NUTS

The Stan user manual [3] provides an implementation of LDA, which we have copy and pasted below:

```stan
data {
  int<lower=2> K;               // num topics
  int<lower=2> V;               // num words
  int<lower=1> M;               // num docs
  int<lower=1> N;               // total word instances
  int<lower=1,upper=V> w[N];    // word n
  int<lower=1,upper=M> doc[N];  // doc ID for word n
  vector<lower=0>[K] alpha;     // topic prior
  vector<lower=0>[V] beta;      // word prior
```

```
}
parameters {
  simplex[K] theta[M];   // topic dist for doc m
  simplex[V] phi[K];     // word dist for topic k
}
model {
  for (m in 1:M)
    theta[m] ~ dirichlet(alpha);  // prior
  for (k in 1:K)
    phi[k] ~ dirichlet(beta);     // prior
  for (n in 1:N) {
    real gamma[K];
    for (k in 1:K)
      gamma[k] = log(theta[doc[n], k]) + log(phi[k, w[n]]);
    target += log_sum_exp(gamma);  // likelihood;
  }
}
```

We try running NUTS on Stan to fit the mean-field LDA model:

```
time_start <- Sys.time()
lda <- stan(
  file = 'LDA.stan',
  data = c('K', 'V', 'M', 'N', 'w', 'doc', 'alpha', 'beta'),
  iter = 2000,
  chains = 4,
  refresh = 0
)
time_end <- Sys.time()
```

Coordinate Ascent Variational Inference

```
beta_mat <- matrix(                        # full prior for words per topic
  rep(beta, K),
  nrow = K,
  byrow = TRUE
)
alpha_mat <- matrix(                       # full prior for topic proportions per doc
  rep(alpha, M),
  nrow = M,
  byrow = TRUE
)
z <- sample(1:K, N, replace = TRUE)        # initialize topic assignment of each word
glo_beta <- beta_mat + table(z, words)     # initial distribution of words per topic
doc_prop <- alpha_mat + table(doc, z)      # initial distribution of topics per document

for(iter in 1:100){                        # fixed iterations, but should look at ELBO
  beta_updater <- beta_mat
  for(d in 1:M){                           # loop over documents
    print(d)
    words_in_doc <- words[doc == d]        # get only words in the document
    word_indices <- which(doc == d)
    theta <- list()
```

```r
    old_prop <- alpha

    # repeat until document topic proportions have converged

    while(sum(abs(old_prop - doc_prop[d,]) > 10^(-4)) > 0){
      old_prop <- doc_prop[d,]
      for(w in 1:length(words_in_doc)){  # loop over words in document
        theta_word <- rep(NA, K)         # probability of topics for each word
        for(k in 1:K){
          theta_word[k] <- exp(
            digamma(doc_prop[d, k]) +
              digamma(glo_beta[k, words_in_doc[w]]) -
              digamma(sum(glo_beta[k,]))
          )
        }
        theta_word <- theta_word / sum(theta_word)
        theta[[w]] <- theta_word
      }
      theta_matrix <- matrix(
        unlist(theta),
        nrow = length(theta),
        byrow = TRUE
      )
      doc_prop[d,] <- alpha + colSums(theta_matrix)
    }
    z[word_indices] <- apply(
      theta_matrix,
      1,
      which.max
    )
    for(w in 1:length(words_in_doc)){
      beta_updater[,words_in_doc[w]] <-
        beta_updater[,words_in_doc[w]] +
        theta_matrix[w,]
    }
  }
  glo_beta <- beta_updater
}
```

Stochastic Variational Inference

```r
get_ELBO <- function(
  glo_beta, doc_prop, z, words, doc,
  words_in_doc, local_theta_matrix
){
  ELBO <- 0
  for(k in 1:K){                          # global parameters
    normalizer <- sum(glo_beta[k,])
    ELBO <- ELBO + sum(
      glo_beta[k,] / normalizer *
        (beta - 1) * log(glo_beta[k,] / normalizer)
    )
```

```
  }

  for(d in 1:M){                            # topic proportions
    normalizer <- sum(doc_prop[d,])
    ELBO <- ELBO + sum(
      doc_prop[d,] / normalizer *
        (alpha - 1) * log(doc_prop[d,] / normalizer)
      )
  }

  for(d in 1:M){                            # assignment probability
    assignments <- z[which(doc == d)]
    prob_assignment <- doc_prop[d,] / sum(doc_prop[d,])
    ELBO <- ELBO + sum(
      prob_assignment[assignments] * log(prob_assignment[assignments])
      )
    for(k in 1:K){                          # word probability
      ELBO <- ELBO + prob_assignment[k] * (
        digamma(glo_beta[k, words[w]]) -
            digamma(sum(glo_beta[k,]))
      )
    }
  }

  for(k in 1:K){                            # global parameters
    normalizer <- sum(glo_beta[k,])
    ELBO <- ELBO - sum(
      glo_beta[k,] / normalizer *
        (glo_beta[k,] - 1) * log(glo_beta[k,] / normalizer)
    )
  }

  for(d in 1:M){                            # topic proportions
    normalizer <- sum(doc_prop[d,])
    ELBO <- ELBO - sum(
      doc_prop[d,] / normalizer *
        (doc_prop[d,] - 1) * log(doc_prop[d,] / normalizer)
      )
  }

  for(i in 1:nrow(local_theta_matrix)){
    normalizer <- sum(local_theta_matrix[i,])
    raw_prob <- local_theta_matrix[i, z[words_in_doc[i]]]
    ELBO <- ELBO - raw_prob / normalizer *
      log(raw_prob / normalizer) *
      length(words) / length(words_in_doc)
  }

  return(ELBO)
}
```

```
beta_mat <- matrix(                         # full prior for words per topic
  rep(beta, K),
  nrow = K,
```

```r
  byrow = TRUE
)
alpha_mat <- matrix(                      # full prior for topic proportions per doc
  rep(alpha, M),
  nrow = M,
  byrow = TRUE
)
z <- sample(1:K, N, replace = TRUE)       # initialize topic assignment of each word
glo_beta <- beta_mat + table(z, words)    # initial distribution of words per topic
doc_prop <- alpha_mat + table(doc, z)     # initial distribution of topics per document
counter <- 1                              # counter for step size
trace_ELBO <- list()
```

After the initialization above, we can run the code chunk below any number of times we want, or stop any time we want:

```r
for(iter in 1:20000){                     # fixed iterations, but should look at ELBO
  eps <- (1 + counter)^(-0.75)
  beta_updater <- beta_mat
  d <- sample(1:M, 1)
  words_in_doc <- words[doc == d]         # get only words in the document
  word_indices <- which(doc == d)
  theta <- list()
  old_prop <- alpha

  # repeat until document topic proportions have converged

  while(sum(abs(old_prop - doc_prop[d,]) > 10^(-1)) > 0){
    old_prop <- doc_prop[d,]
    for(w in 1:length(words_in_doc)){     # loop over words in document
      theta_word <- rep(NA, K)            # probability of topics for each word
      for(k in 1:K){
        theta_word[k] <- exp(
          digamma(doc_prop[d, k]) +
            digamma(glo_beta[k, words_in_doc[w]]) -
            digamma(sum(glo_beta[k,]))
        )
      }
      theta_word <- theta_word / sum(theta_word)
      theta[[w]] <- theta_word
    }
    theta_matrix <- matrix(
      unlist(theta),
      nrow = length(theta),
      byrow = TRUE
    )
    doc_prop[d,] <- alpha + colSums(theta_matrix)       # update document topics prop
  }
  z[word_indices] <- apply(
    theta_matrix,
    1,
    which.max
  )
  for(w in 1:length(words_in_doc)){                      # noisy global parameters
```

```
    beta_updater[, words_in_doc[w]] <-
      beta_updater[, words_in_doc[w]] +
      length(words) / length(words_in_doc) *
      theta_matrix[w,]
  }
  glo_beta <- (1 - eps) * glo_beta + eps * beta_updater # the stochastic update
  counter <- counter + 1
  if(counter %% 50 == 0){print(counter)}
}
```

Statement of Contribution

Sebastian Ibanez

- Wrote the Introduction, Dataset, and Preprocessing sections of the final paper
- Coded the preprocess.r and tableizer.r scripts and performed preprocessing on the dataset

Wicaksono Wijono

- Coded the algorithms to fit the model and discussed the results and limitations

Bibliography

These are clickable links:

[1] **LDA** Blei et al. (2003) Latent Dirichlet Allocation

[2] **Data** Thompson (2015) All the news

[3] **Stan User's Guide** Stan Development Team. Latent Dirichlet Allocation

[4] **VI Review** Blei et al. (2017) Variational Inference: A Review for Statisticians

[5] **SVI** Hoffman et al. (2013) Stochastic Variational Inference

[6] **Robbins-Monro** Robbins and Monro (1951) A Stochastic Approximation Method