**A topic model for movie reviews**

In this document, we fit an [LDA topic model](http://en.wikipedia.org/wiki/Latent_Dirichlet_allocation) to the [Cornell Movie Review Data](http://www.cs.cornell.edu/people/pabo/movie-review-data/) introduced by Pang, Lee, and Vaidyanathan in their 2002 EMNLP paper, where we use 'polarity dataset version 2.0' (introduced in a subsequent Pang and Lee 2004 ACL paper). To fit the model, we used the R package [lda](http://cran.r-project.org/web/packages/lda/) and we visualize the output using [LDAvis](https://github.com/cpsievert/LDAvis).

**The data**

For convenience, the R package 'LDAvisData' provides data used to supplement LDAvis examples. The package provides an object named reviews which is a character vector of length 2000. Each element of that vector contains a single movie review. Note that reviews was created using [this script](https://github.com/cpsievert/moviereviews/blob/master/data-raw/reviews.R).

*# LDAvisData can be installed from GitHub via 'devtools::install\_github("cpsievert/LDAvisData")'*

data(reviews, package = "LDAvisData")

**Pre-processing**

Before fitting a topic model, we need to tokenize the text. This dataset is already fairly clean, so we only remove punctuation and some common [stop words](http://en.wikipedia.org/wiki/Stop_words). In particular, we use the english stop words from the [SMART information retrieval system](http://en.wikipedia.org/wiki/SMART_Information_Retrieval_System), available in the R package **tm**.

*# read in some stopwords:*

**library**(tm)

stop\_words <- stopwords("SMART")

*# pre-processing:*

reviews <- gsub("'", "", reviews) *# remove apostrophes*

reviews <- gsub("[[:punct:]]", " ", reviews) *# replace punctuation with space*

reviews <- gsub("[[:cntrl:]]", " ", reviews) *# replace control characters with space*

reviews <- gsub("^[[:space:]]+", "", reviews) *# remove whitespace at beginning of documents*

reviews <- gsub("[[:space:]]+$", "", reviews) *# remove whitespace at end of documents*

reviews <- tolower(reviews) *# force to lowercase*

*# tokenize on space and output as a list:*

doc.list <- strsplit(reviews, "[[:space:]]+")

*# compute the table of terms:*

term.table <- table(unlist(doc.list))

term.table <- sort(term.table, decreasing = TRUE)

*# remove terms that are stop words or occur fewer than 5 times:*

del <- names(term.table) %in% stop\_words | term.table < 5

term.table <- term.table[!del]

vocab <- names(term.table)

*# now put the documents into the format required by the lda package:*

get.terms <- **function**(x) {

index <- match(x, vocab)

index <- index[!is.na(index)]

rbind(as.integer(index - 1), as.integer(rep(1, length(index))))

}

documents <- lapply(doc.list, get.terms)

**Using the R package 'lda' for model fitting**

The object documents is a length-2000 list where each element represents one document, according to the specifications of the **lda** package. After creating this list, we compute a few statistics about the corpus:

*# Compute some statistics related to the data set:*

D <- length(documents) *# number of documents (2,000)*

W <- length(vocab) *# number of terms in the vocab (14,568)*

doc.length <- sapply(documents, **function**(x) sum(x[2, ])) *# number of tokens per document [312, 288, 170, 436, 291, ...]*

N <- sum(doc.length) *# total number of tokens in the data (546,827)*

term.frequency <- as.integer(term.table) *# frequencies of terms in the corpus [8939, 5544, 2411, 2410, 2143, ...]*

Next, we set up a topic model with 20 topics, relatively diffuse priors for the topic-term distributions (*η*  = 0.02) and document-topic distributions (*α*  = 0.02), and we set the collapsed Gibbs sampler to run for 5,000 iterations (slightly conservative to ensure convergence). A visual inspection of fit$log.likelihood shows that the MCMC algorithm has converged after 5,000 iterations. This block of code takes about 24 minutes to run on a laptop using a single core 1.7Ghz processor (and 8GB RAM).

*# MCMC and model tuning parameters:*

K <- 20

G <- 5000

alpha <- 0.02

eta <- 0.02

*# Fit the model:*

**library**(lda)

set.seed(357)

t1 <- Sys.time()

fit <- lda.collapsed.gibbs.sampler(documents = documents, K = K, vocab = vocab,

num.iterations = G, alpha = alpha,

eta = eta, initial = NULL, burnin = 0,

compute.log.likelihood = TRUE)

t2 <- Sys.time()

t2 - t1 *# about 24 minutes on laptop*

**Visualizing the fitted model with LDAvis**

To visualize the result using [LDAvis](https://github.com/cpsievert/LDAvis/), we'll need estimates of the document-topic distributions, which we denote by the *D*×*K*  matrix *θ*  , and the set of topic-term distributions, which we denote by the *K*×*W*  matrix *ϕ*  . We estimate the “smoothed” versions of these distributions (“smoothed” means that we've incorporated the effects of the priors into the estimates) by cross-tabulating the latent topic assignments from the last iteration of the collapsed Gibbs sampler with the documents and the terms, respectively, and then adding pseudocounts according to the priors. A better estimator might average over multiple iterations of the Gibbs sampler (after convergence, assuming that the MCMC is sampling within a local mode and there is no label switching occurring), but we won't worry about that for now.

theta <- t(apply(fit$document\_sums + alpha, 2, **function**(x) x/sum(x)))

phi <- t(apply(t(fit$topics) + eta, 2, **function**(x) x/sum(x)))

We've already computed the number of tokens per document and the frequency of the terms across the entire corpus. We save these, along with *ϕ*  , *θ*  , and vocab, in a list as the data object MovieReviews, which is included in the **LDAvis** package.

MovieReviews <- list(phi = phi,

theta = theta,

doc.length = doc.length,

vocab = vocab,

term.frequency = term.frequency)

Now we're ready to call the createJSON() function in **LDAvis**. This function will return a character string representing a JSON object used to populate the visualization. The createJSON() function computes topic frequencies, inter-topic distances, and projects topics onto a two-dimensional plane to represent their similarity to each other. It also loops through a grid of values of a tuning parameter, 0≤*λ*≤1  , that controls how the terms are ranked for each topic, where terms are listed in decreasing of *relevance*, where the relevance of term *w*  to topic *t*  is defined as *λ*×*p*(*w*∣*t*)+(1−*λ*)×*p*(*w*∣*t*)/*p*(*w*)  . Values of *λ*  near 1 give high relevance rankings to *frequent* terms within a given topic, whereas values of *λ*  near zero give high relevance rankings to *exclusive* terms within a topic. The set of all terms which are ranked among the top-R most relevant terms for each topic are pre-computed by the createJSON() function and sent to the browser to be interactively visualized using D3 as part of the JSON object.

**library**(LDAvis)

*# create the JSON object to feed the visualization:*

json <- createJSON(phi = MovieReviews$phi,

theta = MovieReviews$theta,

doc.length = MovieReviews$doc.length,

vocab = MovieReviews$vocab,

term.frequency = MovieReviews$term.frequency)

The serVis() function can take json and serve the result in a variety of ways. Here we'll write json to a file within the 'vis' directory (along with other HTML and JavaScript required to render the page). You can see the result [here](http://cpsievert.github.io/LDAvis/reviews/vis).

serVis(json, out.dir = 'vis', open.browser = FALSE)

If you discover something interesting in your data using **LDAvis**, you can share the result via a URL since the state of the visualization is stored in the URL at all times. For example, in the movie review data, you can quickly see that Topic 7 is broadly about comedies by linking directly to the state of LDAvis where the selected Topic is “7” and the value of *λ*  is 0.6 with the following URL:

<http://cpsievert.github.io/LDAvis/reviews/vis/#topic=7&lambda=0.6&term=>

You can also link to the term that is hovered. For example, when you look at the 30 most relevant terms for Topic 5 using a relevance setting of *λ*=0.5  , the term “action” is the 6th bar from the top (i.e. the 6th most relevant term for this topic). The widths of the red and blue bars indicate that there is at least one other topic in which the term “action” appears frequently. By hovering over “action”, we see from the following state of **LDAvis** that term “action” also appears frequently in Topic 14 (as the 9th most relevant term):

<http://cpsievert.github.io/LDAvis/reviews/vis/#topic=14&lambda=0.5&term=action>

Comparing these two topics, we can see that Topic 5 discusses action in the context of movies about crime and police, whereas in Topic 14, the term “action”“ is also used frequently, but the topic is specifically about kung fu movies with Chinese actors (Jackie Chan and Jet Li, for example). These two topics both make heavy use of the word "action” but in slightly different contexts (i.e. slightly different styles of movies).

To encode a state of the visualization in the URL, you must include a string after the “/” of the form “#topic=k&labmda=l&term=s”, where “k”, “l”, and “s” are strings representing the topic to be selected, the value of *λ*  to be used in the relevance calculation, and the term to be hovered, respectively. If no term hovering is desired, omit “s” from the URL. The topic, “k”, will be forced to an integer in {0,1,..,*K*}  , and the value of *λ*  will be forced to the interval [0,1]  , with non-numeric values returning the default state of the visualization (topic = 0, *λ*  = 1, term = “”).