Practical Skills for Document Clustering in R*

Brandon M. Stewart

June 15, 2010

This lab is split into three exercises allowing you to use three different datasets and three different models to perform unsupervised document clustering in R. Unlike supervised learning, unsupervised learning has no human annotated examples. The first model k-means is a simple and very common algorithm in computer science. The second two models are Bayesian hierarchical models designed to capture the *ex ante* known structure of the documents and substantive interest. Because different people work at different paces, I've included boxes with extra material. The material is delineated: beginner, medium and hard. The beginner material requires little to no additional programming, the medium material usually requires a little bit of coding and some statistical background, the advanced material is for those who really want to look under the hood of these methods and have considerable background in R.

When performing document clustering you should follow the same basic steps:

1. Choose a Corpus of Documents
2. Quantitatively Represent Your Text
3. Choose/Specify a Model
4. Label Clusters
5. Validate, Validate, Validate

We are (mostly) going to focus on steps 3 and 4 today.

Warning: Unsupervised learning is a little bit like herding cats. Because you are uncovering the latent structure in the documents you may not get the exact conceptual categories that you want. If you have very strong expectations/desires for categories you need to be doing Supervised learning.

1 k-means using participant papers

Before the lab I wanted to know what scholars in this group were interested in and writing about in their own research. So I decided to construct a topic model of scholarly articles.

*Handout prepared for the Tools for Text Workshop at the University of Washington. I am grateful to Justin Grimmer for providing code and data from his dissertation.

1.1 Choose a Corpus of Documents

For this lab I have downloaded every paper written by the Tools for Text Workshop participants that I could easily find (i.e. available on personal websites, Google Scholar etc.). I took the raw PDF's and converted them to a term-document matrix using the following steps:

1. Identify the corpus using the list of 45 participants and download PDF's
2. OCR scanned papers and convert them all to plain UTF-8 .txt form
3. Created a term-document matrix using a Python script which also:
    - Stemmed words using the Porter Stemming Algorithm
    - Removed words appearing in less than 10% of the documents
    - Removed words appearing in more than 10% of the documents
    - Removed common stop words

The result is a comma-separated value (csv) formatted file that we will now load. The file contains 88 different papers. Start by reading the matrix into Excel using the following code:

```
##Import and minimally format the Document-Term Matrix
file <- read.csv("english_count.csv")      #Read in the matrix.
papersDTM <- as.matrix(file[,-1])          #This removes the filenames from the first column
rownames(papersDTM) <- file[,1]            #This adds the filenames as row names
filenames <- as.character(file[,1])        #This adds an extra vector for labels.
rm(file)                                   #Removes this file that we no longer need
```

Now we have two items in our workspace which we can look at using the `head()` command to see the first few lines and the `dim()` command to see the dimensions of the matrix.

```
head(papersDTM)                            #Our Document-Term Matrix
head(filenames)                            #Our List of Filenames
```

You can see that there are 88 rows (articles) and 3189 columns (unique terms). Each cell is a count of the number of times that term appears in the article. The object `filenames` is just a separate vector of the filenames that we will use as a reference.

1.2 Quantitatively Represent Your Text

Now we have already chosen many elements of our representation by the way in which we constructed our term document matrix. Because the lengths of our documents vary so wildly (see Figure 1) we may want to row-standardize our document matrix (divide each entry by the number of terms in that document). We can perform this in `R` using the following code:

```
rowTotals <- apply(papersDTM, 1, sum)      #Find the sum of words in each Document
input <- papersDTM/rowTotals               #Divide each row by those totals
```

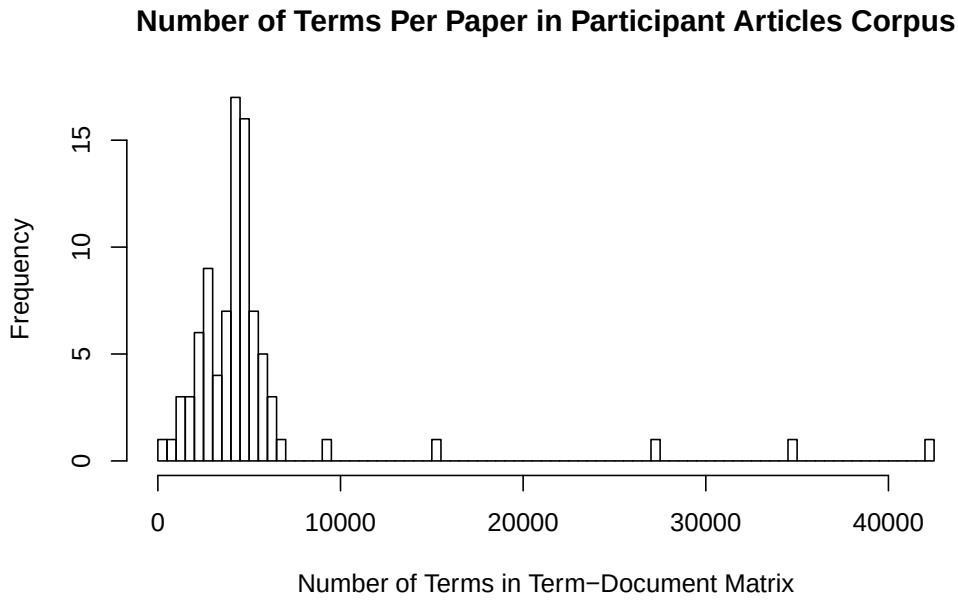

Figure 1: This histogram shows that there is considerable variation in article lengths within our corpus. Long documents may become troublesome outliers in our algorithm so we need to be aware of heterogeneity in the corpus like this. If you want to create this histogram yourself use the code:
`hist(apply(papersDTM, 1, sum), xlab="Number of Terms in Term-Document Matrix", main="Number of Terms Per Paper in Participant Articles Corpus", breaks=100)`

Now every cell corresponds to the proportion of the document that is occupied by that term (after removing all the stop words, most frequent and least frequent words).

⇒ **Extra (Beginner):** One of the limitations of standard k-means is that it has no mechanism for handling outliers (although you can always add one). Once you have completed the rest of this exercise, try running k-means on the original data without this transformation and notice how the same documents are always outliers.

1.3 Choose a Model

We are going to analyze this corpus using the k-means algorithm. k-means is the most popular (and arguably important) flat clustering algorithm. "Its objective is to minimize the average squared Euclidean distance of documents from their cluster centers where a cluster center is defined as the mean or centroid $\overrightarrow{\mu}$ of the documents in a cluster $\omega : \overrightarrow{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\overrightarrow{x} \in \omega} \overrightarrow{x}$ (Manning, Raghavan and Schütze, 2008, 360).

The k-means algorithm is non-deterministic. In order to get the same results each time we will set the random seed. We will need to rerun this code immediately before executing the model if we want to guarantee the same results.

```
set.seed(12345)                          #This sets the random seed.
```

Obviously we can try other seeds to see if we get different results.

The `kmeans()` function in `R` takes three relevant inputs: our data matrix (`input`)) our k or the number of clusters we are looking for and the number of starts. The number of starts will initialize the algorithm with however many random starting points the user requests. Then it displays the results with the lowest residual sum of squares. If your computer is running slowly you can always decrease the number of starts.

```
results <- kmeans(input,                 #Our input term document matrix
                  centers=10,            #The number of clusters
                  nstart=25)             #The number of starts chosen by the algorithm
```

We can type `results` to view our results however they are not very useful until we do some additional work to develop cluster labels.

1.4 Label Clusters

We should always label our clusters in two different ways:

1. Use some form of automated labelling
2. Manually read a sample of documents

We'll start by developing a very simple procedure for automated labelling, the difference in the mean word use between the cluster and the documents not in the cluster. It is very simple:

1. For each cluster c:
   (a) For each word w:
      i. Find the mean use of word w in the documents that make up cluster c
      ii. Find the mean use of word w in the documents that are not in cluster c
      iii. Take the mean in cluster c and subtract the mean not in cluster c
   (b) Rank order all the words so that the highest values come first
2. Look at the top 20 words for each cluster

There are many potential problems with this technique but it is a useful heuristic that captures the key words in the cluster. The following code implements this in `R`

```
for (i in 1:length(results$withinss)) {
  #For each cluster, this defines the documents in that cluster
  inGroup <- which(results$cluster==i)
  within <- papersDTM[inGroup,]
  if(length(inGroup)==1) within <- t(as.matrix(within))
  out <- papersDTM[-inGroup,]
  words <- apply(within,2,mean) - apply(out,2,mean) #Take the difference in means for each ter
  print(c("Cluster", i), quote=F)
  labels <- order(words, decreasing=T)[1:20] #Take the top 20 Labels
```

```
  print(names(words)[labels], quote=F)     #From here down just labels
  if(i==length(results$withinss)) {
    print("Cluster Membership")
    print(table(results$cluster))
    print("Within cluster sum of squares by cluster")
    print(results$withinss)
    }
  }
```

This command also puts out some additional useful information.
⇒ **Extra (Beginner):** This mechanism for cluster labelling has a number of flaws. The most obvious is that it does not take into account the rarity of the words in the corpus or the sampling variability of the document selection. Think about how this problem could be solved and what other situations we might want to be aware of when labelling documents this way.

We also want to manually read a set of documents. We can write a few lines of code to see the file names of the documents in each cluster as well as randomly select documents from the cluster for us to look at:

```
clusterNum <- 1          #Set the Cluster Number you want to look at
#See list of all documents
filenames[which(results$cluster==clusterNum)]
#See a random sample of one document
filenames[which(results$cluster==clusterNum)][sample(sum(results$cluster==clusterNum),1)]
```

The variable `clusterNum` sets the cluster you are going to look at. The first line of code shows all the file names in that cluster. The second line of code randomly selects a document.

Pick a cluster and skim a few of the articles. Do they match the automatic labels?

1.5 Repeat!

Try different values of k and use the automatic labels to determine whether or not it is worth reading more deeply. Settle on a model and label the resulting topics. Think about possible validations using one of the criteria defined in Quinn et al. (Forthcoming).

1. **Semantic Validity:** All categories are coherent and meaningful
2. **Convergent Construct Validity:** Topics concur with existing measures in critical details.
3. **Discriminant Construct Validity:** The topics differ from existing measures in productive ways.
4. **Predictive Measure:** Data generated from the model corresponds to external events in expected ways.
5. **Hypothesis Validity:** Data generated from the model can be used to test substantive hypotheses.

Congratulations, you've just run a topic model!

1.6 Additional Exercises

⇒ **Extra (Easy):** Try analyzing another dataset using the same data. In addition to real data available in existing packages or your own data you can generate simulated data to explore the algorithm more in depth. For a very simple dataset try the following simulated 2-D example from the `kmeans()` help file:

```
x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
           matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2))
```

⇒ **Extra (Medium):** If you have experience with `R` try coding `kmeans()` yourself from scratch. It will help you better understand the basics of unsupervised learning. You can use Manning, Raghavan and Schütze (2008) chapter 16 as a reference.

⇒ **Extra (Medium):** While k-means is a very common algorithm for document clustering it is more frequently used with a distance metric called *cosine distance* which has the advantage of normalizing based on the document length. Try coding cosine distance using the formula from Manning, Raghavan and Schütze (2008), in chapter 6.3.

`http://nlp.stanford.edu/IR-book/pdf/06vect.pdf`

⇒ **Extra (Medium/Hard):** While the difference in means is a useful labelling heuristic a more common labelling mechanism is mutual information. Try coding up this metric using the formula from Manning, Raghavan and Schütze (2008) chapter 13.5.1

`http://nlp.stanford.edu/IR-book/pdf/13bayes.pdf`

2 Latent Dirichlet Allocation for Literature Review

The code and data from this example come from the first chapter of Justin Grimmer's dissertation Grimmer (Forthcoming). Justin wanted to review the literature on Fenno's highly cited *Homestyle* to determine whether or not the fundamental idea of how congressmen communicate with their constituents was still being actively researched. He decided to use a hierarchical topic model called Latent Dirichlet Allocation (**?**) to provide a quick overview of the literature. In this exercise we will analyze the same corpus but entirely in `R`.

2.1 Choose a Corpus of Documents

First we identify the corpus, 377 scholarly articles which cite Fenno's *Homestyle*, and download a term-document matrix from JSTOR's new Data For Research site `dfr.jstor.org`. After some standard preprocessing in `R` including stemming and converting the files into the format for the `lda()` package we can load the following data workspace along with the libraries of functions we need.

```
load("Exercise2.RData")
library(lda)
library(ggplot2)
```

The `lda` library is an implementation of Latent Dirichlet Allocation using a Collapsed Gibbs Sampler written in `C`. It also has a variety of useful functions for preprocessing the data and analyzing the model results. The `ggplot2` package is a graphics package that we will use at the end to make a graph of the topic mixtures.
We can start by looking at the objects in our newly loaded workspace.

```
summary(ldadata)            #This is a strange format needed for this model.
head(file.order)            #This is our normal Term-Document Matrix
head(data.order)            #This is information on the articles we are analyzing
```

2.2 Quantitatively Represent Your Text

We are going to leave the data representation as it is as word counts are assumed by the model.

2.3 Choose Your Model

Grimmer (Forthcoming) uses Latent Dirichlet Allocation because the model allows for each article to be a mixture of topics. We might intuitively believe that most scholarly articles are actually about multiple topics (partially about statistics and partially about voting behavior for instance). We start by setting the random seed, choosing a number of topics and then running the model:

```
set.seed(12345)                    #This sets the random seed

#Now we run the Latent Dirichlet Allocation Model
K <- 50
result <- lda.collapsed.gibbs.sampler(
              ldadata$documents,  #This is the set of documents
              K,                 #This is the number of clusters
              ldadata$vocab,      #This is the vocab set
              25,                 #These are additional model parameters
              0.1,
              0.1))
```

There are three critical inputs: the two matrices that make up the data `ldadata$documents` and `ldadata$vocab` and the number of clusters, set in the example here to 50.

2.4 Label

We can now label our clusters using a simple built in function.

```
top.words <- top.topic.words(result$topics, 5, by.score=TRUE)
#The "5" is the number of labels
top.words
```

Examine the labels and think about whether they represent plausible, discrete and cohesive topics within the documents at large.

2.5 Validate

To help us understand what LDA looks like on documents in practice we can run a quick visualization of a sample of 10 documents from the LDA demo. Execute the following code:

```
N <- 10 #Choose the number of documents to display

topic.proportions <- t(result$document_sums) / colSums(result$document_sums)
topic.proportions <- topic.proportions[sample(1:dim(topic.proportions)[1], N),]
topic.proportions[is.na(topic.proportions)] <-  1 / K
colnames(topic.proportions) <- apply(top.words, 2, paste, collapse=" ")
topic.proportions.df <- melt(cbind(data.frame(topic.proportions),
                                   document=factor(1:N)),
                             variable_name="topic",
                             id.vars = "document")
qplot(topic, value, fill=document, ylab="proportion",
       data=topic.proportions.df, geom="bar") +
   opts(axis.text.x = theme_text(angle=90, hjust=1)) +
   coord_flip() +
   facet_wrap(~ document, ncol=5)
```

2.6 Additional Exercises

Try the model with different topics and then consider moving into another corpus of documents. You can use the corpus from exercise one or you can use the corpus of documents that is built in to the software itself using the command `demo(lda)`.

3 Expressed Agenda Model for Analyzing Press Releases

This code and example comes from Justin Grimmer's Expressed Agenda model in which he analyzes the proportion of attention that senators dedicate to different issues in their press releases.

3.1 Choose a Corpus of Documents

I've put together a subset of the 64,000 press releases that are available on Justin Grimmer's dataverse. Load up the documents using the following code:

```
rm(list=ls(all=TRUE)) #Clear out our workspace

#Load the Files
load("Exercise3.RData")
library(MCMCpack)

#Take a look at the files
author  #A listing of the authors of the documents
head(filenames) #A listing of file names
head(pressTDM) #Our typical term-document matrix
```

3.2 Choose Your Model

We can use some very simple code to estimate the model. I have arbitrarily set the number of categories to 15 but I encourage you to try other values.

```
set.seed(12345)       #This sets the random seed

#Run the Model
i <- 15               #This sets the number of categories
results <- exp.agenda.vonmon(pressTDM, author, i)
```

Labels

We can assign labels through a very simple process using the output of the model. Execute the following code and examine the labels.

```
words <- colnames(pressTDM) #create a separate vector of word titles

#Assign Labels (Don't worry about this too much)
x <- c()
for (j in 1:i) {
temp <- words[order(results$mus[,j] -
      apply(results$mus[,-j], 1, mean),
           decreasing=T)[1:20]]
x <- rbind(x, temp)
}
rownames(x) <- NULL
t(x)   #Displays our Labels
categories <- x
```

Validation

We also want to look at the expressed agenda priorities themselves as well as the assignment of topics by document. The following code snippets will estimate these relationships.

```
#Show the Expressed Priorities by Simulating from the Posterior
hist(rdirichlet(10000,
                alpha=results$thetas[2,])[,1],
                #The first index "2" is the senator,
                #the second index "1" is the topic number
                main="Posterior Distribution of Clinton2007 on Topic 1")
```

We can also assign documents to their most probable categories and output them for analysis.

```
#Assign Each Document to Its Categories and add its FileName
topics<- apply(results$rs, 1, which.max)
names(topics) <- filenames

##Optionally remove the comment sign to write out a spreadsheet of results.
#write.csv(topics, file = "TopicModels.csv", quote = FALSE)
```

References cited here are shown below but additional resources will be posted on the website by the beginning of next week.

References

Grimmer, Justin. Forthcoming. "A Bayesian Hierarchical Topic Model for Political Texts: Measuring Expressed Agendas in Senate Press Releases." *Political Analysis* .

Manning, Christopher D., Prahakar Raghavan and Hinrich Schütze. 2008. *An Introduction to Information Retrieval*. Cambridge, England: Cambridge University Press.

Quinn, Keven M., Burt L. Monroe, Michael Colaresi, Michael H. Crespin and Dragomir R. Radev. Forthcoming. "How to Analyze Political Attention with Minimal Assumptions and Costs." *American Journal of Political Science* .