Lab #5: Document Classification and Clustering

Due in Sakai March 1, 2019 (by 10:00a)

Introduction

For this lab, you will get hands-on experience with some popular forms of supervised and unsupervised machine learning with texts—document classification and document clustering, respectively.

Getting Started

First, set your working directory. Open up a fresh RStudio session (if you open RStudio and a previous work session is loaded, be sure you save any R and/or RData files before you close them out). Then set your working directory:

```
setwd("working_directory_here")
```

Then open up a fresh R script. Save it using your first initial, full last name, and then "_lab5." So my script would be titled **MTaylor_lab5.R**.

You should always strive to keep your scripts tidy. At the top of your R script, type this (substituting in your first initial and last name):

```
##############################
##  MTaylor_lab5.R
##  Note: Code for lab assignment #5
##  Author: Marshall A. Taylor
##############################

###BEGIN###
```

You are ready to begin your code. Be sure to include all the code necessary for me to check your work. Document your code thoroughly (using "#").

Remember that saving your R work is a two-step process. You save your R script using Cmd+Enter (Mac) or Cntrl+Enter (Windows). You save your R data objects like this:

```
save.image("MTaylor_lab5.RData")
```

Lastly, prep a Word, Pages, or LaTeX document that has the same title structure: e.g., **MTaylor_lab5.docx**. This is where you put your write-ups and visualizations.

You should turn in three documents to Sakai: your **R script** that shows the code you used, **RData file** that provides the data your scraped, and **Word document** (or whatever text processor you choose to use) with your write-ups.

Assignment

Note that some of these functions may take a while to run. Plan your time accordingly.

There are two components to this lab. First, you will use a provided corpus to train and apply a two-class na"ive Bayes classifier. Second, you will use hierarchical clustering to group together documents into different corpus.

The first corpus comes from Figure Eight. The researchers scraped a total of 10,876 tweets that used disaster-related terms: e.g., "aftershock," "twister," "rescued," "siren," and so on. They then had a team of coders go through the data and code each tweet as being "relevant"—that is, actually about some sort of disaster—or "not relevant." Tweets were considered not relevant if they were in reference to something other than an actual disaster: a movie review or a joke are a couple of examples the researchers provide.

The second corpus comes from the UCI Machine Learning Repository. The dataset includes 3,159 scraped tweets from a variety of health-related news sources in 2015. I put the data together following the script provided here (Baron 2018).[1] I have provided a random sample of 500 of these tweets to reduce computation time.

Section #1: Classification

For this section, you will divide the first corpus into a training set and a testing set. You will then train a na"ive Bayes classifier to see how well we can predict whether a tweet is about a real disaster based on the words in the tweets.

The corpus file is called "disaster_classify.rds." You can read in the corpus using the `readRDS()` function. Note that I removed 16 tweets that did not have a "relevant" or "not relevant" code. That leaves you with 10,860 tweets.

1. Preprocess your corpus and convert it to a DTM. Keep in mind that these are Twitter data, which introduces a couple of extra text cleaning challenges (see the "Hints" section below for how I preprocessed these tweets). Copy and paste the text of the first three tweets into your Word document.
2. Divide your text data into training and test sets. Make the training set the first 3,000 tweets. The remaining tweets will compose your test set.
3. Check and make sure your proportions across the two coding categories are comparable between your training and test sets (say, within a 3 to 5 percentage points). Copy and paste the proportion distributions for both the training and test sets into your Word document.
4. Make sure your training and test DTMs have the same words (columns). Set the columns in both sets to be all of the terms that occur at least once in the training DTM. How many terms does that give you across the two DTMs?
5. Binarize your matrices. What does that mean, and why do we do it?
6. Train your naïve Bayes classifier. What are the probabilities for the words "riot" and "intrusion" appearing in documents belonging to the "relevant" and "not relevant" categories, conditional on the word appearing at least once in the document? What do these numbers tell us?
7. Use your trained classifier to predict the "relevant"-"not relevant" codes in the test data. This may take a little while; be patient.
8. Take a look at the confusion matrix. How well did we do? How do you know?

Section #2: Clustering

Now let's say we don't have the relevancy codes, and we just want to cluster tweets together that address/talk about similar things.

I'm going to have you make one small change to your clustering algorithm. Tweet data seem to separate into more crisp clusters with Ward's merging criterion rather than the complete-linkage criterion we talked about in class. So I want you to use Ward's method. This is a simple change: instead of using `"complete"` for the `hc_method` argument, you will instead use `"ward.D2"`.

[1] Baron also provides a cluster analysis example using these data.

The corpus file is called "health_cluster.rds." We won't worry about making a dendrogram here since it would be difficult to interpret with this many cases.

1. Convert your corpus to a preprocessed DTM. You can follow the same steps you did for cleaning up the classification data. Copy and paste the text of the first three tweets into your Word document.
2. Convert your DTM into a square matrix of tweet-by-tweet cosine similarities. How do you interpret any given cell in this matrix?
3. Convert your cosine similarities into cosine dissimilarities. Why is it important to make this conversion? How does it change the interpretation?
4. Calculate the average silhouette width statistic for 1- to 20-cluster solutions (this may take a while). Make a line plot showing how the average silhouette width changes. What does this plot tell us?
5. Run the clustering algortihm with the Ward's merging criterion. Set `k` to whatever number of clusters you think is appropriate.
6. What words are most distinctively associated with each cluster? How would you interpret these clusters, given the words most highly associated with each of them?

Hints

The R scripts for the dictionary and sentiment analysis slides (pres61.R and pres62.R) might be very helpful. Like, *really really helpful*.

This is the preprocessing workflow I used to clean the tweets. Feel free to customize it any way you want.

```
corpus <- VCorpus(VectorSource(disaster_classify$text))

removeURL <- content_transformer(function(x) gsub("http[^[:space:]]*", "", x))
userNames <- content_transformer(function(x) gsub("\\B@\\w+", "", x))
removeSpecial <- content_transformer(function(x) gsub("[^\x20-\x7E]", "", x))
removeMostPunctuation <-
  content_transformer(function(x, preserve_intra_word_dashes = FALSE)
  {
    rmpunct <- function(x) {
      x <- gsub("#", "\002", x)
      x <- gsub("[[:punct:]]+", "", x)
      gsub("\002", "#", x, fixed = TRUE)
    }
    if (preserve_intra_word_dashes) {
      x <- gsub("(\\w)-(\\w)", "\\1\001\\2", x)
      x <- rmpunct(x)
      gsub("\001", "-", x, fixed = TRUE)
    } else {
      rmpunct(x)
    }
  })
removeRogueTags <- content_transformer(function(x) {return(gsub("# ", "", x))})
attachHash <- content_transformer(function(x) {return(gsub("# ", "#", x))})
    #The lemmatize_strings() function separates the hashtag from its corresponding text
    #(e.g., #hashtag becomes # hashtag). This function re-attaches the hashtags back to
    #their texts.

corpus <- tm_map(corpus, removeURL)
```

```
corpus <- tm_map(corpus, userNames)
corpus <- tm_map(corpus, removeSpecial)
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, removeMostPunctuation)
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removeWords, stopwords("english"))
corpus <- tm_map(corpus, removeRogueTags)
corpus <- tm_map(corpus, stripWhitespace)

corpus <- convert.tm.to.character(corpus)
corpus <- lemmatize_strings(corpus,
                            dictionary=lexicon::hash_lemmas)

corpus <- VCorpus(VectorSource(corpus))
corpus <- tm_map(corpus, attachHash)

inspect(corpus[[1]]) #Make sure everything looks good
```

The `removeURL` bit of code is adapted from here, the `userNames` bit from here, the `removeSpecial` from here, and the `removeMostPunctuation` code is adapted from here.

References

Baron, Justine. 2018. "Text Clustering using R: An Introduction for Data Scientists." Retrieved February 21, 2019 (link).