Text as Data: Homework 2

Jeff Ziegler

August 16, 2017

In this homework assignment we're going to compare the press releases of two senators—Richard Shelby and Jeff Sessions, Republican senators from Alabama. To make this comparison, we're going to download a bigger collection of Senate press releases and then focus on the releases from Shelby and Sessions. We encourage you to spend some time processing these texts this week, because we will use this collection for the next homework assignment as well.

Downloading the Data

The press release collection are stored here:
`https://github.com/lintool/GrimmerSenatePressReleases`

Download the collection as a `.zip` *file, unzip the file on your computer.*

Creating a Document-Term Matrix

We're going to use the files from Richard Shelby and Jeff Sessions to make two different kinds of Document-Term Matrices. The first will consider only the 1000 most used unigrams, while the second (separate) DTM will use the 500 most common trigrams. To create the document-term matrices, use the following recipe.

1) Create two nested dictionaries for both the Shelby and Sessions press releases. The nested dictionary should contain, for each press release:

   – Month of release
   – Year of release
   – Day of release
   – Author (either Shelby or Sessions)
   – The text of the press release

   To create the nested dictionary:

   i) Use `os.listdir` to create lists of both the Sessions and Shelby press releases

ii) The file names are formatted as `DayMonthYearAuthorNumber.txt`. Devise a parsing rule to extract the month, year, day, of the releases

iii) Store all the information in a nested dictionary

```python
# import libraries
from urllib import urlopen
import re
import os
import csv
import nltk
from nltk.corpus import stopwords
import collections

# load all press releases from Shelby and Sessions into nested dictionaries
# first, designate folder where press releases are stored in gitHub
PRfolder = 'Downloads/GrimmerSenatePressReleases-master/raw/'

# create empty list within each dictionary key
# to be filled with press releases and their associated info
pressReleases = {}
pressReleases['day']= []
pressReleases['month']= []
pressReleases['year']= []
pressReleases['senator']= []
pressReleases['text'] = []
# iterate over both Sessions' and Shelby's PRs
for senator in [PRfolder + 'Sessions', PRfolder + 'Shelby']:
  # for each senator, iterate over each press release
    for PR in os.listdir(senator):
        # and append pressReleases with relevant info
        # since the relevant info is located in the file name
        # formatted: DayMonthYearAuthorNumber.txt
        # so, the first two elements of the file name are the days
        pressReleases['day'].append(PR[:2])
        # then the month is the next three elements
        pressReleases['month'].append(PR[2:5])
        # then the year is the next four
        pressReleases['year'].append(PR[5:9])
        # find the characters that precede the file extension .txt
        # can't just take elements since there are extra numbers in file names
        pressReleases['senator'].append(re.sub('[0-9]+.txt', '', PR[9:]))
        # open press release by file name and read in text as string
        pressReleases['text'].append(open(senator + '/' + PR, 'r').read())
```

2) Next, we will find the 1000 most used unigrams and the 500 most used trigrams, after removing/simplifying a set of words

i) discard punctuation, capitalization, and use `word_tokenize` to split the text on white space

ii) Apply the Porter Stemmer to the tokenized documents.

iii) Use the stop words from
`‘http://jmlr.org/papers/volume5/lewis04a/a11-smart-stop-list/english.stop’`
Append to the list:

* shelby
* sessions
* richard
* jeff
* email
* press
* room
* member
* senate

Apply the Porter Stemmer to this list of stop words and discard all stemmed stop words from the press releases.

iv) Form the list of trigrams using the `trigrams` function from `NLTK`

v) Use a python dictionary to count the number of times each unigram is used and a second dictionary to count the number of times each trigram is used. These should be counts over the *whole corpus* (that is, both senators' press releases).

3) Identify the 1000 unigrams used most often and the 500 most often used trigrams. If you're writing trigrams to a csv to analyze somewhere else, be sure to represent each `tuple` without commas.

4) Write a document-term matrix, where each row contains
`Speaker`, $\texttt{Count}_1$, $\texttt{Count}_2$, ..., $\texttt{Count}_{1000}$
for unigrams, and
`Speaker`, $\texttt{Count}_1$, $\texttt{Count}_2$, ..., $\texttt{Count}_{500}$
for trigrams.

Remember, if `foo` is a list, you can count the number of times `x` occurs with
`foo.count(x)`

5) Write the document term matrix for the unigrams and trigrams to separate .csv files. Remember that you'll need to reformat the trigram `tuples` so that you don't end up with extra commas in your column names. We recommend defining a function in python that takes a `tuple`, like
`‘wabash’, ‘college’, ‘best’`

and converts it to
`wabash.college.best`

```python
### Problem 2 through 5

# create function to use Porter stemmer
def porterStem(unstemmedList):
  return [nltk.stem.PorterStemmer().stem(words) for words in unstemmedList]

# create new lists for unigrams and trigrams to be filled with tokens
pressReleases['unigramTokens']= []
pressReleases['trigramTokens']= []

# load a set of stop words from nlkt
# with the other stop word additions
stopWords = stopwords.words('english') + ['shelby', 'sessions', 'richard', 'jeff', 'email', 'press', 'room', 'member', 'senate']
# apply Porter stemmer to stop words
stopWords = porterStem(stopWords)

# edit the text of pressReleases
for PR in range(0,len(pressReleases['text'])):
  # remove capitalization
  textTokens = pressReleases['text'][PR].lower()
  # discard punctuation by removing non-word characters
  textTokens = re.sub("\W", " ", textTokens)
  # and apply Porter stem to tokenized PRs
  textTokens = porterStem(nltk.word_tokenize(textTokens))
  # remove stop words
  textTokens = [x for x in textTokens if x not in stopWords]
  # then append unigramTokens and trigramTokens
  pressReleases['unigramTokens'].append(textTokens)
  trigramTokens = nltk.trigrams(textTokens)
  # create list to be filled with trigrams
  trigramList = []
  # iterate over all trigram tokens and append into list
  for i in trigramTokens:
    trigramList.append(i)
  pressReleases['trigramTokens'].append(trigramList)

# Use a python dictionary to count the number of times each unigram is used
# and a second dictionary to count the number of times each trigram is used.
# These should be counts over the whole corpus (that is, both senators
# press releases).

# create empty dictionaries to be filled with counts of across-document frequency
# for unigrams and trigrams
unigramDict = {}
trigramDict = {}

```

```
# for each press release
for PR in range(0,len(pressReleases['text'])):
  # add counts to totals
    for word in pressReleases['unigramTokens'][PR]:
        if word not in unigramDict:
            unigramDict[word] = 1
        else:
            unigramDict[word] += 1
    # add counts to totals
    for word in pressReleases['trigramTokens'][PR]:
        if word not in trigramDict:
            trigramDict[word] = 1
        else:
            trigramDict[word] += 1

# sort unigrams and trigrams into new lists
mostNunigrams = []
mostNtrigrams = []

# create function to take the most used words
def extractTopN(topsList, mostNgrams):
  # loop over dictionary and append new list
  # by value, rather than key
  # then sort list
  return sorted(topsList, key=topsList.get, reverse=True)

# extract the most used unigrams
mostNunigrams = extractTopN(unigramDict, mostNunigrams)
# take only the top 1000
mostNunigrams = mostNunigrams[:1000]
# extract the most used trigrams
mostNtrigrams = extractTopN(trigramDict, mostNtrigrams)
# take only the top 500
mostNtrigrams = mostNtrigrams[:500]

# create DTM matrix and write it to .csv
# task: we need to check whether each of the top 1000 words
# is in each press release, and count their frequency
# will iterate over each press release
# first, open up .csv writer
with open('Documents/Git/WUSTL_textAnalysis/PRunigrams.csv', 'wb') as f:
    w = csv.writer(f)
    # create header to be written to .csv as variable names
    # 1st column is senator name, preceding columns
    # represent top 1000 unigrams
    csvHeader = mostNunigrams
    csvHeader.insert(0,'senator')
    # write header first
    w.writerow(csvHeader)
    # then, we need to create counts of all the words
  # in each document (NOT across authors or
```

```
  # documents like the previous problem)
  # so, for each press release
    for PR in range(0,len(pressReleases['text'])):
    # create a clear row
  rowEntry = []
  # and give each unigram count in that press release
  for unigram in mostNunigrams:
    rowEntry.append(pressReleases['unigramTokens'][PR].count(unigram))
  rowEntry.insert(0,pressReleases['senator'][PR])
  # write row
  w.writerow(rowEntry)

# now do for trigrams as well
with open('Documents/Git/WUSTL_textAnalysis/PRtrigrams.csv', 'wb') as f:
    w = csv.writer(f)
    csvHeader = mostNtrigrams
    csvHeader.insert(0,'senator')
    w.writerow(csvHeader)
    for PR in range(0,len(pressReleases['text'])):
  rowEntry = []
  for trigram in mostNtrigrams:
    rowEntry.append(pressReleases['trigramTokens'][PR].count(trigram))
  rowEntry.insert(0,pressReleases['senator'][PR])
  w.writerow(rowEntry)
```

Applying Word Separating Algorithms

1) Using the document-term matrix, for both unigrams and trigrams create the following three measures of word separation

   i) Independent linear discriminant⇝ measure used in Mosteller and Wallace (1963)

   ii) Standardized mean difference⇝ For each word J calculate:

$$\text{std diff} \;=\; \frac{\text{Difference in author means}}{\text{Standard error, diff. in means}}$$

   iii) Standardized Log Odds⇝ as described in Monroe, Colaresi, and Quinn (2009). To create the scores, set $\alpha_j = 1$

```
# load libraries and .csv files
library(foreign); library(dplyr); library(tidyr); library(matrixStats);
    library(lsa); library(KRLS)

# since I was unable to properly clean the files in python...
# remove weird second column
unigrams <- read.csv("~/Documents/Git/WUSTL_textAnalysis/PRunigrams.csv", row.
    names=NULL)[,-2]
names(unigrams)[1] <- "senator"
# include check.names=F, otherwise you will get odd variable names
```

```r
trigrams <- read.csv("~/Documents/Git/WUSTL_textAnalysis/PRtrigrams.csv", row.names=NULL, check.names = FALSE)[,-2]
# then clean up rest of names by removing additional text
# every variable name should just have word1.word2.word3 now
names(trigrams) <- gsub(" u'", "", names(trigrams), fixed = TRUE)
names(trigrams) <- gsub("(u'", "", names(trigrams), fixed = TRUE)
names(trigrams) <- gsub(",", ".", names(trigrams), fixed = TRUE)
names(trigrams) <- gsub("'", "", names(trigrams), fixed = TRUE)
names(trigrams) <- gsub(")", "", names(trigrams), fixed = TRUE); names(trigrams)[1] <- "senator"

### Problem 2

# create function that performs all three measures of word separation:
# 1) linear discriminant analysis,
# 2) standardized mean difference,
# 3) and standardized log odds

wordSeparation <- function(unigramDTM){
  # first, create separate DTMs for each senator
  # and remove 'senator' variable
  sessionsDTM <- unigramDTM[grep('Sessions', unigramDTM$senator),-1]
  shelbyDTM <- unigramDTM[-(grep('Sessions', unigramDTM$senator)),-1]

  # need to calculate means and variances for each column
  # in both senator DTMs
  sessionsMeans <- colSums(sessionsDTM) / sum(colSums(sessionsDTM))
  sessionsVariances <- colVars(as.matrix(sessionsDTM))
  shelbyMeans <- colSums(shelbyDTM) / sum(colSums(shelbyDTM))
  shelbyVariances <- colVars(as.matrix(shelbyDTM))

  # 1) linear discriminant analysis
  # which is difference in unigram means between authors over sum of variances across author
  wordSepTable <- data.frame("linearDiscriminant" = cbind((sessionsMeans - shelbyMeans) / (sessionsVariances + shelbyVariances)))

  # 2) standardized mean difference
  # take the differences of means between authors
  # over standard error, difference of means
  # (or in other words standardize by taking the sqrt of the sum of authors variance/n)
  wordSepTable$standMeanDiff <- (sessionsMeans - shelbyMeans) / sqrt((sessionsVariances/sum(colSums(sessionsDTM))) +
                                                                      (shelbyVariances/sum(colSums(shelbyDTM))))

  # 3) and standardized log odds
  # we need to first take each column sum + alpha (which = 1)
  # over the total n of the author DTM + sum of alphas (which = # of cols - 1)
```

```
  piSessions <- (colSums(sessionsDTM) + 1) / (sum(colSums(sessionsDTM)) + ncol
    (sessionsDTM)-1)
  piShelby <- (colSums(shelbyDTM) + 1) / (sum(colSums(shelbyDTM)) + ncol(
    shelbyDTM)-1)
  # to get the log odds ratio
  # take log (pi1/1-p1) - log(pi2/1-pi2)
  logOdds <- log(piSessions/(1-piSessions)) - log(piShelby / (1-piShelby))
  # finally standardize the log odds ratio
  # by taking the sqrt of the variance of the ratio
  # ~ 1/(x1+1) + 1/(x2+1)
  wordSepTable$standLogOdds <- logOdds/sqrt(var(logOdds))

  # return table of different word separation measures
  return(wordSepTable)
}

# execute function separately for senator, and then by type (unigram, trigram)
unigramWordSep <- wordSeparation(unigrams); unigramWordSep$grams <- rownames(
    unigramWordSep)
trigramWordSep <- wordSeparation(trigrams); trigramWordSep$grams <- rownames(
    trigramWordSep)
```

2) Create a plot for each of the measures that shows the most discriminating words. Some helpful functions are
   `plot`, but set `pch = ‘’`
   `text` allows the placement of texts on plots. Can we learn anything about how Jeff Sessions and Richard Shelby present their work to their constituents?

```
# part 2: create plots

# take random sample of 10 observations (unigrams and trigrams)
set.seed(4); plotUnigrams <- sample_n(unigramWordSep, 15); plotTrigrams <-
    sample_n(trigramWordSep, 15)
pdf("~/Documents/Git/WUSTL_textAnalysis/HW2wordDistanceUnigrams.pdf")
par(mfrow=c(1,3))
# plot 1: linear discriminant
plot(plotUnigrams$linearDiscriminant, pch='', xaxt="n", xlab="", ylab="Weight"
    , main="Linear Discriminant")
text(plotUnigrams$linearDiscriminant, label=plotUnigrams$grams, cex=.9)

# plot 2: standardized mean difference
plot(plotUnigrams$standMeanDiff, pch='', xaxt="n", xlab="", ylab="Weight",
    main="Standardized Mean Difference")
text(plotUnigrams$standMeanDiff, label=plotUnigrams$grams, cex=.9)

# plot 2: standardized mean difference
plot(plotUnigrams$standLogOdds, pch='', xaxt="n", xlab="", ylab="Weight", main
    ="Standardized Log Odds")
```

```r
text(plotUnigrams$standLogOdds, label=plotUnigrams$grams, cex=.9)
dev.off()

# same for trigrams
pdf("~/Documents/Git/WUSTL_textAnalysis/HW2wordDistanceTrigrams.pdf")
par(mfrow=c(1,3))
# plot 1: linear discriminant
plot(plotTrigrams$linearDiscriminant, pch='', xaxt="n", xlab="", ylab="Weight"
    , main="Linear Discriminant")
text(plotTrigrams$linearDiscriminant, label=plotTrigrams$grams, cex=.7)

# plot 2: standardized mean difference
plot(plotTrigrams$standMeanDiff, pch='', xaxt="n", xlab="", ylab="Weight",
    main="Standardized Mean Difference")
text(plotTrigrams$standMeanDiff, label=plotTrigrams$grams, cex=.7)

# plot 2: standardized mean difference
plot(plotTrigrams$standLogOdds, pch='', xaxt="n", xlab="", ylab="Weight", main
    ="Standardized Log Odds")
text(plotTrigrams$standLogOdds, label=plotTrigrams$grams, cex=.7)
dev.off()
```

3) Compare the discriminating measures in 3 plots. What are the primary differences across the measures?

Figure 1: Word separation measures for unigrams.

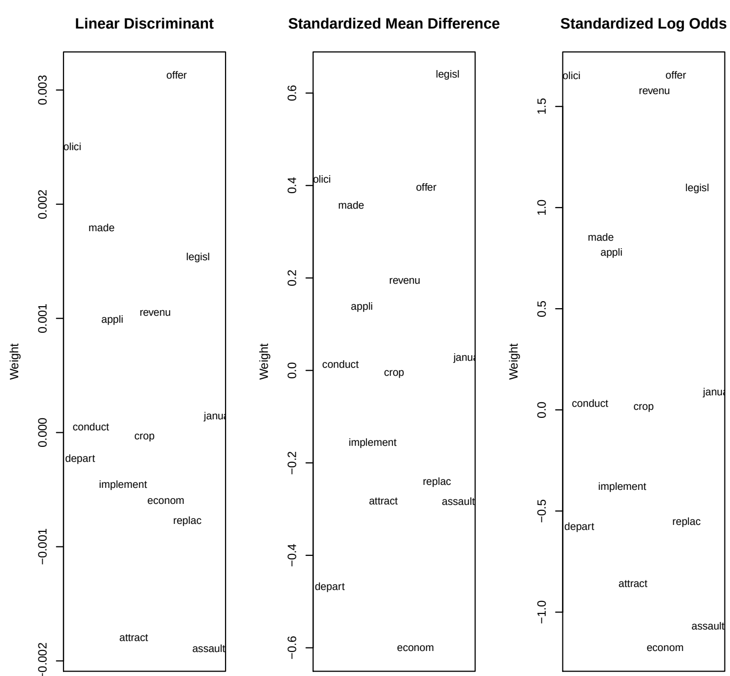
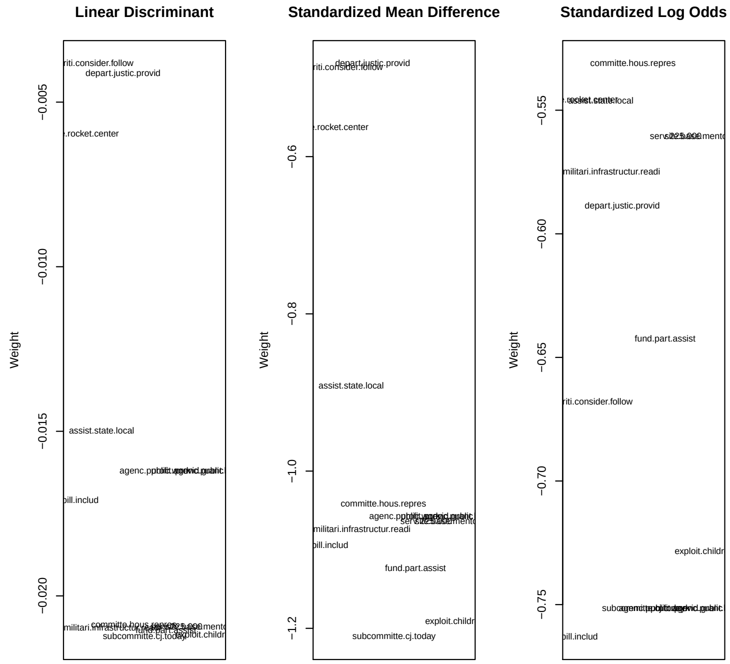


Figure 2: Word separation measures for trigrams.



Both senators seem to talk about legislation and politics.

Comparing Document Similarity

Using the trigram word document matrix, let's compare 100 Shelby press releases to 100 Sessions press releases.

1) Devise a method to sample 100 press releases from each senator's collection

2) Create the following six matrices:

   i) Euclidean distance between documents
   ii) Euclidean distance between documents with tf-idf weights
   iii) Cosine similarity between documents
   iv) Cosine similarity between documents with tf-idf weights
   v) Normalize the rows of the trigram document term matrix. For row i,

$$\boldsymbol{x}_i^* = \frac{\boldsymbol{x}_i}{\sum_{j=1}^{500} x_{ij}}$$

   Then apply the `Gaussian` kernel to the normalized matrices

   vi) Use the same normalization, but now with tf-idf weights. Apply the Gaussian kernel.

3) Using the matrices, identify the most similar (smallest distance) and dissimilar (greatest distance) press releases. Read the pairs of press releases—do they appear to actually be similar? Which method appears to perform best?

```r
### Problem 3

# sample 100 documents from Shelby and Sessions
trigramsSample <- rbind(trigrams[sample(which(trigrams$senator=="Shelby"), 100, replace=F),],
                        trigrams[sample(which(trigrams$senator=="Sessions"), 100, replace=F),])

# find different document difference measures

# i) Euclidean distance
# since we now have the sample in a matrix
# compute the distance matrix by measuring
# distances between the rows of a data matrix
# since we're measuring the length between vectors
# and the press releases are represented as vectors in each row
euclideanDistMatrix <- as.matrix(dist(trigramsSample, method = "euclidean"))

# ii) Euclidean distance with tf-idf weights
# first, get idf = log((total documents)/(number of docs with the term))
# since these are trigrams, this will severely down weight
euclideanIDF <- log(nrow(euclideanDistMatrix)/colSums(euclideanDistMatrix))
# create TI-IDF matrix to fill
euclideanTFIDF <- euclideanDistMatrix
# now take the inner product
for(word in names(euclideanIDF)){
  euclideanTFIDF[,word] <- euclideanDistMatrix[,word] * euclideanIDF[word]
}

# iii) Cosine similarity
# calculate cosine of the each transposed row
cosineSimilarMatrix <- as.matrix(cosine((euclideanDistMatrix)))

# iv) Cosine similarity with tf-idf weights
cosineIDF <- log(nrow(cosineSimilarMatrix)/colSums(cosineSimilarMatrix))
# create TI-IDF matrix to fill
cosineTFIDF <- cosineSimilarMatrix
# now take the inner product
for(word in names(cosineIDF)){
  cosineTFIDF[,word] <- cosineSimilarMatrix[,word] * cosineIDF[word]
}

# v) normalize rows of the trigram document term matrix
trigramNorm <- trigramsSample[,-1]
for (i in 1:nrow(trigramNorm)){
  trigramNorm[i,]<- trigramNorm[i,]/sum(trigramNorm[i,])
}
```

```r
# w/ Gaussian kernel
# k(x_i,x_j) = exp(   || x_i    x+j || ^2) / sigma^2
trigramNormGaussian <- gausskernel(trigramNorm,1)

# vi) normalize Gaussian kernel with tf-idf weights
ngkIDF <- log(nrow(trigramNormGaussian)/colSums(trigramNormGaussian))
# create TI-IDF matrix to fill
ngkTFIDF <- trigramNormGaussian
# now take the inner product
for(word in names(ngkIDF)){
  ngkTFIDF[,word] <- trigramNormGaussian[,word] * ngkIDF[word]
}

### Problem 3

# find most similar documents in each matrix
findSimilarDocs <- function(inputMatrix){
  which(inputMatrix == max(inputMatrix), arr.ind = TRUE)
}

# there are a lot of ties, especially with the TF-IDF weights
findSimilarDocs(euclideanDistMatrix)
findSimilarDocs(euclideanTFIDF)
findSimilarDocs(cosineSimilarMatrix)
findSimilarDocs(cosineTFIDF)
findSimilarDocs(trigramNormGaussian)
findSimilarDocs(ngkTFIDF)
```

```
# there are a lot of ties, but some reasonable reasonable similar documents were:
> findSimilarDocs(ngkTFIDF)
    row col
941  67  67
> findSimilarDocs(euclideanDistMatrix)
    row col
747  43  36
799  36  43
> findSimilarDocs(ngkTFIDF)
    row col
941  67  67
> findSimilarDocs(cosineTFIDF)
    row col
648  32  32
```