

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

```

1 # import libraries
2 from urllib import urlopen
3 import re
4 import os
5 import csv
6 import nltk
7 from nltk.corpus import stopwords
8 import collections
9
10 # load all press releases from Shelby and Sessions into nested
    dictionaries
11 # first, designate folder where press releases are stored in gitHub
12 PRfolder = 'Downloads/GrimmerSenatePressReleases-master/raw/'
13
14 # create empty list within each dictionary key
15 # to be filled with press releases and their associated info
16 pressReleases = {}
17 pressReleases['day'] = []
18 pressReleases['month'] = []
19 pressReleases['year'] = []
20 pressReleases['senator'] = []
21 pressReleases['text'] = []
22 # iterate over both Sessions' and Shelby's PRs
23 for senator in [PRfolder + 'Sessions', PRfolder + 'Shelby']:
24     # for each senator, iterate over each press release
25     for PR in os.listdir(senator):
26         # and append pressReleases with relevant info
27         # since the relevant info is located in the file name
28         # formatted: DayMonthYearAuthorNumber.txt
29         # so, the first two elements of the file name are the days
30         pressReleases['day'].append(PR[:2])
31         # then the month is the next three elements
32         pressReleases['month'].append(PR[2:5])
33         # then the year is the next four
34         pressReleases['year'].append(PR[5:9])
35         # find the characters that precede the file extension .txt
36         # can't just take elements since there are extra numbers in file
    names
37         pressReleases['senator'].append(re.sub('[0-9]+.txt', '', PR[9:]))
38         # open press release by file name and read in text as string
39         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, Count₁, Count₂, ..., Count₁₀₀₀
for unigrams, and
Speaker, Count₁, Count₂, ..., Count₅₀₀
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

```
1 ##### Problem 2 through 5
2
3 # create function to use Porter stemmer
4 def porterStem(unstemmedList):
5     return [nltk.stem.PorterStemmer().stem(words) for words in unstemmedList]
6
7 # create new lists for unigrams and trigrams to be filled with tokens
8 pressReleases['unigramTokens'] = []
9 pressReleases['trigramTokens'] = []
10
11 # load a set of stop words from nlkt
12 # with the other stop word additions
13 stopWords = stopwords.words('english') + ['shelby', 'sessions', 'richard', 'jeff', 'email', 'press', 'room', 'member', 'senate']
14 # apply Porter stemmer to stop words
15 stopWords = porterStem(stopWords)
16
17 # edit the text of pressReleases
18 for PR in range(0, len(pressReleases['text'])):
19     # remove capitalization
20     textTokens = pressReleases['text'][PR].lower()
21     # discard punctuation by removing non-word characters
22     textTokens = re.sub("\W", " ", textTokens)
23     # and apply Porter stem to tokenized PRs
24     textTokens = porterStem(nltk.word_tokenize(textTokens))
25     # remove stop words
26     textTokens = [x for x in textTokens if x not in stopWords]
27     # then append unigramTokens and trigramTokens
28     pressReleases['unigramTokens'].append(textTokens)
29     trigramTokens = nltk.trigrams(textTokens)
30     # create list to be filled with trigrams
31     trigramList = []
32     # iterate over all trigram tokens and append into list
33     for i in trigramTokens:
34         trigramList.append(i)
35     pressReleases['trigramTokens'].append(trigramList)
36
37 # Use a python dictionary to count the number of times each unigram is used
38 # and a second dictionary to count the number of times each trigram is used.
39 # These should be counts over the whole corpus (that is, both senators
40 # press releases).
41
42 # create empty dictionaries to be filled with counts of across-document
    frequency
43 # for unigrams and trigrams
44 unigramDict = {}
45 trigramDict = {}
46
```

```

47 # for each press release
48 for PR in range(0, len(pressReleases[ 'text' ])):
49     # add counts to totals
50     for word in pressReleases[ 'unigramTokens' ][PR]:
51         if word not in unigramDict:
52             unigramDict[word] = 1
53         else:
54             unigramDict[word] += 1
55     # add counts to totals
56     for word in pressReleases[ 'trigramTokens' ][PR]:
57         if word not in trigramDict:
58             trigramDict[word] = 1
59         else:
60             trigramDict[word] += 1
61
62 # sort unigrams and trigrams into new lists
63 mostNunigrams = []
64 mostNtrigrams = []
65
66 # create function to take the most used words
67 def extractTopN(topsList, mostNgrams):
68     # loop over dictionary and append new list
69     # by value, rather than key
70     # then sort list
71     return sorted(topsList, key=topsList.get, reverse=True)
72
73 # extract the most used unigrams
74 mostNunigrams = extractTopN(unigramDict, mostNunigrams)
75 # take only the top 1000
76 mostNunigrams = mostNunigrams[:1000]
77 # extract the most used trigrams
78 mostNtrigrams = extractTopN(trigramDict, mostNtrigrams)
79 # take only the top 500
80 mostNtrigrams = mostNtrigrams[:500]
81
82 # create DIM matrix and write it to .csv
83 # task: we need to check whether each of the top 1000 words
84 # is in each press release, and count their frequency
85 # will iterate over each press release
86 # first, open up .csv writer
87 with open('Documents/Git/WUSTL_textAnalysis/PRunigrams.csv', 'wb') as f:
88     w = csv.writer(f)
89     # create header to be written to .csv as variable names
90     # 1st column is senator name, preceding columns
91     # represent top 1000 unigrams
92     csvHeader = mostNunigrams
93     csvHeader.insert(0, 'senator')
94     # write header first
95     w.writerow(csvHeader)
96     # then, we need to create counts of all the words
97     # in each document (NOT across authors or

```

```

98 # documents like the previous problem)
99 # so, for each press release
100     for PR in range(0, len(pressReleases[ 'text' ])):
101         # create a clear row
102         rowEntry = []
103         # and give each unigram count in that press release
104         for unigram in mostNunigrams:
105             rowEntry.append(pressReleases[ 'unigramTokens' ][PR].count(unigram))
106         rowEntry.insert(0, pressReleases[ 'senator' ][PR])
107         # write row
108         w.writerow(rowEntry)
109
110 # now do for trigrams as well
111 with open( 'Documents/Git/WUSTL_textAnalysis/PRtrigrams.csv', 'wb') as f:
112     w = csv.writer(f)
113     csvHeader = mostNtrigrams
114     csvHeader.insert(0, 'senator')
115     w.writerow(csvHeader)
116     for PR in range(0, len(pressReleases[ 'text' ])):
117         rowEntry = []
118         for trigram in mostNtrigrams:
119             rowEntry.append(pressReleases[ 'trigramTokens' ][PR].count(trigram))
120         rowEntry.insert(0, pressReleases[ 'senator' ][PR])
121         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 $\alpha_j = 1$

```

1 # load libraries and .csv files
2 library(foreign); library(dplyr); library(tidyr); library(matrixStats)
3
4 # since I was unable to properly clean the files in python...
5 # remove weird second column
6 unigrams <- read.csv("~/Documents/Git/WUSIL_textAnalysis/PRunigrams.csv", row.
    names=NULL)[, -2]
7 names(unigrams)[1] <- "senator"
8 # include check.names=F, otherwise you will get odd variable names
9 trigrams <- read.csv("~/Documents/Git/WUSIL_textAnalysis/PRtrigrams.csv", row.
    names=NULL, check.names = FALSE)[, -2]

```

```

10 # then clean up rest of names by removing additional text
11 # every variable name should just have word1.word2.word3 now
12 names(trigrams) <- gsub(" u'", "", names(trigrams), fixed = TRUE)
13 names(trigrams) <- gsub("(u'", "", names(trigrams), fixed = TRUE)
14 names(trigrams) <- gsub(", ", ".", names(trigrams), fixed = TRUE)
15 names(trigrams) <- gsub("'", "", names(trigrams), fixed = TRUE)
16 names(trigrams) <- gsub(")", "", names(trigrams), fixed = TRUE); names(
    trigrams)[1] <- "senator"
17
18 ### Problem 2
19
20 # create function that performs all three measures of word separation:
21 # 1) linear discriminant analysis,
22 # 2) standardized mean difference,
23 # 3) and standardized log odds
24
25 wordSeparation <- function(unigramDTM){
26   # first, create separate DTMs for each senator
27   # and remove 'senator' variable
28   sessionsDTM <- unigramDTM[grep('Sessions', unigramDTM$senator), -1]
29   shelbyDTM <- unigramDTM[-(grep('Sessions', unigramDTM$senator)), -1]
30
31   # need to calculate means and variances for each column
32   # in both senator DTMs
33   sessionsMeans <- colSums(sessionsDTM) / sum(colSums(sessionsDTM))
34   sessionsVariances <- colVars(as.matrix(sessionsDTM))
35   shelbyMeans <- colSums(shelbyDTM) / sum(colSums(shelbyDTM))
36   shelbyVariances <- colVars(as.matrix(shelbyDTM))
37
38   # 1) linear discriminant analysis
39   # which is difference in unigram means between authors over sum of variances
    across author
40   wordSepTable <- data.frame("linearDiscriminant" = cbind((sessionsMeans -
    shelbyMeans) / (sessionsVariances + shelbyVariances)))
41
42   # 2) standardized mean difference
43   # take the differences of means between authors
44   # over standard error, difference of means
45   # (or in other words standardize by taking the sqrt of the sum of authors
    variance/n)
46   wordSepTable$standMeanDiff <- (sessionsMeans - shelbyMeans) / sqrt((
    sessionsVariances/sum(colSums(sessionsDTM))) +
47   (
    shelbyVariances/sum(colSums(shelbyDTM))))
48
49   # 3) and standardized log odds
50   # we need to first take each column sum + alpha (which = 1)
51   # over the total n of the author DIM + sum of alphas (which = # of cols - 1)
52   piSessions <- (colSums(sessionsDTM) + 1) / (sum(colSums(sessionsDTM)) + ncol
    (sessionsDTM)-1)

```

```

53 piShelby <- (colSums(shelbyDTM) + 1) / (sum(colSums(shelbyDTM)) + ncol(
    shelbyDTM) - 1)
54 # to get the log odds ratio
55 # take log (pi1/(1-p1) - log(pi2/(1-pi2))
56 logOdds <- log(piSessions/(1-piSessions)) - log(piShelby / (1-piShelby))
57 # finally standardize the log odds ratio
58 # by taking the sqrt of the variance of the ratio
59 # ~ 1/(x1+1) + 1/(x2+1)
60 wordSepTable$standLogOdds <- logOdds/sqrt(var(logOdds))
61
62 # return table of different word separation measures
63 return(wordSepTable)
64 }
65
66 # execute function separately for senator, and then by type (unigram, trigram)
67 unigramWordSep <- wordSeparation(unigrams); unigramWordSep$grams <- rownames(
    unigramWordSep)
68 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?

```

1 # part 2: create plots
2
3 # take random sample of 10 observations (unigrams and trigrams)
4 set.seed(4); plotUnigrams <- sample_n(unigramWordSep, 15); plotTrigrams <-
    sample_n(trigramWordSep, 15)
5 pdf("~/Documents/Git/WUSIL_textAnalysis/HW2wordDistanceUnigrams.pdf")
6 par(mfrow=c(1,3))
7 # plot 1: linear discriminant
8 plot(plotUnigrams$linearDiscriminant, pch='', xaxt="n", xlab="", ylab="Weight",
    , main="Linear Discriminant")
9 text(plotUnigrams$linearDiscriminant, label=plotUnigrams$grams, cex=.9)
10
11 # plot 2: standardized mean difference
12 plot(plotUnigrams$standMeanDiff, pch='', xaxt="n", xlab="", ylab="Weight",
    , main="Standardized Mean Difference")
13 text(plotUnigrams$standMeanDiff, label=plotUnigrams$grams, cex=.9)
14
15 # plot 2: standardized mean difference
16 plot(plotUnigrams$standLogOdds, pch='', xaxt="n", xlab="", ylab="Weight", main=
    "Standardized Log Odds")
17 text(plotUnigrams$standLogOdds, label=plotUnigrams$grams, cex=.9)
18 dev.off()

```



```

19
20 # same for trigrams
21 pdf("~/Documents/Git/WUSIL_textAnalysis/HW2wordDistanceTrigrams.pdf")
22 par(mfrow=c(1,3))
23 # plot 1: linear discriminant
24 plot(plotTrigrams$linearDiscriminant, pch='', xaxt="n", xlab="", ylab="Weight",
      , main="Linear Discriminant")
25 text(plotTrigrams$linearDiscriminant, label=plotTrigrams$grams, cex=.7)
26
27 # plot 2: standardized mean difference
28 plot(plotTrigrams$standMeanDiff, pch='', xaxt="n", xlab="", ylab="Weight",
      , main="Standardized Mean Difference")
29 text(plotTrigrams$standMeanDiff, label=plotTrigrams$grams, cex=.7)
30
31 # plot 2: standardized mean difference
32 plot(plotTrigrams$standLogOdds, pch='', xaxt="n", xlab="", ylab="Weight", main=
      ="Standardized Log Odds")
33 text(plotTrigrams$standLogOdds, label=plotTrigrams$grams, cex=.7)
34 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.

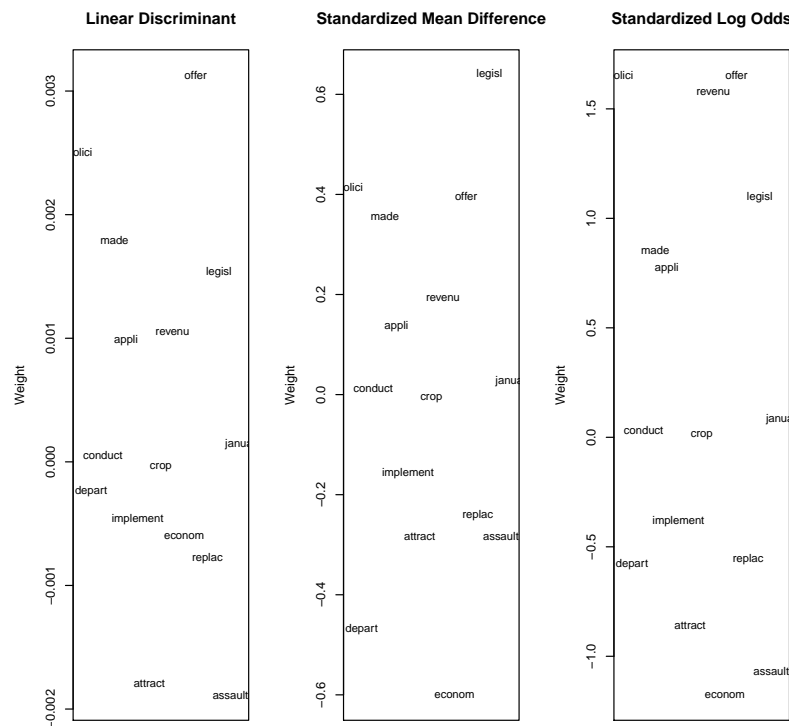
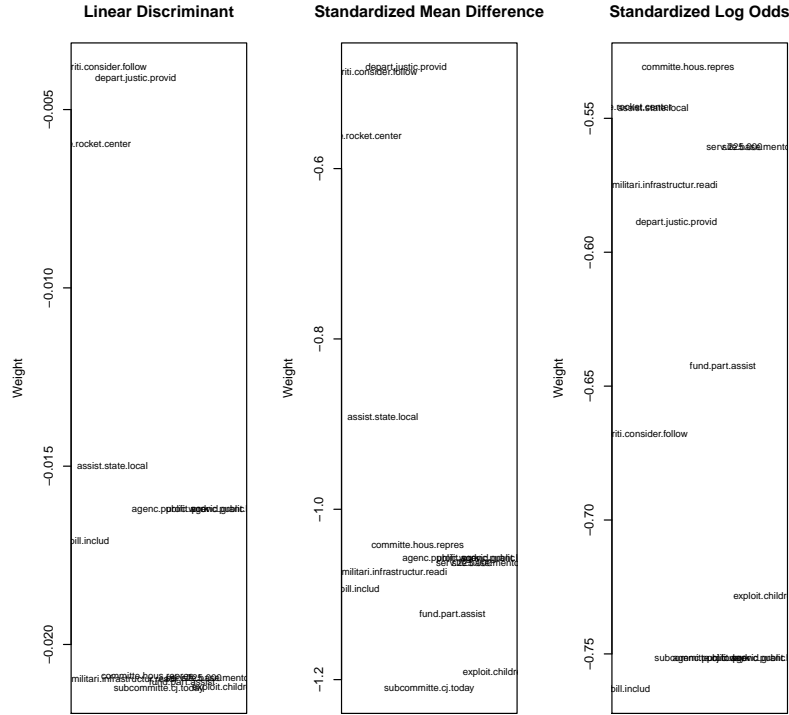


Figure 2: Word separation measures for trigrams.



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 ,

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

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?