An Introduction to Information Retrieval, Part II

STA 325, Supplemental Material

Install packages

```r
library("xtable")
```

```
## Warning: package 'xtable' was built under R version 3.4.
```

Information retrieval applied to NY Times

Let's consider applying our information retrieval skills to a corpus of documents from the NY Times.

NY Times corpus

Let's start by seeing what kind of data we have. To do this, we're going to use some pre-written functions located in `ir.R` that you are welcome to use.

Loading scripts

```r
source("scripts/ir.R")
```

Reading in the data

```
library(XML)
music <- read.directory("nyt_corpus/music")
art <- read.directory("nyt_corpus/art")
```

Let's examine the data

```r
length(music)
```

```
## [1] 45
```

```r
length(art)
```

```
## [1] 57
```

We have 45 music articles and 57 art articles from the NYTimes.

Let's examine the data

1. What command would you use to extract the 37th word of story number 1595645 in art? (That word is "experiencing".)
2. Give a command to count the number of times the word "the" appears in that story.

(Try this on your own and don't look at the solution.)

Solution

```
which(dir("nyt_corpus/art")=="1595645.xml")
```

```
## [1] 48
```

The 37th word is

```
art[[48]][37]
```

```
## [1] "experiencing"
```

Solution (continued)

One way to see how often "the" appears in the story above is using the following command:

```
sum(art[[48]] == "the")
```

```
## [1] 103
```

Bag of words

3. Give the commands you would use to construct a bag-of-words data-frame from the document vectors for the art and music stories.

(Try this on your own and don't look at the solution.)

Solution

lapply is one of the most useful functions in R: it takes two arguments, a data structure (vector, list, array,. . .) and a function, and returns a list which it gets by applying the function to each element of the data structure.

```
art.bow <- lapply(art,table)
music.bow <- lapply(music,table)
is.list(music.bow)
```

```
## [1] TRUE
```

```
length(music.bow)
```

```
## [1] 45
```

Solution (continued)

```
nyt.frame <- make.BoW.frame(c(art.bow,music.bow))
dim(nyt.frame)
```

```
## [1]  102 4431
```

The number of rows should equal the total number of stories, and $45+57 = 102$.

Distance matrices

Create distance matrices from this data frame for (a) the straight Euclidean distance, (b) the distance with word-count normalization and (c) the distance with vector-length scaling, and then for all three again with inverse-document-frequency weighting. Be sure to give the commands that you use. [1]

[1] Note: There are six different distance commands that can be found in the function `distances`. Some of these were covered in class and some of these are new ones that you have not seen before that you will explore during the assignment.

Partial solution

Let's just look at seeing how to tackle this problem.

```
dist.plain = distances(nyt.frame)
dim(dist.plain)
```

```
## [1] 102 102
```

```
round(dist.plain[1:5,1:5],2)
```

```
##        [,1]   [,2]   [,3]   [,4]   [,5]
## [1,]   0.00 163.82 116.99 129.27 132.92
## [2,] 163.82   0.00  87.26  87.42  78.60
## [3,] 116.99  87.26   0.00  68.26  78.89
## [4,] 129.27  87.42  68.26   0.00  80.46
## [5,] 132.92  78.60  78.89  80.46   0.00
```

```
dist.wordcount = distances(div.by.sum(nyt.frame))
dist.euclen = distances(div.by.euc.length(nyt.frame))
```

On your own

Now, do the idf weights on your own.

On your own

```
nyt.frame.idf = idf.weight(nyt.frame)
dist.idf.plain = distances(nyt.frame.idf)
dist.idf.wordcount = distances(div.by.sum(nyt.frame.idf))
dist.idf.euclen = distances(div.by.euc.length(nyt.frame.idf))
```

Average distances between stories

For each of the six different difference measures in the function `distances`, what is the average distance between stories in the same category and between stories in different categories?

Solution

There are multiple ways to do this.

The simplest is to realize that, in this case, the first 57 stories are all art, and the last 45 are all music.

- So if d is a distance matrix, the within-category entries are `d[1:57,1:57]` and `d[58:102,58:102]`, and the between-category entries are `d[1:57,58:102]`.
- Then `mean(c(d[1:57,1:57],d[58:102,58:102]))`} would give the average distance between stories in the same category, similarly `mean(d[58:102,1:57])` for the between-category average.

Solution

The `outer()` function takes three arguments: two data-structures and another function. (Here the function is `!=`, which I put in quotes so that R realizes I'm naming a function, and not asking it to evaluate an expression.)

It returns a matrix which it gets from applying the function to each pair of components from its first two arguments.

Here those first two arguments are vectors of length 102, so what it gives back is a 102×102 matrix, where `are.different[i,j]` shows whether `class.label[i] != class.label[j]`. In other words, it's TRUE if documents i and j belong to different classes.

Solution

```
class.labels = c(rep("art",57),rep("music",45))
head(class.labels)
```

```
## [1] "art" "art" "art" "art" "art" "art"
```

```
are.different = outer(class.labels,class.labels,"!=")
head(are.different)
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,1
## [1,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
## [2,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
## [3,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
## [4,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
## [5,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
## [6,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
##      [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,2
## [1,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
## [2,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
## [3,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
## [4,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
## [5,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
## [6,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
```

Solution

And a logical array picks out elements from another array: `mean(d[are.different])` is the average distance between classes. To average within classes, `mean(d[!are.different])` Not only does this work if the classes are intermingled (we just have to get the `class.labels` vector right), we can also use this to not include the distance from a document to itself in the within-class average:

With-in class differences (with self)

```
# calculate with-in class differences, with self
are.same = !are.different
diag(are.same) = TRUE
mean(dist.plain[are.same])
```

```
## [1] 76.43675
```

```
mean(dist.wordcount[are.same])
```

```
## [1] 0.101356
```

```
mean(dist.euclen[are.same])
```

```
## [1] 0.7293624
```

```
mean(dist.idf.plain[are.same])
```

```
## [1] 87.93946
```

```
mean(dist.idf.wordcount[are.same])
```

With-in class differences (without self)

```
# calculate with-in class differences, without self
are.same = !are.different
diag(are.same) = FALSE
mean(dist.plain[are.same])
```

```
## [1] 77.94421
```

```
mean(dist.wordcount[are.same])
```

```
## [1] 0.1033549
```

```
mean(dist.euclen[are.same])
```

```
## [1] 0.7437465
```

```
mean(dist.idf.plain[are.same])
```

```
## [1] 89.67376
```

```
mean(dist.idf.wordcount[are.same])
```

Between category averages

```
# calculate between category averages
mean(dist.plain[58:102,1:57])
```

```
## [1] 78.06805
```

```
mean(dist.wordcount[58:102,1:57])
```

```
## [1] 0.1069678
```

```
mean(dist.euclen[58:102,1:57])
```

```
## [1] 0.7670132
```

```
mean(dist.idf.plain[58:102,1:57])
```

```
## [1] 88.25976
```

```
mean(dist.idf.wordcount[58:102,1:57])
```

```
## [1] 0.1322932
```

Output into an xtable

```
wselfwdif<-c(mean(dist.plain[are.same]),
             mean(dist.wordcount[are.same]),
             mean(dist.euclen[are.same]),
             mean(dist.idf.plain[are.same]),
             mean(dist.idf.wordcount[are.same]),
             mean(dist.idf.euclen[are.same]))
wselfwodif<-c(mean(dist.plain[are.same]),
              mean(dist.wordcount[are.same]),
              mean(dist.euclen[are.same]),
              mean(dist.idf.plain[are.same]),
              mean(dist.idf.wordcount[are.same]),
              mean(dist.idf.euclen[are.same]))
betweendif<-c(mean(dist.plain[58:102,1:57]),
              mean(dist.wordcount[58:102,1:57]),
              mean(dist.euclen[58:102,1:57]),
              mean(dist.idf.plain[58:102,1:57]),
              mean(dist.idf.wordcount[58:102,1:57]),
              mean(dist.idf.euclen[58:102,1:57]))
mat<-cbind(wselfwdif, wselfwodif, betweendif)
rownames(mat)<-c("Plain {without IDF}",
                 "Sum-normed {without IDF}", "Length-normed
                 {without IDF}", "Plain {with IDF}", "Sum-normed
                 {with IDF}", "Length-normed {with IDF}")
colnames(mat)<-c("Within-class, with self", "Within-class, without               self", "Between-class")
```

Print xtable

```r
print(xtable(mat))
```

```
## % latex table generated in R 3.4.1 by xtable 1.8-3 package
## % Tue Oct  2 10:10:16 2018
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrr}
##   \hline
##  & Within-class, with self & Within-class, without              sel
##   \hline
## Plain \{without IDF\} & 77.94 & 77.94 & 78.07 \\
##   Sum-normed \{without IDF\} & 0.10 & 0.10 & 0.11 \\
##   Length-normed
##                 \{without IDF\} & 0.74 & 0.74 & 0.77 \\
##   Plain \{with IDF\} & 89.67 & 89.67 & 88.26 \\
##   Sum-normed
##                 \{with IDF\} & 0.13 & 0.13 & 0.13 \\
##   Length-normed \{with IDF\} & 1.37 & 1.37 & 1.38 \\
##    \hline
## \end{tabular}
## \end{table}
```

Multidimensional scaling plots

Create multidimensional scaling plots for the different distances, and describe what you see. (Include the code you used, the plots, and explanations for the code).

Solution

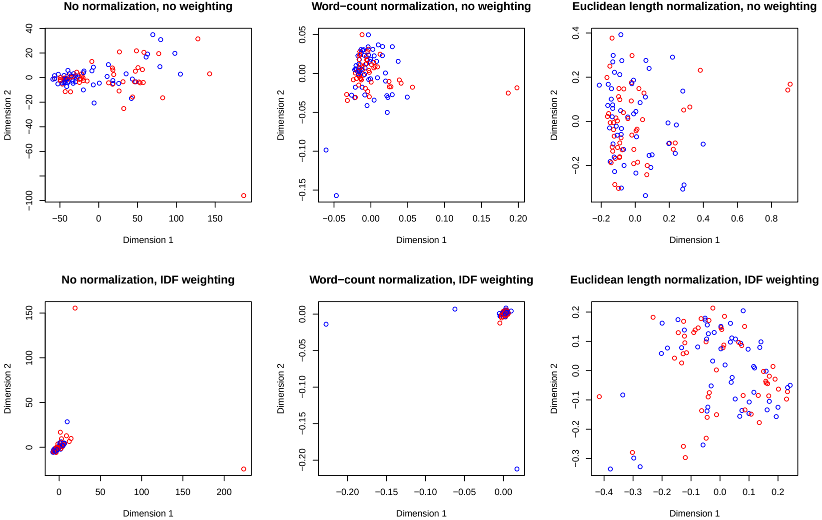
Recall the point of multi-dimensional scaling is such that we can take our distance matrix and represent this in lower dimensions, namely two, so that we can visualize it.

Solution

```r
# we run classical MDS on our distance matrices
head(cmdscale(as.matrix(dist.plain)))
```

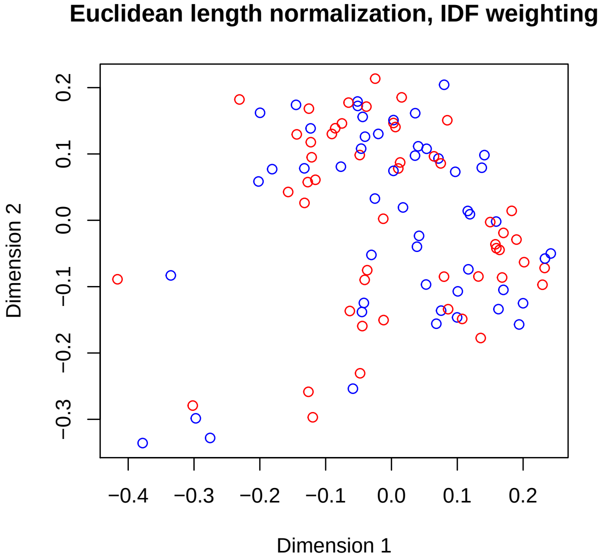
```
##              [,1]      [,2]
## [1,] 142.885340  3.068330
## [2,]  -6.909667  8.437926
## [3,]  53.400617 -4.257171
## [4,]  47.720333 -3.421029
## [5,]  47.205833  5.192395
## [6,]  35.295081  3.055806
```

Solution



Top row, without IDF. Bottom row, with IDF. Left column, un-normalized vectors. Middle column, normalized by word-count. Right column, normalized by Euclidean length. Red circles are art,

Solution



Only with both IDF weights and Euclidean-length normalization do we get reasonable separation of the two categories.