8-Vector_Space_Approaches

Ben Schmidt

3/19/2015

Vector spaces represent an extremely useful way of visualizing and thinking about highly multidimensional data.

Most of the introductory stuff was covered last week: I've also put a more interactive form for exploring this information up at http://benschmidt.org/bookworm/_site/

We'll start by building all the code from last week into a single function to read the SOTUs in as a term-document matrix.

```
library(dplyr)
library(ggplot2)
library(tidyr)

foldername = "data/SOTUS"


folderIntoTD = function (foldername,normalized=F) {
  readSOTU = function(file) {
    message(file)
    text = scan(file,sep="\n",what="raw")
    words = text %>%
      strsplit("[^A-Za-z]") %>%
      unlist
    SOTU = data.frame(word=words,filename=file %>% gsub(foldername,"",.) %>% gsub("/","",.) %>% gsub(".
    return(SOTU)
  }

  SOTUS = list.files(foldername,full.names = TRUE)

  #SOTUS = SOTUS[grep("/19[89]|/20[10]",SOTUS)]

  all = lapply(SOTUS,readSOTU)
  allSOTUs = rbind_all(all)

  counts = allSOTUs %>% mutate(word=tolower(word)) %>% group_by(filename,word) %>% summarize(count=n())

  td = counts %>% group_by(word) %>%
    filter(sum(count)>1000,word!="") %>%
    ungroup %>%
    spread(word,count,fill=0)


  if (normalized) {
      # allow normalization as an argument
      normalized = counts %>% group_by(filename) %>% mutate(ratio = count/sum(count)) %>% ungroup
      norm_data_frame = normalized %>% group_by(word) %>% filter(sum(count)>1000,word!="") %>% select(-
      td  = norm_data_frame %>% ungroup %>% spread(key=word,value=ratio,fill=0)
  }
```

```
  return(td)

}
```

The metadata about parties we'll hold out separately.

```
parties = read.table("data/SOTUparties.tsv",sep="\t",header=T) %>% mutate(year=as.character(year))
```

Note that the function defined above now has *nothing* to do with the state of the unions in particular: you can run it on any folder you choose. In fact, I recommend you do so in class, because it will be helpful for your problem set.

Principal Components Analysis

It should be clear from the demos that there are inexhaustable set of perspective from which to view the term-document matrix. What's the best one?

"Principal Components Analysis" offers one helpful solution to this conundrum.

It takes as input a matrix, rather than a data frame, so we have to remove the filename from our data.

For the state of the Unions, I'll be looking at the *normalized* term document matrix.

```
td = folderIntoTD("data/SOTUS",normalized = T)

justaMatrix = td %>% select(-filename)

model = prcomp(justaMatrix)

#You could also write `justaMatrix %>% prcomp` to keep the dplyr syntax going.
```

The result of this is a model: if you ask for it, you'll see an odd combination of factors.

As we venture out of the world of dplyr into modeling, we'll have to occasionally perform some gymnastics to keep things usable by our familiar arrange, filter, and group functions. The most important will be `as.data.frame`, which can "coerce" the `matrix` format into a `data.frame`.

But to figure out what these objects are composed of, the function `names` is often helpful. You can also always read the documentation, which tells you what you're looking at.

```
names(model)

?prcomp
```

The important elements of principal components are the *weights*, which are available as part of the matrix at `model$rotation`.

We can see what the weights are by

```
model$rotation %>% as.data.frame %>% mutate(word=rownames(model$rotation)) %>% select(word,PC1) %>% arra
model$rotation %>% as.data.frame %>% mutate(word=rownames(model$rotation)) %>% select(word,PC1) %>% arra
# The second "Principal Component"
model$rotation %>% as.data.frame %>% mutate(word=rownames(model$rotation)) %>% select(word,PC2) %>% arra
model$rotation %>% as.data.frame %>% mutate(word=rownames(model$rotation)) %>% select(word,PC2) %>% arra
```

To actually see what this model looks like on our data, we can use the function `predict` with the model we've built.

Again, we have to assign the filename manually: the prediction is the rows as the original term-document matrix, just with new variables. So we can just pull the filenames from there.

```
prediction = predict(model)
prediction = prediction %>% as.data.frame %>% mutate(filename=td$filename) %>% select(filename,PC1,PC2,
```

```
ggplot(prediction) + geom_text(aes(x=PC1,y=PC2,label=filename))

ggplot(prediction) + geom_text(aes(x=PC1,y=as.numeric(filename),label=filename))
ggplot(prediction) + geom_text(aes(x=PC2,y=as.numeric(filename),label=filename))
```

This can be particularly valuable in looking at categories. Here we `inner_join` the parties matrix from earlier on, so that we can colorize by political party.

```
ggplot(prediction %>% mutate(year=filename) %>% inner_join(parties)) + geom_text(aes(x=PC1,y=PC2,label=
```

We can also use any of the progressive different dimensions to look at.

```
ggplot(prediction %>% mutate(year=filename) %>% inner_join(parties)) + geom_text(aes(x=PC2,y=PC3,label=
```

Linear Discriminant Analysis

Another, more exotic option is something called Linear Discriminant Analysis. It, too finds, weights for each vector. (Note that although this is abbreviated as "LDA", it is unrelated to the similarly-named technique that Matt Jockers in *Macroanalysis,* Latent Direchlet Allocation.)

The mechanics here are probably not worth doing yourself, particularly because there are better classification algorithms out there. But it's worth seeing that PCA, which doesn't take classes into account, can be dramatically bested by something that does for the purpose of segregating classes.

```
lda = MASS::lda
withMetadata = td %>% select(-year) %>% mutate(year=filename) %>% select(-filename) %>% inner_join(parti
model = lda(x=withMetadata %>% select(-year,-name,-party),grouping=withMetadata$party)


#That throws an error if you have columns that are the same across classes; you'll need to expand the f

predicted = (withMetadata %>% select(-year,-name,-party) %>% as.matrix) %*% model$scaling

predicted = predicted %>% as.data.frame %>% mutate(year=withMetadata$year,party=withMetadata$party,name=

ggplot(predicted) + geom_text(aes(x=LD1,y=LD2,label=year,color=party))
```

But these sorts of factors aren't necessarily the most useful. You'll see, for example, in the Linear Disciminant example that the first discriminator simply splits apart speeches from the first two party systems from those after Andrew Jackson shook things up, and the second one simultaneously separates Democrats from Republicans and Democratic-Republicans from Federalists. Does this mean that Federalists really share something fundamental with modern Republicans? Almost certainly not. One of the clues is that within the big Republican and Democratic blobs, there's no substantial similarities between individual presidents.

```
ggplot(predicted) + geom_text(aes(x=LD2,y=LD3,label=year,color=party))
#Colorizing by name also shows something different.
ggplot(predicted) + geom_text(aes(x=LD1,y=LD2,label=year,color=name))
```

Distances in Vector Space

```
justaMatrix = td %>% select(-filename)
a = dist(justaMatrix) %>% as.matrix
colnames(a) = td$filename

intermediateFrame = a %>% as.data.frame %>% mutate(element1=td$filename)

tidyDistanceMatrix = intermediateFrame %>% gather(element2,distance,-element1)


tidyDistanceMatrix %>% filter(element1==1994) %>% arrange(-distance) %>% ggplot() + geom_point(aes(x=el
```

You could do the same thing with a log transform. In lots of cases, these turn out to be more appropriate for texts. The only difference in the text below is that I'm adding "logged" into the description.

```
justaMatrix = td %>% select(-filename)
logged = log(justaMatrix + .01)
a = (dist(logged) %>% as.matrix)
colnames(a) = td$filename

intermediateFrame = a %>% as.data.frame %>% mutate(element1=td$filename)

tidyDistanceMatrix = intermediateFrame %>% gather(element2,distance,-element1)


tidyDistanceMatrix %>% filter(element1==1996) %>% arrange(-distance) %>% ggplot() + geom_point(aes(x=el
```

Cosine Similarity