Thinking of Texts as Data

Ben Schmidt

March 5, 2015

For most of the rest of this semester, we'll be talking about a particular *sort* of textual analysis: what are called *bag of words* approaches, and their most important subset, the *vector space* model of texts.

The key feature of a bag of words approach to text analysis is that it ignores the order of words in the original text. These two sentences clearly mean different things:

1. "Not I, said the blind man, as he picked up his hammer and saw."
2. "And I saw, said the not blind man, as he picked up his hammer."

But they consist of the same core set of words. The key question in working with this kind of data is the following: what do similarities like that mean? You'll sometimes encounter a curmudgeon who proudly declares that "literature is not data" before stomping off. This is obviously true. And, as in the concordances we constructed last week or the digital humanities literature that relies more heavily on tools from natural language processing, there are many ways to bring some of the syntactic elements of language into play.

But bag-of-words approaches have their own set of advantages: they make it possible to apply a diverse set of tools and approaches from throughout statistics, machine learning, and data visualization to texts in ways that can often reveal surprising insights. In fact, methods developed to work with non-textual data can often be fruitfully applied to textual data, and methods primarily used on textual data can be used on sources that are not texts. You may have an individual project using humanities data that is *not* textual. For example, we saw last week how Zipf's law of word distributions applies to our city sizes dataset just as it applies to word counts. Almost all of the methods we'll be looking at can be applied as easily, and sometimes as usefully, to other forms of humanities data. (Streets in a city, soldiers in an army, or bibliographic data.) But to do so requires transforming them into a useful model.

The term-document matrix

The key concept in most of these is something called the "term-document matrix." This is a data structure extremely similar to the ones that we have been building so far: in it, each *row* refers to a column.

This is not "tidy data" in Wickham's sense, so it can be easily generated from it using the `spread` function in his `tidyr` package. Let's investigate by looking at some state of the Union Addresses. This is code from last week, but modified to only read in State of the Unions written since 1980. (Can you see where?)

```r
library(dplyr)
library(ggplot2)

readSOTU = function(file) {
  message(file)
  text = scan(file,sep="\n",what="raw")
  words = text %>%
    strsplit("[^A-Za-z]") %>%
    unlist
  SOTU = data.frame(word=words,year=gsub(".*(\\d{4}).*","\\1",file),stringsAsFactors = FALSE)
  return(SOTU)
}
```

```
SOTUS = list.files("data/SOTUS",full.names = TRUE)

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

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

With state of the union addresses, the most obvious definition of a document is a speech: we have that here in the frame "year".

We also want to know how many times each word is used. You know how to do that: through a group_by and `summarize` function.

```
counts = allSOTUs %>% mutate(word=tolower(word)) %>% group_by(year,word) %>% summarize(count=n()) %>% u
```

In a term-document matrix, the terms are the column names, and the values are the counts. To make this into a term-document matrix, we simply have to spread out the terms along these lines.

```
library(tidyr)
td = counts %>% spread(word,count,fill=0)
```

Ordinarily, you might want to look at a `data.frame` in this situation. That's a bad idea here. The function `dim` lets you see how big a data.frame is: in this case, we have one that is 36 rows by 10,000 columns. Those 10,000 columns are the distinct rows.

Later, we'll work with these huge matrices. But for now, let's set up a filter to make the frame a little more manageable.

```
td = counts %>% group_by(word) %>%
  filter(sum(count)>100) %>%
  ungroup %>%
  spread(word,count,fill=0)

dim(td)
```

```
ggplot(td) + aes(x=america,y=freedom,label=year) + geom_text() + geom_path()

ggplot(td) + aes(x=the,y=and,label=year) + geom_text() + geom_path()
```

With this sort of data, it's more useful to compare against a different set of texts.

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

td = td %>% inner_join(parties)
```

Here's an elaborate ggplot of usage of "and" and "for"

```
ggplot(td) + aes(x=`for`,y=and,label=name,color=party,pch=name) + geom_point(size=5)
```

Note that we can get even more complicated: for instance, adding together lots of different words by using math in the aesthetic definition.

(Also note a little trick here. I want to use backticks around the word "if" because it's a **reserved word** in the R syntax.)

```r
ggplot(td) + aes(x=and+or,y=but+`if`,label=name,color=party,pch=name) + geom_point(size=5)
```

As an in-class challenge–can you come up with a chart that accurately distinguishes between Democrats and Republicans?

There's a big problem with this data: it's not *normalized.* Bill Clinton used to talk, a lot, in his state of the union speeches. (We can prove this with a plot)

```r
counts %>% group_by(year) %>%
  summarize(count=sum(count)) %>% ggplot() + aes(x=as.numeric(year),y=count) + geom_line()
```

```r
counts %>%
  filter(word=="i") %>% ggplot() + aes(x=as.numeric(year),y=count) + geom_line()
```

We can normalize these plots so that the term-document matrix gives the relative use per words of text.

```r
normalized = counts %>% group_by(year) %>% mutate(ratio = count/sum(count)) %>% ungroup


normalized %>% filter(word=="we") %>% ggplot() + aes(x=as.numeric(year),y=ratio) + geom_line()
```

```r
norm_data_frame = normalized %>% group_by(word) %>% filter(mean(ratio) > .001) %>% select(-count)

norm_td  = norm_data_frame %>% ungroup %>% spread(key=word,value=ratio,fill=0)
```

If you install the `rgl` library, you can look at some of these plots in 3 dimensions. This is confusing. (And in the version I'm giving you, maybe broken.)

```r
rgl::plot3d(x=as.numeric(norm_td$a),y=norm_td$freedom,z=norm_td$education,col="red")
```