(de)Bugging Ramsay—A Last Stab at tfidf and “The Waves”

Chris Forster

`http://cforster.com/2013/02/returning-to-tfidf`

In my last post I talked about some of the challenges of reproducing the analysis, offered in Stephen Ramsay’s *Reading Machines* of the distinctiveness of characters’ vocabulary in Woolf’s *The Waves*. Here I follow up a bit more on why I was unable to get the same results Ramsay reports.

`R`, `tm`, and `weightTfIdf`

To get `R` to generate tf—idf scores, one can do the following:

{% highlight r %} # Generate a Document Term Matrix dtm <- DocumentTermMatrix(corpus, control=list(weighting=weightTfIdf)) {% endhighlight %}

This was how I got my scores last time; and those scores didn’t match the one’s reported by Ramsay. So, one explanation is that I bollocks’d the data; the other is that there may be something going on under the hood.

To explore this latter possibility, I had a look at the `weightTfIdf` function.Doing so is easy; if you’re in an `R` console, with the `tm` package loaded, just type `weightTfIdf` and you’ll get all 26 lines of the code. It is worth noting that by default, the function normalizes its scores according to text. That is, rather than calculating the tf-idf score using a raw count of the number of times a particular term appears in a single document (in our case, the number of times a particular character uses a word), it divides the raw term frequency by the total number of words in that document (in our case, the total number of words said by a character). This normalizing process explains why the scores Ramsay reports for Louis (12) start at 5.9, while my own scores were always small numbers (0.006…; 0.003…, etc).

Short digression on misunderstanding the code

When I first dug into `weightTfIdf` code, I thought I discovered a difference in implementation:In order to work with the data, this function transposes the Document Term Matrix into a Term Document Matrix; this means that rather than the columns being terms and the rows texts, the rows are terms and the columns documents. I don't know *why* the function does this; but it does, and that explains why we're looking at row sums in what follows rather than column sums.

{% highlight r %} lnrs <- log2(nDocs(m)/rs) {% endhighlight %}

Here we're calculating part of the *tf-idf* score—the part that corresponds to Ramsay's log(N/df). Ignore the `log2` for a moment and look at that divisor, `rs`. That should represent the number of documents which contain a particular term. How is that calculated? Well, the code says:

{% highlight r %} rs <- row_sums(m > 0) {% endhighlight %}

That `row_sums` function comes the `slam` package; and when I first saw this, I thought I had my explanation. Ah ha! If the divisor in the logarithm is the sum of a row, then it is dividing the number of documents *not* by the number of documents which contain the specified term, but by the sum of the row—that is, by the total occurrences of that term! Here is our explanation!

"Well, but what is that comparison, `m > 0`, doing in the function call," you ask? Good question! Well, I assumed that it was just a way to pass non-zero values to the `row_sums` function, and went about trying to rewrite the function properly. This heady delirium lead to questions on StackOverflow and wasted time. Because, of course, I totally misunderstood how this part of the code works.

The trick is that a relational operator (like `>`) on a matrix, returns a matrix with boolean values, evaluating that expression for each item in the matrix. So, each cell in the term document matrix is now either `TRUE` or `FALSE` based on whether the value of that term was greater than 0. So it might look something like this:

```
             Docs
Terms         bernard.txt jinny.txt louis.txt neville.txt
  absorption         TRUE     FALSE     FALSE       FALSE
  abstract           TRUE     FALSE     FALSE       FALSE
  abstraction       FALSE     FALSE      TRUE       FALSE
  absurd             TRUE     FALSE     FALSE        TRUE
  absurdity          TRUE     FALSE     FALSE        TRUE
  absurdly          FALSE     FALSE     FALSE        TRUE
```

And, since TRUE is treated as numerically equivalent to 1, and FALSE to 0, if we were to sum these rows, we would get... exactly what we were looking for; i.e., the number of documents in which the term appears. So, that was wasted time.

Another thing to note here is that `weightTfIdf` uses `log2()`, the binary logarithmWhich returns the exponent to which you would raise 2 in order to get the specified term; i.e. log2(16)=4, b/c 2^4=16.. The logarithm here works to essentially scale the term's frequency based on how specific the term is to any particular document; terms which occur in all documents will have `N` nearly equal to `df`, or `N/df` nearly equal to 1. And `log(1) == log2(1) == 0`. That is, it will push the weight to (or towards) 0. Whereas a term which occurs in only one document will maximize `N/1`. A logarithmic function is used simply to dampen that effect, preventing scores linearly increasing in the case of terms highly specific to a single document.

Does it make a difference whether one uses `log2()` or the natural logarithm or the base 10 “common” logarithm I learned back in school? I don’t really know; I doubt it. The particular logarithmic function one chooses will change the *score*, but it wouldn’t change the relationship among the terms (which had the *highest* score).

Hand Simulations

After all that, I tried to tinker a bit to bring results into line. I stopped normalizing my data and tried different log functions in an attempt to match Ramsay’s scores. Well, okay, let’s take one last look.

Despairing, I returned to the clearest data Ramsay gives us—the list of terms and scores which appears on page 12 and picked a few, to see if I could manage this by hand.My high school computer science teacher (we switched from Pascal to C++ midway through my high school career) used to make us “hand simulate” algorithms; print out the source code, jot down the variable names, and debug by hand. Smart guy. (Though the phrase “hand simulation” may have been unfortunate.) (I don’t *really* mean by hand of course. Just… more slowly.) Here are the terms Ramsay lists for Louis, which have a score of 5.0021615: australian, beast, grained, though, wilt.

So, I returned to my data.

{% highlight r %} > rawdtm <- DocumentTermMatrix(characters) # A basic Document Term Matrix of raw frequency counts > terms <- c(‘australian’,‘beast’,‘grained’,‘thou’,‘wilt’) > inspect(rawdtm[,terms]) A document-term matrix (6 documents, 5 terms)

```
Non-/sparse entries: 6/24
Sparsity           : 80%
Maximal term length: 10
Weighting          : term frequency (tf)

             Terms
Docs          australian beast grained thou wilt
  bernard.txt          1     0       0    0    0
  jinny.txt            0     0       0    0    0
  louis.txt            6     6       6    6    6
  neville.txt          0     0       0    0    0
  rhoda.txt            0     0       0    0    0
  susan.txt            0     0       0    0    0
```

{% endhighlight %}

This is already puzzling. These five terms all had the same score. Yet, *australian* occurs six times in Louis, but across two texts; while the other terms occurs six time in Louis, and *only* in Louis.

So, back to the data. I open up the original gutenberg and search through it for australian. And, indeed, looks like 7 total occurrences; one of which is from Bernard:

> I thought how Louis would mount those steps in his neat suit with his cane in his hand and his angular, rather detached gait. With his Australian accent ("My father, a banker at Brisbane") he would come, I thought, with greater respect to these old ceremonies than I do, who have heard the same lullabies for a thousand years.

Let's check *grained.* I found six occurrences; all in Louis's speech and, as I was doing this by hand, I noticed that all those occurrences of *grained* were in the phrase *grained oak*:

- "the grained oak panel of Mr Wickham's door"
- "...Mr Wickham's grained oak door"
- "...I am also one who will force himself to desert these windy

and moonlit territories, these midnight wanderings, and confront grained oak doors.”

- “I, the companion of Plato, of Virgil, will knock at the grained oak door.”
- “I brought down my fist on my master’s grained oak door.”
- “…yet brought down my fist on the grained oak door”

Why does *grained* occur in Ramsay & Steger’s list, but not *oak*? Well, maybe *everyone* talks about oak trees, but only Louis talks about *grained oak.* Back to our data:

{% highlight r %} >inspect(rawdtm[,c(‘grained’,‘oak’)]) A document-term matrix (6 documents, 2 terms)

Non-/sparse entries: 3/9 Sparsity : 75% Maximal term length: 7 Weighting : term frequency (tf) Terms Docs grained oak bernard.txt 0 2 jinny.txt 0 0 louis.txt 6 10 neville.txt 0 0 rhoda.txt 0 0 susan.txt 0 0 {% endhighlight %}

Hmm… well, let’s do the math for these three terms (the first two of which got the same score in Ramsay’s analysis; the last of which didn’t appear at all). For these three terms, *australian*, *oak*, and *grained* for Louis, we have:

{% highlight r %} tf df N 1+tf*(log2(N/df)) 1+tf*(log10(N/df)) 1+tf*(log(N/df)) australian 6 2 6 10.50978 3.862728 7.59164 grained 6 1 6 16.50978 5.668909 11.75056 oak 10 2 6 16.84963 5.771213 11.98612 {% endhighlight %}

So oak should have the *highest* score of the three.

Something is clearly wrong.

I’ve returned to the raw counts, double checked them against the original file; and computed these things individually. I’ve tried to imagine all possible ways, but have been unable to produce any of the scores listed on pg 12.

I suspect that there may be a problem in the Ramsay and Steger’s data. There is significant overlap in Ramsay and Steger's results and my own. It is for Louis that our data is *most* consistent; we share 19 of the top 24 terms. It is least similar for Bernard (sharing only *4/24* terms) and Rhoda (11/24). Did the Ramsay & Steger's analysis, perhaps, discard the final chapter where only Bernard talks (perhaps deliberately)? That would obviously *greatly* change the Bernard data set; and, by removing so much text, it could easily affect the *df* term for other characters, accounting for the other discrepancies. (But it wouldn't explain *oak*.)

It may very well be *my* data that is the problem, but if so I've been unable to locate the error. I'm abandoning trying to reconcile these approaches to a method that *should*, in principle, be eminently reproducible. I will though summarize the top 24 terms with highest tf-idf scores, using the best data I've got, and not normalizing.

{% highlight r %} > louis[1:24] western oak beast grained thou wilt accent 23.264663 15.849625 15.509775 15.509775 15.509775 15.509775 14.000000 boasting nile average clerks stamps australian boys 12.679700 12.679700 10.339850 10.339850 10.000000 9.509775 8.000000 pitchers steel beaten boast bobbing custard eatingshop 7.924813 7.924813 7.754888 7.754888 7.754888 7.754888 7.754888 england eyres ham 7.754888 7.754888 7.754888 > bernard[1:24] curiosity hampton letter phrases byron elderly heaven married 25.84963 23.77444 23.26466 22.22858 22.18948 20.67970 20.67970 20.67970 observed dinner phrase willow fin simple describe self 20.67970 20.60451 20.00000 19.01955 18.09474 18.09474 17.43459 17.43459 stick sense story nature pictures thinking canopy enemy 17.43459 16.37895 16.00000 15.84963 15.84963 15.84963 15.50978 15.50978 > neville[1:24] story doomed immitigable papers byron catullus 12.000000 10.339850 10.339850 10.339850 7.924813 7.924813 cheep perfection camel detect don hosepipes 7.924813 7.924813 7.754888 7.754888 7.754888 7.754888 hubbub loads mallet marvel squirting waits 7.754888 7.754888 7.754888 7.754888 7.754888 7.754888 boys founder knives pocket scene shakespeare 7.000000 6.339850 6.339850 6.339850 6.339850 6.339850 > jinny[1:24] tunnel prepared billowing game native peers quicker 9.509775 7.924813 7.754888 7.754888 7.754888 7.754888 7.754888 melancholy bodies band cabinet coach crag dazzle 6.339850 5.264663 5.169925 5.169925 5.169925 5.169925 5.169925 deftly equipped eyebrows felled haymarket jump lockets 5.169925 5.169925 5.169925 5.169925 5.169925 5.169925 5.169925 matthews murmured prepare 5.169925 5.169925 5.169925 > rhoda[1:24] oblong dips tiger fuller swallow fallen steep suspended 18.094738 12.924813 11.094738 10.339850 9.509775 8.000000 7.924813 7.924813 cliffs minnows pond terror bunch foam party puddle 7.754888 7.754888 7.754888 7.000000 6.339850 6.339850 6.339850 6.339850 pools violets bow caverns chirp choke column columns 6.000000 6.000000 5.169925 5.169925 5.169925 5.169925 5.169925 5.169925 > susan[1:24] kitchen setter washing bury cart gate apron 15.849625 10.339850 10.339850 8.000000 7.924813 7.924813 7.754888 seasons squirrel windowpane beds butter clean wet 7.754888 7.754888 7.754888 6.339850 6.339850 6.339850 6.339850 blown winter baby bitten boil cabbages carbolic 6.000000 5.264663 5.169925 5.169925 5.169925 5.169925 5.169925 clara cradle eggs 5.169925 5.169925 5.169925 {% endhighlight %}