Introduction to Narrative

Christopher Lovell

July 6, 2015

Introduction

The *Narrative* package is designed to extend on the ubiquitous *tm* package, making time series analysis of corpora easier, as well additional functionality.

If you have not done any text mining in R before I recommend reading the tm introductory vignette first. The following document provides only a quick introduction to the *tm* package before moving on to demonstrate the additional features provided by *Narrative*.

tm

For this demo we'll be using a corpus of Reuters documents provided with the *tm* package. To load your own data, Narrative provides the `readSeparateText()` function for reading a directory full of .txt files. This is discussed in detail later.

```
library(tm)
```

```
## Loading required package: NLP
```

```
## Warning: package 'NLP' was built under R version 3.2.1
```

```
reut21578 <- system.file("texts", "crude", package = "tm")
reuters <- VCorpus(DirSource(reut21578),readerControl = list(reader = readReut21578XMLasPlain))
```

```
reuters
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 20
```

The corpus contains 20 documents. To look at a document in detail, use `as.character()`.

```
as.character(reuters[[1]])
```

```
## [1] "Diamond Shamrock Corp said that\neffective today it had cut its contract prices for crude oil by
```

The meta data contains the title, language, some high level topics, place of origin, and a time stamp; this will be important later when we begin to use the *Narrative* package.

```
meta(reuters[[1]])
```

```
##   author       : character(0)
##   datetimestamp: 1987-02-26 17:00:56
##   description  :
##   heading      : DIAMOND SHAMROCK (DIA) CUTS CRUDE PRICES
##   id           : 127
##   language     : en
##   origin       : Reuters-21578 XML
##   topics       : YES
##   lewissplit   : TRAIN
##   cgisplit     : TRAINING-SET
##   oldid        : 5670
##   topics_cat   : crude
##   places       : usa
##   people       : character(0)
##   orgs         : character(0)
##   exchanges    : character(0)
```

After reading in your data the next stage is to cleanse it and stem it. *tm* provides a numnber of functions for doing so.

```
reuters <- tm_map(reuters, content_transformer(tolower))
reuters <- tm_map(reuters, content_transformer(removeNumbers))
reuters <- tm_map(reuters, content_transformer(removePunctuation))
reuters <- tm_map(reuters, content_transformer(removeWords), stopwords("english"))
reuters <- tm_map(reuters, stemDocument)
reuters <- tm_map(reuters, content_transformer(stripWhitespace))
```

```
as.character(reuters[[1]])
```

```
## [1] "diamond shamrock corp said effect today cut contract price crude oil dlrs barrel reduct bring p
```

An incredibly useful representation of the corpus is as a *document term matrix*. This is a *td* matrix, where t is a vector of all terms used in the corpus, and d is a vector of all documents in the corpus. Each row then represents the frequency of a term in each document, and each row shows a documents constituent terms and their frequency.

```
dtm <- DocumentTermMatrix(reuters)
inspect(dtm[1:5,1:20])
```

```
## <<DocumentTermMatrix (documents: 5, terms: 20)>>
## Non-/sparse entries: 6/94
## Sparsity           : 94%
## Maximal term length: 10
## Weighting          : term frequency (tf)
##
##      Terms
## Docs  abdulaziz abil abl abroad accept accord across act activity add
##   127         0    0   0      0      0      0      0   0        0   0
##   144         0    2   0      0      0      0      0   0        0   0
##   191         0    0   0      0      0      0      0   0        0   0
##   194         0    0   0      0      0      0      0   0        0   0
##   211         0    0   0      0      0      0      0   0        0   0
```

```
##      Terms
## Docs  added address adher advantag advisers agenc agr agre agreement
##   127     0       0     0        0        0     0   0    0         0
##   144     1       4     0        1        0     0   1    0         1
##   191     0       0     0        0        0     0   0    0         0
##   194     0       0     0        0        0     0   0    0         0
##   211     0       0     0        0        0     0   0    0         0
##      Terms
## Docs  agricultur
##   127          0
##   144          0
##   191          0
##   194          0
##   211          0
```

Some useful operations can be applied to the term document matrix, such as finding the most frequent terms, or terms that are highly correlated.

```
findFreqTerms(dtm,lowfreq = 15)
```

```
##  [1] "barrel" "bpd"    "crude"  "dlrs"   "kuwait" "last"   "market"
##  [8] "mln"    "offici" "oil"    "one"    "opec"   "price"  "reuter"
## [15] "said"   "saudi"  "will"
```

```
findAssocs(dtm, "opec" ,corlimit = 0.85)
```

```
## $opec
##      emerg       meet    analyst production        oil
##       0.92       0.92       0.91       0.87       0.86
```

Narrative

Now that I have summarised the basic text mining functionality provided by *tm*, I'll move on to the additional functionality written in *Narrative*. I will use the `Narrative::function` syntax to distinguish between functions written in Narrative or other packages.

Word & Character Counts Word and character counts are often useful for normalising statistics generated from your corpus. We can generate vectors of each using the `characterCount` and `wordCount` functions. `characterCount` accepts the corpus as an argument, whereas `wordCount` uses a document term matrix.

```
wc <- Narrative::wordCount(dtm)
cc <- Narrative::characterCount(reuters)
```

Each function returns a numeric vector giving the word or character count for each document in our corpus.

Meta data These vectors can be added to our corpus metadata using the `addToMetaData` function.

```
reuters <- Narrative::addToMetaData(corpus = reuters, vector = wc, tag = "word_count")
reuters <- Narrative::addToMetaData(corpus = reuters, vector = cc, tag = "character_count")
meta(reuters[[1]])
```

```
##   author         : character(0)
##   datetimestamp  : 1987-02-26 17:00:56
##   description    :
##   heading        : DIAMOND SHAMROCK (DIA) CUTS CRUDE PRICES
##   id             : 127
##   language       : en
##   origin         : Reuters-21578 XML
##   topics         : YES
##   lewissplit     : TRAIN
##   cgisplit       : TRAINING-SET
##   oldid          : 5670
##   topics_cat     : crude
##   places         : usa
##   people         : character(0)
##   orgs           : character(0)
##   exchanges      : character(0)
##   word_count     : 57
##   character_count: 353
```

IO Narrative provides a couple of IO functions that make handling meta data associated with a corpus easier. To demonstrate, I'll first write our *reuters* corpus to disk using the `saveCorpus` function. This function takes a corpus, and writes it to the specified directory, along with an .Rdata file containing all associated metadata.

```
library(Narrative)
```

```
## Narrative v0.2
```

```
Narrative::saveCorpus(reuters,"C:\\Users\\324240\\Desktop\\test_files", save_metadata = T)
```

```
## metadata saved to " C:\Users\324240\Desktop\test_files " as " reuters_metadata.RDS "
```

We can then read this data back in to R, including the metadata, using the `readSeparateText` function.

```
reuters2 <- Narrative::readSeparateText("C:\\Users\\324240\\Desktop\\test_files",load_metadata = T,meta
```

Term Frequency Analysis A document term matrix provides a ready made data structure for searching. In our example corpus, we can find the number of occurences of the term "oil" by subsetting the DTM on columns.

```
inspect(dtm[,c("oil")])
```

```
## <<DocumentTermMatrix (documents: 20, terms: 1)>>
## Non-/sparse entries: 20/0
## Sparsity           : 0%
## Maximal term length: 3
## Weighting          : term frequency (tf)
##
##      Terms
## Docs  oil
```

```
##   127   5
##   144  12
##   191   2
##   194   1
##   211   1
##   236   7
##   237   3
##   242   3
##   246   5
##   248   9
##   273   5
##   349   4
##   352   5
##   353   4
##   368   3
##   489   4
##   502   5
##   543   3
##   704   3
##   708   1
```

This can be used to create a time series, showing how the usage of this term has changed over time. To do this, we convert the subsetted DTM in to a numeric vector.

```
search.result <- as.matrix(dtm[,c("oil")])
```

We then use this to generate an `xts` object, which is an R time series data type. To do this, we pass the search result as well as the date-time data from the corpus meta data to the `xtsGenerate` function.

```
xts.search <- Narrative::xtsGenerate(time = do.call(c,meta(reuters,"datetimestamp")),
                                     value = search.result)
xts.search
```

```
##                     [,1]
## 1987-02-26 17:00:56    5
## 1987-02-26 17:34:11   12
## 1987-02-26 18:18:00    2
## 1987-02-26 18:21:01    1
## 1987-02-26 19:00:57    1
## 1987-03-01 03:25:46    7
## 1987-03-01 03:39:14    3
## 1987-03-01 05:27:27    3
## 1987-03-01 08:22:30    5
## 1987-03-01 18:31:44    9
## 1987-03-02 01:05:49    5
## 1987-03-02 07:39:23    4
## 1987-03-02 07:43:22    5
## 1987-03-02 07:43:41    4
## 1987-03-02 08:25:42    3
## 1987-03-02 11:20:05    4
## 1987-03-02 11:28:26    5
## 1987-03-02 12:13:46    3
## 1987-03-02 14:38:34    3
## 1987-03-02 14:49:06    1
```

The xts data type allows us then to aggregate our data by time bin. Below, we aggregate by day to return the total number of occurences across all documents on this day.

```
xts.search.aggregate <- Narrative::xtsAggregate(xts.search,
                                                time_aggregate = "daily",
                                                normalisation = F)
xts.search.aggregate
```

```
##                     [,1]
## 1987-02-26 19:00:57   21
## 1987-03-01 18:31:44   27
## 1987-03-02 14:49:06   37
```

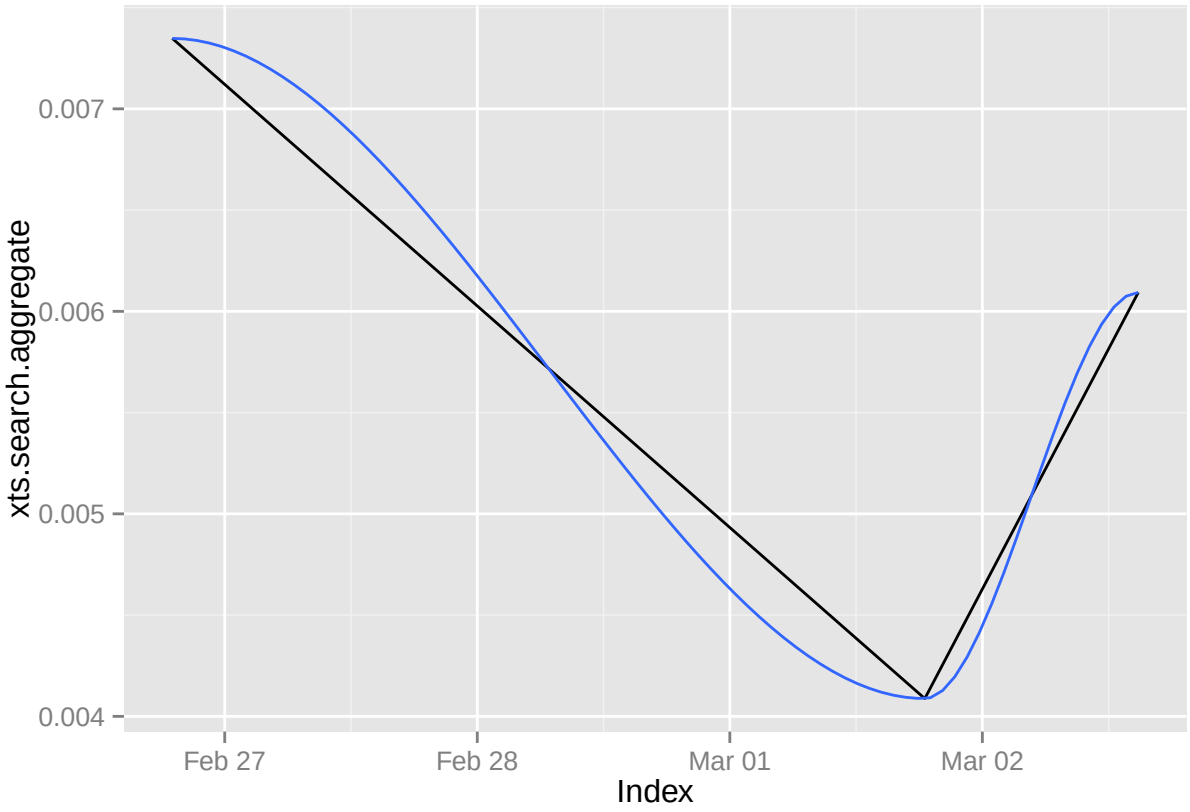
There are a number of different normalisation options when aggregating your time series data. If you wish to normalise by the number of documents in a given window, set the normalisation flag to true.

```
xts.search.aggregate <- Narrative::xtsAggregate(xts.search,
                                                time_aggregate = "daily",
                                                normalisation = T)
xts.search.aggregate
```

```
##                     [,1]
## 1987-02-26 19:00:57  4.2
## 1987-03-01 18:31:44  5.4
## 1987-03-02 14:49:06  3.7
```

Alternatively, you can normalise by any other dimension associated with your data. Just pass a numeric vector of normalisation values to the `normalisation` argument. Below, we use the character count calculated previously to normalise by size of document, then plot this along with a Loess smoother.

```
xts.search.aggregate <- Narrative::xtsAggregate(xts.search,
                                                time_aggregate = "daily",
                                                normalisation = do.call(c,meta(reuters,"character_count

p <- zoo::autoplot.zoo(xts.search.aggregate) + ggplot2::stat_smooth()
p
```

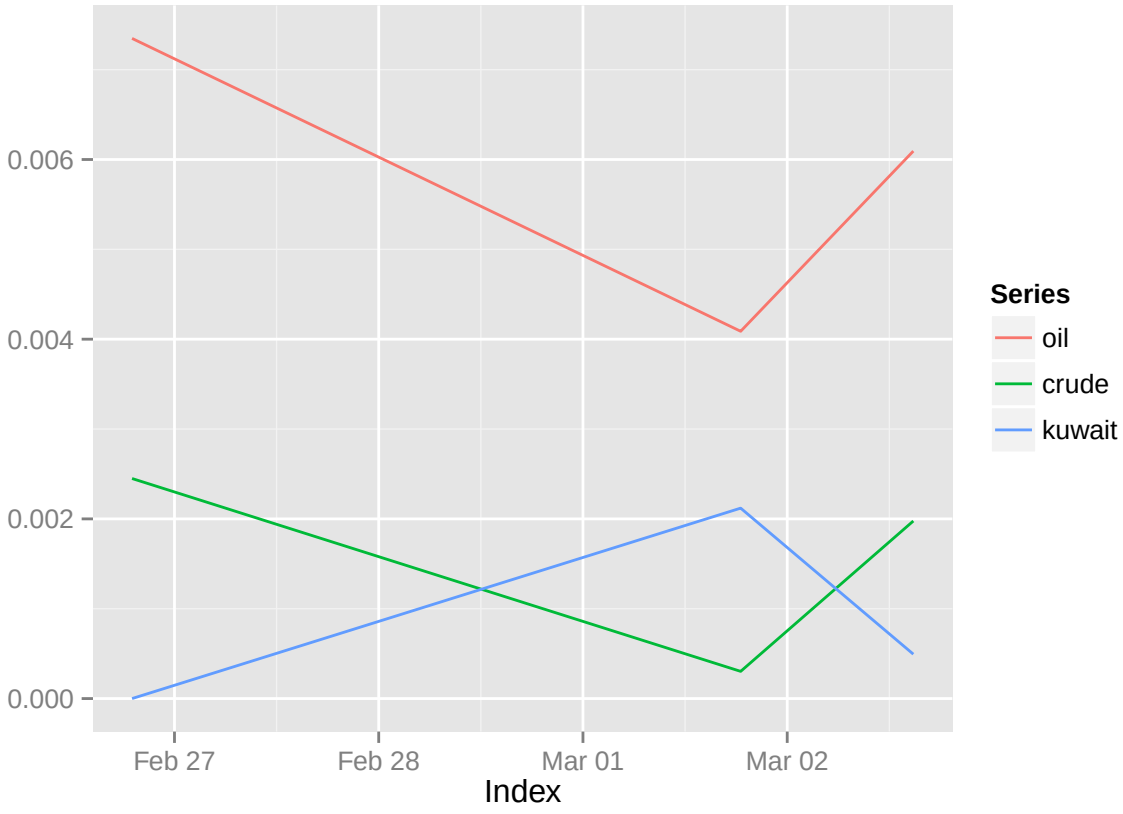


You can search for and compare multiple terms at the same time.

```
terms <- c("oil","crude","kuwait")
search.result <- as.matrix(dtm[,terms])

xts.search <- Narrative::xtsGenerate(time = do.call(c,meta(reuters,"datetimestamp")),
                                     value = search.result)

xts.search.aggregate <- Narrative::xtsAggregate(xts.search,
                                                time_aggregate = "daily",
                                                normalisation = do.call(c,meta(reuters,"character_count
```

```
## Warning in if (normalisation == F) {: the condition has length > 1 and only
## the first element will be used

## Warning in if (normalisation == T) {: the condition has length > 1 and only
## the first element will be used
```

```
p <- zoo::autoplot.zoo(xts.search.aggregate, facets = NULL)
p
```


0.006
0.004
0.002
0.000
Feb 27
Feb 28
Mar 01
Mar 02
Index
Series
oil
crude
kuwait