Getting Started with quanteda

- Introduction
- quanteda Features
  - Corpus management tools
  - Natural-Language Processing tools
  - Document-Feature Matrix analysis tools
  - Additional and planned features
  - Working with other text analysis packages
- How to Install
- Creating and Working with a Corpus
  - Currently available corpus sources
    - Example: building a corpus from a character vector
    - Example: loading in files using `textfile()`
    - Creating a corpus from a Twitter search
  - How a quanteda corpus works
    - Corpus principles
  - Tools for handling corpus objects
    - Adding two corpus objects together
    - subsetting corpus objects
  - Exploring corpus texts
- Extracting Features from a Corpus
  - Tokenizing texts
  - Constructing a document-frequency matrix
    - Viewing the document-frequency matrix
    - Grouping documents by document variable
    - Grouping words by dictionary or equivalence class
- Further Examples
  - Similarities between texts
  - Scaling document positions
  - Topic models

Introduction

An R package for managing and analyzing text, by Ken Benoit and Paul Nulty.[1]

quanteda makes it easy to manage texts in the form of a corpus, defined as a collection of texts that includes document-level variables specific to each text, as well as meta-data for documents and for the collection as a whole. **quanteda** includes tools to make it easy and fast to manuipulate the texts in a corpus, by performing the most common natural language processing tasks simply and quickly, such as tokenizing, stemming, or forming ngrams. **quanteda**'s functions for tokenizing texts and forming multiple tokenized documents into a *document-feature matrix* are both extremely fast and extremely simple to use. **quanteda** can segment texts easily by words, paragraphs, sentences, or even user-supplied

delimiters and tags.

Built on the text processing functions in the **stringi** package, which is in turn built on C++ implementation of the ICU libraries for Unicode text handling, **quanteda** pays special attention to fast and correct implementation of Unicode and the handling of text in any character set, following conversion internally to UTF-8.

quanteda is built for efficiency and speed, through its design around three infrastructures: the **string** package for text processing, the **data.table** package for indexing large documents efficiently, and the **Matrix** package for sparse matrix objects. If you can fit it into memory, **quanteda** will handle it quickly. (And eventually, we will make it possible to process objects even larger than available memory.)

quanteda is principally designed to allow users a fast and convenient method to go from a corpus of texts to a selected matrix of documents by features, after defining what the documents and features. The package makes it easy to redefine documents, for instance by splitting them into sentences or paragraphs, or by tags, as well as to group them into larger documents by document variables, or to subset them based on logical conditions or combinations of document variables. The package also implements common NLP feature selection functions, such as removing stopwords and stemming in numerous languages, selecting words found in dictionaries, treating words as equivalent based on a user-defined “thesaurus”, and trimming and weighting features based on document frequency, feature frequency, and related measures such as *tf-idf*.

quanteda Features

Corpus management tools

The tools for getting texts into a corpus object include:

- loading texts from directories of individual files
- loading texts ``manually” by inserting them into a corpus using helper functions
- managing text encodings and conversions from source files into corpus texts
- attaching variables to each text that can be used for grouping, reorganizing a corpus, or simply recording additional information to supplement quantitative analyses with non-textual data
- recording meta-data about the sources and creation details for the corpus.

The tools for working with a corpus include:

- summarizing the corpus in terms of its language units
- reshaping the corpus into smaller units or more aggregated units
- adding to or extracting subsets of a corpus
- resampling texts of the corpus, for example for use in non-parametric bootstrapping of the texts
- Easy extraction and saving, as a new data frame or corpus, key words in context (KWIC)

Natural-Language Processing tools

For extracting features from a corpus, `quanteda` provides the following tools:

- extraction of word types
- extraction of word n-grams
- extraction of dictionary entries from user-defined dictionaries
- feature selection through
    - stemming
    - random selection
    - document frequency
    - word frequency
- and a variety of options for cleaning word types, such as capitalization and rules for handling punctuation.

Document-Feature Matrix analysis tools

For analyzing the resulting *document-feature* matrix created when features are abstracted from a corpus, `quanteda` provides:

- scaling methods, such as correspondence analysis, Wordfish, and Wordscores
- topic models, such as LDA
- classifiers, such as Naive Bayes or k-nearest neighbour
- sentiment analysis, using dictionaries

Additional and planned features

Additional features of quanteda include:

- the ability to explore texts using *key-words-in-context*;
- fast computation of a variety of readability indexes;
- fast computation of a variety of lexical diversity measures;
- quick computation of word or document association measures, for clustering or to compute similarity scores for other purposes; and
- a comprehensive suite of descriptive statistics on text such as the number of sentences, words, characters, or syllables per document.

Planned features coming soon to **quanteda** are:

- bootstrapping methods for texts that makes it easy to resample texts from pre-defined units, to

facilitate computation of confidence intervals on textual statistics using techniques of non-parametric bootstrapping, but applied to the original texts as data.

- expansion of the document-feature matrix structure through a standard interface called `textmodel()`. (As of version 0.8.0, textmodel works in a basic fashion only for the "Wordscores" and "wordfish" scaling models.)

Working with other text analysis packages

`quanteda` is hardly unique in providing facilities for working with text – the excellent *tm* package already provides many of the features we have described. `quanteda` is designed to complement those packages, as well to simplify the implementation of the text-to-analysis workflow. `quanteda` corpus structures are simpler objects than in *tm*s, as are the document-feature matrix objects from `quanteda`, compared to the sparse matrix implementation found in *tm*. However, there is no need to choose only one package, since we provide translator functions from one matrix or corpus object to the other in `quanteda`.

Once constructed, a **quanteda** "dfm"**"** can be easily passed to other text-analysis packages for additional analysis of topic models or scaling, such as:

- topic models (including converters for direct use with the **topicmodels**, **LDA**, and **stm** packages)
- document scaling (using **quanteda**'s own functions for the "wordfish" and "Wordscores" models, direct use with the **ca** package for correspondence analysis, or scaling with the **austin** package)
- document classification methods, using (for example) Naive Bayes, k-nearest neighbour, or Support Vector Machines
- more sophisticated machine learning through a variety of other packages that take matrix or matrix-like inputs.
- graphical analysis, including word clouds and strip plots for selected themes or words.

How to Install

As of version 0.8.0, the GitHub master repository will always contain the development version of quanteda, while the CRAN version will contain the latest "stable" version. You therefore have two options for installing the package:

1. From CRAN, using your R package installer, or simply

```
install.packages("quanteda")
```

2. (For the development version) From GitHub, using

```
devtools::install_github("kbenoit/quanteda")
```

Because this compiles some C++ source code, you will need a compiler installed. If you are using a Windows platform, this means you will need also to install the Rtools software available from CRAN. If you are using OS X, you will probably need to install XCode, available for free from the App Store.

3. (Optional) You can install some additional corpus data from **quantedaData** using

```
## devtools required to install quanteda from Github
devtools::install_github("kbenoit/quantedaData")
```

Creating and Working with a Corpus

```
require(quanteda)
```

Currently available corpus sources

quanteda has a simple and powerful tool for loading texts: `textfile()`. This function takes a file or fileset from disk or a URL, and loads it as a special class of pre-corpus object, known as a `corpusSource` object, from which a corpus can be constructed using a second command, `corpus()`.

`textfile()` works on:

- text (`.txt`) files;
- comma-separated-value (`.csv`) files;
- XML formatted data;
- data from the Facebook API, in JSON format;
- data from the Twitter API, in JSON format; and
- generic JSON data.

The corpus constructor command `corpus()` works directly on:

- a vector of character objects, for instance that you have already loaded into the workspace using other tools;
- a `corpusSource` object created using `textfile()`; and
- a `VCorpus` corpus object from the **tm** package.

Example: building a corpus from a character vector

The simplest case is to create a corpus from a vector of texts already in memory in R. This gives the

advanced R user complete flexbility with his or her choice of text inputs, as there are almost endless ways to get a vector of texts into R.

If we already have the texts in this form, we can call the corpus constructor function directly. We can demonstrate this on the built-in character vector of 57 US president inaugural speeches called `inaugTexts`.

```
str(inaugTexts)  # this gives us some information about the object
#>  Named chr [1:57] "Fellow-Citizens of the Senate and of the House of Representatives:\n\nAmong the
#>  - attr(*, "names")= chr [1:57] "1789-Washington" "1793-Washington" "1797-Adams" "1801-Jefferson"
myCorpus <- corpus(inaugTexts)  # build the corpus
summary(myCorpus, n = 5)
#> Corpus consisting of 57 documents, showing 5 documents.
#>
#>             Text Types Tokens Sentences
#>  1789-Washington   626   1540        24
#>  1793-Washington    96    147         4
#>       1797-Adams   826   2584        37
#>   1801-Jefferson   716   1935        42
#>   1805-Jefferson   804   2381        45
#>
#> Source:  /private/var/folders/3_/7s7qq3wx08b8htzt5l9sdm6m0000gr/T/RtmphORmdD/Rbuild70655f5fec07/qu
#> Created: Sun Feb 21 16:43:49 2016
#> Notes:
```

If we wanted, we could add some document-level variables – what quanteda calls `docvars` – to this corpus.

We can do this using the R's `substring()` function to extract characters from a name – in this case, the name of the character vector `inaugTexts`. This works using our fixed starting and ending positions with `substring()` because these names are a very regular format of `YYYY-PresidentName`.

```
docvars(myCorpus, "President") <- substring(names(inaugTexts), 6)
docvars(myCorpus, "Year") <- as.integer(substring(names(inaugTexts), 1, 4))
summary(myCorpus, n=5)
#> Corpus consisting of 57 documents, showing 5 documents.
#>
#>             Text Types Tokens Sentences  President Year
#>  1789-Washington   626   1540        24 Washington 1789
#>  1793-Washington    96    147         4 Washington 1793
#>       1797-Adams   826   2584        37      Adams 1797
#>   1801-Jefferson   716   1935        42  Jefferson 1801
#>   1805-Jefferson   804   2381        45  Jefferson 1805
#>
#> Source:  /private/var/folders/3_/7s7qq3wx08b8htzt5l9sdm6m0000gr/T/RtmphORmdD/Rbuild70655f5fec07/qu
#> Created: Sun Feb 21 16:43:49 2016
#> Notes:
```

If we wanted to tag each document with additional meta-data not considered a document variable of

interest for analysis, but rather something that we need to know as an attribute of the document, we could also add those to our corpus.

```
metadoc(myCorpus, "language") <- "english"
metadoc(myCorpus, "docsource")  <- paste("inaugTexts", 1:ndoc(myCorpus), sep = "_")
summary(myCorpus, n = 5, showmeta = TRUE)
#> Corpus consisting of 57 documents, showing 5 documents.
#>
#>             Text Types Tokens Sentences  President Year _language
#>  1789-Washington   626   1540        24 Washington 1789   english
#>  1793-Washington    96    147         4 Washington 1793   english
#>       1797-Adams   826   2584        37      Adams 1797   english
#>   1801-Jefferson   716   1935        42  Jefferson 1801   english
#>   1805-Jefferson   804   2381        45  Jefferson 1805   english
#>    _docsource
#>  inaugTexts_1
#>  inaugTexts_2
#>  inaugTexts_3
#>  inaugTexts_4
#>  inaugTexts_5
#>
#> Source:  /private/var/folders/3_/7s7qq3wx08b8htzt5l9sdm6m0000gr/T/RtmphORmdD/Rbuild70655f5fec07/qu
#> Created: Sun Feb 21 16:43:49 2016
#> Notes:
```

The last command, `metadoc`, allows you to define your own document meta-data fields. Note that in assiging just the single value of `"english"`, R has recycled the value until it matches the number of documents in the corpus. In creating a simple tag for our custom metadoc field `docsource`, we used the quanteda function `ndoc()` to retrieve the number of documents in our corpus. This function is deliberately designed to work in a way similar to functions you may already use in R, such as `nrow()` and `ncol()`.

Example: loading in files using `textfile()`

```
# Twitter json
mytf1 <- textfile("~/Dropbox/QUANTESS/social media/zombies/tweets.json")
myCorpusTwitter <- corpus(mytf1)
summary(myCorpusTwitter, 5)
# generic json - needs a textField specifier
mytf2 <- textfile("~/Dropbox/QUANTESS/Manuscripts/collocations/Corpora/sotu/sotu.json",
                  textField = "text")
summary(corpus(mytf2), 5)
# text file
mytf3 <- textfile("~/Dropbox/QUANTESS/corpora/project_gutenberg/pg2701.txt", cache = FALSE)
summary(corpus(mytf3), 5)
# multiple text files
mytf4 <- textfile("~/Dropbox/QUANTESS/corpora/inaugural/*.txt", cache = FALSE)
summary(corpus(mytf4), 5)
# multiple text files with docvars from filenames
mytf5 <- textfile("~/Dropbox/QUANTESS/corpora/inaugural/*.txt",
                  docvarsfrom="filenames", sep="-", docvarnames=c("Year", "President"))
```

```
summary(corpus(mytf5), 5)
# XML data
mytf6 <- textfile("~/Dropbox/QUANTESS/quanteda_working_files/xmlData/plant_catalog.xml",
                  textField = "COMMON")
summary(corpus(mytf6), 5)
# csv file
write.csv(data.frame(inaugSpeech = texts(inaugCorpus), docvars(inaugCorpus)),
          file = "/tmp/inaugTexts.csv", row.names = FALSE)
mytf7 <- textfile("/tmp/inaugTexts.csv", textField = "inaugSpeech")
summary(corpus(mytf7), 5)
```

Creating a corpus from a Twitter search

`quanteda` provides an interface to retrieve and store data from a twitter search as a corpus object. The REST API query uses the twitteR package, and an API authorization from twitter is required. The process of obtaining this authorization is described in detail here: https://openhatch.org/wiki/Community_Data_Science_Workshops/Twitter_authentication_setup, correct as of October 2014. The twitter API is a commercial service, and rate limits and the data returned are determined by twitter.

Four keys are required, to be passed to `quanteda`'s `getTweets` source function, in addition to the search query term and the number of results required. The maximum number of results that can be obtained is not exactly identified in the API documentation, but experimentation indicates an upper bound of around 1500 results from a single query, with a frequency limit of one query per minute.

The code below performs authentication and runs a search for the string 'quantitative'. Many other functions for working with the API are available from the twitteR package. An R interface to the streaming API is also available link.

```
# These keys are examples and may not work! Get your own key at dev.twitter.com
consumer_key="vRLy03ef6OFAZB7oCL4jA"
consumer_secret="wWF35Lr1raBrPerVHSDyRftv8qB1H7ltV0T3Srb3s"
access_token="1577780816-wVbOZEED8KZs70PwJ2q5ld2w9CcvcZ2kC6gPnAo"
token_secret="IeC6iYlgUK9csWiP524Jb4UNM8RtQmHyetLi9NZrkJA"


tw <- getTweets('quantitative', numResults=20, consumer_key, consumer_secret, access_token, token_sec
```

The return value from the above query is a source object which can be passed to quanteda's corpus constructor, and the document variables are set to correspond with tweet metadata returned by the API.

```
twCorpus <- corpus(tw)
names(docvars(twCorpus))
```

How a quanteda corpus works

Corpus principles

A corpus is designed to be a “library” of original documents that have been converted to plain, UTF-8 encoded text, and stored along with meta-data at the corpus level and at the document-level. We have a special name for document-level meta-data: *docvars*. These are variables or features that describe attributes of each document.

A corpus is designed to be a more or less static container of texts with respect to processing and analysis. This means that the texts in corpus are not designed to be changed internally through (for example) cleaning or pre-processing steps, such as stemming or removing punctuation. Rather, texts can be extracted from the corpus as part of processing, and assigned to new objects, but the idea is that the corpus will remain as an original reference copy so that other analyses – for instance those in which stems and punctuation were required, such as analyzing a reading ease index – can be performed on the same corpus.

To extract texts from a a corpus, we use an extractor, called `texts()`.

```
texts(inaugCorpus)[2]
#>
#> "Fellow citizens, I am again called upon by the voice of my country to execute the functions of it
```

To summarize the texts from a corpus, we can call a `summary()` method defined for a corpus.

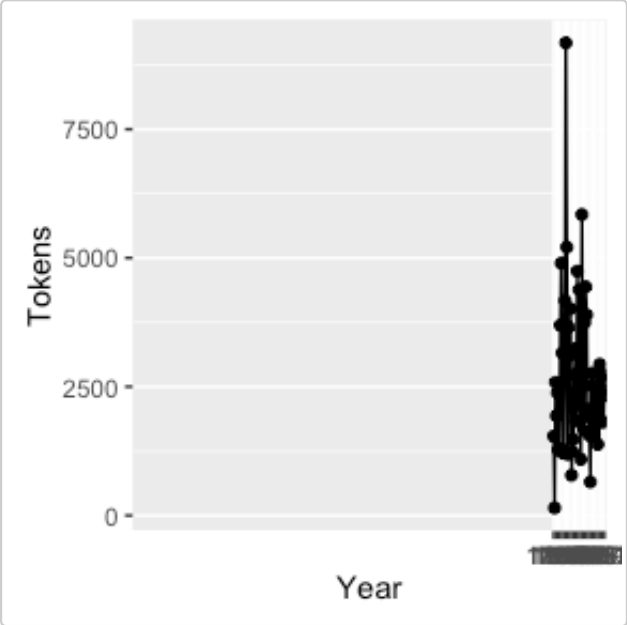
```
summary(ie2010Corpus)
#> Corpus consisting of 14 documents.
#>
#>                                 Text Types Tokens Sentences year debate
#>        2010_BUDGET_01_Brian_Lenihan_FF  1949   8733       404 2010 BUDGET
#>       2010_BUDGET_02_Richard_Bruton_FG  1042   4478       217 2010 BUDGET
#>         2010_BUDGET_03_Joan_Burton_LAB  1621   6429       309 2010 BUDGET
#>        2010_BUDGET_04_Arthur_Morgan_SF  1589   7185       345 2010 BUDGET
#>          2010_BUDGET_05_Brian_Cowen_FF  1618   6697       252 2010 BUDGET
#>           2010_BUDGET_06_Enda_Kenny_FG  1151   4254       155 2010 BUDGET
#>      2010_BUDGET_07_Kieran_ODonnell_FG   681   2309       133 2010 BUDGET
#>       2010_BUDGET_08_Eamon_Gilmore_LAB  1183   4217       202 2010 BUDGET
#>     2010_BUDGET_09_Michael_Higgins_LAB   490   1288        44 2010 BUDGET
#>        2010_BUDGET_10_Ruairi_Quinn_LAB   442   1290        60 2010 BUDGET
#>      2010_BUDGET_11_John_Gormley_Green   404   1036        50 2010 BUDGET
#>        2010_BUDGET_12_Eamon_Ryan_Green   512   1651        90 2010 BUDGET
#>      2010_BUDGET_13_Ciaran_Cuffe_Green   444   1248        45 2010 BUDGET
#>  2010_BUDGET_14_Caoimhghin_OCaolain_SF  1188   4094       177 2010 BUDGET
#>  number      foren     name party
#>      01      Brian  Lenihan    FF
#>      02    Richard   Bruton    FG
#>      03       Joan   Burton   LAB
#>      04     Arthur   Morgan    SF
#>      05      Brian    Cowen    FF
#>      06       Enda    Kenny    FG
#>      07     Kieran ODonnell    FG
```

```
#>      08      Eamon  Gilmore   LAB
#>      09    Michael  Higgins   LAB
#>      10     Ruairi    Quinn   LAB
#>      11       John  Gormley Green
#>      12      Eamon     Ryan Green
#>      13     Ciaran    Cuffe Green
#>      14 Caoimhghin OCaolain    SF
#>
#> Source:  /home/paul/Dropbox/code/quantedaData/* on x86_64 by paul
#> Created: Tue Sep 16 15:58:21 2014
#> Notes:
```

We can save the output from the summary command as a data frame, and plot some basic descriptive statistics with this information:

```
tokenInfo <- summary(inaugCorpus)
#> Corpus consisting of 57 documents.
#>
#>             Text Types Tokens Sentences Year  President       FirstName
#>  1789-Washington   626   1540        24 1789 Washington          George
#>  1793-Washington    96    147         4 1793 Washington          George
#>       1797-Adams   826   2584        37 1797      Adams            John
#>   1801-Jefferson   716   1935        42 1801  Jefferson          Thomas
#>   1805-Jefferson   804   2381        45 1805  Jefferson          Thomas
#>     1809-Madison   536   1267        21 1809    Madison           James
#>     1813-Madison   542   1304        33 1813    Madison           James
#>      1817-Monroe  1040   3696       121 1817     Monroe           James
#>      1821-Monroe  1262   4898       131 1821     Monroe           James
#>       1825-Adams  1004   3154        74 1825      Adams     John Quincy
#>     1829-Jackson   517   1210        25 1829    Jackson          Andrew
#>     1833-Jackson   499   1271        30 1833    Jackson          Andrew
#>    1837-VanBuren  1315   4175        95 1837  Van Buren          Martin
#>    1841-Harrison  1893   9178       210 1841   Harrison   William Henry
#>        1845-Polk  1330   5211       153 1845       Polk      James Knox
#>      1849-Taylor   497   1185        22 1849     Taylor         Zachary
#>      1853-Pierce  1166   3657       104 1853     Pierce        Franklin
#>    1857-Buchanan   945   3106        89 1857   Buchanan           James
#>     1861-Lincoln  1075   4016       138 1861    Lincoln         Abraham
#>     1865-Lincoln   362    780        27 1865    Lincoln         Abraham
#>       1869-Grant   486   1243        41 1869      Grant      Ulysses S.
#>       1873-Grant   552   1479        44 1873      Grant      Ulysses S.
#>       1877-Hayes   829   2730        59 1877      Hayes   Rutherford B.
#>    1881-Garfield  1018   3240       112 1881   Garfield        James A.
#>   1885-Cleveland   674   1828        44 1885  Cleveland          Grover
#>    1889-Harrison  1355   4744       157 1889   Harrison        Benjamin
#>   1893-Cleveland   823   2135        58 1893  Cleveland          Grover
#>    1897-McKinley  1236   4383       130 1897   McKinley         William
#>    1901-McKinley   857   2449       100 1901   McKinley         William
#>   1905-Roosevelt   404   1089        33 1905  Roosevelt        Theodore
#>        1909-Taft  1436   5844       159 1909       Taft  William Howard
#>      1913-Wilson   661   1896        68 1913     Wilson         Woodrow
#>      1917-Wilson   549   1656        60 1917     Wilson         Woodrow
#>     1921-Harding  1172   3743       149 1921    Harding       Warren G.
#>    1925-Coolidge  1221   4442       197 1925   Coolidge          Calvin
#>      1929-Hoover  1086   3895       171 1929     Hoover         Herbert
```

```
#>   1933-Roosevelt   744   2064        85 1933  Roosevelt     Franklin D.
#>   1937-Roosevelt   729   2027        96 1937  Roosevelt     Franklin D.
#>   1941-Roosevelt   527   1552        68 1941  Roosevelt     Franklin D.
#>   1945-Roosevelt   276    651        26 1945  Roosevelt     Franklin D.
#>      1949-Truman   781   2531       116 1949     Truman        Harry S.
#>  1953-Eisenhower   903   2765       123 1953 Eisenhower       Dwight D.
#>  1957-Eisenhower   621   1933        93 1957 Eisenhower       Dwight D.
#>     1961-Kennedy   566   1568        52 1961    Kennedy         John F.
#>     1965-Johnson   569   1725        98 1965    Johnson   Lyndon Baines
#>       1969-Nixon   743   2437       106 1969      Nixon Richard Milhous
#>       1973-Nixon   545   2018        69 1973      Nixon Richard Milhous
#>      1977-Carter   528   1380        53 1977     Carter           Jimmy
#>      1981-Reagan   904   2798       128 1981     Reagan          Ronald
#>      1985-Reagan   925   2935       125 1985     Reagan          Ronald
#>        1989-Bush   795   2683       143 1989       Bush          George
#>     1993-Clinton   644   1837        81 1993    Clinton            Bill
#>     1997-Clinton   773   2451       112 1997    Clinton            Bill
#>        2001-Bush   622   1810        97 2001       Bush       George W.
#>        2005-Bush   772   2325       101 2005       Bush       George W.
#>       2009-Obama   939   2729       112 2009      Obama          Barack
#>       2013-Obama   814   2335        90 2013      Obama          Barack
#>
#> Source:  /home/paul/Dropbox/code/quanteda/* on x86_64 by paul
#> Created: Fri Sep 12 12:41:17 2014
#> Notes:
if (require(ggplot2))
    ggplot(data=tokenInfo, aes(x=Year, y=Tokens, group=1)) + geom_line() + geom_point() +
        scale_x_discrete(labels=c(seq(1789,2012,12)), breaks=seq(1789,2012,12) )
#> Loading required package: ggplot2
```




```
# Longest inaugural address: William Henry Harrison
tokenInfo[which.max(tokenInfo$Tokens),]
#>                        Text Types Tokens Sentences Year President
#> 1841-Harrison 1841-Harrison  1893   9178       210 1841  Harrison
#>                   FirstName
```

```
#> 1841-Harrison William Henry
```

Tools for handling corpus objects

Adding two corpus objects together

The `+` operator provides a simple method for concatenating two corpus objects. If they contain different sets of document-level variables, these will be stitched together in a fashion that guarantees that no information is lost. Corpus-level medata data is also concatenated.

```
library(quanteda)
mycorpus1 <- corpus(inaugTexts[1:5], note = "First five inaug speeches.")
mycorpus2 <- corpus(inaugTexts[53:57], note = "Last five inaug speeches.")
mycorpus3 <- mycorpus1 + mycorpus2
summary(mycorpus3)
#> Corpus consisting of 10 documents.
#>
#>             Text Types Tokens Sentences
#>  1789-Washington   626   1540        24
#>  1793-Washington    96    147         4
#>       1797-Adams   826   2584        37
#>   1801-Jefferson   716   1935        42
#>   1805-Jefferson   804   2381        45
#>     1997-Clinton   773   2451       112
#>        2001-Bush   622   1810        97
#>        2005-Bush   772   2325       101
#>       2009-Obama   939   2729       112
#>       2013-Obama   814   2335        90
#>
#> Source:  Combination of corpuses mycorpus1 and mycorpus2
#> Created: Sun Feb 21 16:43:50 2016
#> Notes:   First five inaug speeches. Last five inaug speeches.
```

subsetting corpus objects

There is a method of the `subset()` function defined for corpus objects, where a new corpus can be extracted based on logical conditions applied to docvars:

```
summary(subset(inaugCorpus, Year > 1990))
#> Corpus consisting of 6 documents.
#>
#>          Text Types Tokens Sentences Year President FirstName
#>  1993-Clinton   644   1837        81 1993   Clinton      Bill
#>  1997-Clinton   773   2451       112 1997   Clinton      Bill
#>     2001-Bush   622   1810        97 2001      Bush George W.
#>     2005-Bush   772   2325       101 2005      Bush George W.
#>    2009-Obama   939   2729       112 2009     Obama    Barack
#>    2013-Obama   814   2335        90 2013     Obama    Barack
#>
```

```
#> Source:  /home/paul/Dropbox/code/quanteda/* on x86_64 by paul
#> Created: Fri Sep 12 12:41:17 2014
#> Notes:
summary(subset(inaugCorpus, President == "Adams"))
#> Corpus consisting of 2 documents.
#>
#>        Text Types Tokens Sentences Year President   FirstName
#>  1797-Adams   826   2584        37 1797     Adams        John
#>  1825-Adams  1004   3154        74 1825     Adams John Quincy
#>
#> Source:  /home/paul/Dropbox/code/quanteda/* on x86_64 by paul
#> Created: Fri Sep 12 12:41:17 2014
#> Notes:
```

Exploring corpus texts

The `kwic` function (KeyWord In Context) performs a search for a word and allows us to view the contexts in which it occurs:

```
options(width = 200)
kwic(inaugCorpus, "terror")
#>                                                contextPre keyword                       contex
#>    [1797-Adams, 1327]              fraud or violence, by [  terror ] , intrigue, or venality
#> [1933-Roosevelt, 112] nameless, unreasoning, unjustified [  terror ] which paralyzes needed effor
#> [1941-Roosevelt, 289]      seemed frozen by a fatalistic [  terror ] , we proved that this
#>   [1961-Kennedy, 868]    alter that uncertain balance of [  terror ] that stays the hand of
#>    [1981-Reagan, 821]     freeing all Americans from the [  terror ] of runaway living costs.
#>  [1997-Clinton, 1055]        They fuel the fanaticism of [  terror ] . And they torment the
#>  [1997-Clinton, 1655]  maintain a strong defense against [  terror ] and destruction. Our childre
#>    [2009-Obama, 1646]     advance their aims by inducing [  terror ] and slaughtering innocents, w
kwic(inaugCorpus, "terror", valuetype = "regex")
#>                                                contextPre   keyword                        conte
#>    [1797-Adams, 1327]              fraud or violence, by [  terror   ] , intrigue, or venality
#> [1933-Roosevelt, 112] nameless, unreasoning, unjustified [  terror   ] which paralyzes needed effo
#> [1941-Roosevelt, 289]      seemed frozen by a fatalistic [  terror   ] , we proved that this
#>   [1961-Kennedy, 868]    alter that uncertain balance of [  terror   ] that stays the hand of
#>   [1961-Kennedy, 992]          of science instead of its [  terrors  ] . Together let us explore
#>    [1981-Reagan, 821]     freeing all Americans from the [  terror   ] of runaway living costs.
#>   [1981-Reagan, 2204]   understood by those who practice [ terrorism ] and prey upon their neighbo
#>  [1997-Clinton, 1055]        They fuel the fanaticism of [  terror   ] . And they torment the
#>  [1997-Clinton, 1655]  maintain a strong defense against [  terror   ] and destruction. Our childr
#>    [2009-Obama, 1646]     advance their aims by inducing [  terror   ] and slaughtering innocents,
kwic(inaugCorpus, "communist*")
#>                                          contextPre    keyword                  contextPost
#>  [1949-Truman, 838] the actions resulting from the [ Communist  ] philosophy are a threat to
#> [1961-Kennedy, 519]             -- not because the [ Communists ] may be doing it,
```

In the above summary, `Year` and `President` are variables associated with each document. We can access such variables with the `docvars()` function.

```
# inspect the document-level variables
```

```
head(docvars(inaugCorpus))
#>                 Year  President FirstName
#> 1789-Washington 1789 Washington    George
#> 1793-Washington 1793 Washington    George
#> 1797-Adams      1797      Adams      John
#> 1801-Jefferson  1801  Jefferson    Thomas
#> 1805-Jefferson  1805  Jefferson    Thomas
#> 1809-Madison    1809    Madison     James

# inspect the corpus-level metadata
metacorpus(inaugCorpus)
#> $source
#> [1] "/home/paul/Dropbox/code/quanteda/* on x86_64 by paul"
#>
#> $created
#> [1] "Fri Sep 12 12:41:17 2014"
#>
#> $notes
#> NULL
#>
#> $citation
#> NULL
```

More corpora are available from the quantedaData package.

Extracting Features from a Corpus

In order to perform statistical analysis such as document scaling, we must extract a matrix associating values for certain features with each document. In quanteda, we use the `dfm` function to produce such a matrix. “dfm” is short for *document-feature matrix*, and always refers to documents in rows and “features” as columns. We fix this dimensional orientation because is is standard in data analysis to have a unit of analysis as a row, and features or variables pertaining to each unit as columns. We call them “features” rather than terms, because features are more general than terms: they can be defined as raw terms, stemmed terms, the parts of speech of terms, terms after stopwords have been removed, or a dictionary class to which a term belongs. Features can be entirely general, such as ngrams or syntactic dependencies, and we leave this open-ended.

Tokenizing texts

To simply tokenize a text, quanteda provides a powerful command called `tokenize()`. This produces an intermediate object, consisting of a list of tokens in the form of character vectors, where each element of the list corresponds to an input document.

`tokenize()` is deliberately conservative, meaning that it does not remove anything from the text unless told to do so.

```
txt <- c(text1 = "This is $10 in 999 different ways,\n up and down; left and right!",
         text2 = "@kenbenoit working: on #quanteda 2day\t4ever, http://textasdata.com?page=123.")
```

```
tokenize(txt)
#> tokenizedText object from 2 documents.
#> text1 :
#>  [1] "This"      "is"        "$"         "10"        "in"        "999"       "different" "ways"
#> [17] "!"
#>
#> text2 :
#>  [1] "@kenbenoit"     "working"        ":"              "on"             "#quanteda"      "2day"
#> [12] "/"              "textasdata.com" "?"              "page"           "="              "123"
tokenize(txt, removeNumbers = TRUE, removePunct = TRUE)
#> tokenizedText object from 2 documents.
#> text1 :
#>  [1] "This"      "is"        "in"        "different" "ways"      "up"        "and"       "down"
#>
#> text2 :
#> [1] "@kenbenoit"     "working"        "on"             "#quanteda"      "2day"           "4ever"
tokenize(txt, removeNumbers = FALSE, removePunct = TRUE)
#> tokenizedText object from 2 documents.
#> text1 :
#>  [1] "This"      "is"        "10"        "in"        "999"       "different" "ways"      "up"
#>
#> text2 :
#>  [1] "@kenbenoit"     "working"        "on"             "#quanteda"      "2day"           "4ever"
tokenize(txt, removeNumbers = TRUE, removePunct = FALSE)
#> tokenizedText object from 2 documents.
#> text1 :
#>  [1] "This"      "is"        "$"         "in"        "different" "ways"      ","         "up"
#>
#> text2 :
#>  [1] "@kenbenoit"     "working"        ":"              "on"             "#quanteda"      "2day"
#> [12] "/"              "textasdata.com" "?"              "page"           "="              "."
tokenize(txt, removeNumbers = FALSE, removePunct = FALSE)
#> tokenizedText object from 2 documents.
#> text1 :
#>  [1] "This"      "is"        "$"         "10"        "in"        "999"       "different" "ways"
#> [17] "!"
#>
#> text2 :
#>  [1] "@kenbenoit"     "working"        ":"              "on"             "#quanteda"      "2day"
#> [12] "/"              "textasdata.com" "?"              "page"           "="              "123"
tokenize(txt, removeNumbers = FALSE, removePunct = FALSE, removeSeparators = FALSE)
#> tokenizedText object from 2 documents.
#> text1 :
#>  [1] "This"      " "         "is"        " "         "$"         "10"        " "         "in"
#> [17] " "         "up"        " "         "and"       " "         "down"      ";"         " "
#>
#> text2 :
#>  [1] "@kenbenoit"     " "              "working"        ":"              " "              "on"
#> [12] "4ever"          ","              " "              "http"           ":"              "/"
#> [23] "123"            "."
```

We also have the option to tokenize characters:

```
tokenize("Great website: http://textasdata.com?page=123.", what = "character")
#> tokenizedText object from 1 document.
```

```
#> Component 1 :
#>  [1] "G" "r" "e" "a" "t" "w" "e" "b" "s" "i" "t" "e" ":" "h" "t" "t" "p" ":" "/" "/" "t" "e" "x"
tokenize("Great website: http://textasdata.com?page=123.", what = "character",
         removeSeparators = FALSE)
#> tokenizedText object from 1 document.
#> Component 1 :
#>  [1] "G" "r" "e" "a" "t" " " "w" "e" "b" "s" "i" "t" "e" ":" " " "h" "t" "t" "p" ":" "/" "/" "t"
```

and sentences:

```
# sentence level
tokenize(c("Kurt Vongeut said; only assholes use semi-colons.",
           "Today is Thursday in Canberra:  It is yesterday in London.",
           "En el caso de que no puedas ir con ellos, ¿quieres ir con nosotros?"),
         what = "sentence")
#> tokenizedText object from 3 documents.
#> Component 1 :
#> [1] "Kurt Vongeut said; only assholes use semi-colons."
#>
#> Component 2 :
#> [1] "Today is Thursday in Canberra:  It is yesterday in London."
#>
#> Component 3 :
#> [1] "En el caso de que no puedas ir con ellos, ¿quieres ir con nosotros?"
```

Constructing a document-frequency matrix

Tokenizing texts is an intermediate option, and most users will want to skip straight to constructing a document-feature matrix. For this, we have a Swiss-army knife function, called `dfm()`, which performs tokenization and tabulates the extracted features into a matrix of documents by features. Unlike the conservative approach taken by `tokenize()`, the `dfm()` function applies certain options by default, such as `toLower()` – a separate function for lower-casing texts – and removes punctuation. All of the options to `tokenize()` can be passed to `dfm()`, however.

```
myCorpus <- subset(inaugCorpus, Year > 1990)

# make a dfm
myDfm <- dfm(myCorpus)
#> Creating a dfm from a corpus ...
#>    ... lowercasing
#>    ... tokenizing
#>    ... indexing documents: 6 documents
#>    ... indexing features: 2,303 feature types
#>    ... created a 6 x 2303 sparse dfm
#>    ... complete.
#> Elapsed time: 0.016 seconds.
myDfm[, 1:5]
#> Document-feature matrix of: 6 documents, 5 features.
#> 6 x 5 sparse Matrix of class "dfmSparse"
#>                features
```

```
#> docs           my fellow citizens today we
#>   1993-Clinton  7      5        2    10 52
#>   1997-Clinton  6      7        7     5 42
#>   2001-Bush     3      1        9     2 47
#>   2005-Bush     2      3        6     3 37
#>   2009-Obama    2      1        1     6 62
#>   2013-Obama    3      3        6     4 68
```

Other options for a `dfm()` include removing stopwords, and stemming the tokens.

```
# make a dfm, removing stopwords and applying stemming
myStemMat <- dfm(myCorpus, ignoredFeatures = stopwords("english"), stem = TRUE)
#> Creating a dfm from a corpus ...
#>    ... lowercasing
#>    ... tokenizing
#>    ... indexing documents: 6 documents
#>    ... indexing features: 2,303 feature types
#>    ... removed 115 features, from 174 supplied (glob) feature types
#>    ... stemming features (English), trimmed 504 feature variants
#>    ... created a 6 x 1684 sparse dfm
#>    ... complete.
#> Elapsed time: 0.028 seconds.
myStemMat[, 1:5]
#> Document-feature matrix of: 6 documents, 5 features.
#> 6 x 5 sparse Matrix of class "dfmSparse"
#>               features
#> docs           fellow citizen today celebr mysteri
#>   1993-Clinton      5       2    10      4       1
#>   1997-Clinton      7       8     6      1       0
#>   2001-Bush         1      10     2      0       0
#>   2005-Bush         3       7     3      2       0
#>   2009-Obama        1       1     6      2       0
#>   2013-Obama        3       8     6      1       0
```

The option `ignoredFeatures` provides a list of tokens to be ignored. Most users will supply a list of pre-defined “stop words”, defined for numerous languages, accessed through the `stopwords()` function:

```
head(stopwords("english"), 20)
#>  [1] "i"          "me"         "my"         "myself"     "we"         "our"        "ours"       "
#> [16] "his"        "himself"    "she"        "her"        "hers"
head(stopwords("russian"), 10)
#>  [1] "и"   "в"   "во"  "не"  "что" "он"  "на"  "я"   "с"   "со"
head(stopwords("arabic"), 10)
#>  [1] "قوة" "هي"  "هو"  "من"  "له"  "لن"  "لم"  "كل"  "في"  "فى"
```

Viewing the document-frequency matrix

The dfm can be inspected in the Enviroment pane in RStudio, or by calling R’s `View` function. Calling `plot` on a dfm will display a wordcloud using the wordcloud package

```
mydfm <- dfm(ukimmigTexts, ignoredFeatures=c("will", stopwords("english")))
#>
#>    ... lowercasing
#>    ... tokenizing
#>    ... indexing documents: 9 documents
#>    ... indexing features: 1,586 feature types
#>    ... removed 97 features, from 175 supplied (glob) feature types
#>    ... created a 9 x 1489 sparse dfm
#>    ... complete.
#> Elapsed time: 0.014 seconds.
mydfm
#> Document-feature matrix of: 9 documents, 1,489 features.
```

To access a list of the most frequently occurring features, we can use `topfeatures()`:

```
topfeatures(mydfm, 20)  # 20 top words
#> immigration     british      people      asylum     britain          uk      system  population
#>          66          37          35          29          28          27          27          21
#>         bnp     illegal        work     percent
#>          13          13          13          12
```

Plotting a word cloud is very simple, since this is the default `plot()` method for a `dfm` class object:

```
plot(mydfm)
```

The `plot.dfm()` method passes arguments through to `wordcloud()` from the **wordcloud** package, and can prettify the plot using the same arguments:

```
if (require(RColorBrewer))
    plot(mydfm, max.words = 100, colors = brewer.pal(6, "Dark2"), scale = c(8, .5))
#> Loading required package: RColorBrewer
```

Grouping documents by document variable

Often, we are interested in analysing how texts differ according to substantive factors which may be encoded in the document variables, rather than simply by the boundaries of the document files. We can group documents which share the same value for a document variable when creating a dfm:

```
byPartyDfm <- dfm(ie2010Corpus, groups = "party", ignoredFeatures = stopwords("english"))
#> Creating a dfm from a corpus ...
#>    ... grouping texts by variable: party
#>    ... lowercasing
#>    ... tokenizing
#>    ... indexing documents: 5 documents
#>    ... indexing features: 4,881 feature types
#>    ... removed 117 features, from 174 supplied (glob) feature types
#>    ... created a 5 x 4764 sparse dfm
#>    ... complete.
#> Elapsed time: 0.05 seconds.
```

We can sort this dfm, and inspect it:

```
sort(byPartyDfm)[, 1:10]
#> Document-feature matrix of: 5 documents, 10 features.
#> 5 x 10 sparse Matrix of class "dfmSparse"
#>        features
#> docs    will people budget government public minister tax economy pay jobs
#>   FF     212     23     44         47     65       11  60      37  41   41
#>   FG      93     78     71         61     47       62  11      20  29   17
#>   Green   59     15     26         19      4        4  11      16   4   15
#>   LAB     89     69     66         36     32       54  47      37  24   20
#>   SF     104     81     53         73     31       39  34      50  24   27
```

Note that the most frequently occurring feature is “will”, a word usually on English stop lists, but one that is not included in quanteda’s built-in English stopword list.

Grouping words by dictionary or equivalence class

For some applications we have prior knowledge of sets of words that are indicative of traits we would like to measure from the text. For example, a general list of positive words might indicate positive sentiment in a movie review, or we might have a dictionary of political terms which are associated with a particular ideological stance. In these cases, it is sometimes useful to treat these groups of words as equivalent for the purposes of analysis, and sum their counts into classes.

For example, let’s look at how words associated with terrorism and words associated with the economy vary by President in the inaugural speeches corpus. From the original corpus, we select Presidents since Clinton:

```
recentCorpus <- subset(inaugCorpus, Year > 1991)
```

Now we define a demonstration dictionary:

```
myDict <- dictionary(list(terror = c("terrorism", "terrorists", "threat"),
                          economy = c("jobs", "business", "grow", "work")))
```

We can use the dictionary when making the dfm:

```
byPresMat <- dfm(recentCorpus, dictionary = myDict)
#> Creating a dfm from a corpus ...
#>    ... lowercasing
#>    ... tokenizing
#>    ... indexing documents: 6 documents
#>    ... indexing features: 2,303 feature types
#>    ... applying a dictionary consisting of 2 keys
#>    ... created a 6 x 2 sparse dfm
#>    ... complete.
#> Elapsed time: 0.024 seconds.
byPresMat
#> Document-feature matrix of: 6 documents, 2 features.
#> 6 x 2 sparse Matrix of class "dfmSparse"
#>               features
#> docs           terror economy
#>   1993-Clinton      0       8
#>   1997-Clinton      1       8
#>   2001-Bush         0       4
#>   2005-Bush         1       6
#>   2009-Obama        1      10
#>   2013-Obama        1       6
```

The constructor function `dictionary()` also works with two common “foreign” dictionary formats: the LIWC and Provalis Research’s Wordstat format. For instance, we can load the LIWC and apply this to

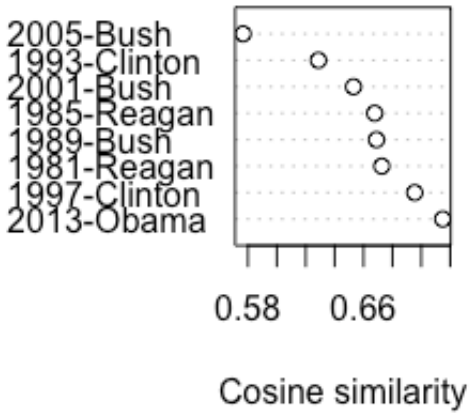
the Presidential inaugural speech corpus:

```
liwcdict <- dictionary(file = "~/Dropbox/QUANTESS/dictionaries/LIWC/LIWC2001_English.dic",
                       format = "LIWC")
liwcdfm <- dfm(inaugTexts[52:57], dictionary = liwcdict, verbose = FALSE)
liwcdfm[, 1:10]
```

Further Examples

Similarities between texts

```
presDfm <- dfm(subset(inaugCorpus, Year>1980),
               ignoredFeatures = stopwords("english"),
               stem=TRUE, verbose=FALSE)
obamaSimil <- similarity(presDfm, c("2009-Obama" , "2013-Obama"), n = NULL,
                             margin = "documents", method = "cosine", normalize = FALSE)
dotchart(obamaSimil$`2009-Obama`, xlab = "Cosine similarity")
```



We can use these distances to plot a dendrogram, clustering presidents:

```
data(SOTUCorpus, package="quantedaData")
presDfm <- dfm(subset(SOTUCorpus, year > 1960), verbose = FALSE, stem = TRUE,
               ignoredFeatures = stopwords("english"))
presDfm <- trim(presDfm, minCount=5, minDoc=3)
# hierarchical clustering - get distances on normalized dfm
presDistMat <- dist(as.matrix(weight(presDfm, "relFreq")))
# hiarchical clustering the distance object
presCluster <- hclust(presDistMat)
# label with document names
presCluster$labels <- docnames(presDfm)
# plot as a dendrogram
plot(presCluster)
```

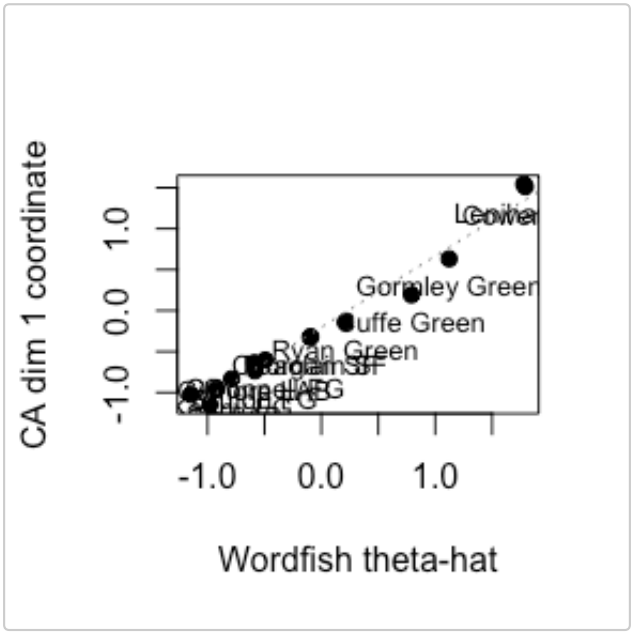
We can also look at term similarities:

```
similarity(presDfm, c("fair", "health", "terror"), method = "cosine", margin = "features", n = 20)
#> similarity Matrix:
#> $fair
#>   economi     begin      mani jefferson    author     howev     faith       god   struggl       ca
#>    0.9080    0.9076    0.9039    0.8981    0.8944    0.8944    0.8867    0.8723    0.8660    0.86
#>
#> $terror
#>    factori  adversari commonplac     miracl     racial     bounti     martin   guarante       sol
#>     0.9526     0.9526     0.9428     0.9428     0.9428     0.9428     0.9428     0.8944     0.894
#>   industri       open
#>     0.8433     0.8433
#>
#> $health
#> knowledg    shape  generat    wrong    defin   common    child     fear   demand   planet    powe
#>   0.9428   0.9045   0.8971   0.8944   0.8893   0.8889   0.8889   0.8889   0.8845   0.8819   0.879
```

Scaling document positions

We have a lot of development work to do on the `textmodel()` function, but here is a demonstration of unsupervised document scaling comparing the "wordfish" model to scaling from correspondence analysis:

```
# make prettier document names
docnames(ie2010Corpus) <-
    paste(docvars(ie2010Corpus, "name"), docvars(ie2010Corpus, "party"))
ieDfm <- dfm(ie2010Corpus, verbose = FALSE)
wf <- textmodel(ieDfm, model = "wordfish", dir=c(2,1))
#> Warning in if (dispersion == "poisson" & dispersionFloor != 0) warning("dispersionFloor argument
wca <- textmodel(ieDfm, model = "ca")
# plot the results
plot(wf@theta, -1*wca$rowcoord[,1],
     xlab="Wordfish theta-hat", ylab="CA dim 1 coordinate", pch=19)
text(wf@theta, -1*wca$rowcoord[,1], docnames(ieDfm), cex=.8, pos=1)
abline(lm(-1*wca$rowcoord[,1] ~ wf@theta), col="grey50", lty="dotted")
```



Topic models

```
quantdfm <- dfm(ie2010Corpus, verbose = FALSE,
                ignoredFeatures = c("will", stopwords("english")))

if (require(topicmodels)) {
    myLDAfit20 <- LDA(convert(quantdfm, to = "topicmodels"), k = 20)
    get_terms(myLDAfit20, 5)
    topics(myLDAfit20, 3)
}
#> Loading required package: topicmodels
#>      Lenihan FF Bruton FG Burton LAB Morgan SF Cowen FF Kenny FG ODonnell FG Gilmore LAB Higgins L
#> [1,]         17         11          12         18        16         5           2            6
#> [2,]          9         10           4          8        15         2           8           19
#> [3,]         14          7           1          1         1         1          19            2
```

---

1. This research was supported by the European Research Council grant ERC-2011-StG 283794-QUANTESS. Code contributors to the project include Ben Lauderdale, Pablo Barberà, and Kohei Watanabe.↩