05 Converting to and from Non-Tidy Formats

This project will focus on the process of tidying document-term matrices, as well as casting a tidy data frame into a sparse matrix. I will also expore how to tidy Corpus objects, which combine raw text with document metadata, into text data frames, leading to a case study of ingesting and analyzing financial articles.

05_01 Tidying a Document-Term Matrix

One of the most common structures that text mining packages work with is the document-term matrix (or DTM). This is a matrix where:

- each row represents one document (such as a book or article),
- each column represents one term, and
- each value (typically) contains the number of appearances of that term in that document.

05_01_01 Tidying DocumentTermMatrix Objects

Perhaps the most widely used implementation of DTMs in R is the DocumentTermMatrix class in the tm package. Many available text mining datasets are provided in this format. For example, consider the collection of Associated Press newspaper articles included in the topicmodels package.

```
data("AssociatedPress", package = "topicmodels")
AssociatedPress
```

```
## <<DocumentTermMatrix (documents: 2246, terms: 10473)>>
## Non-/sparse entries: 302031/23220327
## Sparsity           : 99%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

I see that this dataset contains documents (each of them an AP article) and terms (distinct words). Notice that this DTM is 99% sparse (99% of document-word pairs are zero). I could access the terms in the document with the Terms() function.

```
terms <- Terms(AssociatedPress)
head(terms)
```

```
## [1] "aaron"      "abandon"    "abandoned"  "abandoning" "abbott"     "abboud"
```

If I wanted to analyze this data with tidy tools, I would first need to turn it into a data frame with one-token-per-document-per-row. The broom package introduced the tidy() verb, which takes a non-tidy object and turns it into a tidy data frame. The tidytext package implements this method for DocumentTermMatrix objects.

```
ap_td <- tidy(AssociatedPress)
ap_td
```

```
## # A tibble: 302,031 × 3
##    document       term count
##       <int>      <chr> <dbl>
## 1         1     adding     1
## 2         1      adult     2
## 3         1        ago     1
## 4         1    alcohol     1
```

```
## 5         1  allegedly     1
## 6         1      allen     1
## 7         1 apparently     2
## 8         1   appeared     1
## 9         1   arrested     1
## 10        1    assault     1
## # ... with 302,021 more rows
```

I now have a tidy three-column tbl_df, with variables document, term, and count.

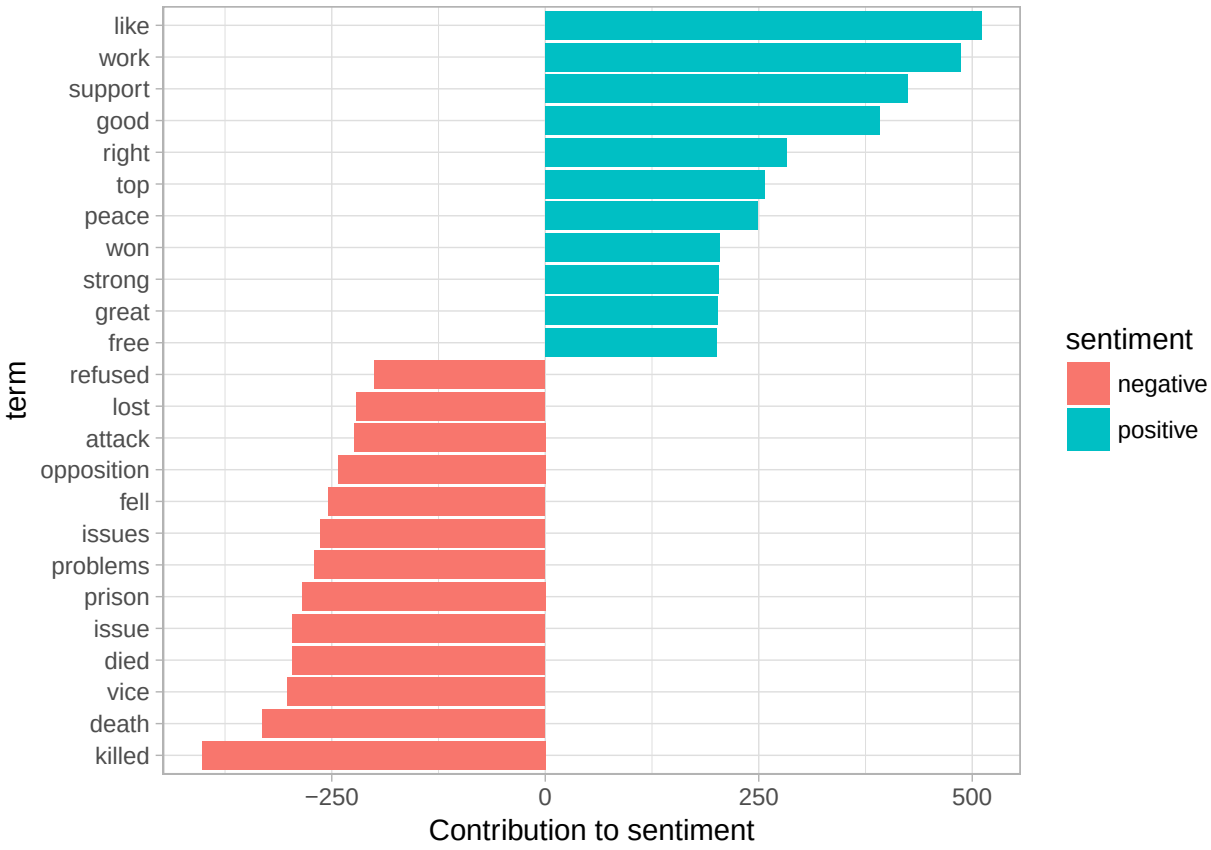
```
ap_sentiments <- ap_td %>%
  inner_join(get_sentiments("bing"), by = c(term = "word"))

ap_sentiments
```

```
## # A tibble: 30,094 × 4
##    document    term count sentiment
##       <int>   <chr> <dbl>     <chr>
## 1         1 assault     1  negative
## 2         1 complex     1  negative
## 3         1   death     1  negative
## 4         1    died     1  negative
## 5         1    good     2  positive
## 6         1 illness     1  negative
## 7         1  killed     2  negative
## 8         1    like     2  positive
## 9         1   liked     1  positive
## 10        1 miracle     1  positive
## # ... with 30,084 more rows
```

This would let me visualize which words from the AP articles most often contributed to positive or negative sentiment. I can see that the most common positive words include “like”, “work”, “support”, and “good”, while the most negative words include “killed”, “death”, and “vice”. (The inclusion of “vice” as a negative term is probably a mistake on the algorithm’s part, since it likely usually refers to “vice president”).

```
ap_sentiments %>%
  count(sentiment, term, wt = count) %>%
  ungroup() %>%
  filter(n >= 200) %>%
  mutate(n = ifelse(sentiment == "negative", -n, n)) %>%
  mutate(term = reorder(term, n)) %>%
  ggplot(aes(term, n, fill = sentiment)) +
  geom_bar(stat = "identity") +
  ylab("Contribution to sentiment") +
  coord_flip()
```



05_01_02 Tidying DocumentTermMatrix Objects

Just as some existing text mining packages provide document-term matrices as sample data or output, some algorithms expect such matrices as input. Therefore, tidytext provides cast_ verbs for converting from a tidy form to these matrices.

For example, I could take the tidied AP dataset and cast it back into a document-term matrix using the cast_dtm() function.

```
data("data_corpus_inaugural", package = "quanteda")
inaug_dfm <- quanteda::dfm(data_corpus_inaugural, verbose = FALSE)
```

```
inaug_dfm
```

```
## Document-feature matrix of: 58 documents, 9,232 features (91.6% sparse).
```

The tidy method works on these document-feature matrices as well, turning them into a one-token-per-document-per-row table:

```
inaug_td <- tidy(inaug_dfm)
inaug_td
```

```
## # A tibble: 44,725 × 3
##           document   term count
##              <chr>  <chr> <dbl>
## 1  1789-Washington fellow     3
## 2  1793-Washington fellow     1
## 3       1797-Adams fellow     3
```

```
## 4   1801-Jefferson fellow     7
## 5   1805-Jefferson fellow     8
## 6     1809-Madison fellow     1
## 7     1813-Madison fellow     1
## 8      1817-Monroe fellow     6
## 9      1821-Monroe fellow    10
## 10      1825-Adams fellow     3
## # ... with 44,715 more rows
```

I am interested in finding the words most specific to each of the inaugural speeches.

```
inaug_tf_idf <- inaug_td %>%
  bind_tf_idf(term, document, count) %>%
  arrange(desc(tf_idf))

inaug_tf_idf
```

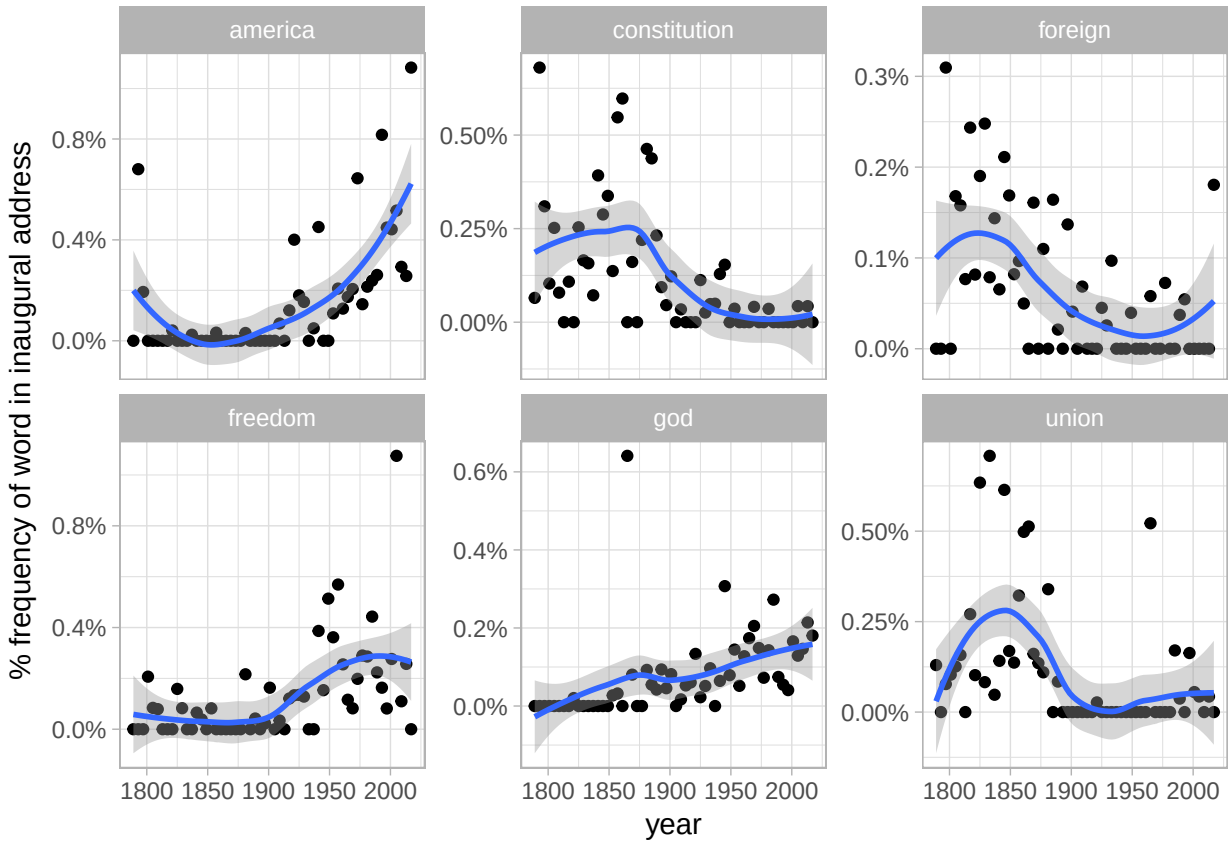
```
## # A tibble: 44,725 × 6
##           document        term count          tf      idf     tf_idf
##              <chr>       <chr> <dbl>       <dbl>    <dbl>      <dbl>
## 1  1793-Washington      arrive     1 0.006802721 4.060443 0.02762206
## 2  1793-Washington upbraidings     1 0.006802721 4.060443 0.02762206
## 3  1793-Washington    violated     1 0.006802721 3.367296 0.02290677
## 4  1793-Washington   willingly     1 0.006802721 3.367296 0.02290677
## 5  1793-Washington   incurring     1 0.006802721 3.367296 0.02290677
## 6  1793-Washington    previous     1 0.006802721 2.961831 0.02014851
## 7  1793-Washington   knowingly     1 0.006802721 2.961831 0.02014851
## 8  1793-Washington injunctions     1 0.006802721 2.961831 0.02014851
## 9  1793-Washington   witnesses     1 0.006802721 2.961831 0.02014851
## 10 1793-Washington     besides     1 0.006802721 2.674149 0.01819149
## # ... with 44,715 more rows
```

As another example of a visualization possible with tidy data, I could extract the year from each document's name, and compute the total number of words within each year.

```
year_term_counts <- inaug_td %>%
  extract(document, "year", "(\\d+)", convert = TRUE) %>%
  complete(year, term, fill = list(count = 0)) %>%
  group_by(year) %>%
  mutate(year_total = sum(count))
```

Further, I can see that over time, American presidents became less likely to refer to the country as the "Union" and more likely to refer to "America". They also became less likely to talk about the "constitution" and foreign" countries, and more likely to mention "freedom" and "God".

```
year_term_counts %>%
  filter(term %in% c("god", "america", "foreign", "union", "constitution", "freedom")) %>%
  ggplot(aes(year, count / year_total)) +
  geom_point() +
  geom_smooth() +
  facet_wrap(~ term, scales = "free_y") +
  scale_y_continuous(labels = scales::percent_format()) +
  ylab("% frequency of word in inaugural address")
```



These examples show how I can use tidytext, and the related suite of tidy tools, to analyze sources even if their origin was not in a tidy format.

05_02 Casting tidy text data into a matrix

Just as some existing text mining packages provide document-term matrices as sample data or output, some algorithms expect such matrices as input. Therefore, tidytext provides cast_ verbs for converting from a tidy form to these matrices.

For example, I could take the tidied AP dataset and cast it back into a document-term matrix using the cast_dtm() function.

```
ap_td %>%
  cast_dtm(document, term, count)
```

```
## <<DocumentTermMatrix (documents: 2246, terms: 10473)>>
## Non-/sparse entries: 302031/23220327
## Sparsity           : 99%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

Similarly, I could cast the table into a dfm object from quanteda's dfm with cast_dfm().

```
ap_td %>%
  cast_dfm(term, document, count)
```

```
## Document-feature matrix of: 10,473 documents, 2,246 features (98.7% sparse).
```

Some tools simply require a sparse matrix:

```
# cast into a Matrix object
m <- ap_td %>%
  cast_sparse(document, term, count)

class(m)
```

```
## [1] "dgCMatrix"
## attr(,"package")
## [1] "Matrix"
```

```
dim(m)
```

```
## [1]  2246 10473
```

This kind of conversion could easily be done from any of the tidy text structures I have used so far in this project. For example, I could create a DTM of Jane Austen's books in just a few lines of code.

```
austen_dtm <- austen_books() %>%
  unnest_tokens(word, text) %>%
  count(book, word) %>%
  cast_dtm(book, word, n)

austen_dtm
```

```
## <<DocumentTermMatrix (documents: 6, terms: 14520)>>
## Non-/sparse entries: 40379/46741
## Sparsity           : 54%
## Maximal term length: 19
## Weighting          : term frequency (tf)
```

This casting process allows for reading, filtering, and processing to be done using dplyr and other tidy tools, after which the data can be converted into a document-term matrix for machine learning applications.

05_03 Tidying Corpus Objects with Metadata

Some data structures are designed to store document collections before tokenization, often called a "corpus". One common example is Corpus objects from the tm package. These store text alongside metadata, which may include an ID, date/time, title, or language for each document.

For example, the tm package comes with the acq corpus, containing 50 articles from the news service Reuters.

```
data("acq")
acq
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 50
```

```
# first document
acq[[1]]
```

```
## <<PlainTextDocument>>
## Metadata:  15
## Content:  chars: 1287
```

A corpus object is structured like a list, with each item containing both text and metadata (see the tm documentation for more on working with Corpus documents). This is a flexible storage method for documents, but doesn't lend itself to processing with tidy tools.

Next, I construct a table with one row per document, including the metadata (such as id and datetimestamp) as columns alongside the text.

```
acq_td <- tidy(acq)
acq_td
```

```
## # A tibble: 50 × 16
##                       author       datetimestamp description
##                        <chr>              <dttm>       <chr>
## 1                       <NA> 1987-02-26 16:18:06
## 2                       <NA> 1987-02-26 16:19:15
## 3                       <NA> 1987-02-26 16:49:56
## 4  By Cal Mankowski, Reuters 1987-02-26 16:51:17
## 5                       <NA> 1987-02-26 17:08:33
## 6                       <NA> 1987-02-26 17:32:37
## 7      By Patti Domm, Reuter 1987-02-26 17:43:13
## 8                       <NA> 1987-02-26 17:59:25
## 9                       <NA> 1987-02-26 18:01:28
## 10                      <NA> 1987-02-26 18:08:27
##                                            heading    id language            origin topics
##                                              <chr> <chr>    <chr>             <chr>  <chr>
## 1   COMPUTER TERMINAL SYSTEMS <CPML> COMPLETES SALE    10       en Reuters-21578 XML    YES
## 2    OHIO MATTRESS <OMT> MAY HAVE LOWER 1ST QTR NET    12       en Reuters-21578 XML    YES
## 3     MCLEAN'S <MII> U.S. LINES SETS ASSET TRANSFER    44       en Reuters-21578 XML    YES
## 4    CHEMLAWN <CHEM> RISES ON HOPES FOR HIGHER BIDS    45       en Reuters-21578 XML    YES
## 5    <COFAB INC> BUYS GULFEX FOR UNDISCLOSED AMOUNT    68       en Reuters-21578 XML    YES
## 6          INVESTMENT FIRMS CUT CYCLOPS <CYL> STAKE    96       en Reuters-21578 XML    YES
## 7  AMERICAN EXPRESS <AXP> SEEN IN POSSIBLE SPINNOFF   110       en Reuters-21578 XML    YES
## 8   HONG KONG FIRM UPS WRATHER<WCO> STAKE TO 11 PCT   125       en Reuters-21578 XML    YES
## 9               LIEBERT CORP <LIEB> APPROVES MERGER   128       en Reuters-21578 XML    YES
## 10     GULF APPLIED TECHNOLOGIES <GATS> SELLS UNITS   134       en Reuters-21578 XML    YES
## # ... with 40 more rows, and 8 more variables: lewissplit <chr>, cgisplit <chr>, oldid <chr>,
## #   places <list>, people <lgl>, orgs <lgl>, exchanges <lgl>, text <chr>
```

So, for example, I can find the most common words across the 50 Reuters articles, or the ones most specific to each article.

```
acq_tokens <- acq_td %>%
  select(-places) %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = "word")

# most common words
acq_tokens %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 1,566 × 2
##        word     n
##       <chr> <int>
## 1      dlrs   100
## 2       pct    70
## 3       mln    65
## 4   company    63
## 5    shares    52
## 6    reuter    50
## 7     stock    46
```

```
## 8     offer    34
## 9     share    34
## 10 american    28
## # ... with 1,556 more rows
```

```
# tf-idf
acq_tokens %>%
  count(id, word) %>%
  bind_tf_idf(word, id, n) %>%
  arrange(desc(tf_idf))
```

```
## Source: local data frame [2,853 x 6]
## Groups: id [50]
##
##       id     word     n         tf      idf    tf_idf
##    <chr>    <chr> <int>      <dbl>    <dbl>     <dbl>
## 1    186   groupe     2 0.13333333 3.912023 0.5216031
## 2    128  liebert     3 0.13043478 3.912023 0.5102639
## 3    474  esselte     5 0.10869565 3.912023 0.4252199
## 4    371  burdett     6 0.10344828 3.912023 0.4046920
## 5    442 hazleton     4 0.10256410 3.912023 0.4012331
## 6    199  circuit     5 0.10204082 3.912023 0.3991860
## 7    162 suffield     2 0.10000000 3.912023 0.3912023
## 8    498     west     3 0.10000000 3.912023 0.3912023
## 9    441      rmj     8 0.12121212 3.218876 0.3901668
## 10   467  nursery     3 0.09677419 3.912023 0.3785829
## # ... with 2,843 more rows
```

Summary

Text analysis requires working with a variety of tools, many of which have inputs and outputs that aren't in a tidy form. This project showed how to convert between a tidy text data frame and sparse document-term matrices, as well as how to tidy a Corpus object containing document metadata.