04 Relationships between Words

So far we've considered words as individual units, and considered their relationships to sentiments or to documents. However, many interesting text analyses are based on the relationships between words, whether examining which words tend to follow others immediately, or that tend to co-occur within the same documents.

In this project, I will explore some of the methods tidytext offers for calculating and visualizing relationships between words in text dataset.

04_01 Tokenizing by n-gram

I have been using the unnest_tokens function to tokenize by word, or sometimes by sentence, which is useful for the kinds of sentiment and frequency analyses I have been doing so far. But I can also use the function to tokenize into consecutive sequences of words, called n-grams. By seeing how often word X is followed by word Y, I can then build a model of the relationships between them.

```
austen_bigrams <- austen_books() %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)

austen_bigrams
```

```
## # A tibble: 725,048 × 2
##                   book          bigram
##                 <fctr>           <chr>
## 1  Sense & Sensibility       sense and
## 2  Sense & Sensibility and sensibility
## 3  Sense & Sensibility  sensibility by
## 4  Sense & Sensibility         by jane
## 5  Sense & Sensibility     jane austen
## 6  Sense & Sensibility     austen 1811
## 7  Sense & Sensibility    1811 chapter
## 8  Sense & Sensibility       chapter 1
## 9  Sense & Sensibility           1 the
## 10 Sense & Sensibility      the family
## # ... with 725,038 more rows
```

04_01_01 Counting and Filtering n-grams

```
austen_bigrams %>%
  count(bigram, sort = TRUE)
```

```
## # A tibble: 211,237 × 2
##      bigram     n
##       <chr> <int>
## 1    of the  3017
## 2     to be  2787
## 3    in the  2368
## 4    it was  1781
## 5      i am  1545
## 6   she had  1472
## 7    of her  1445
## 8    to the  1387
```

```
## 9   she was  1377
## 10 had been  1299
## # ... with 211,227 more rows
```

In the next step, I split a column into multiple based on a delimiter. This lets me separate it into two columns, "word1" and "word2", at which point I can remove cases where either is a stop-word.

```
bigrams_separated <- austen_bigrams %>%
  separate(bigram, c("word1", "word2"), sep = " ")

bigrams_filtered <- bigrams_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

# new bigram counts:
bigram_counts <- bigrams_filtered %>%
  count(word1, word2, sort = TRUE)

bigram_counts
```

```
## Source: local data frame [33,421 x 3]
## Groups: word1 [6,711]
##
##      word1     word2     n
##      <chr>     <chr> <int>
## 1      sir    thomas   287
## 2     miss  crawford   215
## 3  captain wentworth   170
## 4     miss woodhouse   162
## 5    frank churchill   132
## 6     lady   russell   118
## 7     lady   bertram   114
## 8      sir    walter   113
## 9     miss   fairfax   109
## 10 colonel   brandon   108
## # ... with 33,411 more rows
```

I see that names (whether first and last or with a salutation) are the most common pairs in Jane Austen books.

For other analyses, I recombine the columns into one. Thus, "separate/filter/count/unite" let us find the most common bigrams not containing stop-words.

```
bigrams_united <- bigrams_filtered %>%
  unite(bigram, word1, word2, sep = " ")

bigrams_united
```

```
## # A tibble: 44,784 × 2
##                   book                   bigram
## *               <fctr>                    <chr>
## 1  Sense & Sensibility              jane austen
## 2  Sense & Sensibility              austen 1811
## 3  Sense & Sensibility             1811 chapter
## 4  Sense & Sensibility                chapter 1
## 5  Sense & Sensibility             norland park
## 6  Sense & Sensibility surrounding acquaintance
```

```
## 7  Sense & Sensibility              late owner
## 8  Sense & Sensibility            advanced age
## 9  Sense & Sensibility      constant companion
## 10 Sense & Sensibility            happened ten
## # ... with 44,774 more rows
```

In other analyses the most common trigrams, which are consecutive sequences of 3 words, can be also interesting.

```
austen_books() %>%
  unnest_tokens(trigram, text, token = "ngrams", n = 3) %>%
  separate(trigram, c("word1", "word2", "word3"), sep = " ") %>%
  filter(!word1 %in% stop_words$word,
         !word2 %in% stop_words$word,
         !word3 %in% stop_words$word) %>%
  count(word1, word2, word3, sort = TRUE)
```

```
## Source: local data frame [8,757 x 4]
## Groups: word1, word2 [7,462]
##
##        word1     word2     word3     n
##        <chr>     <chr>     <chr> <int>
## 1       dear      miss woodhouse    23
## 2       miss        de    bourgh    18
## 3       lady catherine        de    14
## 4  catherine        de    bourgh    13
## 5       poor      miss    taylor    11
## 6        sir    walter    elliot    11
## 7        ten  thousand    pounds    11
## 8       dear       sir    thomas    10
## 9     twenty  thousand    pounds     8
## 10   replied      miss  crawford     7
## # ... with 8,747 more rows
```

04_01_02 Analyzing Bigrams

This one-bigram-per-row format is helpful for exploratory analyses of the text. As a simple example, the most common "streets" mentioned in each book can be interesting:

```
bigrams_filtered %>%
  filter(word2 == "street") %>%
  count(book, word1, sort = TRUE)
```

```
## Source: local data frame [34 x 3]
## Groups: book [6]
##
##                   book       word1     n
##                 <fctr>       <chr> <int>
## 1  Sense & Sensibility    berkeley    16
## 2  Sense & Sensibility      harley    16
## 3      Northanger Abbey    pulteney    14
## 4      Northanger Abbey      milsom    11
## 5        Mansfield Park     wimpole    10
## 6    Pride & Prejudice gracechurch     9
## 7  Sense & Sensibility     conduit     6
```

```
## 8  Sense & Sensibility        bond     5
## 9           Persuasion      milsom     5
## 10          Persuasion      rivers     4
## # ... with 24 more rows
```

A bigram can also be treated as a term in a document in the same way that individual words are treated.

```
bigram_tf_idf <- bigrams_united %>%
  count(book, bigram) %>%
  bind_tf_idf(bigram, book, n) %>%
  arrange(desc(tf_idf))

bigram_tf_idf
```

```
## Source: local data frame [36,217 x 6]
## Groups: book [6]
##
##                   book            bigram     n         tf      idf
##                 <fctr>             <chr> <int>      <dbl>    <dbl>
## 1           Persuasion captain wentworth   170 0.02985599 1.791759
## 2       Mansfield Park       sir thomas   287 0.02873160 1.791759
## 3       Mansfield Park     miss crawford   215 0.02152368 1.791759
## 4           Persuasion      lady russell   118 0.02072357 1.791759
## 5           Persuasion        sir walter   113 0.01984545 1.791759
## 6                 Emma    miss woodhouse   162 0.01700966 1.791759
## 7     Northanger Abbey       miss tilney    82 0.01594400 1.791759
## 8  Sense & Sensibility   colonel brandon   108 0.01502086 1.791759
## 9                 Emma   frank churchill   132 0.01385972 1.791759
## 10   Pride & Prejudice    lady catherine   100 0.01380453 1.791759
## # ... with 36,207 more rows, and 1 more variables: tf_idf <dbl>
```

Much as I discovered in earlier, the units that distinguish each Austen book are almost exclusively names. I also notice some pairings of a common verb and a name, such as "replied elizabeth" in Pride & Prejudice, or "cried emma" in Emma.

There are advantages and disadvantages to examining the tf-idf of bigrams rather than individual words. Pairs of consecutive words might capture structure that isn't present when one is just counting single words, and may provide context that makes tokens more understandable (for example, "pulteney street", in Northanger Abbey, is more informative than "pulteney").

04_01_03 Using Bigrams to provide Context in Setiment Analysis

One of the problems with this approach is that a word's context can matter nearly as much as its presence. For example, the words "happy" and "like" will be counted as positive, even in a sentence like "I'm not happy and I don't like it!"Now that we have the data organized into bigrams, it's easy to tell how often words are preceded by a word like "not":

```
bigrams_separated %>%
  filter(word1 == "not") %>%
  count(word1, word2, sort = TRUE)
```

```
## Source: local data frame [1,246 x 3]
## Groups: word1 [1]
##
##    word1 word2     n
##    <chr> <chr> <int>
```

```
## 1    not    be   610
## 2    not    to   355
## 3    not  have   327
## 4    not  know   252
## 5    not     a   189
## 6    not think   176
## 7    not  been   160
## 8    not   the   147
## 9    not    at   129
## 10   not    in   118
## # ... with 1,236 more rows
```

By performing sentiment analysis on the bigram data, I can examine how often sentiment-associated words are preceded by "not" or other negating words. I could use this to ignore or even reverse their contribution to the sentiment score.

I use the AFINN lexicon for sentiment analysis with positive or negative numbers indicating the direction of the sentiment.

```
AFINN <- get_sentiments("afinn")

AFINN
```

```
## # A tibble: 2,476 × 2
##           word score
##          <chr> <int>
## 1      abandon    -2
## 2    abandoned    -2
## 3     abandons    -2
## 4     abducted    -2
## 5    abduction    -2
## 6   abductions    -2
## 7        abhor    -3
## 8     abhorred    -3
## 9    abhorrent    -3
## 10      abhors    -3
## # ... with 2,466 more rows
```

I can then examine the most frequent words that were preceded by "not" and were associated with a sentiment.

```
not_words <- bigrams_separated %>%
  filter(word1 == "not") %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word2, score, sort = TRUE) %>%
  ungroup()

not_words
```

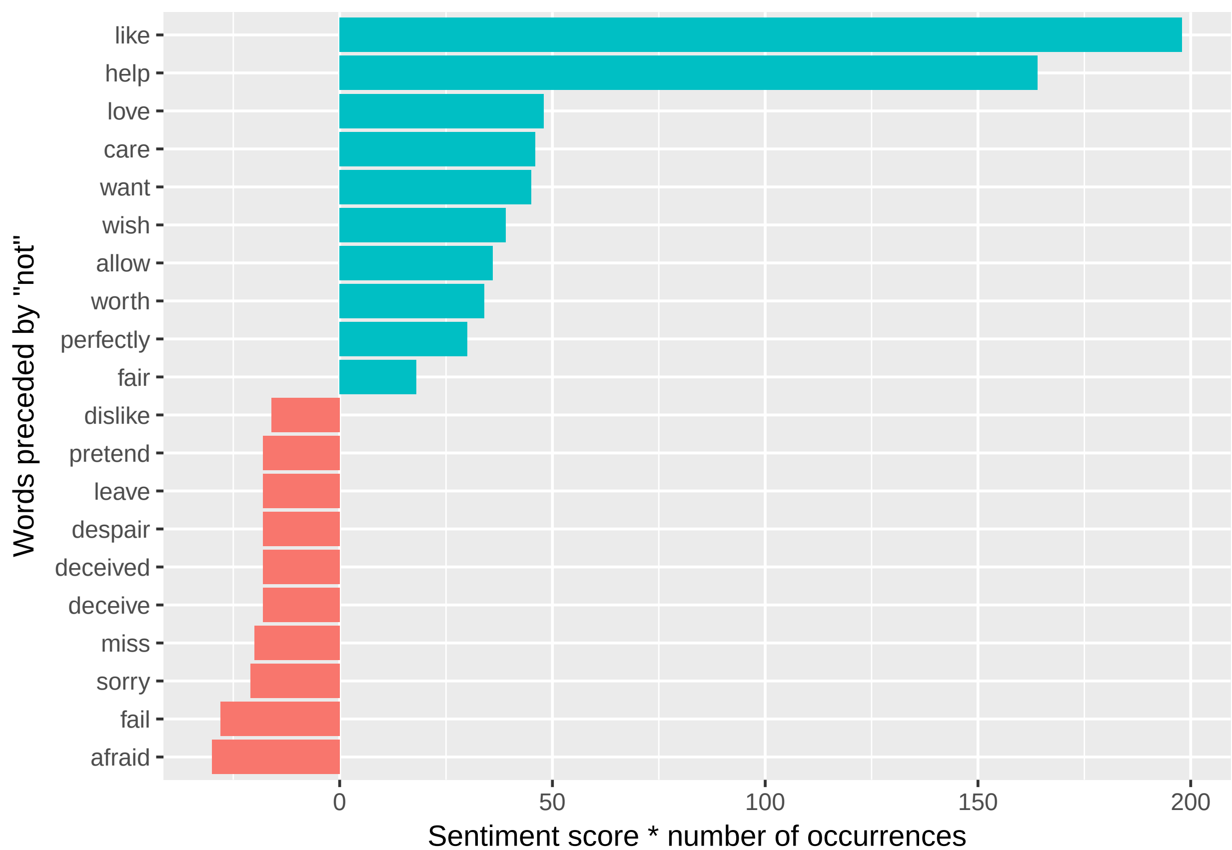
```
## # A tibble: 245 × 3
##      word2 score     n
##      <chr> <int> <int>
## 1     like     2    99
## 2     help     2    82
## 3     want     1    45
## 4     wish     1    39
## 5    allow     1    36
## 6     care     2    23
```

```
## 7     sorry    -1    21
## 8     leave    -1    18
## 9   pretend    -1    18
## 10    worth     2    17
## # ... with 235 more rows
```

For example, the most common sentiment-associated word to follow "not" was "like", which would normally have a (positive) score of 2.

It's worth asking which words contributed the most in the "wrong" direction. To compute that, I can multiply their score by the number of times they appear (so that a word with a score of +3 occurring 10 times has as much impact as a word with a sentiment score of +1 occurring 30 times). I visualize the result with a bar plot.

```r
not_words %>%
  mutate(contribution = n * score) %>%
  arrange(desc(abs(contribution))) %>%
  head(20) %>%
  mutate(word2 = reorder(word2, contribution)) %>%
  ggplot(aes(word2, n * score, fill = n * score > 0)) +
  geom_col(show.legend = FALSE) +
  xlab("Words preceded by \"not\"") +
  ylab("Sentiment score * number of occurrences") +
  coord_flip()
```



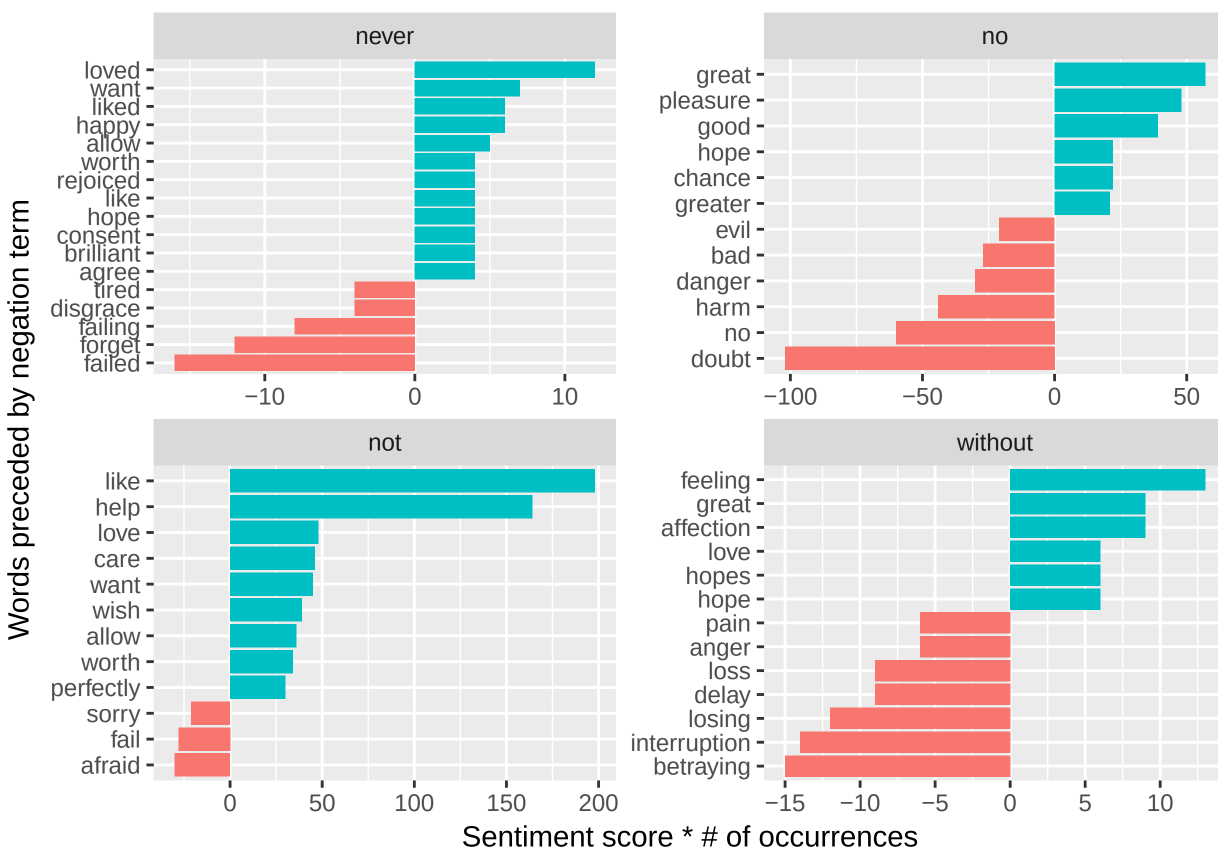
The bigrams "not like" and "not help" were overwhelmingly the largest causes of misidentification, making the text seem much more positive than it is. But I can see phrases like "not afraid" and "not fail" sometimes suggest text is more negative than it is.

I could pick four common words (or more) that negate the subsequent term, and use the same joining and counting approach to examine all of them at once. So I can then visualize what the most common words to follow each particular negation are.

```
negation_words <- c("not", "no", "never", "without")

negated_words <- bigrams_separated %>%
  filter(word1 %in% negation_words) %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word1, word2, score, sort = TRUE) %>%
  ungroup()
```

```
negated_words %>%
  mutate(contribution = n * score,
         word2 = reorder(paste(word2, word1, sep = "__"), contribution)) %>%
  group_by(word1) %>%
  top_n(12, abs(contribution)) %>%
  ggplot(aes(word2, contribution, fill = n * score > 0)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ word1, scales = "free") +
  scale_x_discrete(labels = function(x) gsub("__.+$", "", x)) +
  xlab("Words preceded by negation term") +
  ylab("Sentiment score * # of occurrences") +
  coord_flip()
```



While “not like” and “not help” are still the two most common examples, we can also see pairings such as “no great” and “never loved.” We could combine this with the approaches in Chapter 2 to reverse the AFINN scores of each word that follows a negation. These are just a few examples of how finding consecutive words can give context to text mining methods.

04_01_04 Visualizing a Network of Bigrams with ggraph

The igraph package has many powerful functions for manipulating and analyzing networks. One way to create an igraph object from tidy data is the graph_from_data_frame() function, which takes a data frame of edges with columns for "from", "to", and edge attributes (in this case n):

```
# original counts
bigram_counts
```

```
## Source: local data frame [33,421 x 3]
## Groups: word1 [6,711]
##
##      word1     word2     n
##      <chr>     <chr> <int>
## 1      sir    thomas   287
## 2     miss  crawford   215
## 3  captain wentworth   170
## 4     miss woodhouse   162
## 5    frank churchill   132
## 6     lady   russell   118
## 7     lady   bertram   114
## 8      sir    walter   113
## 9     miss   fairfax   109
## 10 colonel   brandon   108
## # ... with 33,411 more rows
```

```
# filter for only relatively common combinations
bigram_graph <- bigram_counts %>%
  filter(n > 20) %>%
  graph_from_data_frame()

bigram_graph
```

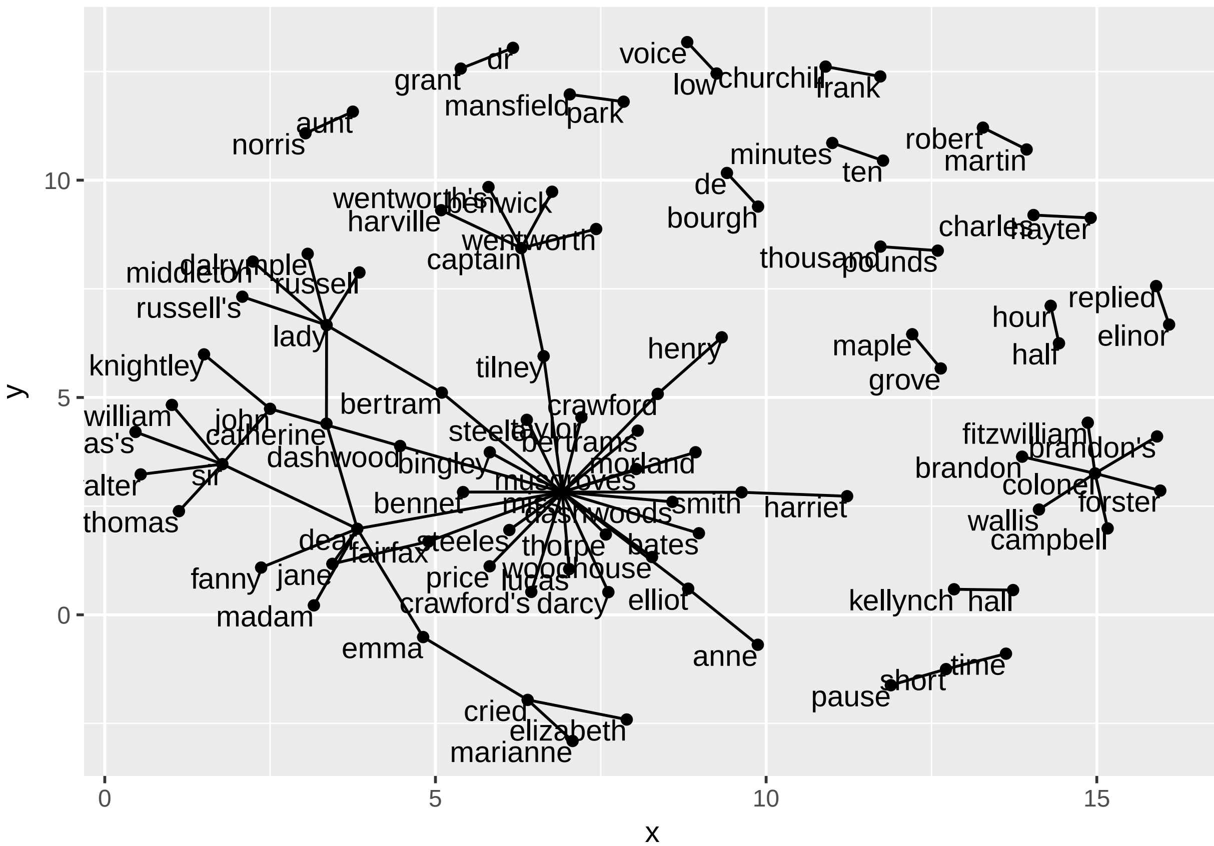
```
## IGRAPH DN-- 91 77 --
## + attr: name (v/c), n (e/n)
## + edges (vertex names):
##  [1] sir     ->thomas     miss    ->crawford   captain ->wentworth
##  [4] miss    ->woodhouse  frank   ->churchill  lady    ->russell
##  [7] lady    ->bertram    sir     ->walter     miss    ->fairfax
## [10] colonel ->brandon    miss    ->bates      lady    ->catherine
## [13] sir     ->john       jane    ->fairfax    miss    ->tilney
## [16] lady    ->middleton  miss    ->bingley    thousand->pounds
## [19] miss    ->dashwood   miss    ->bennet     john    ->knightley
## [22] miss    ->morland    captain ->benwick    dear    ->miss
## + ... omitted several edges
```

I can convert an igraph object into a ggraph with the ggraph function, after which I add layers to it, much like layers are added in ggplot2. For example, for a basic graph I need to add three layers: nodes, edges, and text.

```
set.seed(2017)

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link() +
```

```
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```



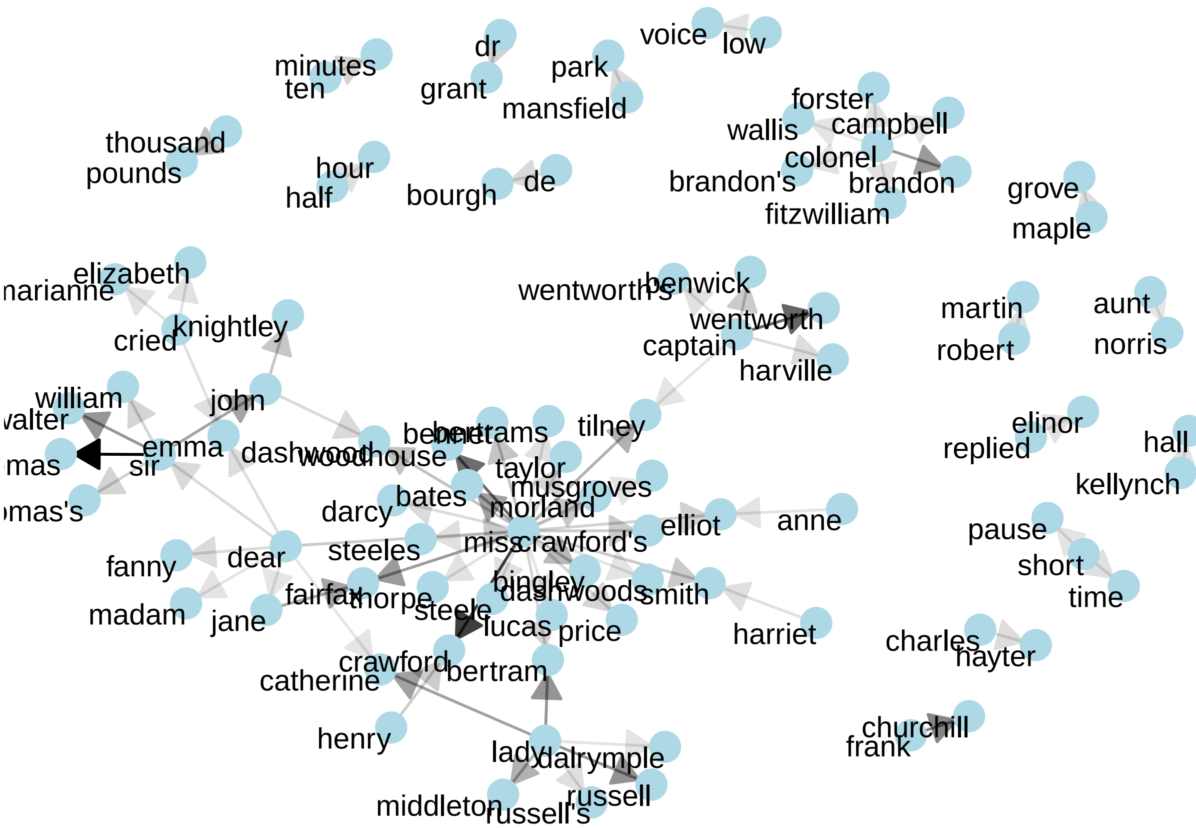
I can visualize some details of the text structure. For example, I see that salutations such as “miss”, “lady”, “sir”, “and”colonel” form common centers of nodes, which are often followed by names. I also see pairs or triplets along the outside that form common short phrases (“half hour”, “thousand pounds”, or “short time/pause”).

With a few polishing operations I want to make a better looking graph:

```
set.seed(2016)

a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE,
                 arrow = a, end_cap = circle(.07, 'inches')) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  theme_void()
```



04_01_05 Visualizing Bigrams in other Texts

I went to a good amount of work in cleaning and visualizing bigrams on a text dataset, so I want to collect it into a function so that I easily perform it on other text datasets.

```
count_bigrams <- function(dataset) {
  dataset %>%
    unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
    separate(bigram, c("word1", "word2"), sep = " ") %>%
    filter(!word1 %in% stop_words$word,
           !word2 %in% stop_words$word) %>%
    count(word1, word2, sort = TRUE)
}

visualize_bigrams <- function(bigrams) {
  set.seed(2016)
  a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

  bigrams %>%
    graph_from_data_frame() %>%
    ggraph(layout = "fr") +
    geom_edge_link(aes(edge_alpha = n), show.legend = FALSE, arrow = a) +
    geom_node_point(color = "lightblue", size = 5) +
    geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
    theme_void()
}
```

At this point, I can visualize bigrams in other works, such as the King James Version of the Bible:

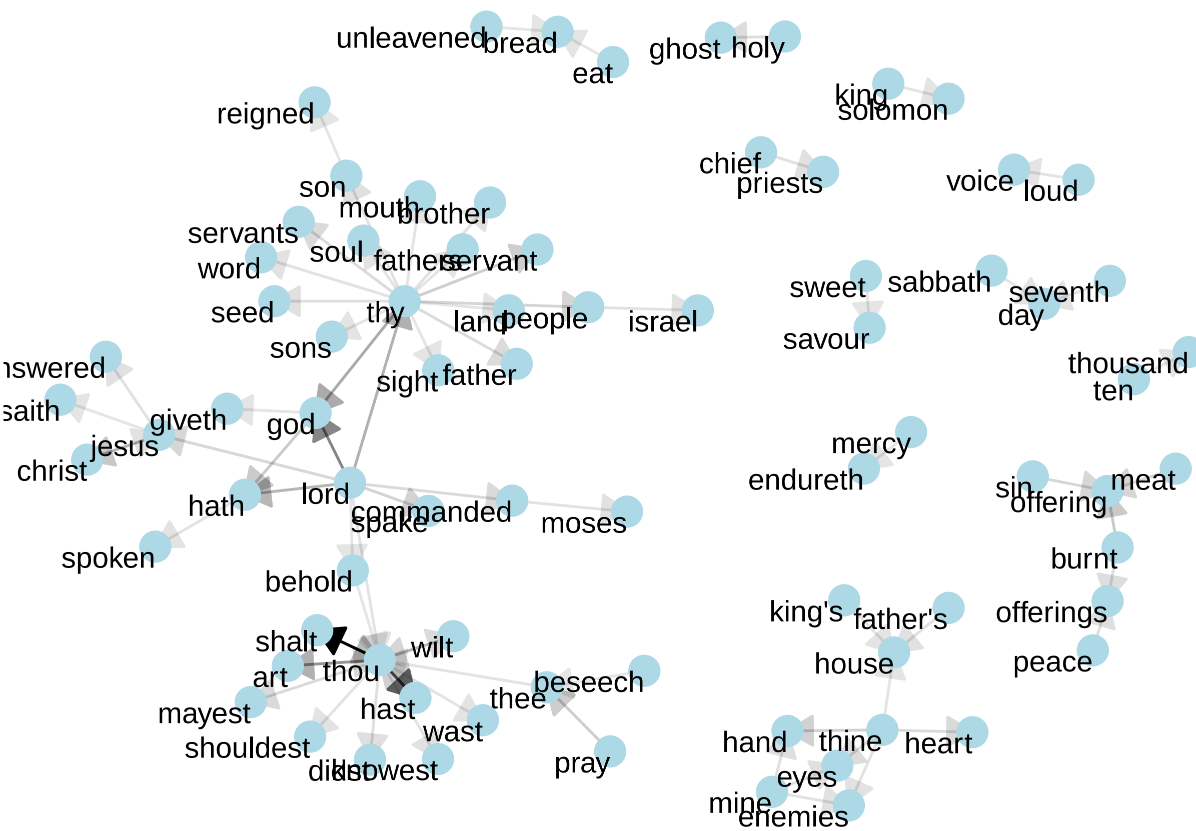
```r
# the King James version is book 10 on Project Gutenberg:
kjv <- gutenberg_download(10)
```

```
## Determining mirror for Project Gutenberg from http://www.gutenberg.org/robot/harvest
```

```
## Using mirror http://aleph.gutenberg.org
```

```r
kjv_bigrams <- kjv %>%
  count_bigrams()

# filter out rare combinations, as well as digits
kjv_bigrams %>%
  filter(n > 40,
         !str_detect(word1, "\\d"),
         !str_detect(word2, "\\d")) %>%
  visualize_bigrams()
```



04_02 Counting and Correlating Pairs of Words with the Widyr Package

Tokenizing by n-gram is a useful way to explore pairs of adjacent words. However, I may also be interested in words that tend to co-occur within particular documents or particular chapters, even if they don't occur next to each other.

Tidy data is a useful structure for comparing between variables or grouping by rows, but it can be challenging to compare between rows: for example, to count the number of times that two words appear within the same document, or to see how correlated they are. Most operations for finding pairwise counts or correlations need to turn the data into a wide matrix first.

04_02_01 Counting and Correlating among Sections

Consider the book "Pride and Prejudice" divided into 10-line sections. I am interested in what words tend to appear within the same section.

```
austen_section_words <- austen_books() %>%
  filter(book == "Pride & Prejudice") %>%
  mutate(section = row_number() %/% 10) %>%
  filter(section > 0) %>%
  unnest_tokens(word, text) %>%
  filter(!word %in% stop_words$word)

austen_section_words
```

```
## # A tibble: 37,240 × 3
##                   book section         word
##                 <fctr>   <dbl>        <chr>
## 1  Pride & Prejudice       1        truth
## 2  Pride & Prejudice       1  universally
## 3  Pride & Prejudice       1 acknowledged
## 4  Pride & Prejudice       1       single
## 5  Pride & Prejudice       1   possession
## 6  Pride & Prejudice       1      fortune
## 7  Pride & Prejudice       1         wife
## 8  Pride & Prejudice       1     feelings
## 9  Pride & Prejudice       1        views
## 10 Pride & Prejudice       1     entering
## # ... with 37,230 more rows
```

```
# count words co-occuring within sections
word_pairs <- austen_section_words %>%
  pairwise_count(word, section, sort = TRUE)

word_pairs
```

```
## # A tibble: 796,008 × 3
##        item1     item2     n
##        <chr>     <chr> <dbl>
## 1      darcy elizabeth   144
## 2  elizabeth     darcy   144
## 3       miss elizabeth   110
## 4  elizabeth      miss   110
## 5  elizabeth      jane   106
## 6       jane elizabeth   106
## 7       miss     darcy    92
## 8      darcy      miss    92
## 9  elizabeth   bingley    91
## 10   bingley elizabeth    91
## # ... with 795,998 more rows
```

For example, I can see that the most common pair of words in a section is "Elizabeth" and "Darcy" (the two main characters). I can easily find the words that most often occur with Darcy:

```
word_pairs %>%
  filter(item1 == "darcy")
```

```
## # A tibble: 2,930 × 3
```

```
##    item1     item2     n
##    <chr>     <chr> <dbl>
## 1  darcy elizabeth   144
## 2  darcy      miss    92
## 3  darcy   bingley    86
## 4  darcy      jane    46
## 5  darcy    bennet    45
## 6  darcy    sister    45
## 7  darcy      time    41
## 8  darcy      lady    38
## 9  darcy    friend    37
## 10 darcy   wickham    37
## # ... with 2,920 more rows
```

04_02_02 Pairwise Correlation

Pairs like "Elizabeth" and "Darcy" are the most common co-occurring words, but that's not particularly meaningful since they're also the most common individual words. I may instead want to examine correlation among words, which indicates how often they appear together relative to how often they appear separately.

The pairwise_cor() function in widyr lets us find the phi coefficient between words based on how often they appear in the same section. Its syntax is similar to pairwise_count().

```
# we need to filter for at least relatively common words first
word_cors <- austen_section_words %>%
  group_by(word) %>%
  filter(n() >= 20) %>%
  pairwise_cor(word, section, sort = TRUE)

word_cors
```

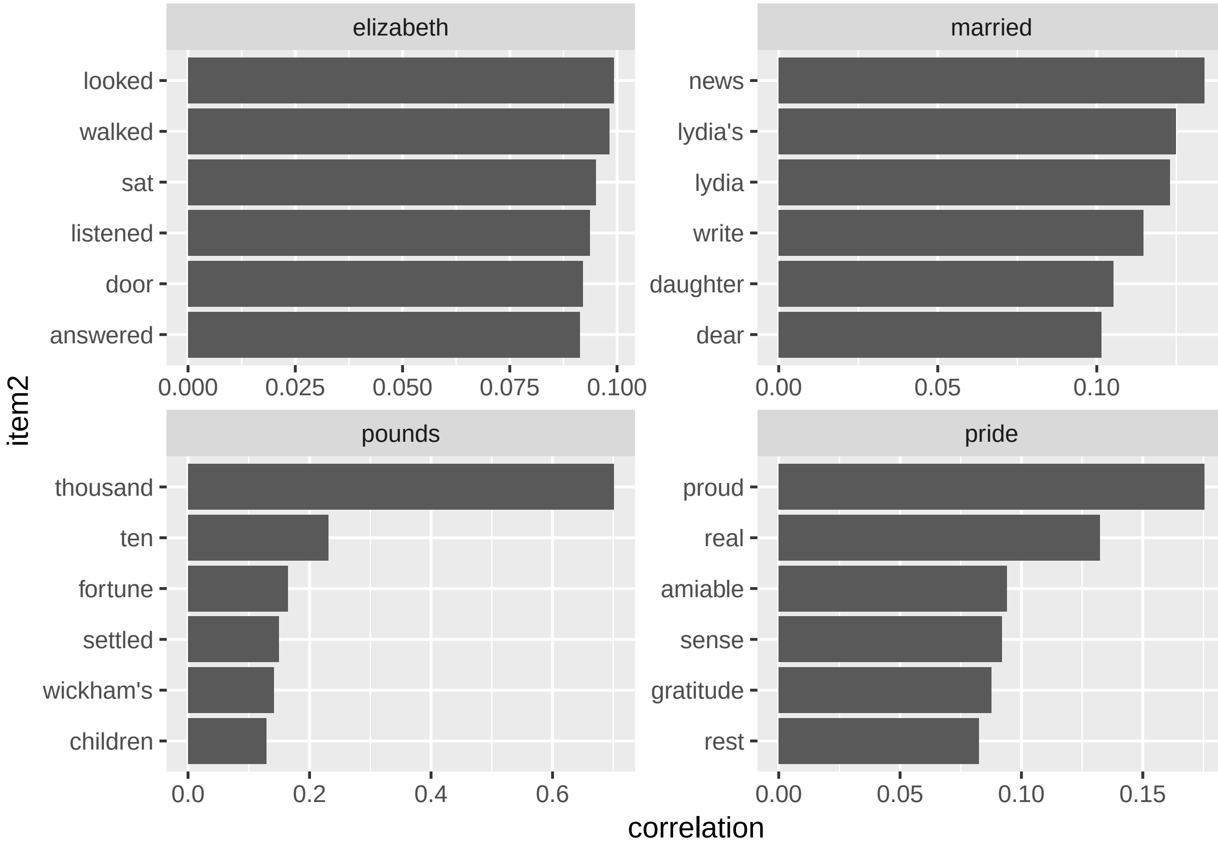
```
## # A tibble: 154,842 × 3
##        item1     item2 correlation
##        <chr>     <chr>       <dbl>
## 1     bourgh        de   0.9508501
## 2         de    bourgh   0.9508501
## 3     pounds  thousand   0.7005808
## 4   thousand    pounds   0.7005808
## 5    william       sir   0.6644719
## 6        sir   william   0.6644719
## 7  catherine      lady   0.6633048
## 8       lady catherine   0.6633048
## 9    forster   colonel   0.6220950
## 10   colonel   forster   0.6220950
## # ... with 154,832 more rows
```

This lets me pick particular interesting words and find the other words most associated with them.

```
word_cors %>%
  filter(item1 %in% c("elizabeth", "pounds", "married", "pride")) %>%
  group_by(item1) %>%
  top_n(6) %>%
  ungroup() %>%
  mutate(item2 = reorder(item2, correlation)) %>%
  ggplot(aes(item2, correlation)) +
```

```r
  geom_bar(stat = "identity") +
  facet_wrap(~ item1, scales = "free") +
  coord_flip()
```

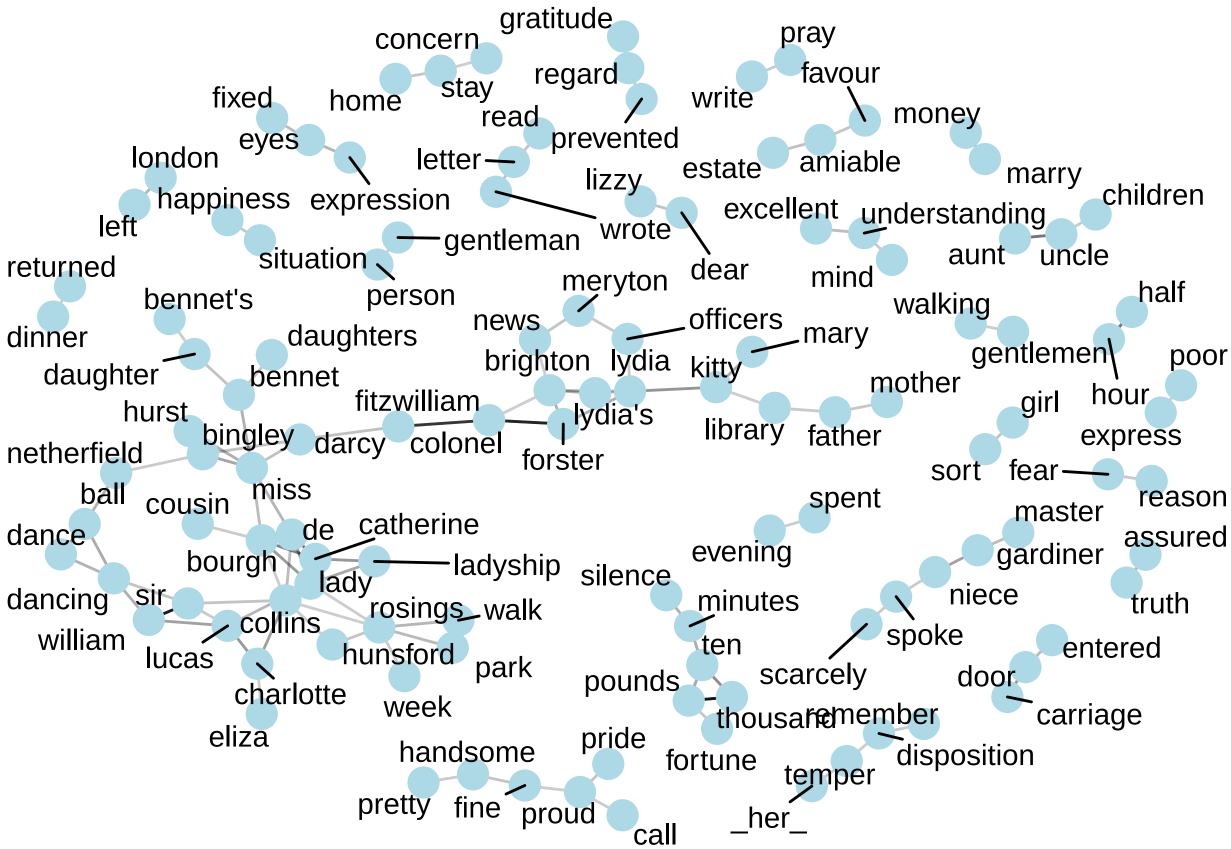
```
## Selecting by correlation
```



Just as I used ggraph to visualize bigrams, I can use it to visualize the correlations and clusters of words that were found by the widyr package.

```r
set.seed(2016)

word_cors %>%
  filter(correlation > .15) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(edge_alpha = correlation), show.legend = FALSE) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_void()
```



Unlike the bigram analysis, the relationships here are symmetrical, rather than directional (there are no arrows). I can also see that while pairings of names and titles that dominated bigram pairings are common, such as “colonel/fitzwilliam”, I can also see pairings of words that appear close to each other, such as “walk” and “park”, or “dance” and “ball”.

Summary

This project showed how the tidy text approach is useful not only for analyzing individual words, but also for exploring the relationships and connections between words. Such relationships can involve n-grams, which enable me to see what words tend to appear after others, or co-occurences and correlations, for words that appear in proximity to each other. Further, this project also demonstrated the ggraph package for visualizing both of these types of relationships as networks. These network visualizations are a flexible tool for exploring relationships.