Chapter4. Relationships between words: n-grams and correlations

Geonwoo Ban

2021 3 29

이번 장에서는 단어들간의 관계성을 계산하고 시각화하는 방법들을 소개하고 있다.

4.1 Tokenizing by n-gram

-앞선 chapter에서는 단어별로 토큰화를 한다거나 문장별로 토큰화를 하기 위해 `unnest_tokens()` 함수를 사용했었다. 이 함수는 정서분석이나 빈도분석에도 유용하게 사용되었음을 확인하였다. 이 함수를 사용해 ***n-grams***라고 하는 연속적인 단어 sequence로 토큰화할 수도 있다.

-단어 X에 단어 Y가 얼마나 자주 이어서 나오는지를 봄으로써 이 단어들 사이의 관계 모델을 구축할 수 있다.

- `unnest_tokens()` 에 `token = "ngrams"` 옵션을 추가하고, 각 엔그램에서 파악하려는 단어의 수를 n으로 설정하여 작업을 수행한다.

-n을 2로 설정하면 bigrams라고 부르기도 하는 두 개의 연속 단어 쌍을 검사해보자.

```
library(dplyr)
library(tidytext)
library(janeaustenr)
library(ggplot2)
library(forcats)

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

austen_bigrams
```

```
## # A tibble: 675,025 x 2
##    book                bigram
##    <fct>               <chr>
##  1 Sense & Sensibility sense and
##  2 Sense & Sensibility and sensibility
##  3 Sense & Sensibility <NA>
##  4 Sense & Sensibility by jane
##  5 Sense & Sensibility jane austen
##  6 Sense & Sensibility <NA>
##  7 Sense & Sensibility <NA>
##  8 Sense & Sensibility <NA>
##  9 Sense & Sensibility <NA>
## 10 Sense & Sensibility <NA>
## # ... with 675,015 more rows
```

- 이 데이터 구조는 tidy text structure의 형식이다.
- 각 행당 하나의 토큰으로 구성되어 있지만, 각 토큰은 이제 바이그램을 나타낸다.

4.1.1 Counting and filtering n-grams

-엔그램 개수를 세어 어떤 엔그램이 많이 반복되는지를 확인해보자.

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

```
## # A tibble: 193,210 x 2
##    bigram      n
##    <chr>   <int>
##  1 <NA>    12242
##  2 of the   2853
##  3 to be    2670
##  4 in the   2221
##  5 it was   1691
##  6 i am     1485
##  7 she had  1405
##  8 of her   1363
##  9 to the   1315
## 10 she was  1309
## # ... with 193,200 more rows
```

- 당연하게도 가장 많이 반복되는 단어쌍은 분석에 있어 의미가 없는 단어들로 구성되어 있음을 확인할 수 있다.
- 이러한 불용어들을 제거하기 위해서는 단어를 분리하여 따로 저장할 필요가 있다.
- tiryr의 `seperate()` 를 사용하면 구분 기호에 따라 1개 열을 여러 열로 분해하여 새로운 dataset을 만들 수 있다.
- 이렇게 분해한 dataset으로 분해한 열 중에 하나라도 불용어가 있는 경우 제거하는 방법을 사용하여 불용어를 제거한다.

```
library(tidyr)

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
```

```
## # A tibble: 28,975 x 3
##    word1   word2         n
##    <chr>   <chr>     <int>
##  1 <NA>    <NA>      12242
##  2 sir     thomas      266
##  3 miss    crawford    196
##  4 captain wentworth   143
##  5 miss    woodhouse   143
##  6 frank   churchill   114
##  7 lady    russell     110
##  8 sir     walter      108
##  9 lady    bertram     101
## 10 miss    fairfax      98
## # ... with 28,965 more rows
```

- 제인 오스틴의 책들에서 이름과 성 또는 경칭을 포함하는 단어들이 가장 흔한 쌍을 이룰는 것을 알 수 있다.
- tidyr의 `unite()` 함수는 `seperate()` 의 역함수로서 여러 열을 하나로 재결합한다.

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

bigrams_united
```

```
## # A tibble: 51,155 x 2
##    book                bigram
##    <fct>               <chr>
##  1 Sense & Sensibility NA NA
##  2 Sense & Sensibility jane austen
##  3 Sense & Sensibility NA NA
##  4 Sense & Sensibility NA NA
##  5 Sense & Sensibility NA NA
##  6 Sense & Sensibility NA NA
##  7 Sense & Sensibility NA NA
##  8 Sense & Sensibility NA NA
##  9 Sense & Sensibility chapter 1
## 10 Sense & Sensibility NA NA
## # ... with 51,145 more rows
```

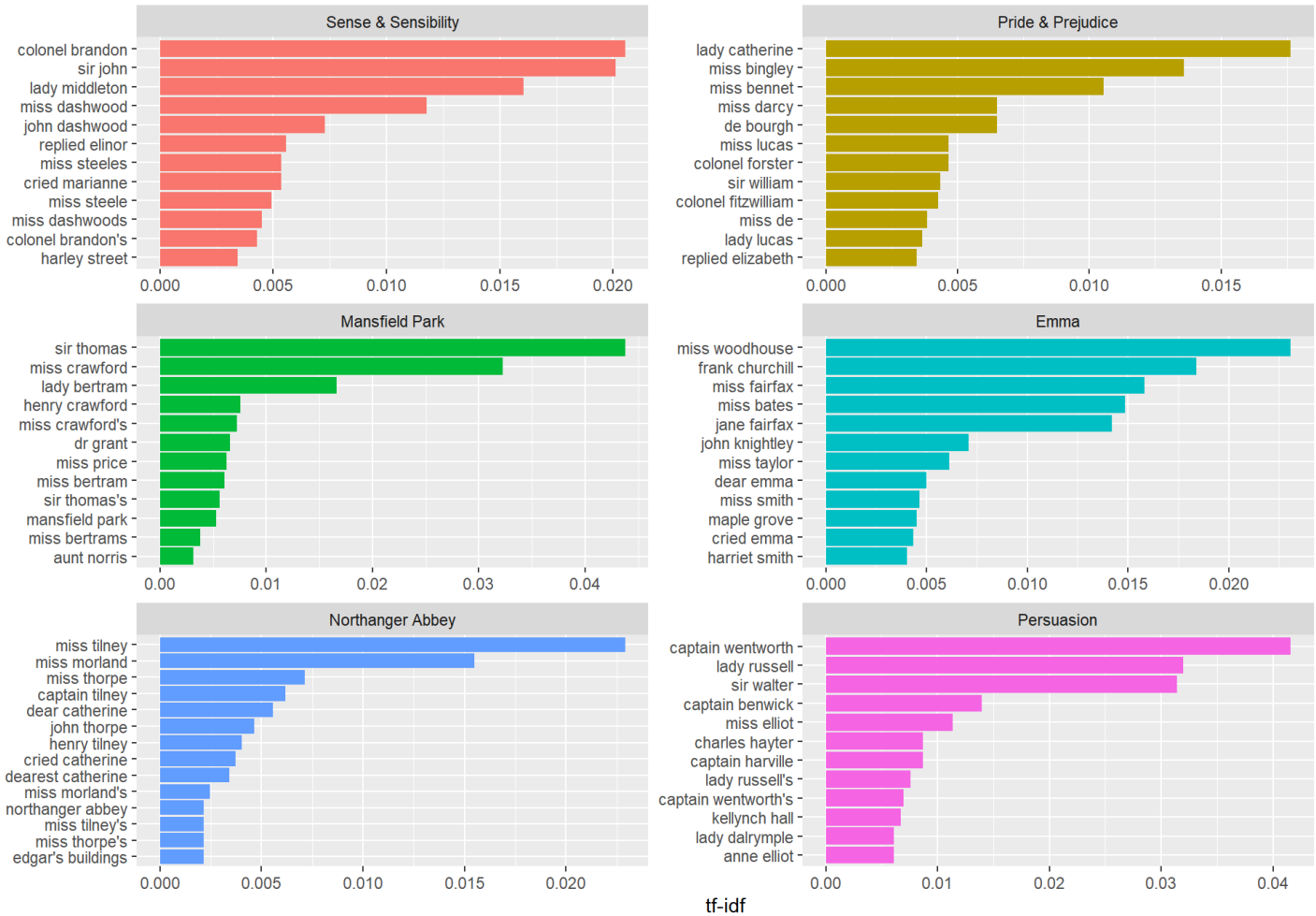
4.1.2 Analyzing bigrams

-1행당 1바이그램 형식은 텍스트의 탐색적 분석에 도움이 된다.

-바이그램은 앞서 다루었던 tf-idf값 또한 계산하여 시각화가 가능하다.

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

bigram_tf_idf %>%
  group_by(book) %>%
  slice_max(tf_idf, n = 12) %>%
  ungroup() %>%
  ggplot(aes(tf_idf, fct_reorder(bigram, tf_idf), fill = book)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~book, ncol = 2, scales = "free") +
  labs(x = "tf-idf", y = NULL)
```



- 앞서 개별 단어에 대한 tf-idf 값을 계산하여 정렬해본 결과, 대부분이 인물들의 이름으로 구성되어 있었다.
- 바이그램에서 tf-idf를 계산하여 정렬해보니 인물들 이름에 동사나 경칭, 성을 붙인 단어들이 높은 tf-idf값을 가짐을 확인할 수 있다.

- 이러한 개별 단어가 아닌 바이그램의 tf-idf를 보는 것에는 장단점이 존재한다.
    - 연속된 단어 쌍은 한 단어만 보았을 때는 존재하지 않는 구조를 파악할 수 있게 하며 토큰을 더 이해할 수 있게 하는 문맥 흐름을 제공한다.
    - 그러나 두 단어의 쌍은 각 한 단어의 쌍보다 더 드물게 나오기 때문에 각 바이그램의 개수 또한 sparse 하다.
    - 따라서 바이그램은 매우 큰 텍스트 데이터셋을 가지고 있을 때 더욱 유용할 수 있다.

4.1.3 Using bigrams to provide context in sentiment analysis

-Chap2에서 사용한 sentiment analysis는 단순히 참조용 lexicon에 맞춰 긍정 단어나 부정 단어의 빈도를 계산하였다.

-이러한 방법의 문제점 중 하나는 단어의 맥락이 단어의 존재 여부만큼이나 중요할 수 있다는 것이다.

-예를 들어 “happy”와 “like” 라는 단어는 “I’m not happy and I don’t like it”과 같은 문장에서 긍정 단어로 세어질 수 있다.

-바이그램으로 표현을 하면 단어 앞에 “not”과 같은 단어가 얼마나 자주 나오는지를 쉽게 알 수 있다.

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

```
## # A tibble: 1,178 x 3
##    word1 word2     n
##    <chr> <chr> <int>
##  1 not   be      580
##  2 not   to      335
##  3 not   have    307
##  4 not   know    237
##  5 not   a       184
##  6 not   think   162
##  7 not   been    151
##  8 not   the     135
##  9 not   at      126
## 10 not   in      110
## # ... with 1,168 more rows
```

- 바이그램 데이터에 대한 정서분석을 수행함으로써 정서와 연관된 단어 앞에 ‘not’ 또는 그 밖의 부정 단어가 얼마나 자주 나오는지 조사할 수 있다.
- 긍정 단어 앞에 나오는 부정 단어를 조사함으로써 긍정 단어가 정서 점수에 기여하는 정도를 무시하거나, 아예 점수를 반대 정서로 되돌리는 데 이용할 수 있다.
- 숫자로 sentiment를 표현하여 음수나 양수로 sentiment의 방향성을 알 수 있는 AFINN lexicon을 사용해 sentiment analysis를 시행해보자.

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

AFINN
```

```
## # A tibble: 2,477 x 2
##    word       value
##    <chr>      <dbl>
##  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 2,467 more rows
```

- 이제 단어의 앞에 'not'이 나오면서 sentiment와 관련이 있는, 가장 빈도수가 높은 단어를 찾아보자.

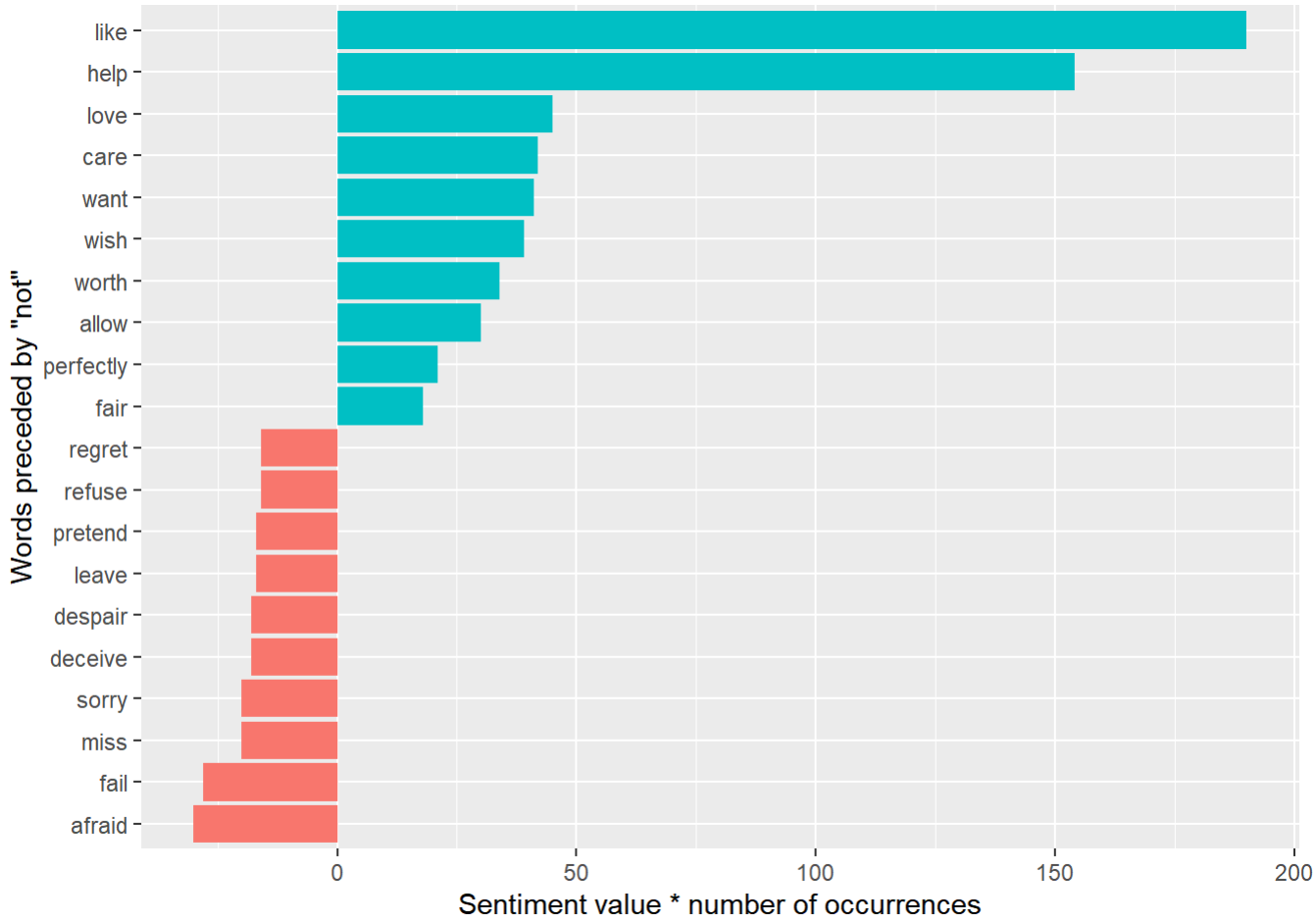
```
not_words <- bigrams_separated %>%
  filter(word1 == "not") %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word2, value, sort = TRUE)

not_words
```

```
## # A tibble: 229 x 3
##    word2   value     n
##    <chr>   <dbl> <int>
##  1 like        2    95
##  2 help        2    77
##  3 want        1    41
##  4 wish        1    39
##  5 allow       1    30
##  6 care        2    21
##  7 sorry      -1    20
##  8 leave      -1    17
##  9 pretend    -1    17
## 10 worth       2    17
## # ... with 219 more rows
```

- 앞에 'not'이 있으면서 가장 많이 나온 단어는 like로 AFINN lexicon을 사용하여 sentiment score를 계산하면 2점에 해당하는 단어이다.
- 각 단어들이 얼마나 많이 잘못된 sentiment로 계산되었는지 따져보기위해 출현 횟수 $\times$ sentiment score 를 계산하여 시각화해보자.

```
not_words %>%
  mutate(contribution = n * value) %>% # 기여도를 계산.
  arrange(desc(abs(contribution))) %>% # 기여도의 절댓값순으로 정렬
  head(20) %>% # 그중 20개만 뽑아서 사용.
  mutate(word2 = reorder(word2, contribution)) %>% # 기여도 순으로 reorder
  ggplot(aes(n * value, word2, fill = n * value > 0)) + # 긍정과 부정에 기여한 정도에 따라 fill 사
용.
  geom_col(show.legend = FALSE) +
  labs(x = "Sentiment value * number of occurrences",
       y = "Words preceded by ₩"not₩"")
```

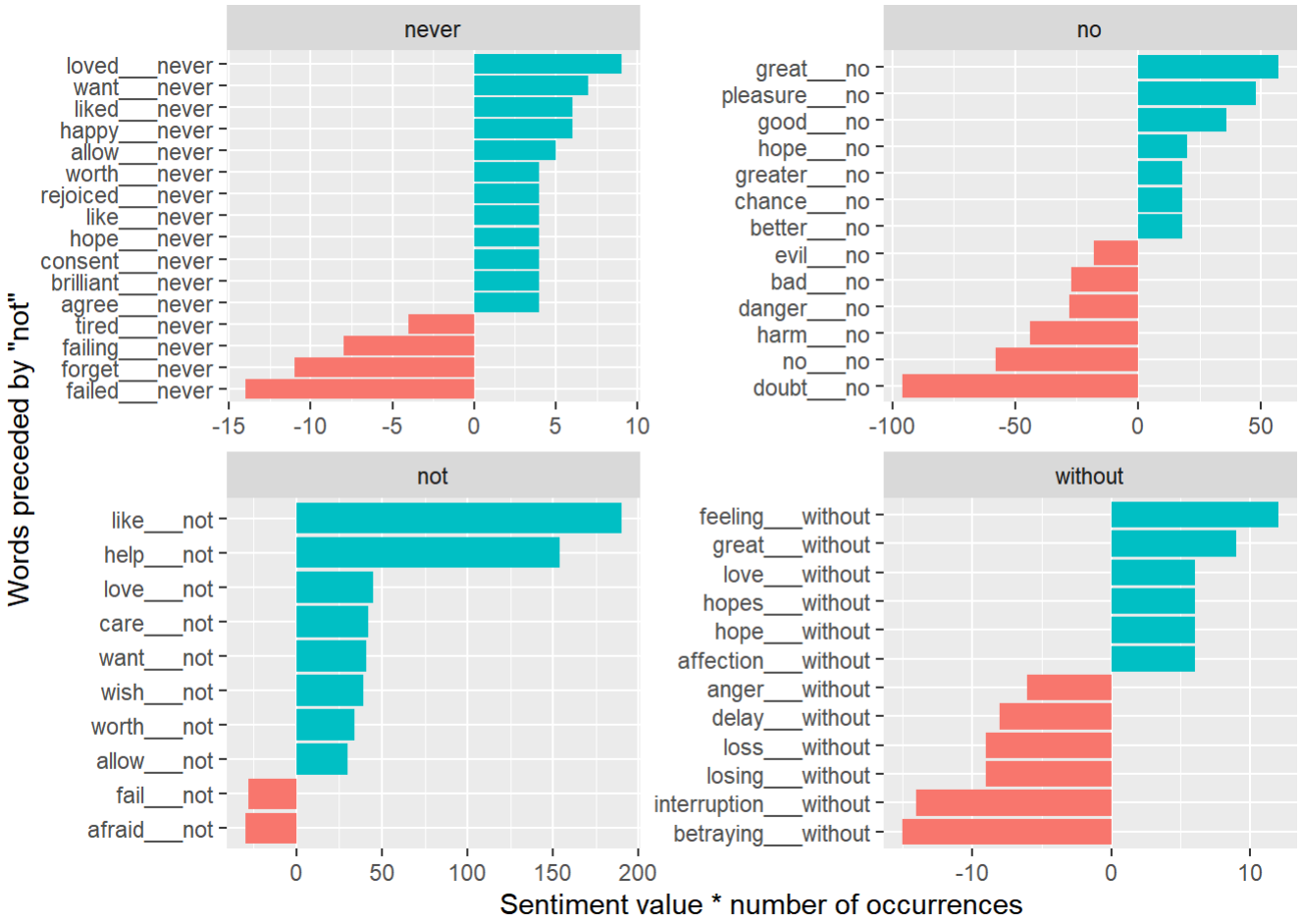


- 앞에 not이 나오는 단어들 중에서 긍정적인 방향으로든 부정적인 방향으로든 정서 점수에 가장 큰 영향을 끼친 20개 단어를 시각화한 것.
- “not like”나 “not help”라는 바이그램이 가장 많이 잘못된 sentiment에 영향을 주었음을 확인할 수 있다.
- “not afraid”나 “not fail”과 같은 구절은 텍스트가 의도된 의미보다 더 부정적이게 보이게 했다는 점 또한 확인할 수 있다.
- “not”이라는 단어 외에도 뒤에 나오는 단어들을 부정하는 네 가지 흔한 단어를 골라내어 한번에 검토해보자.

```
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, value, sort = TRUE) %>%
  ungroup()


negated_words %>%
  group_by(word1) %>%
  mutate(contribution = n * value) %>% # 기여도를 계산.
  top_n(10,abs(contribution)) %>%
  mutate(word2=reorder_within(word2, contribution, word1)) %>%
  ggplot(aes(x=contribution, y=word2, fill=contribution > 0)) + # 긍정과 부정에 기여한 정도에 따라 fill 사용.
  geom_col(show.legend = FALSE) +
  labs(x = "Sentiment value * number of occurrences",
       y = "Words preceded by W"notW"")+
  facet_wrap(~word1, scales='free')
```



- 각 부정어 별 가장 흔하게 나오는 후발단어를 확인할 수 있다.

4.1.4 Visualizing a network of bigrams with ggraph

-여러 단어 사이의 모든 관계를 동시에 시각화하는 방법 중 한 가지로 단어를 *network* 로, 즉 '그래프'로 정렬하는 방식을 들 수 있다.

-여러 단어간의 노드를 통해 다양한 조합을 사용하여 시각화할 수 있다.

- from : 노드가 나가는 정점
- to : 노드가 향하는 정점
- weight : 각 노드와 연관된 가중치

- `igraph` 패키지를 사용하여 시각화해보자.

```
library(igraph)

# original counts
bigram_counts
```

```
## # A tibble: 28,975 x 3
##    word1   word2         n
##    <chr>   <chr>     <int>
##  1 <NA>    <NA>      12242
##  2 sir     thomas      266
##  3 miss    crawford    196
##  4 captain wentworth   143
##  5 miss    woodhouse   143
##  6 frank   churchill   114
##  7 lady    russell     110
##  8 sir     walter      108
##  9 lady    bertram     101
## 10 miss    fairfax      98
## # ... with 28,965 more rows
```

```
# filter for only relatively common combinations
bigram_graph <- bigram_counts %>%
  filter(n > 20) %>%
  graph_from_data_frame() # 각 단어별 노드를 나타내주는 함수.

bigram_graph
```

```
## IGRAPH 521afc9 DN-- 86 71 --
## + attr: name (v/c), n (e/n)
## + edges from 521afc9 (vertex names):
##  [1] NA      ->NA         sir     ->thomas     miss    ->crawford
##  [4] captain ->wentworth  miss    ->woodhouse  frank   ->churchill
##  [7] lady    ->russell    sir     ->walter     lady    ->bertram
## [10] miss    ->fairfax    colonel ->brandon    sir     ->john
## [13] miss    ->bates      jane    ->fairfax    lady    ->catherine
## [16] lady    ->middleton  miss    ->tilney     miss    ->bingley
## [19] thousand->pounds     miss    ->dashwood   dear    ->miss
## [22] miss    ->bennet     miss    ->morland    captain ->benwick
## + ... omitted several edges
```

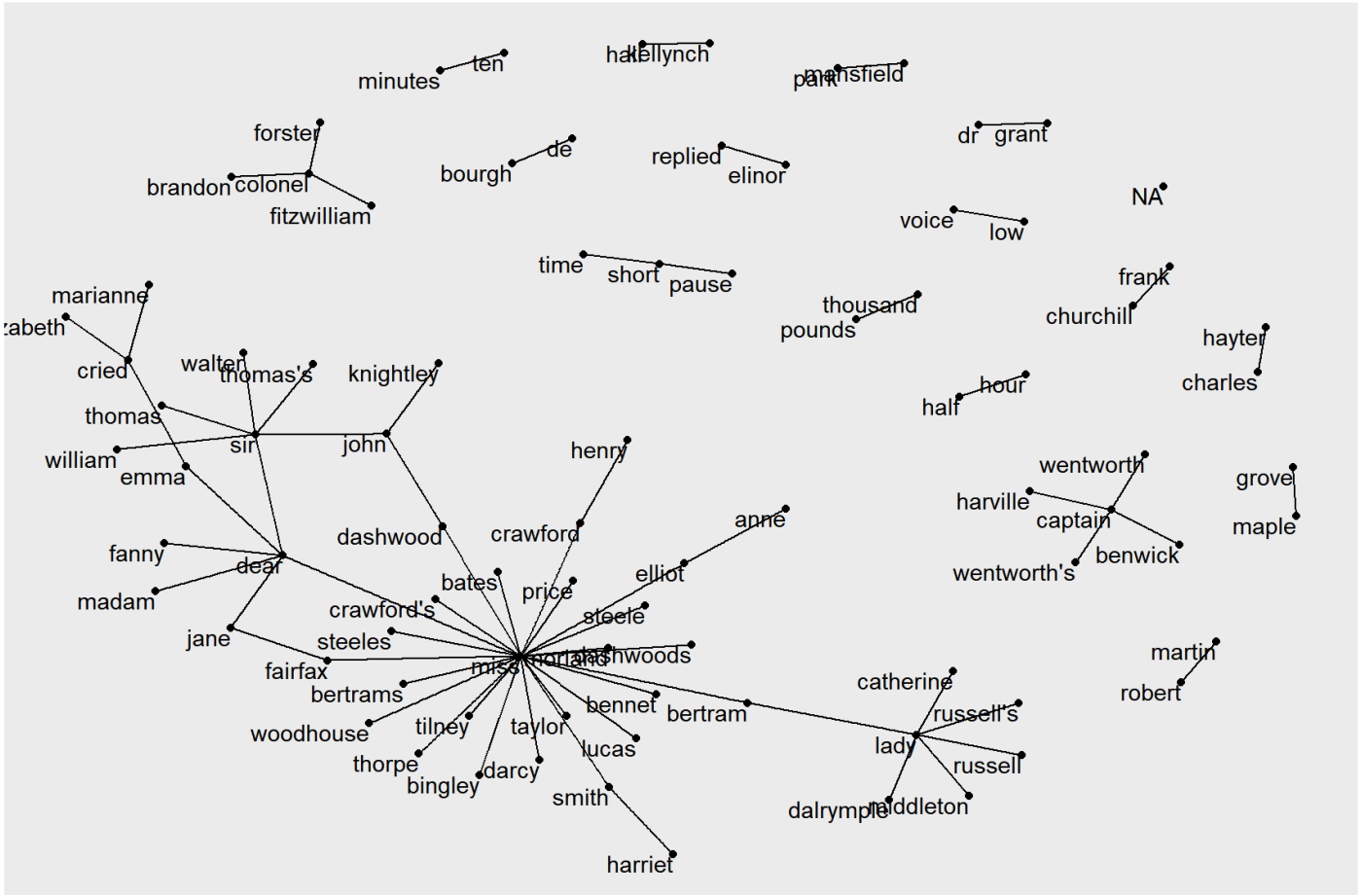
- 이렇게 노드로 단어를 연결시킨 데이터셋을 얻었으면, ggraph 라는 패키지의 ggraph() 함수를 사용하여 시각화를 할 수 있다.
- geom_edge_link() : 각 점을 찍어줌.
- geom_node_point() : 각 점별 노드를 연결.
- geom_node_text() : 텍스트 레이어

```
library(ggraph)
```

```
## Warning: package 'ggraph' was built under R version 4.0.4
```

```
set.seed(2021)

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```

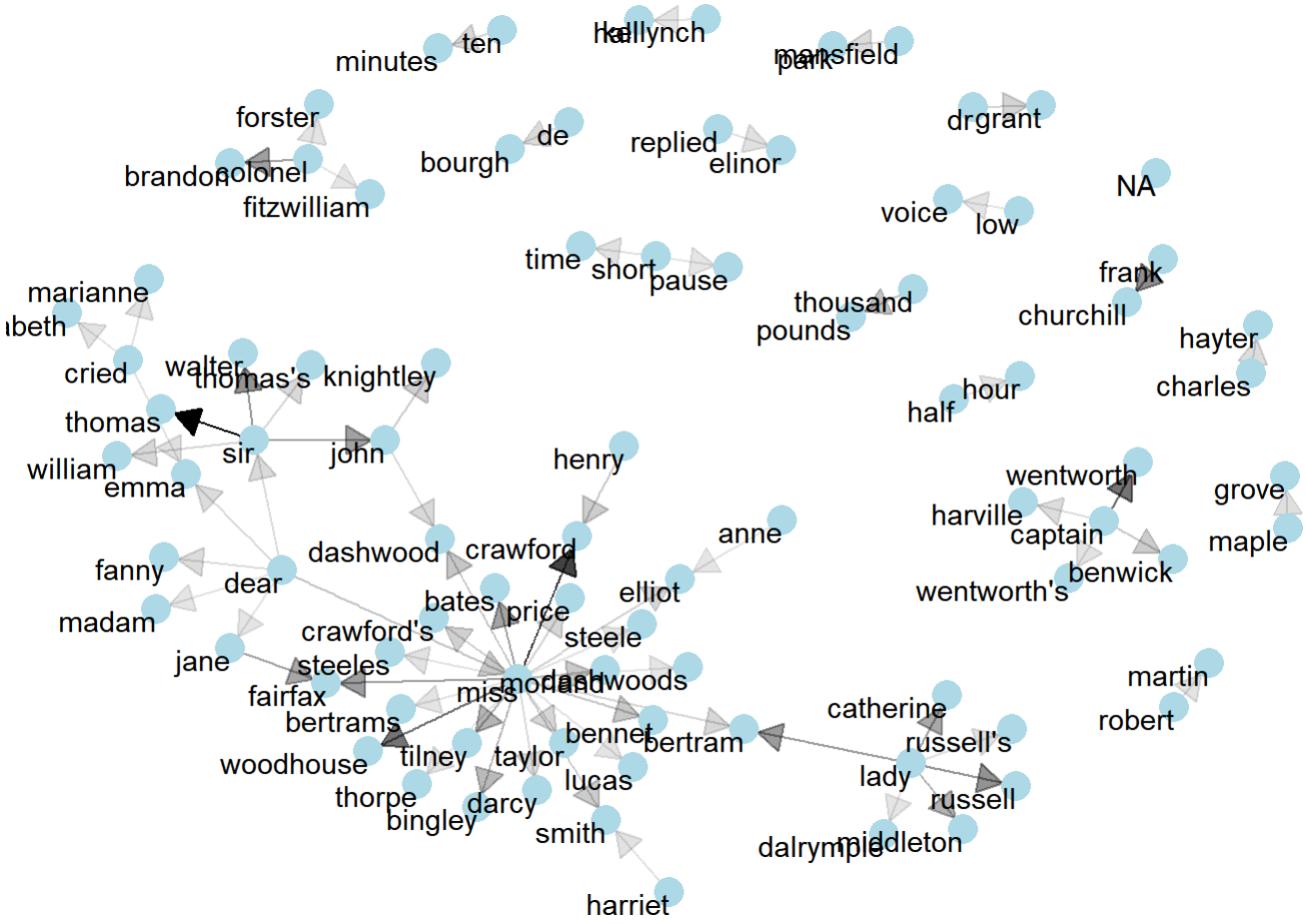


- 이 그림을 통해 텍스트 구조의 일부 세부 사항을 볼 수 있다.
- 예를 들어 “miss”, “lady”, “captain”, “sir”과 같은 경칭은 대부분의 단어 조합에 겹치는 단어로 공통 중심을 형성한다는 것을 알 수 있다.
- 추가적인 작업을 통해 그래프를 더 깔끔하게 그려보자.
- 먼저 link 레이어에 edge_alpha라는 옵션을 주어 희소한 단어일수록 투명하게 만든다.
- `grid::arrow()` 를 사용하여 화살표로 방향성을 추가하며, 화살표가 점에 닿기 전에 끝나게 하는 `end_cap` 옵션을 포함하였다.
-

```
set.seed(2021)

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()
```



- 위에서 시각화를 하기위해 사용한 텍스트 처리 모델은 **Markov chain**을 사용하였다.
- **Markov chain**에서 단어의 각 선택은 이전 단어에만 의존한다.
- 이러한 연결망 구조는 데이터간 관계를 시각화하는데에 유용한 시각화방법이다.

4.1.5 Visualizing bigrams in other texts

-앞서 바이그램으로 나누어 단어간의 관계까지 보는 과정을 함수화 시킨다면 소설책 뿐만 아니라 더 다양한 분야의 텍스트 분석에도 적용가능 할 것이다.

```
library(dplyr)
library(tidyr)
library(tidytext)
library(ggplot2)
library(igraph)
library(ggraph)

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(2021)
  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()
}
```

- `count_bigrams(dataset)` 은 텍스트 데이터를 바이그램으로 쪼개어 각 바이그램의 개수를 계산하는 함수이다.
- `visualize_bigrams` 은 바이그램으로 이루어진 데이터를 각 단어간의 관계성을 시각화하여 그림을 제공하는 함수이다.

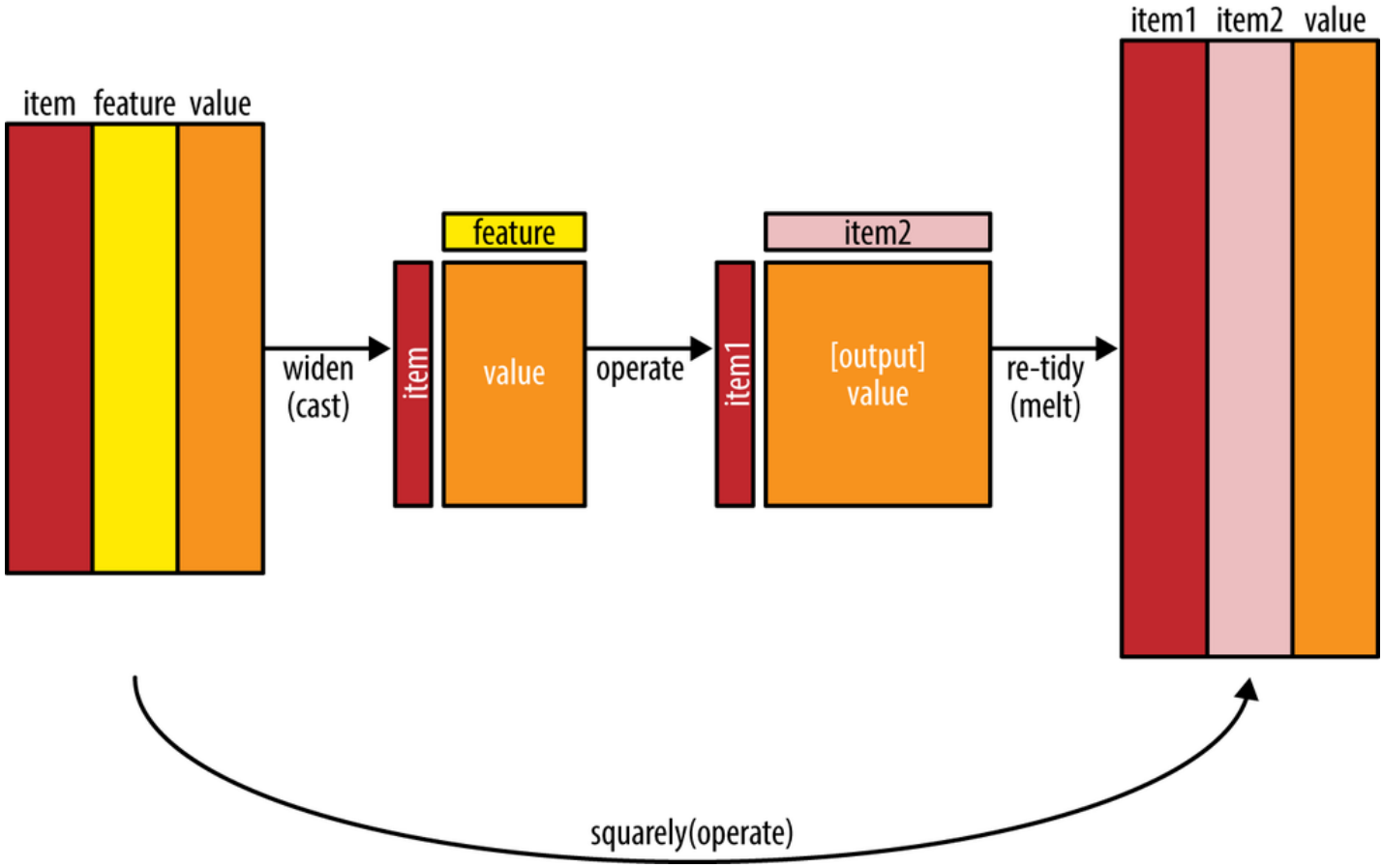
4.2 Counting and correlating pairs of words with the widyr package

-엔그램 토큰화는 인접한 단어들로 구성된 단어쌍을 찾는데에 유용한 방법이다.

-그렇지만 특정 단어들이 서로 연달아 나오지 않지만 해당 단어들이 동시에 출현하는 경우를 관심대상으로 삼을 수도 있다.

-두 단어가 같은 문서에 나온 횟수나 상관을 확인하기 위해서는 먼저 데이터를 **wide matrix**로 변환해야한다. (Chap4 에서는 필요없는 예제임. Chap5에서 다룸.)

- `widyr` 패키지는 '데이터 확장, 연산 수행, 데이터 정돈' 패턴을 단순화하여 단어의 횟수나 상관을 계산하는 작업을 쉽게 수행할 수 있게 해주는 패키지이다.



- widyr 패키지는 먼저 tidy structure dataset을 wide matrix로 변환하는 "casting"단계를 거친 후
- 상관과 같은 연산을 수행한 다음
- 결과를 다시 정의하여 re-tidy 과정을 통해 다시 정의한다.

4.2.1 Counting and correlating among sections

-2장의 sentiment analysis를 했던 방식처럼 각 책에 여러 section들로 나누어서 생각해보자.

-관심있는 단어가 어떤 단어와 서로 같은 단원에 나타나는지에 관심이 있을 수 있다.

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

austen_section_words # 단어쌍 계산을 위한 tidy text structure dataset
```

```
## # A tibble: 37,240 x 3
##    book              section word
##    <fct>               <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
```

- widyr의 함수들 중 `pairwise_count()` 를 사용하여 위에서 만든 tidy text structure dataset의 각 단어 쌍에 대해 하나의 행을 생성해볼 수 있다.
- 또한 이 함수를 사용하면 한 개 단원 안에 흔하게 나오는 단어 쌍을 셀 수 있다.

```
library(widyr)

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

word_pairs
```

```
## # A tibble: 796,008 x 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
```

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

```
## # A tibble: 2,930 x 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
```

4.2.2 Pairwise correlation

-“Elizabath”와 “Darcy” 같은 쌍은 가장 흔한 단어쌍이면서 가장 흔하게 나오는 독립 단어이기도 하기 때문에 두 단어를 보는 관점에서는 의미가 없다.

-대신 단어 사이의 상관성을 계산할 수 있다.

-상관성을 의미하는 지표로는 **phi coefficient**를 사용한다.

$$\phi = \frac{n_{11}n_{00} - n_{10}n_{01}}{\sqrt{n_{1\cdot}n_{0\cdot}n_{\cdot 0}n_{\cdot 1}}}$$

- `widyr` 의 `pairwise_cor()` 함수를 사용하면 동일한 절에 나타나는 빈도에 따라 단어 사이의 파이 계수를 찾을 수 있다.

```
# 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 x 3
##    item1     item2     correlation
##    <chr>     <chr>           <dbl>
##  1 bourgh    de              0.951
##  2 de        bourgh          0.951
##  3 pounds    thousand        0.701
##  4 thousand  pounds          0.701
##  5 william   sir             0.664
##  6 sir       william         0.664
##  7 catherine lady            0.663
##  8 lady      catherine       0.663
##  9 forster   colonel         0.622
## 10 colonel   forster         0.622
## # ... with 154,832 more rows
```

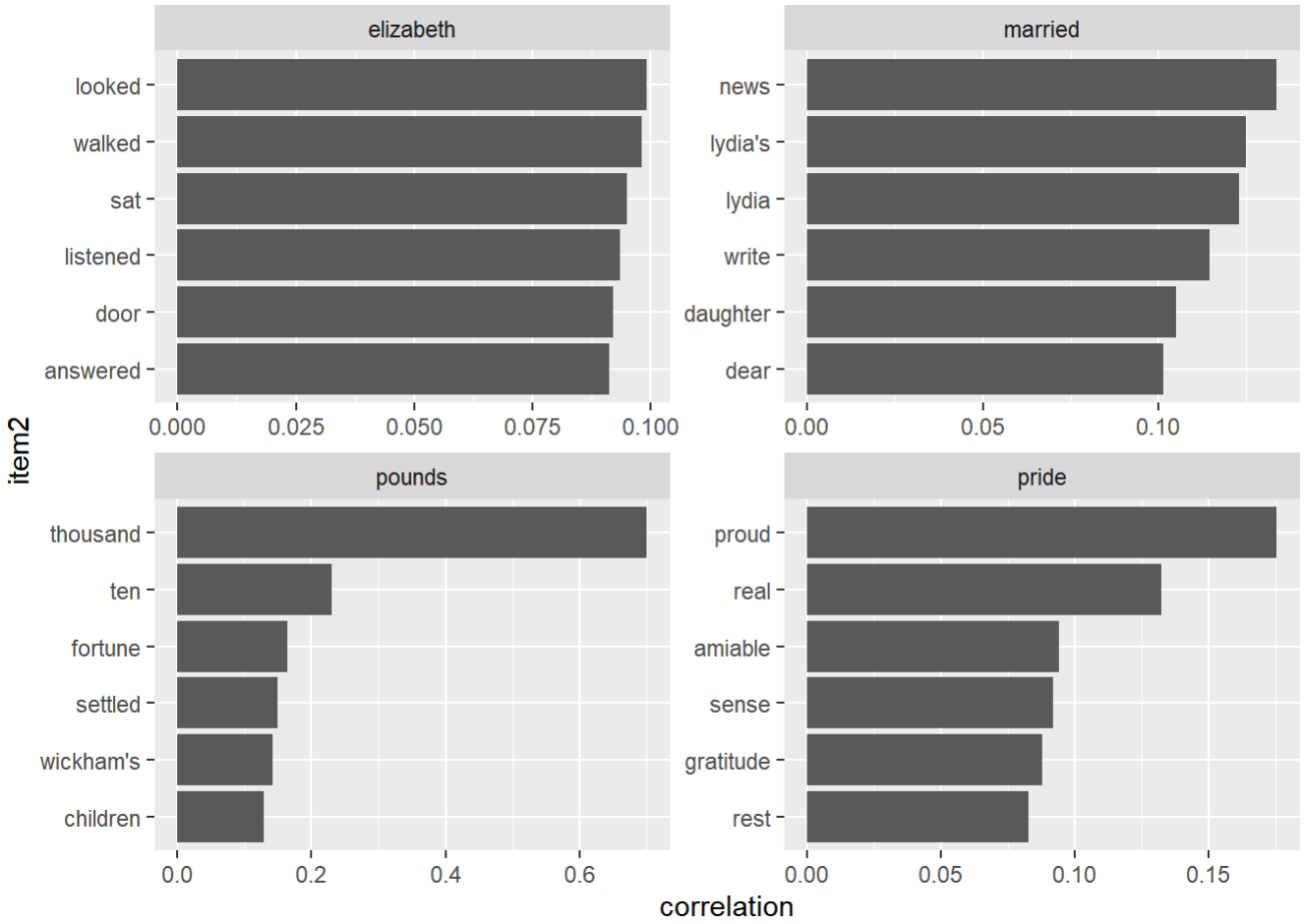
- correlation을 계산 한 후에 관심있는 단어에 대해 관련성이 높은 단어를 찾을 수 있다.

```
word_cors %>%
  filter(item1 == "pounds")
```

```
## # A tibble: 393 x 3
##    item1  item2     correlation
##    <chr>  <chr>           <dbl>
##  1 pounds thousand       0.701
##  2 pounds ten            0.231
##  3 pounds fortune        0.164
##  4 pounds settled        0.149
##  5 pounds wickham's      0.142
##  6 pounds children       0.129
##  7 pounds mother's       0.119
##  8 pounds believed       0.0932
##  9 pounds estate         0.0890
## 10 pounds ready          0.0860
## # ... with 383 more rows
```

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

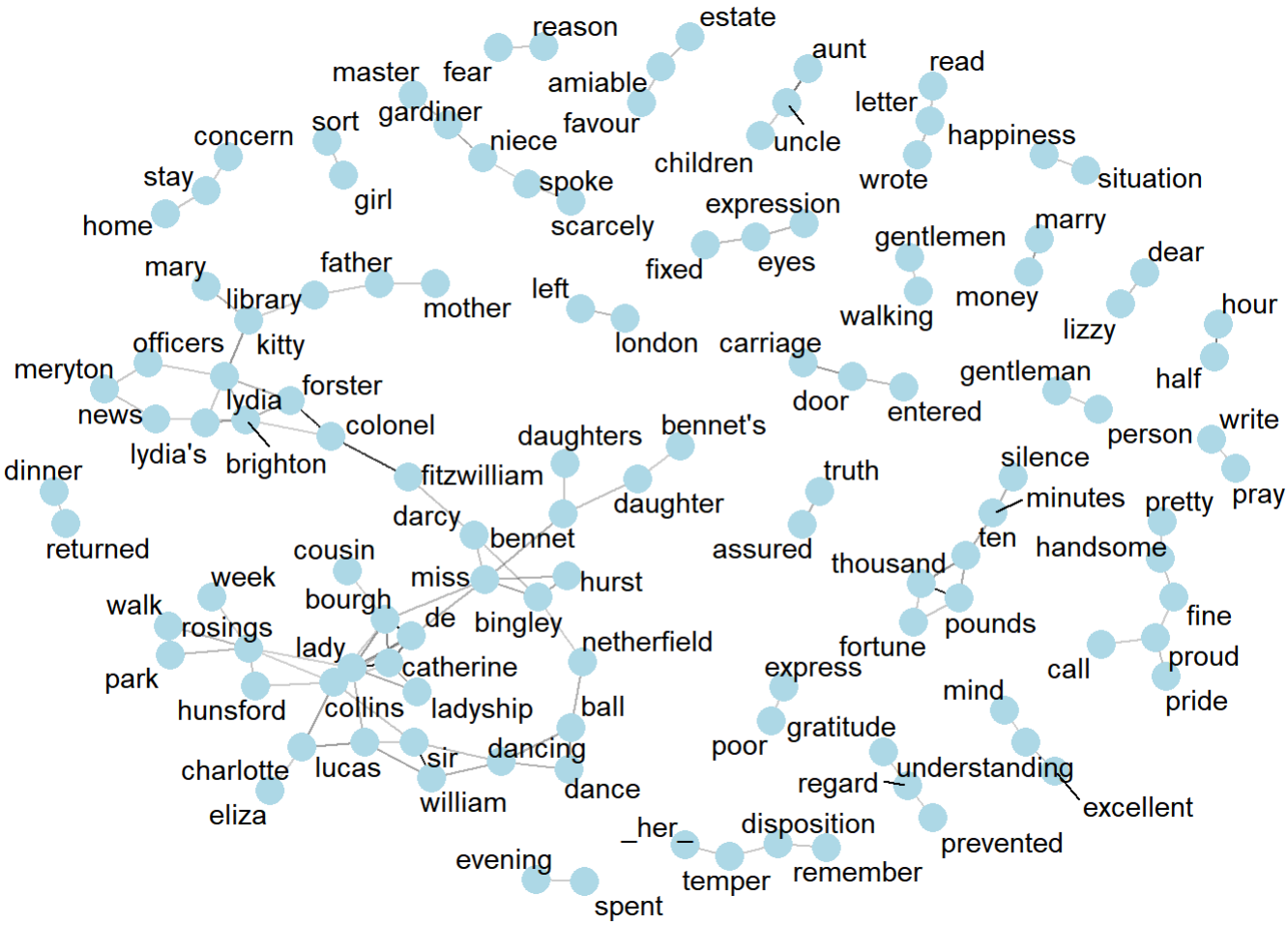
```
## Selecting by correlation
```



- 오만과 편견 속의 각 단어와 상관이 가장 높은 단어들에 대한 plot이다.

```
set.seed(2021)

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()
```



- 바이그램 분석과 달리 여기서의 관계는 방향성을 띄고있는 것이 아니라 상호관계적임을 주의해야한다.
- “colonel”이나 “fitzwilliam”과 같이 바이그램 짝 중에 거의 대부분을 차지하는 이름이나 제목의 짝이 흔하다는 점을 확인할 수 있고, 서로 인접해서 출현하는 단어의 조합도 확인할 수 있다.

Summary

-이번 장에서는 개별 단어를 분석하여 문장이나 책 한권에 적용하는 것이 아닌 단어간의 관계 및 연결을 탐색하는 방법들을 살펴보았다.

-관계를 보는 방법으로는 엔그램이 포함될 수 있으며, 이는 어떤 단어가 다른 단어 뒤에 나타나는 경향이 있는지와, 서로 인접한 곳에 출현하는 단어 간의 상관을 볼 수 있게 하는 방법임을 확인할 수 있었다.

-이러한 관계를 연결망으로 시각화하여 확인할 수 있었으며, 이러한 연결망 시각화는 관계에 대해 관심있게 볼때에 쓰기 적절한 도구임을 알 수 있었다.