Topic 2: The NLP Pipeline

Taylor Arnold[1] Lauren Tilton[2]

04 July 2017

[1] Assistant Professor of Statistics
University of Richmond
@statsmaths

[2] Assistant Professor of Digital Humanities
University of Richmond
@nolauren

Setup

Packages

To run the following code, make sure you have loaded (and installed) the following packages. We like to set **ggplot2** to use the minimal theme, but this is of course entirely optional.

```
library(cleanNLP)
library(dplyr)
library(readr)
library(stringi)
library(ggplot2)
theme_set(theme_minimal())
```

Annotation engines

Back ends

We have been able to get some real, interesting results by splitting our raw text into tokens. Some clever filtering and use of external datasets has gotten us some rough results in terms of character identification and the detection of themes.

To go deeper though, we need a more advanced natural language processing engine. These extract more granular features of the text, such as identifying parts of speech and tagging particular known entities.

Back end, cont.

In **cleanNLP**, we currently provide back ends to two of the most well-known such libraries:

- **spaCy** a Python library primarily built for speed and stability
- **CoreNLP** a Java library built to have bleeding-edge functionality

Initialising back ends

To use one of these backends in **cleanNLP**, simply run an alternative `init_` function before annotating the text. Either use:

```
library(cleanNLP)
init_spaCy(model_name = "en")
anno <- run_annotators(paths)
nlp <- get_combine(anno)
```

Or:

```
library(cleanNLP)
init_coreNLP(language = "en")
anno <- run_annotators(paths)
nlp <- get_combine(anno)
```

Annotation results

The resulting data set `nlp` also has one row per token, but now there are many additional features that have been learned from the text:

```
## # A tibble: 551,463 x 15
##      id   sid   tid     word    lemma  upos   pos   cid source
##   <int> <int> <int>    <chr>    <chr> <chr> <chr> <int>  <int>
## 1     1     1     2       To       to   ADP    IN     1      0
## 2     1     1     3 Sherlock sherlock PROPN   NNP     4      4
## 3     1     1     4   Holmes   holmes PROPN   NNP    13      2
## 4     1     1     5      she   -PRON-  PRON   PRP    20      6
## 5     1     1     6       is       be  VERB   VBZ    24      2
## 6     1     1     7   always   always   ADV    RB    27      6
## # ... with 5.515e+05 more rows, and 6 more variables:
## #   relation <chr>, word_source <chr>, lemma_source <chr>,
## #   entity_type <chr>, entity <chr>, spaces <int>
```

Annotation tasks

NLP backends use models to learn features about the words and sentences in our raw text. Common tasks include:

- tokenisation
- lemmatisation
- sentence boundaries
- part of speech tags
- dependencies
- named entities
- coreferences
- sentiment analysis
- word embeddings

A collection of these running together (as they typically need to), is known as an **NLP Pipeline**. We will explain the meaning behind and some applications of many of these annotation tasks in these slides.

Back end details

- options passed to the `init_` functions control which models and annotations are selected
- models have to be trained specifically for every natural language that they support
- more complex annotation tasks need to be trained seperately for different styles of speech (i.e., Twitter versus Newspapers)
- libraries needed for these types of annotations require large external dependencies in order to run correctly
- in the interest of time, today we will simply provide the annotation objects for our corpora of study.

More detailed instructions for setting up either back end can be found on the cleanNLP repository and we are happy to help as best we can during the break or after the tutorial.

Reading data

As mentioned above, we have already run the spaCy annotators on the corpus of Sherlock Holmes stories and made them available in the GitHub repository:

```
paths <- dir("data/holmes_stories", full.names = TRUE)
sh_meta <- data_frame(id = seq_along(paths),
                      story = stri_sub(basename(paths), 4, -5))
sh_nlp <- read_csv("data/sh_nlp.csv.gz")
```

Sentence Boundaries

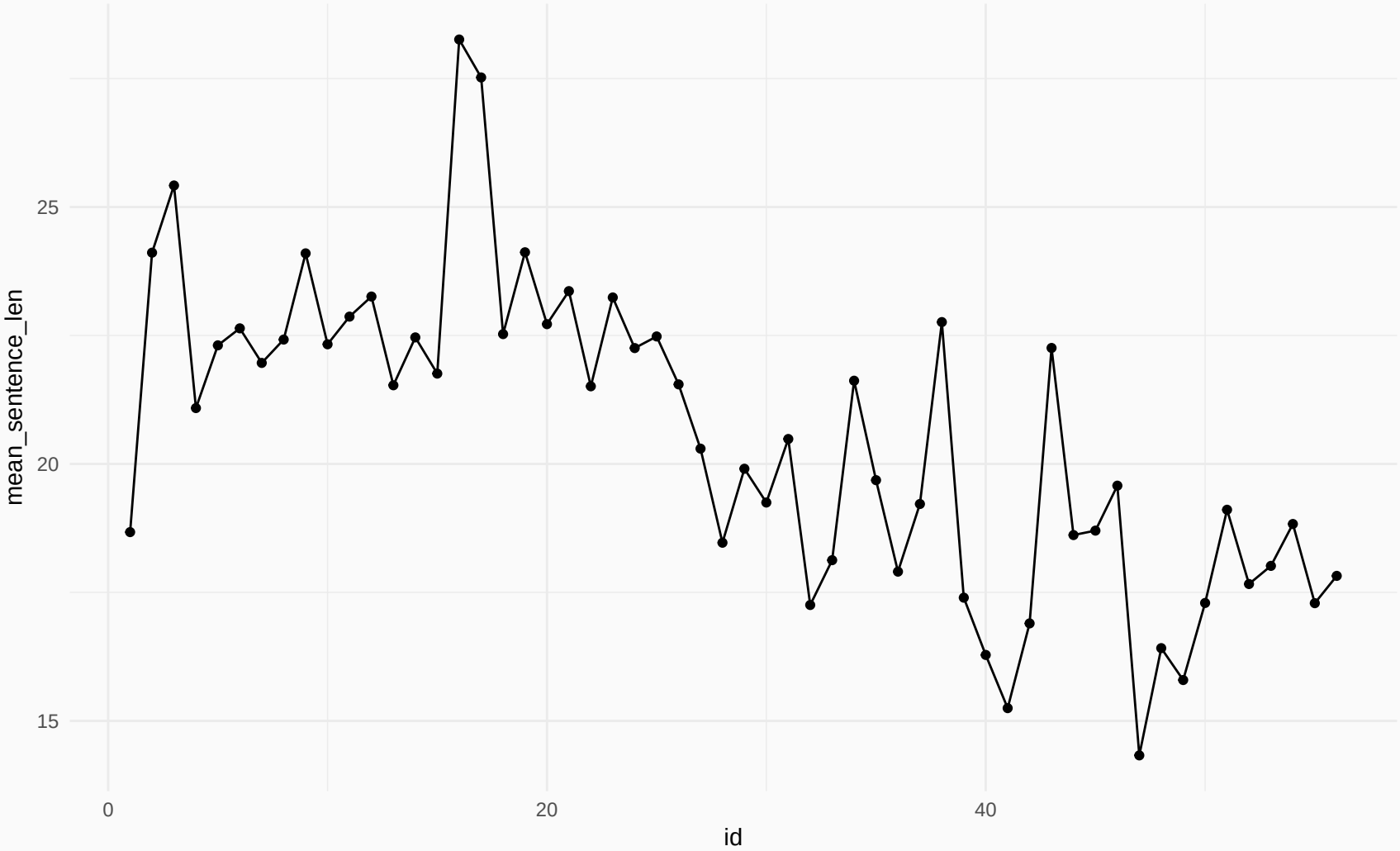
More accurate average sentence length

By using the sentence boundaries learned by the NLP pipeline, we can more accurately count the average length of the sentences in each text.

```
sh_nlp %>%
  group_by(id, sid) %>%
  mutate(sentence_end = tid == max(tid)) %>%
  group_by(id) %>%
  summarize(mean_sentence_len = n() / sum(sentence_end)) %>%
  ggplot(aes(id, mean_sentence_len)) +
    geom_line() +
    geom_point()
```

Errors that might occur in our original method primarily include abbrevations such as “Dr.” and “S.O.S.”.

More accurate average sentence length



Lemmatisation

Lemmas

The most simple new column is the one titled `lemma`. This contains a reduced form of the token, for example converting all verbs into the same tense and all nouns into the singular case.

Lemmas, examples

```
sh_nlp %>% filter(tolower(word) != lemma) %>%
  select(word, lemma) %>% print(n = 10)
```

```
## # A tibble: 134,920 x 2
##        word   lemma
##       <chr>   <chr>
##  1      she  -PRON-
##  2       is      be
##  3        I  -PRON-
##  4    heard    hear
##  5      him  -PRON-
##  6      her  -PRON-
##  7      his  -PRON-
##  8     eyes     eye
##  9      she  -PRON-
## 10 eclipses eclipse
## # ... with 1.349e+05 more rows
```

Using lemmas

While minor, this assists with the topic discovery we did in the previous session by using the lemma frequency rather than the word frequency.

```
sh_lemmafr <- sh_nlp %>%
  left_join(word_frequency, by = c("lemma" = "word")) %>%
  filter(!is.na(frequency)) %>%
  filter(frequency < 0.01) %>%
  filter((tolower(word) == word)) %>%
  count(id, lemma) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id") %>%
  arrange(id, desc(n))
```

Using lemmas

```
sh_lemmafr %>% filter(id == 1) %>% print(n = 12)
```

```
## # A tibble: 0 x 4
## # ... with 4 variables: id <int>, lemma <chr>, n <int>, story <chr>
```

Part of Speech Tags

Using POS tags

Many of the tricks we used in the last session revolved around finding ways to approximate part of speech tags:

- stop words list, for example, removes (amongst other things) punctuation marks, pronouns, conjunctions, and interjections
- checking for upper case marks is really a hunt to identify proper nouns
- the frequency table is largely trying to remove verbs (there are far fewer of these and they tend to be more common), as well as common nouns

Proper part of speech tags can let us do these things more accurately as well as make other types of analysis possible.

POS granularity

In primary or secondary school, you probably learned about a dozen or so parts of speech. These include nouns, verbs, adjectives, and so forth. Linguists in fact identify a far more granular set of part of speech tags, and even amongst themselves do not agree on a fixed set of such tags.

A commonly used one, and the one implemented by spaCy, are the Penn Treebank codes. These are given in our dataset under the `pos` variable.

Penn Treebank

Table 2
The Penn Treebank POS tagset.

1. CC	Coordinating conjunction	25. TO	*to*
2. CD	Cardinal number	26. UH	Interjection
3. DT	Determiner	27. VB	Verb, base form
4. EX	Existential *there*	28. VBD	Verb, past tense
5. FW	Foreign word	29. VBG	Verb, gerund/present participle
6. IN	Preposition/subordinating conjunction	30. VBN	Verb, past participle
7. JJ	Adjective	31. VBP	Verb, non-3rd ps. sing. present
8. JJR	Adjective, comparative	32. VBZ	Verb, 3rd ps. sing. present
9. JJS	Adjective, superlative	33. WDT	*wh*-determiner
10. LS	List item marker	34. WP	*wh*-pronoun
11. MD	Modal	35. WP$	Possessive *wh*-pronoun
12. NN	Noun, singular or mass	36. WRB	*wh*-adverb
13. NNS	Noun, plural	37. #	Pound sign
14. NNP	Proper noun, singular	38. $	Dollar sign
15. NNPS	Proper noun, plural	39. .	Sentence-final punctuation
16. PDT	Predeterminer	40. ,	Comma
17. POS	Possessive ending	41. :	Colon, semi-colon
18. PRP	Personal pronoun	42. (	Left bracket character
19. PP$	Possessive pronoun	43.)	Right bracket character
20. RB	Adverb	44. "	Straight double quote
21. RBR	Adverb, comparative	45. ‘	Left open single quote
22. RBS	Adverb, superlative	46. “	Left open double quote
23. RP	Particle	47. ’	Right close single quote
24. SYM	Symbol (mathematical or scientific)	48. ”	Right close double quote

Universal part of speech

Work has also been done to map these granular codes to language-agnostic codes known as universal parts of speech. Coincidentally, these universal parts of speech mimic those commonly taught in schools:

- *VERB*: verbs (all tenses and modes)
- *NOUN*: nouns (common and proper)
- *PRON*: pronouns
- *ADJ*: adjectives
- *ADV*: adverbs
- *ADP*: adpositions (prepositions and postpositions)
- *CONJ*: conjunctions
- *DET*: determiners
- *NUM*: cardinal numbers
- *PRT*: particles or other function words
- *X*: other: foreign words, typos, abbreviations
- .: punctuation

These are contained in the variable `upos`, and for today will be the most useful for our analysis.

Top characters, again

Here, for example, is the analysis of key characters with our trick replaced by filtering on the proper noun tag “PROPN”:

```
sh_topchar <- sh_nlp %>%
  filter(upos == "PROPN") %>%
  count(id, word) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id") %>%
  arrange(id, desc(n))
```

Top characters, again

```
sh_topchar %>% filter(id == 1) %>% print(n = Inf)
```

```
## # A tibble: 0 x 4
## # ... with 4 variables: id <int>, word <chr>, n <int>, story <chr>
```

Compound words (optional)

A major shortcoming in our tabulation of proper nouns is that many of the proper nouns, in fact most in this case, are actually compound words. The proper way to analyse this data would be to collapse the compound words into a single combined token. It is relatively easy to do this in a slow way with loops. A fast, vectorized method with **dplyr** verbs is show in the code chunk below:

```
sh_compound <- sh_nlp %>%
  filter(upos == "PROPN") %>%
  group_by(id, sid) %>%
  mutate(d = tid - lag(tid) - 1) %>%
  mutate(d = ifelse(is.na(d), 1, d)) %>%
  ungroup() %>%
  mutate(d = cumsum(d)) %>%
  group_by(d) %>%
  summarize(id = first(id), sid = first(sid),
            tid = first(tid),
            thing = stri_c(word, collapse = " ")) %>%
  select(-d) %>%
  inner_join(sh_nlp, by = c("id", "sid", "tid"))
```

Compound words (optional)

```
sh_compound %>% select(id, thing) %>% print(n = 10)
```

```
## # A tibble: 12,793 x 2
##       id           thing
##    <int>           <chr>
##  1     1 Sherlock Holmes
##  2     1     Irene Adler
##  3     1     Irene Adler
##  4     1          Holmes
##  5     1          Holmes
##  6     1    Baker Street
##  7     1          Odessa
##  8     1         Trepoff
##  9     1        Atkinson
## 10     1     Trincomalee
## # ... with 1.278e+04 more rows
```

Named Entity Recognition

Entities

The task of finding characters, places, and other references to proper objects is common enough that it has been wrapped up into a specific annotation task known as named entity recognition (NER). Here are the first few entities from the annotation of our stories

```
results <- sh_nlp %>%
  select(id, entity, entity_type) %>%
  filter(!is.na(entity))
```

Entities, cont.

```
results %>% print(n = 10)
```

```
## # A tibble: 18,939 x 3
##       id           entity entity_type
##    <int>            <chr>       <chr>
##  1     1 Sherlock Holmes      PERSON
##  2     1     Irene Adler      PERSON
##  3     1             one    CARDINAL
##  4     1            Grit         FAC
##  5     1             one    CARDINAL
##  6     1             one    CARDINAL
##  7     1     Irene Adler      PERSON
##  8     1          Holmes      PERSON
##  9     1           first     ORDINAL
## 10     1          Holmes      PERSON
## # ... with 1.893e+04 more rows
```

NER characters

One benefit of this is that NER distinguishes between people and places, making our tabulation even more accurate:

```
sh_nerchar <- sh_nlp %>%
  select(id, entity, entity_type) %>%
  filter(!is.na(entity)) %>%
  filter(entity_type == "PERSON") %>%
  count(id, entity) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id") %>%
  arrange(id, desc(n))
```

```
sh_nerchar <- ungroup(sh_nerchar)
```

NER characters, cont.

```
sh_nerchar %>% filter(id == 1) %>% print(n = Inf)
```

```
## # A tibble: 0 x 4
## # ... with 4 variables: id <int>, entity <chr>, n <int>,
## #   story <chr>
```

Other entity categories

There are many other categories of named entities available within the spaCy and CoreNLP libraries, including:

- *ORGA*: Companies, agencies, institutions, etc.
- *MONEY*: Monetary values, including unit.
- *PERCENT*: Percentages.
- *DATE*: Absolute or relative dates or periods.
- *TIME*: Times smaller than a day.
- *NORP*: Nationalities or religious or political groups.
- *FACILITY*: Buildings, airports, highways, bridges, etc.
- *GPE*: Countries, cities, states.
- *LOC*: Non-GPE locations, mountain ranges, bodies of water.
- *PRODUCT*: Objects, vehicles, foods, etc. (Not services.)
- *EVENT*: Named hurricanes, battles, wars, sports events, etc.
- *WORK_OF_ART*: Titles of books, songs, etc.
- *LANGUAGE*: Any named language.
- *QUANTITY*: Measurements, as of weight or distance.
- *ORDINAL*: “first”, “second”, etc.
- *CARDINAL*: Numerals that do not fall under another type.

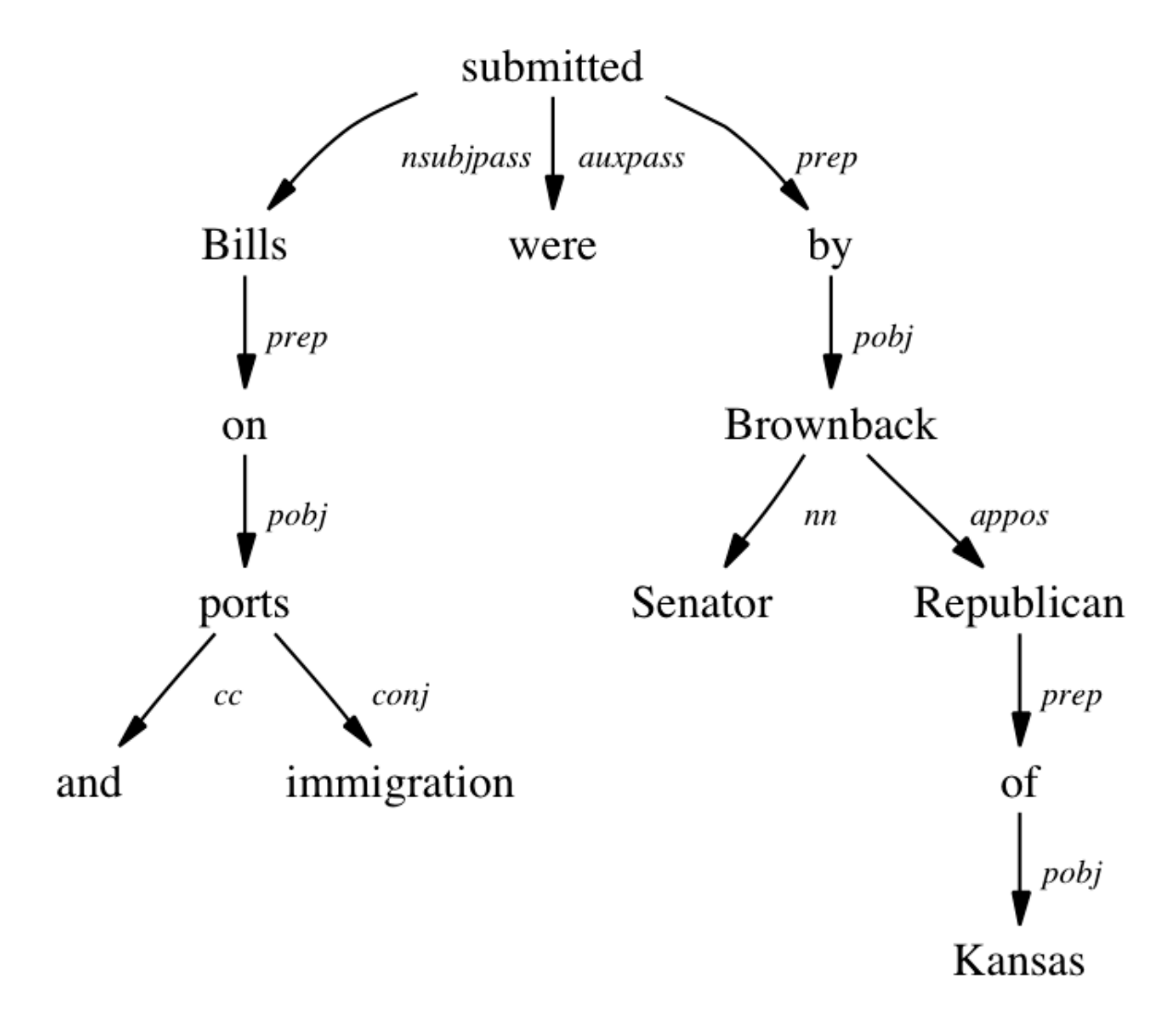
Dependency Parsing

Dependencies

Dependencies are links between tokens within a sentence that indicate grammatical relationships.

For example, they link adjectives to the nouns they describe and adverbs to the verbs they modify. One of the most common dependencies is the direct object tag “dobj”, linking a verb to the noun that receives the action of the verb.

Fully parsed sentence



Dependencies, example

```
sh_nlp %>% filter(id == 1, sid == 1) %>%
  select(word, source, relation, word_source)
```

```
## # A tibble: 9 x 4
##       word source relation word_source
##      <chr>  <int>    <chr>       <chr>
## 1       To      0     ROOT        ROOT
## 2 Sherlock      4 compound      Holmes
## 3   Holmes      2     pobj          To
## 4      she      6    nsubj          is
## 5       is      2    ccomp          To
## 6   always      6   advmod          is
## # ... with 3 more rows
```

What are characters doing?

One way that dependencies can be useful is by determining which verbs are associated with each character by way of the 'nsubj' relation. Amongst other things, this can help identify sentiment, biases, and power dynamics.

In our corpus, we can use the 'nsubj' tag to identify verbs associated with our main characters:

```
sh_whatchar <- sh_nlp %>%
  filter(relation == "nsubj") %>%
  filter(upos == "PROPN") %>%
  count(id, word, lemma_source) %>%
  filter(n > 1)
```

What are characters doing?

```
sh_whatchar %>% print(n = 12)
```

```
## # A tibble: 344 x 4
##       id          word lemma_source     n
##    <int>         <chr>        <chr> <int>
##  1     1       Holmes       murmur     2
##  2     1       Holmes          say     8
##  3     2       Holmes       remark     2
##  4     2       Holmes          say    11
##  5     2           I.          say     2
##  6     2 Merryweather           be     2
##  7     2         Ross           be     3
##  8     2    Spaulding          say     2
##  9     2       Wilson           be     2
## 10     2       Wilson          say     4
## 11     3        Angel         come     2
## 12     3       Holmes       remark     2
## # ... with 332 more rows
```