Topic 1: Exploratory Analysis of Tokenized Text

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

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)
library(tokenizers)
theme_set(theme_minimal())
```

Tokenisation

Splitting text into words

Consider a string in R containing the first paragraph of text from the novel *L'Étranger* of Albert Camus (we'll use `stri_wrap` just to fit the output on the slide):

```
stri_wrap(letranger)
```

```
## [1] "Aujourd'hui, maman est morte. Ou peut-être hier, je ne"
## [2] "sais pas.J'ai reçu un télégramme de l'asile: «Mère décédée."
## [3] "Enterrement demain.Sentiments distingués.» Cela ne veut rien"
## [4] "dire. C'était peut-êtrehier."
```

Splitting text into words

Consider a string in R containing the first paragraph of text from the novel *L'Étranger* of Albert Camus (we'll use `stri_wrap` just to fit the output on the slide):

```
stri_wrap(letranger)
```

```
## [1] "Aujourd'hui, maman est morte. Ou peut-être hier, je ne"
## [2] "sais pas.J'ai reçu un télégramme de l'asile: «Mère décédée."
## [3] "Enterrement demain.Sentiments distingués.» Cela ne veut rien"
## [4] "dire. C'était peut-êtrehier."
```

In order to work with this text, a good first step is to split it apart into its constituent words.

Splitting with whitespace

Splitting on whitespace alone works reasonably well, though there are some issues with punctuation marks:

```
stri_split(letranger, fixed = " ")
```

```
## [[1]]
##  [1] "Aujourd'hui,"      "maman"              "est"
##  [4] "morte."            "Ou"                 "peut-être"
##  [7] "hier,"             "je"                 "ne"
## [10] "sais"              "pas.J'ai"           "reçu"
## [13] "un"                "télégramme"         "de"
## [16] "l'asile:"          "«Mère"              "décédée."
## [19] "Enterrement"       "demain.Sentiments"  "distingués.»"
## [22] "Cela"              "ne"                 "veut"
## [25] "rien"              "dire."              "C'était"
## [28] "peut-êtrehier."
```

Splitting with cleanNLP

There are a number of packages that support the more complex logic needed to deal with many of these errors. Here we'll use the **cleanNLP** package as it will be easy to adapt our approach to work with more complex annotators in the next section.

Running annotations

We start by initialising the tokenizers back end within the **cleanNLP** package. We'll indicate that we want a French locale as this input text is in French.

```
library(cleanNLP)
init_tokenizers(locale = "fr")
```

Running annotations

We start by initialising the tokenizers back end within the **cleanNLP** package. We'll indicate that we want a French locale as this input text is in French.

```
library(cleanNLP)
init_tokenizers(locale = "fr")
```

Then, we run the annotators over the text. We set the option `as_strings` because we are passing the text into the function as a raw string:

```
letranger_anno <- run_annotators(letranger, as_strings = TRUE)
letranger_anno
```

```
##
## A CleanNLP Annotation:
##   num. documents: 1
```

An annotation object

The result seems to be wrapped up in a fairly complex object; however, it is nothing more than a list of data frames. To collapse all of these lists into a one table summary of the tokenisation process, we will call the function `get_combine` on the annotation object:

```
letranger_tokens <- get_combine(letranger_anno)
```

An annotation object

The result is a data frame with one row for each token. Meta data about each token, such as the sentence number and character offset, are included as columns.

```
letranger_tokens
```

```
## # A tibble: 46 x 6
##       id   sid   tid        word   cid spaces
##    <int> <int> <int>       <chr> <int>  <dbl>
## 1      1     1     1 Aujourd'hui     1      0
## 2      1     1     2           ,    12      1
## 3      1     1     4       maman    14      1
## 4      1     1     6         est    20      1
## 5      1     1     8       morte    24      0
## 6      1     1     9           .    29      1
## # ... with 40 more rows
```

cleanNLP tokenization results

Notice that the resulting tokens fix most of the problems in the original white space based technique:

```
letranger_tokens$word
```

```
##  [1] "Aujourd'hui" ","           "maman"       "est"
##  [5] "morte"       "."           "Ou"          "peut"
##  [9] "-"           "être"        "hier"        ","
## [13] "je"          "ne"          "sais"        "pas"
## [17] "."           "J'ai"        "reçu"        "un"
## [21] "télégramme"  "de"          "l'asile"     ":"
## [25] "«"           "Mère"        "décédée"     "."
## [29] "Enterrement" "demain"      "."           "Sentiments"
## [33] "distingués"  "."           "»"           "Cela"
## [37] "ne"          "veut"        "rien"        "dire"
## [41] "."           "C'était"     "peut"        "-"
## [45] "êtrehier"    "."
```

Corpus Metadata

A dataset where each record is its own text is known as a *corpus*. In the remainder of this session, we will be working with a corpus of the 56 short stories featuring Sherlock Holmes.

We start by constructing a meta data table of these stories:

```
paths <- dir("data/holmes_stories", full.names = TRUE)
sh_meta <- data_frame(id = seq_along(paths),
                      story = stri_sub(basename(paths), 4, -5))
sh_meta %>% print(n = 5)
```

```
## # A tibble: 56 x 2
##      id                        story
##   <int>                        <chr>
## 1     1         a_scandal_in_bohemia
## 2     2         the_redheaded_league
## 3     3           a_case_of_identity
## 4     4 the_boscombe_valley_mystery
## 5     5         the_five_orange_pips
## # ... with 51 more rows
```

Annotating files on disk

We want to construct a similar data frame of tokens for all of the short stories in our corpus. As a first step, we will re-initalise the tokenizers backend using an English locale:

```
library(cleanNLP)
init_tokenizers(locale = "en_GB")
```

Then, we call the annotation engine with the paths to the files instead of the raw text:

```
sh_anno <- run_annotators(paths)
```

And once again, collapse the object into a single table.

```
sh_tokens <- get_combine(sh_anno)
```

Sherlock Holmes tokens

The resulting table, as before, has one row for each token in the original dataset.

```
sh_tokens
```

```
## # A tibble: 550,699 x 6
##      id   sid   tid     word   cid spaces
##   <int> <int> <int>    <chr> <int>  <dbl>
## 1     1     1     2       To     2      1
## 2     1     1     4 Sherlock     5      1
## 3     1     1     6   Holmes    14      1
## 4     1     1     8      she    21      1
## 5     1     1    10       is    25      1
## 6     1     1    12   always    28      1
## # ... with 5.507e+05 more rows
```

Sherlock Holmes tokens, sentence 10

```
sh_tokens %>% filter(id == 1) %>% filter(sid == 10) %>% print(n = 12)
```

```
## # A tibble: 35 x 6
##       id   sid   tid        word   cid spaces
##    <int> <int> <int>       <chr> <int>  <dbl>
##  1     1    10     1        Grit   875      1
##  2     1    10     3          in   880      1
##  3     1    10     5           a   883      1
##  4     1    10     7   sensitive   885      1
##  5     1    10     9  instrument   895      0
##  6     1    10    10           ,   905      1
##  7     1    10    12          or   907      1
##  8     1    10    14           a   910      1
##  9     1    10    16       crack   912      1
## 10     1    10    18          in   918      1
## 11     1    10    20         one   921      1
## 12     1    10    22          of   925      1
## # ... with 23 more rows
```

Tokens and style

Using tokens

One can often learn a lot about a corpus by simply finding the occurances of certain tokens or patterns of tokens within it.

For example:

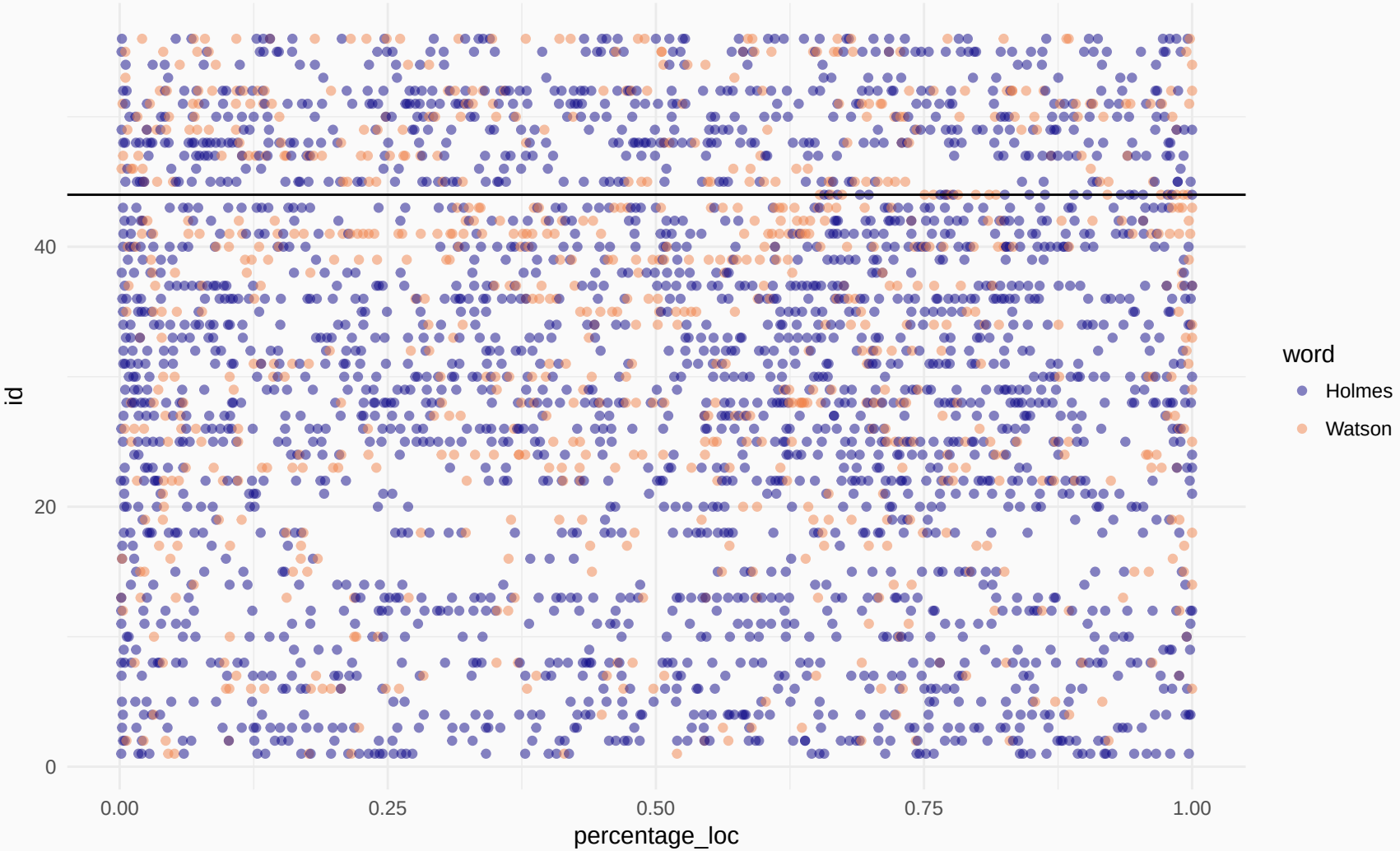
- length of sentences
- number of citations
- presence of known characters
- count of hashtags in a tweet
- ratio of quotes/dialogue to raw text

Visualising Watson and Holmes

Where do Watson and Holmes occur within each text?

```
sh_tokens %>%
  group_by(id) %>%
  mutate(percentage_loc = sid / max(sid)) %>%
  filter(word %in% c("Watson", "Holmes")) %>%
  ggplot(aes(percentage_loc, id)) +
    geom_point(aes(color = word), alpha = 0.5) +
    geom_hline(yintercept = 44)
```

Visualising Watson and Holmes

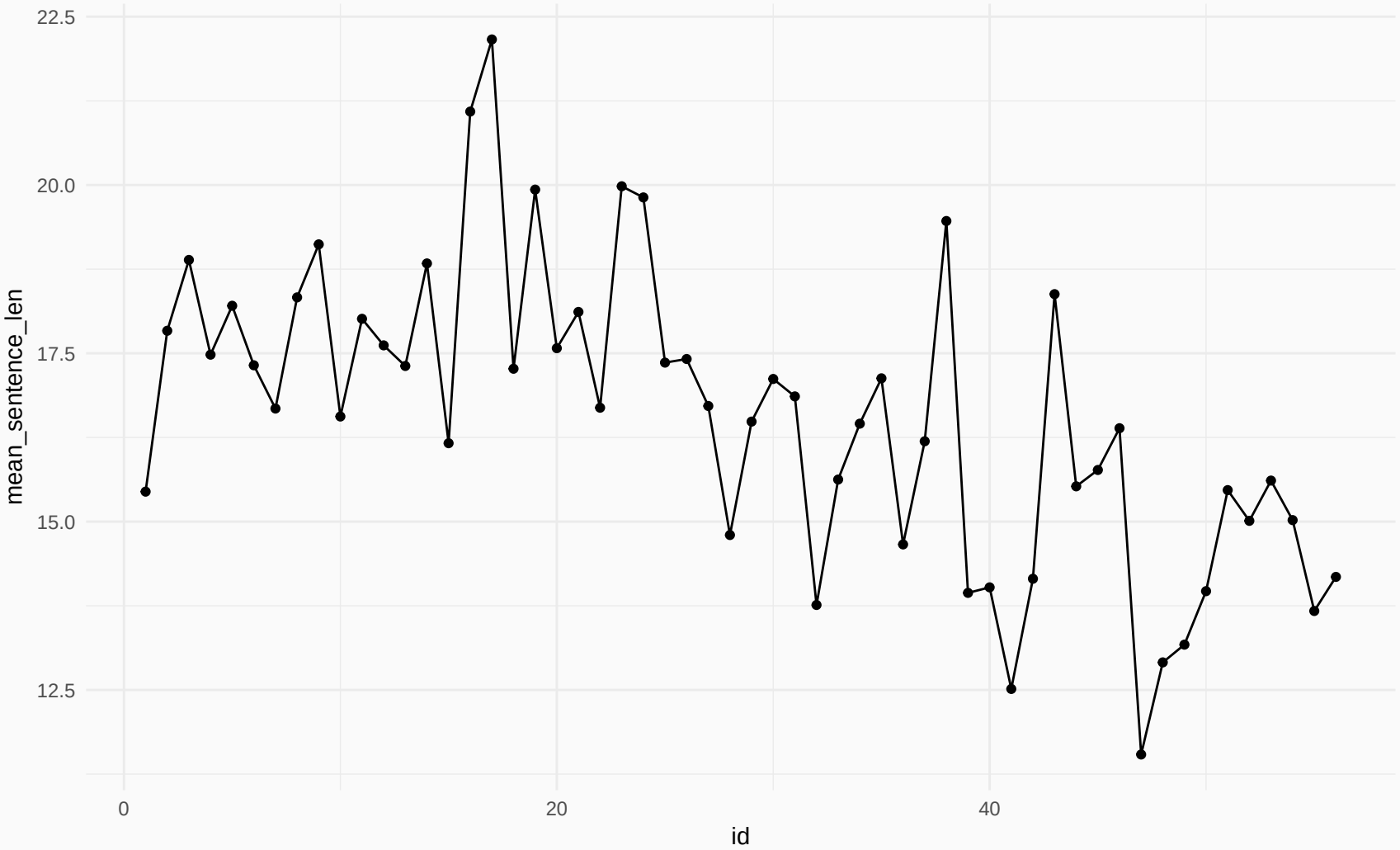


Average sentence length

By counting the number of periods, question marks, and exclamation marks, we can approximate the average length of each sentence in the text.

```
sh_tokens %>%
  mutate(sentence_end = word %in% c(".", "?", "!")) %>%
  group_by(id) %>%
  summarize(mean_sentence_len = n() / sum(sentence_end)) %>%
  ggplot(aes(id, mean_sentence_len)) +
    geom_line() +
    geom_point()
```

Average sentence length



Most frequent tokens

Frequency tokens

In our first set of analyses, we used our prior knowledge to determine which tokens would be interesting to identify and tabulate.

Alternatively, we can let the corpus itself tell us which terms would be the most interesting. Often just knowing which terms are the most frequent, or occur in certain patterns, is interesting itself.

Naïve approach

To begin, we can see what the most common words are across the corpus using the count function:

```
sh_tokens %>%
  count(id, word, sort = TRUE)
```

```
## # A tibble: 105,461 x 3
##       id  word      n
##    <int> <chr> <int>
## 1    22      ,   827
## 2    28      .   823
## 3    37      ,   794
## 4    40      .   778
## 5    22      .   770
## 6    28      ,   765
## # ... with 1.055e+05 more rows
```

Stop words

What is interesting about these top terms? Cynically, we might say very little. The problem is that the most common terms are simply punctuation marks.

If we look farther down the list, common function words such as 'the' and 'my' dominate the counts.

```
## # A tibble: 6 x 3
##      id  word      n
##   <int> <chr> <int>
## 1     1     I   259
## 2     1    to   242
## 3     1    of   235
## 4     1   and   227
## 5     1     a   212
## 6     1    in   152
```

A popular way of dealing with this problem is to define a list of *stop words*, those tokens that are common enough to be thematically uninteresting.

Stop words

We have included a simple list of stop words in the dataset for today, which you should read in with the following.

```
stopwords <- readLines("data/stopwords_en.txt")
sample(stopwords, 25L)
```

```
##  [1] "he'd"     "with"     "few"      "enough"   "away"
##  [6] "long"     "many"     "does"     "older"    "they've"
## [11] "he'll"    "put"      "s"        "n"        "done"
## [16] "was"      "asking"   "perhaps"  "works"    "clearly"
## [21] "take"     "she'd"    "seemed"   "k"        "together"
```

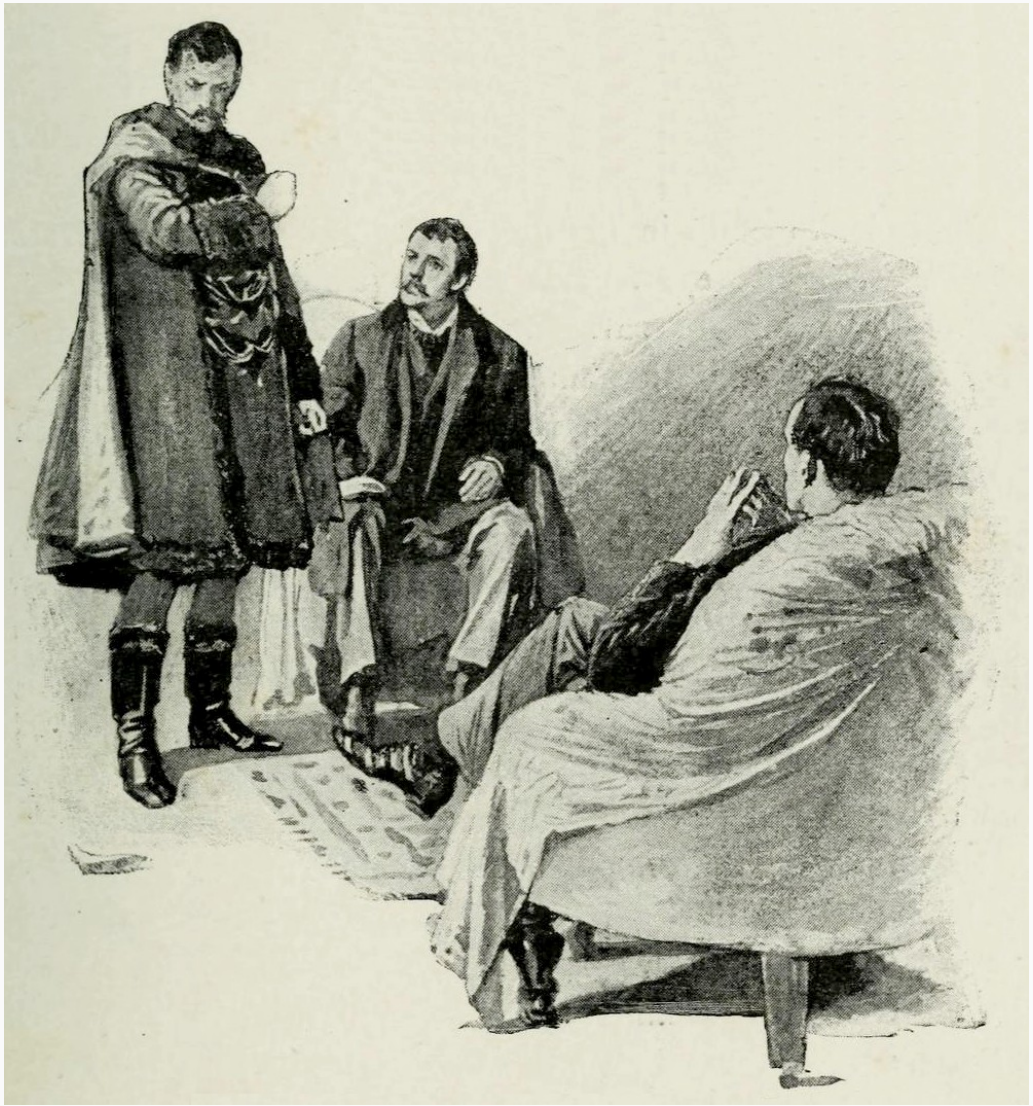
Take a moment to look at some of the words. What parts of speech dominate the list?

Most common non-stopwords

We will make use of the `top_n` function to select just the top 10 occurrences within each text. We also add a call to `left_join` to explicitly add the names of each story to the output:

```
sh_toptokens <- sh_tokens %>%
  filter(!(tolower(word) %in% stopwords)) %>%
  count(id, word, sort = TRUE) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id")
```

‘A Scandal in Bohemia’



Tokens from 'A Scandal in Bohemia'

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

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

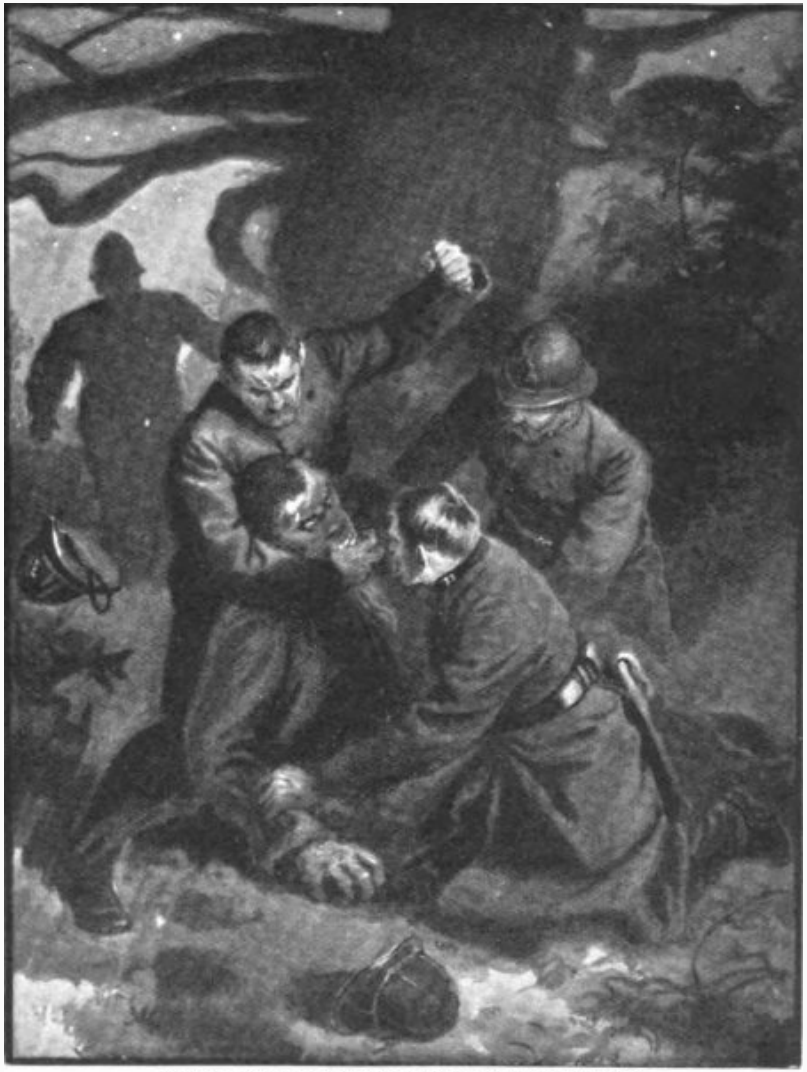
‘The Musgrave Ritual’

Tokens from 'The Musgrave Ritual'

```
sh_toptokens %>% filter(id == 17) %>% print(n = Inf)
```

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

‘His Last Bow’



“THE MAN WALKED INTO THE TRAP AND WAS CAPTURED.”

Tokens from ‘His Last Bow’

```
sh_toptokens %>% filter(id == 44) %>% print(n = Inf)
```

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

Extracting and Locating Characters

Why characters?

One particular category that floats to the top of our lists of most frequent tokens are the main characters for each story. Identifying the people mentioned in a corpus of text has many applications, including:

- on social media it indicates trending issues
- in news articles, the people mentioned give a good clue as to what topics are being discussed (politics, food, culture, local events, ..)
- in fiction, as we have seen, the presence and absence of characters is a major indicator of plot arcs

Proper nouns

Some of the most frequenct non stop words in the texts refer to the names of the characters. How might we extract these directly?

```
sh_propn <- sh_tokens %>%
  filter(!(tolower(word) %in% stopwords)) %>%
  filter((tolower(word) != word)) %>%
  count(id, word, sort = TRUE) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id")
```

Proper nouns from 'A Scandal in Bohemia'

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

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

Proper nouns from ‘The Musgrave Ritual’

```
sh_propn %>% filter(id == 17) %>% print(n = Inf)
```

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

Proper nouns from ‘His Last Bow’

```
sh_propn %>% filter(id == 44) %>% print(n = Inf)
```

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

One top character

Of course, two major characters are always going to be Sherlock Holmes and John Watson. Let us remove them from the list, as well as any names shorter than four characters (these are usually honorific rather than names). We will then take the most mentioned character from each text:

```
holmes_watson <- c("Sherlock", "Holmes", "John", "Watson")
sh_topchar <- sh_tokens %>%
  filter(stri_length(word) > 4) %>%
  filter(!(word %in% holmes_watson)) %>%
  filter(!(tolower(word) %in% stopwords)) %>%
  filter((tolower(word) != word)) %>%
  count(id, word) %>%
  left_join(sh_meta, by = "id") %>%
  top_n(n = 1, n)
```

One top character, cont.

```
sh_topchar %>% filter(id %in% c(1, 12, 17, 45)) %>%
  print(n = Inf)
```

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

Top name

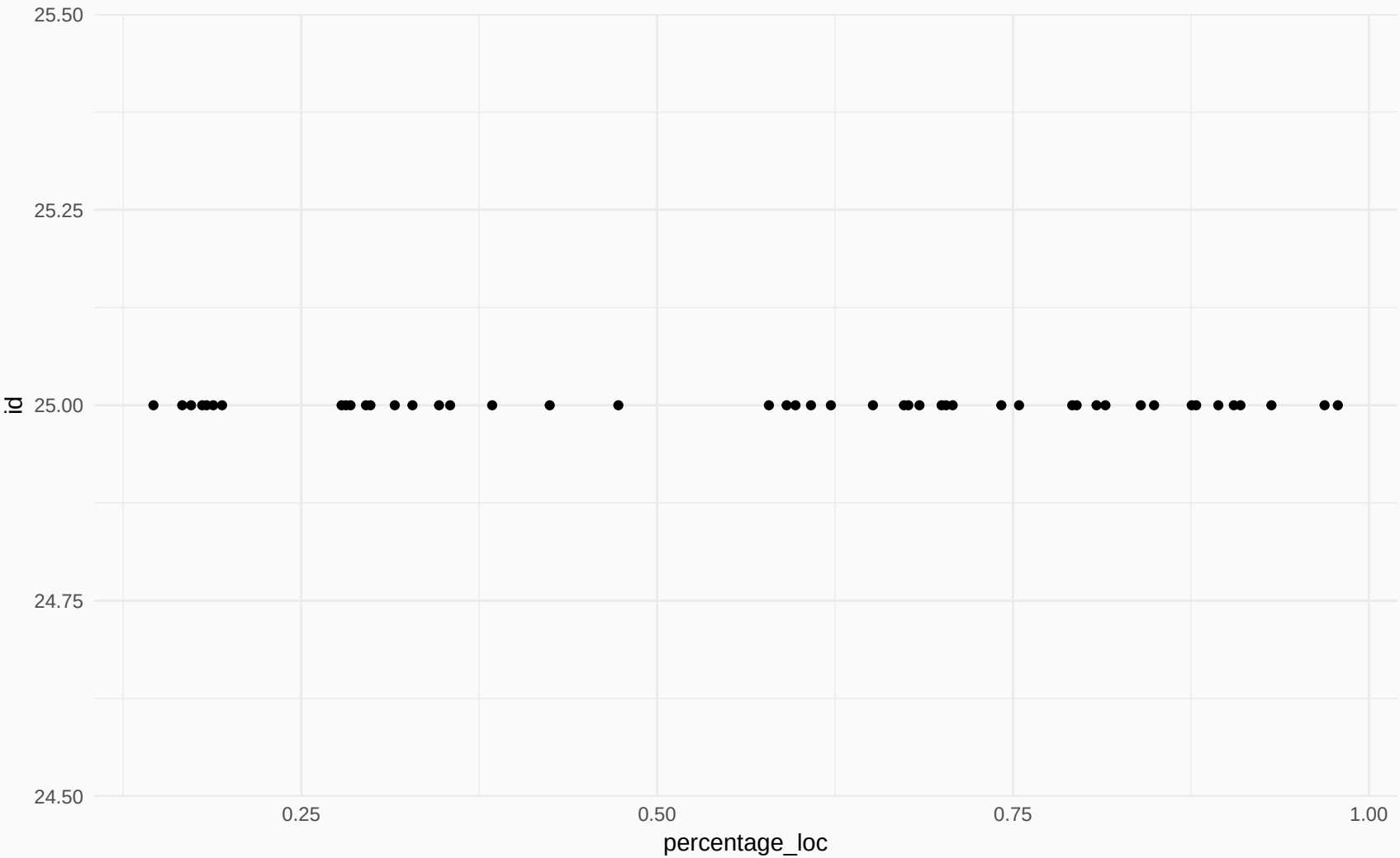
Sometimes this works well, sometimes is picks up the right idea but not enough to really know who the character is (i.e., “Colonel” or “Majesty”), and sometimes it works very well. We will see in the next session how to do a better job of this using more advanced annotation engines.

Visualising main characters

We will make use of the `semi_join` function to plot the character locations:

```
sh_tokens %>%
  group_by(id) %>%
  mutate(percentage_loc = sid / max(sid)) %>%
  semi_join(sh_topchar, by = c("id", "word")) %>%
  ggplot(aes(percentage_loc, id)) +
    geom_point()
```

Visualising main characters



Identifying Themes

Textual topics and themes

When we look back at our original list of top tokens, many of those instances that are not characters describe the main topics, themes, or artefacts of interest in the story.

Finding these frequent, non-proper nouns can indicate the theme or topics of interest within a corpus of texts.

Non-proper words

Our original code can be easily modified to only count those with all lower-case letters:

```
sh_theme <- sh_tokens %>%
  filter(!(tolower(word) %in% stopwords)) %>%
  filter((tolower(word) == word)) %>%
  count(id, word) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id")
```

Non-proper words, cont.

```
sh_theme %>% filter(id == 1) %>% print(n = 10)
```

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

Word frequencies

What we need is something stronger than a stop word list; conveniently such a dataset is included in the **cleanNLP** package as the dataset `word_frequency`.

```
word_frequency %>% print(n = 9)
```

```
## # A tibble: 150,000 x 3
##   language  word frequency
##      <chr> <chr>     <dbl>
## 1       en   the 3.9338375
## 2       en    of 2.2362525
## 3       en   and 2.2100158
## 4       en    to 2.0636764
## 5       en     a 1.5440913
## 6       en    in 1.4400708
## 7       en   for 1.0088552
## 8       en    is 0.8001275
## 9       en    on 0.6376924
## # ... with 1.5e+05 more rows
```

Filtering by word frequency

Instead of a stopword list, we filter out those words with a certain frequency cut-off. By changing this tuning parameter, we can tweak the results until they look reasonable.

```
sh_wordfreq <- sh_tokens %>%
  mutate(lemma = tolower(word)) %>%
  inner_join(word_frequency, by = c("lemma" = "word")) %>%
  filter(frequency < 0.01) %>%
  filter((tolower(word) == word)) %>%
  count(id, word) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id") %>%
  arrange(id, desc(n))
```

A more complex method could compare these global probabilities to the frequency in our text and identify the most deviant probabilities.

Filtering by word frequency, cont.

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

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