Chapter 5: Using Distinctiveness and Distance for Comparative Analyses of Historical Data

Up to now, this book has focused on simple measurements such as word counts. In this chapter, we shift our attention to statistical methods. Specifically, we explore two key concepts widely used in the natural sciences, information sciences, and digital humanities for capturing the defining traits of systems: distinctiveness, which measures how characteristic a word is within a specific context, and distance, which compares the similarity or difference between texts.

Distinctiveness may be used as a way of identifying unique features or patterns within a dataset (e.g., distinguishing words or features that are characteristic of a specific text or corpus compared to another). We will explore "term frequency-inverse document frequency" abbreviated to tf-idf.

Distance measurements are typically used to assign a metric to the similarity or dissimilarity of two sets of data. Just a few of these distance measurements include Jensen-Shannon Divergence (JSD), Euclidean distance, or cosine similarity.

To understand how these metrics might work in research, imagine that our task is comparing two famous children's authors. First, let's use some code to grab two texts, Lewis Carroll's *Through the Looking Glass* (Carroll 1872) and Beatrix Potter's *The Tale of Peter Rabbit* (Potter 1902).

```
# URLs for plain text versions
looking_glass_url <- "https://www.gutenberg.org/files/12/12-0.txt"
peter_url <- "https://www.gutenberg.org/cache/epub/14838/pg14838.txt"

# Extract "Through the Looking-Glass" and save it to RStudio's global
# environment
looking_glass_text <- readLines(looking_glass_url,
                                encoding = "UTF-8",
                                warn = FALSE)

# Extract "The Tale of Peter Rabbit" and save it to RStudio's global
# environment
peter_text <- readLines(peter_url,
                        encoding = "UTF-8",
                        warn = FALSE)
```

You can now see the texts we collected in the Global Environment panel.

Vectorising as Preparation for Measurement If we wanted a comprehensive list of words used frequently by Lewis Carroll but rarely by Beatrix Potter, we would begin by representing each book

as a frequency count of all the words they contain (e.g., “rabbit,” “tea,” “queen” for *Through the Looking Glass*, and “rabbit,” “garden,” “jacket” for *The Tale of Peter Rabbit*). Here is code to tokenize, clean, and count the words for each text:

```
# Load required libraries
library("tidyverse")
library("tidytext")

# Convert to data frames
looking_glass_df <- tibble(writer = "Lewis Carroll", text = looking_glass_text)

peter_df <- tibble(writer = "Beatrix Potter", text = peter_text)

# Combine both texts
books <- bind_rows(looking_glass_df, peter_df)

# Tokenize text by spliting it into individual words
tokens <- books %>%
  unnest_tokens(word, text) %>%
  filter(str_detect(word, "^[a-z']+$"))  # keep only alphabetic words

# Calculate word frequencies per author
word_counts <- tokens %>%
  count(writer, word) %>%
  pivot_wider(names_from = word, values_from = n, values_fill = 0)

# Focus on selected words of interest
target_words <- c("rabbit", "she", "little", "chortled", "shed",
                  "the", "hoe", "slithy", "galumph")
selected <- word_counts %>%
  select(writer, any_of(target_words))

# View the table
print(selected)
```

```
## # A tibble: 2 x 9
##   writer         rabbit   she little chortled  shed   the   hoe slithy
##   <chr>           <int> <int>  <int>    <int> <int> <int> <int>  <int>
## 1 Beatrix Potter      9     7      6        0     4   240     1      0
## 2 Lewis Carroll       0   523    108        1     1  1605     0      3
```

In advanced text mining, this entire set of steps is described as “vectorising” the text, because each text is converted into a list of numerical values corresponding to its word frequencies.

The vectors allow us to compute many different measurements, including distinctiveness, distance, and all of their variants, because all of the difference metrics we will apply begin with these words

Using Distinctiveness to Compare Two Authors

Distinctiveness measures score the relationships between words and documents across two or more sets of texts. These measures can identify words that are characteristic of a specific author, document, year, or political group. Take, for example, Lewis Carroll, a renowned British author of children's literature. Carroll, who had a background in mathematics and logic, was famously inventive with language, especially in his use of nonsense and portmanteau words. He had a strong preference for whimsical and linguistically playful terms—words that stretch meaning or defy conventional usage—such as "slithy," "chortle," and "galumph," – especially visible in his poem *Jabberwocky*, which enriches the dreamlike world of *Through the Looking Glass*.

If we're using distinctiveness algorithms, we would choose tf-idf or log likelihood, which assigns high values to each author–word pair. The pairs *Carroll–slithy* and *Potter–shed* should get a very high score, because these words are very unique to the respective authors. In contrast, *Carroll–she*, *Carroll–the*, *Potter–the*, and *Potter–she* would each get a very low distinctiveness score, as these common words are not unique to either author.

We will explain the code required later, but for now, simply observe the difference between the tf-idf measurements for each author. Which word-writer pairs get a high tf-idf score – which is to say, they are highly distinctive? Which word-author pairs get a low tf-idf score?

```
# Count word frequencies per author (long format)
word_counts2 <- tokens %>%
  count(writer, word, sort = TRUE) %>%
  rename(document = writer, term = word)

# Compute tf-idf values
tfidf_table <- word_counts2 %>%
  bind_tf_idf(term = term, document = document, n = n)

# View top distinctive words for each author
filtered_tfidf <- tfidf_table %>%
  filter(term %in% target_words) %>%
  arrange(document, desc(tf_idf)) %>%
  select(term, tf_idf)
```

```
library(knitr)

make_tab <- function(df) {
  df %>%
    mutate(tf_idf = round(as.numeric(tf_idf), 6)) %>%
    kable(format = "latex", booktabs = TRUE, caption = NULL,
          col.names = c("Word", "TF-IDF")) %>%
    as.character() }

left_tab  <- make_tab(filtered_tfidf[1:6, ])
right_tab <- make_tab(filtered_tfidf[7:12, ])
```

```
two_pane_table <- paste0(
    "\\noindent
    \\begin{minipage}[t]{0.48\\textwidth}
    \\raggedright\\textbf{Some Beatrix Potter Words}\\\\[3pt]",
    left_tab, "
    \\end{minipage}\\hfill
    \\begin{minipage}[t]{0.48\\textwidth}
    \\raggedright\\textbf{Some Lewis Carroll Words}\\\\[3pt]",
    right_tab, "
    \\end{minipage}")

asis_output(two_pane_table)
```

Some Beatrix Potter Words

Word	TF-IDF
rabbit	0.001546
hoe	0.000172
the	0.000000
she	0.000000
little	0.000000
shed	0.000000

Some Lewis Carroll Words

Word	TF-IDF
slithy	7.2e-05
chortled	2.4e-05
the	0.0e+00
she	0.0e+00
little	0.0e+00
shed	0.0e+00

We can use distinctiveness to quickly identify the words that are frequently in Carroll's book but not Potter's. We might expect the results to include a list of words that includes "Jabberwock" (a fearsome creature), the "Snark" (an elusive entity), the "Bandersnatch" (a fast and dangerous beast), and other terms from his peculiar universe. Indeed, many of these words have rarely, if ever, appeared in publications not authored by Carroll himself.

```
library("kableExtra")

distinctively_carroll <- tfidf_table %>%
  filter(document == "Lewis Carroll") %>%
  arrange(desc(tf_idf)) %>%
  select(term, tf_idf)

kable(head(distinctively_carroll, 15),
      format = "latex",
      booktabs = TRUE,
      caption = paste0("\\textbf{\\large Lewis Carroll's Most Distinctive Words}\\\\",
                       "\\textit{(when \\textit{Alice} is compared with \\textit{Peter Rabb
      col.names = c("Word", "TF-IDF"),
      escape = FALSE) %>%
  kable_styling(full_width = TRUE)
```

Interestingly, the Jabberwock, Bandersnatch, and Snark are outranked in this comparison by relatively simple characters – Alice, the Queen, the Red Knight, and Humpty-Dumpty – because none of

Table 1: **Lewis Carroll's Most Distinctive Words**
(when Alice is compared with Peter Rabbit)

Word	TF-IDF
alice	0.0107534
queen	0.0044024
like	0.0029349
me	0.0022132
again	0.0018764
herself	0.0018043
red	0.0016840
king	0.0014915
knight	0.0014193
cried	0.0013472
dumpty	0.0013231
humpty	0.0013231
here	0.0012750
two	0.0012510
moment	0.0010344

these characters appear in *Peter Rabbit*, while all of those characters are mentioned more than the Jabberwock.

If we were to compare *Through the Looking Glass* with a broader corpus of children's writing, we would get different results. Then, words like "queen," "knight," and even "Humpty-Dumpty" might appear outside of Carroll's corpus, but "Bandersnatch" is likely to remain distinctively Carroll-esque.

Using Distance to Compare Two Authors Measuring distance can offer insight into different types of questions than distinctiveness does. Suppose, instead of looking for particular words, we wish to assess how overall linguistically different the profile of any two authors is.

By measuring distance, we can compare the distribution of words in the two books. We would begin with a word-frequency vector, as above. Next, we apply a distance measure to the word counts. The result will be one measurement. A low distance value would indicate that *Alice's Adventures* and *Peter Rabbit* are linguistically similar. A high distance value would suggest that the texts are very dissimilar. However, if we were to compare *Alice's Adventures* to another Carroll book, such as *Through the Looking-Glass*, the calculated distance would be smaller, reflecting the stylistic and lexical similarities between the two texts.

Again, don't worry about the code, but pay attention to the scores in the results:

```
library("text2vec")             # Load the package

# define a function for jsd
jsd <- function(p, q) {
  # Ensure p and q are probability distributions
```

```r
  p <- p / sum(p)
  q <- q / sum(q)
  m <- 0.5 * (p + q)

  # Define a helper function for KL divergence
  kl_div <- function(a, b) {
    a <- ifelse(a == 0, 1, a)  # Avoid log(0)
    b <- ifelse(b == 0, 1, b)
    sum(a * log2(a / b)) }

  0.5 * kl_div(p, m) + 0.5 * kl_div(q, m) }

# load Alice in Wonderland
alice_url <- "https://www.gutenberg.org/files/11/11-0.txt"
alice_text <- readLines(alice_url, encoding = "UTF-8", warn = FALSE)

# Combine into one data frame
books <- bind_rows(
  tibble(booktitle = "Alice", text = alice_text),
  tibble(booktitle = "Peter", text = peter_text),
  tibble(booktitle = "LookingGlass", text = looking_glass_text))

# Tokenise and count
word_freqs <- books %>%
  unnest_tokens(word, text) %>%
  filter(str_detect(word, "^[a-z']+$")) %>%
  count(booktitle, word) %>%
  group_by(booktitle) %>%
  mutate(freq = n / sum(n)) %>%
  ungroup()

# Create a frequency matrix (book x word)
freq_matrix <- word_freqs %>%
  select(booktitle, word, freq) %>%
  pivot_wider(names_from = word, values_from = freq, values_fill = 0) %>%
  column_to_rownames("booktitle") %>%
  as.matrix()

# Compute JSD for each pair of books
book_names <- rownames(freq_matrix)
n_books <- nrow(freq_matrix)

jsd_matrix <- matrix(0, n_books, n_books)
rownames(jsd_matrix) <- book_names
colnames(jsd_matrix) <- book_names

for (i in 1:n_books) {
```

```
  for (j in 1:n_books) {
    jsd_matrix[i, j] <- jsd(freq_matrix[i, ], freq_matrix[j, ]) } }

# View results
kable(jsd_matrix)
```

	Alice	LookingGlass	Peter
Alice	0.00	17559.79	18542.92
LookingGlass	17559.79	0.00	20536.56
Peter	18542.92	20536.56	0.00

Distance measurements give us a score for how "distant" each book is from each other book. With Jensen-Shannon Distance (JSD), smaller values mean more similar texts. *Alice's Adventures in Wonderland* has a distance of 0 from *Alice's Adventures in Wonderland* because they are the same text. Meanwhile, *Alice* has the highest distance from *Peter Rabbit*, while *Alice* is much closer in writing style to *Through the Looking Glass*.

Using Distance to Analyze A Large Corpus Now, imagine we had access to a large repository of books. Instead of comparing two specific authors, we might wish to discover which writers use vocabularies similar to Lewis Carroll's. For instance, we could search for books with a comparable ratio of playful or invented words to conventional ones.

We might be interested in identifying the top ten authors who make frequent use of neologisms or poetic nonsense. If we were doing distance comparisons on a larger set, Carroll might score closer to authors like Edward Lear, Roald Dahl, or Spike Milligan, who also play with absurdity and sound, and farther from Beatrix Potter, whose style is more naturalistic and pastoral.

In sum, distance is useful for quantifying how far apart texts are in terms of their overall linguistic patterns. Variants of distance measurements could make it another alternative for pinpointing the specific features (like "chortle" or "jabberwock") that make a text unique – but in general, we leave this task to distinctiveness (for more on the nuances of each approach, please see Guldi's *The Dangerous Art of Text Mining*).

Distinctiveness and Distance Compared While distinctiveness identifies the unique lexical features of a writer's vocabulary, distance focuses on measuring the relationships and proximities between entire sets of features. We suggest that both of these methods of measurement invite a broader reflection on computational methods by foregrounding the importance of the processes through which we produce historical knowledge.

The ways in which we decide to process and analyse data—whether through distance or distinctiveness—are shaped by the questions we pose. Yet the choices we make during data processing directly influence the interpretive lenses we apply to a corpus. In this way, the computational steps involved in data processing are as much a part of knowledge creation as the act of historical interpretation itself. The methods we choose mediate our access to historical records, and their respective affordances and limitations may lead us to shift our analytical focus—for instance,

from emphasising distinctiveness as an indicator of uniqueness in a corpus, to using distance to reveal relational patterns, or to adopting an entirely different metric more suitable to the historical material at hand. These methodological decisions are interpretive acts in their own right, as they shape how we understand and conceptualize the data before us.

While there are different mathematical formulas for measuring distinctiveness or distance, this chapter introduces a function called term frequency-inverse document frequency (abbreviated "tf-idf") for measuring distinctiveness, and Jensen-Shannon divergence (abbreviated as "JSD") for measuring distance (Fuglede 2004, Lin 1991, Klingenstein 2014).

As we will show, these methods of measurement allow analysts to answers questions like: which words are characteristic of one speaker and not another? Which words are most symptomatic of the 1860s, a decade characterized by an economic crisis related to the cotton industry as well as labor disputes, and what can we learn about the 1860s from applying this method?

We will first demonstrate tf-idf to compare multiple decades back-to-back to surface the words that are most unique to one decade and not the others. We will then use JSD to explore the lexical features of debates surrounding the use of the word "cotton" measured against other decades. The results will demonstrate how changes in the way in which difference is measured reveal different dimensions of a corpus. We will encourage readers to reflect on these differences, using them as a way of expanding their insight into the nature of difference, the use of comparable algorithms as different tools in a process of analysis, and the beginning of a process of critically thinking about data, algorithm, and historical truth.

Distinctiveness applied to History

For the domain of history, distinctiveness measures offer a valuable tool for analyzing political language, allowing analysts to identify which words or patterns set a particular speaker, party, or historical period apart from another.

Here are some examples for how an analyst might think through the lens of distinctiveness when working with the Hansard corpus:

- An analyst might use distinctiveness measures to identify words that were frequently used by Gladstone but not by other prime ministers. This could highlight the linguistic patterns, priorities, or recurring concerns that made Gladstone a distinctive political figure. For example, if Gladstone consistently used words related to reform more often than others, this might reveal key characteristics of his political or rhetorical strategy.
- An analyst might apply distinctiveness to find out the words used by Gladstone early in his career, but not later, and later in his career, but not earlier.
- An analyst could use distinctiveness measures to analyze political parties and the words they used that were excluded from use by other parties at the time. By comparing the vocabulary used by members of different parties, the analyst might uncover terms that were uniquely associated with one party's rhetoric and thereby gain insight into the rhetorical strategies emphasized by each group. This approach can help trace how party identities were constructed and communicated through language.
- An analyst might use distance to compare units of time – weeks, months, or years – with all other units of weeks, months, or years – in order to answer the question, "when were new ideas and concerns coming to parliament's attention? In *The Dangerous Art of Text Mining*, Guldi

explains how distinctiveness can be used to investigate the most distinctive years, months, weeks, and days of any century, with the end of highlighting exceptional conversations that are unlike the day-to-day business that is the matter of any institution.

As these examples indicate, one of the most powerful ways of applying distinctiveness for the historian is about understanding change over time.

Distinctiveness as an approach to change over time Distinctiveness is a powerful concept for understanding historical change, because it allows the analyst to understand which words are most unique to any given political moment in time.

If the word "cotton" appears frequently in the years 1860-69, but cotton is almost never mentioned in the years 1806-1859 or 1970-1911, then "cotton" can be considered distinctive of the 1860s. If the word "cotton" appears with a similar frequency in speeches from each decade in the nineteenth century, it is not distinctive of any given decade but rather a reoccurring concern throughout each decade.

Unpacking Distinctiveness: The tf-idf algorithm Among the most common algorithms used to understand distinctiveness is "tf-idf" (Spärck-Jones 1972, Schnober 2015). The way tf-idf measurements reveal distinctiveness is by ranking the uniqueness of a word-document pair in respect to other words and other documents in a corpus. The algorithm was originally developed by mathematician Karen Spärk-Jones in 1972, and it has been used routinely in library science as a tool for automatically assigning keywords to articles on the basis of how unique a keyword is to the author's article and how unusual the term is for the field as a whole (Spärck-Jones 1972, Belkin et. al 1992).

At a high level, the guiding principle of tf-idf is to assign a high weight to terms that are frequent within a given document but relatively infrequent across the entire collection of documents. In this context, a high weight indicates that a term is particularly distinctive to a specific document, meaning it appears often in that document but is rare in the rest of the corpus. Consider an application to the word "cotton" when tf-idf is used to rank the distinctiveness of terms over time. "Term frequency" (TF) measures how often a term appears in a document, reflecting its local importance. "Inverse document frequency" (IDF) measures how rare a term is across the entire corpus, reducing the weight of common words. Multiplying these values yields the tf-idf score, which quantifies a term's distinctiveness in a given document.

There are multiple ways to define term frequency (TF) in a tf-idf equation. A common and straightforward method is to use the raw count of how many times a word appears in a document. Another approach normalizes term frequency by adjusting for document length, dividing the raw count by the total number of words in the document. This normalization is useful when comparing texts of different lengths, such as speeches from two individuals–one who spoke frequently and one who did not–or when analyzing decades with varying numbers of records.

Several variations of tf-idf exist and may be used for analysis (Manning et. al 2008, Domeniconi et. al 2016, Xiang 2022). For this chapter, however, we will use just raw counts for our tf-idf calculations for the sake of simplicity.

Inverse document frequency (IDF) measures how common or rare a word is within a corpus. It is calculated by dividing the total number of documents in the corpus by the number of documents

that contain the term, then taking the logarithm of that ratio. The primary purpose of IDF is to downweigh very common words, making infrequent terms more distinctive. While this tutorial removes stop words, doing so is not always necessary because IDF naturally reduces the influence of highly frequent terms while emphasizing those that are more unique to specific documents.

Measuring the Distinctiveness of Debates Speeches from the 1830s and 1860s using tf-idf

We will begin our work on Hansard by measuring the distinctiveness of language by decade. The first time we walk through the code, we will use a "toy" example in which we have intentionally downsized the data with the intention of showing how the algorithm works, using the minimal amount of data possible. Later, we will scale the toy code up to a larger corpus, applying distinctiveness to each decade between 1830 and 1870.

The code that follows will use distinctiveness to compare the words used by parliamentary speakers in the 1830s against the 1860s.

First we will load our libraries and our data, which includes the `stop_words` data from `tidytext` and the 1830 and 1860 Hansard debates. We will then use a version of tf-idf from `tidytext`. To use this version of tf-idf, we are going to combine these data frames into one. To track the association of words with their respective decades, we will add a column named "decade" and assign each entry to its corresponding decade.

We can use `bind_rows()` to combine `hansard_1830` and `hansard_1860` into a new data frame by stacking them vertically. This approach differs from using a join function such as `left_join()` or `inner_join()`, which merge data frames by matching rows based on common columns. In contrast, `bind_rows()` does not merge columns; it instead appends the rows of one data frame to the other, even if their column names do not fully align.

```
# Load our needed R packages
library("hansardr")
library("tidyverse")
library("lubridate")
library("tidytext")
library("gt")

# Load tidytext's built-in list of stop words for filtering out common words
data("stop_words")

# Load the Hansard debates from the 1830s and 1860s
data("hansard_1830")
data("hansard_1860")

# Add a 'decade' column to hansard_1830 to tag all rows with the value 1830
hansard_1830 <- hansard_1830 %>%
  mutate(decade = 1830)

# Add a 'decade' column to hansard_1860 to tag all rows with the value 1860
```

```r
hansard_1860 <- hansard_1860 %>%
  mutate(decade = 1860)

# Combine the two Hansard datasets by stacking their rows into one data frame
# 1830 will be stacked above 1860
hansard_data <- bind_rows(hansard_1830, hansard_1860)

# Inspect the data
hansard_data[, 2:3] %>%
  sample_n(5) %>%
  kable(format = "latex", booktabs = TRUE,
        caption = "Sample from Hansard Data") %>%
  kable_styling(latex_options = "hold_position")
```

Table 3: Sample from Hansard Data

text
No man could deny that this was a rotten borough; no man could deny, that the Earl of Radnor sent Members for
It seemed, therefore, that one man might say a thing safely which another must not even hint at.
—
"" Nobody could gainsay the justice of these words, because they merely proposed to provide for the parish that v
What may be the feeling of the people of Luxembourg I cannot undertake to say.

We can now further process our dataset to prepare it for tf-idf analysis. The following code applies a series of transformations: it tokenizes the text into a one-word-per-row format, removes stop words using the predefined list from the `tidytext` package, and filters out digits using a regular expression. By chaining these functions together, we ensure that the data is cleaned and structured appropriately for tf-idf calculations.

Since the following code applies multiple functions at once to a larger subset of decades, it may take longer to finish running.

Next, we prepare a table of counts that are necessary for the distinctiveness algorithm: * *words_per_decade*, a count of how many times each individual word was used per decade

From this table, the tf-idf function will be able to calculate the various components fo the tf-idf algorithm, including: * the number of total unique words for the 1830s * the number of total unique words for the 1860s * the number of total unique words for the corpus overall

Sidenote: Minimizing the impact of data messiness For the sake of our analysis, we are only keeping words that appear more than once per decade–an operation that we perform with `filter(n > 1)`. One reason for this approach is that words appearing only once in the 19th-century Hansard corpus are often the result of OCR errors. When the transcripts were digitized, poor image quality resulted in a mismatch between the digitized word and the word as it was recorded on the transcript. The letter "O," for example, may have been interpreted as the number zero, or the letter "h" might have been interpreted as two "i's" such as "ii." Future versions of the digital Hansard corpus may address these OCR errors.

In the meantime, however, analysts often have to make judgment calls that entail narrowing a corpus to words, whether to eliminate OCR errors or simply to identify more common words. Tweaking the value of the number x in the expression `filter(n > x)` will alter the final results of this chapter in ways that may be meaningful by removing more low frequency words.

```
## # A tibble: 2 x 9
##   hansard_decade success cogent  pith axioms disappears uninspired knitters
##            <dbl>   <int>  <int> <int>  <int>      <int>      <int>    <int>
## 1           1830    2267      0     0      0          3          0        2
## 2           1860       0     80    50      6          0          2        0
## # i 1 more variable: daley <int>
```

These results are ready to be passed to the distinctiveness algorithm so that the tf-idf metric can be computed for each word-decade pair.

We use the function `bind_tf_idf()` to calculate distinctiveness for any word-document pair, using the arguments *word*, *document*, and *n*. To calculate distinctiveness by decade, we set the argument *document* as "decade."

```
# calculate tf-idf (term frequency-inverse document frequency) for each word by decade
# using word frequency and total count
hansard_tf_idf <- words_per_decade %>%
  bind_tf_idf(word, decade, word_count_per_decade)  # compute tf-idf \
```

```
hansard_tf_idf %>%
  sample_n(5) %>%
  arrange(desc(tf_idf))
```

```
##    decade          word word_count_per_decade           tf       idf
##     <num>        <char>                <int>        <num>     <num>
## 1:   1830          isto                    2 2.212553e-07 0.6931472
## 2:   1860       tierney                    7 7.289553e-07 0.0000000
## 3:   1860 irreclaimable                   10 1.041365e-06 0.0000000
## 4:   1860   provocation                  127 1.322533e-05 0.0000000
## 5:   1830        griefs                    3 3.318830e-07 0.0000000
##          tf_idf
##           <num>
## 1: 1.533625e-07
## 2: 0.000000e+00
## 3: 0.000000e+00
## 4: 0.000000e+00
## 5: 0.000000e+00
```

Please note that the output of `bind_tf_idf()` includes the component calculations we mentioned: - one row for each decade in which each word appears - the total number of times each word appears in this decade (given in `word_count_per_decade`, the product of our calculation to produce the `words_per_decade` data frame) - the "term frequency" calculation, a metric of how often each word

appears per decade, normalized against the total number of words in each decade. A high TF means that the word appears frequently in a given decade. - the “inverse document frequency” metric, a calcuation of how uncommon the word is across the entire corpus. A high IDF means that a word is rare across the corpus. - the tf_idf score – the most important score for interpreting the results. A high tf-idf score means that a word is highly distinctive of a decade, i.e., it scarcely appears in the other decade(s) of a corpus.

Now that we have a tf-idf metric for each word-decade pair, we can use these scores to interpret the differences between the 1830s and 1860s.

```
# keep top 15 tf-idf words per decade
top_hansard_tf_idf <- hansard_tf_idf %>%
  group_by(decade) %>% # group by decade
  slice_max(tf_idf, n = 15) %>% # keep top 15 words
  arrange(decade, desc(tf_idf)) %>%
  ungroup()
```

Let’s make a fancy table using `kableExtra` to inspect the data.

```
library("knitr")
library("kableExtra")

make_kable <- function(df, dec_label) {
  df %>%
    select(word, tf_idf) %>%
    arrange(desc(tf_idf)) %>%
    mutate(tf_idf = round(as.numeric(tf_idf), 6)) %>%
    kbl(format = "latex",
        booktabs  = TRUE,
        col.names = c("Word", "TF--IDF Score"),
        align = c("l", "r"),
        caption = paste0("Top TF--IDF Words — ", dec_label),
        escape = TRUE) %>%
    kable_styling(full_width = FALSE) }

target_decades <- if_else(is.numeric(top_hansard_tf_idf$decade),
                          list(c(1830, 1860)),
                          list(c("1830s", "1860s"))) %>%
  unlist()

top_hansard_tf_idf %>%
  filter(decade %in% target_decades) %>%
  group_by(decade) %>%
  group_walk(~ print(make_kable(.x, unique(.x$decade))))
```

```
## Warning: Unknown or uninitialised column: `decade`.
```

Table 4: Top TF–IDF Words —

Word	TF–IDF Score
miguel	7.3e-05
gosford	3.6e-05
pedro	3.1e-05
terceira	2.7e-05
benefitted	2.3e-05
mulgrave	2.0e-05
vigors	1.8e-05
donna	1.7e-05
wetherell	1.6e-05
boroughbridge	1.5e-05
poulter	1.4e-05
predial	1.3e-05
deacle	1.2e-05
carlists	1.2e-05
robison	1.2e-05

Table 5: Top TF–IDF Words —

Word	TF–IDF Score
disestablishment	6.2e-05
schleswig	5.9e-05
churchward	5.7e-05
fenian	4.8e-05
turret	4.2e-05
plated	4.1e-05
disendowment	4.0e-05
cairns	4.0e-05
newdegate	4.0e-05
disraeli	3.9e-05
crimean	3.8e-05
ayrton	3.7e-05
sewage	3.6e-05
competitive	3.2e-05
certificated	3.2e-05

```
## Warning: Unknown or uninitialised column: `decade`.
```

Words that were distinctive in the 1830s, compared to later decades, often reference the Carlist Wars—a series of civil wars in Spain that began in 1833 over the dispute regarding the rightful successor to the throne. Portugal, France and the United Kingdom supported the regency, and even intervened in the conflict by sending forces to confront the Carlist army. Subsequently, the reference to "Carlist" can possibly explain the number of Spanish names/words listed in the bar plot, including "Miguel," "Pedro," and "Terceira."

Other terms that appear as distinctive of the 1830s in comparison to the 1860s are references to individuals and places whose names punctuated routine debates of the period. The year 1831 marked the death of the 1st Earl of Mulgrave, a distinguished politician who had served as Foreign Secretary and was subsequently memorialized in many speeches. From 1830-2, Charles Wetherell represented Boroughbridge in Parliament, explaining the inclusion of his name and the constituency he served. In 1839 the recently colonized city of Gosford, in New South Whales, Australia, is given its name after the 2nd Earl of Gosford, a friend of the governor of New South Whales.

Words distinctive of the 1860s include several references to contemporary matters of foreign affairs unique to the decade, for instance, "schleswig," a reference to the Second Schleswig War (1864), wherein Germany and Denmark commenced an altercation over disputed territory that included the province of Schleswig-Hollsteine. "Fenian" references the 1867 Fenian Uprising, a failed rebellion against British rule in Ireland. The name "Disraeli" marks the prime minister who oversaw the 1867 Reform Act that enfranchised much of Britain's working class. The term "Crimean" references the Crimean War, which broke from 1853-56. "Sewage" references contemporary questions of the rebuilding of London's water supply.

This simple exercise demonstrates tf-idf in action, providing a concrete example of how the method identifies terms that are unique to each document, with minimal overlap between them.

This toy analysis is limited in implication, however, because historians rarely want to compare two distant periods of time. Far more instructive will be the results of analyzing each decade between 1830 to 1860 against each other.

Analyzing the Distinctiveness of Decades Over a Half-Century, from 1830 to 1870

The following code loads the 1840 and 1850 Hansard debates and combines them into a single data frame that will be used to measure tf-idf.

```
# Load Hansard data for the 1840s and 1850s
data("hansard_1840")
data("hansard_1850")

# Add a "decade" column to the 1840s dataset
hansard_1840 <- hansard_1840 %>%
  mutate(decade = 1840)

# Add a "decade" column to the 1850s dataset
```

```r
hansard_1850 <- hansard_1850 %>%
  mutate(decade = 1850)

# Append the 1840s and 1850s data to the existing hansard_data
hansard_data_1830_to_1870 <- hansard_data %>%
  bind_rows(hansard_1840) %>% # Add 1840s data
  bind_rows(hansard_1850) # Add 1850s data
```

In preparation for using tf-idf across multiple decades, we use the following codes to generate word frequencies in text data across specific time periods (e.g., decades). It uses a `for()` loop to iterate through each decade for preprocessing. It preprocesses the text by tokenizing it into words, removing stop words, and filtering out irrelevant tokens (like numbers), then counts the most frequently used words for each decade. The results are stored in a new data frame, `hansard_count`.

While a `for` loop is not needed to perform these actions, we have chosen to use one here to process text data separately for each decade in the dataset. Otherwise, tokenizing all four decade subsets simultaneously could exceed available computer memory. The code may run slow on older or more limited machines – give it time to run.

```r
# define the list of decades to analyze. each value will be used to
# filter the main dataset, enabling decade-by-decade processing.
decades <- c(1830, 1840, 1850, 1860)

# create an empty data frame to hold word frequency counts from each decade.
# this object will be built up by appending results from within the loop.
hansard_words_1830_to_1870 <- data.frame()

# iterate through each specified decade to perform tokenization,
# filtering, and word frequency counting. results are combined into a single table.
for(d in decades) {

  # filter the full hansard dataset for just the current decade (d)
  new_decade <- hansard_data_1830_to_1870 %>%
    filter(decade == d)  # keep only rows from this decade

  # tokenize the text column into individual words, remove stop words,
  # and exclude any tokens that contain numeric digits (e.g., years)
  tokenized_new_decade <- new_decade %>%
    unnest_tokens(word, text) %>%  # convert text to one word per row
    anti_join(stop_words) %>%      # remove stop words
    filter(!str_detect(word, "[:digit:]"))  # remove words with digits

  # count the frequency of each remaining word in the current decade
  # then filter to retain only words appearing more than 20 times
  new_decade_count <- tokenized_new_decade %>%
    count(decade, word, sort = TRUE) %>%  # count word frequencies
    filter(n > 20)  # keep words with frequency > 20
```

```
  # append the word counts from this decade to the overall results table
  hansard_words_1830_to_1870 <- bind_rows(hansard_words_1830_to_1870,
                                          new_decade_count)}  # combine rows
```

```
## Joining with `by = join_by(word)`
## Joining with `by = join_by(word)`
## Joining with `by = join_by(word)`
## Joining with `by = join_by(word)`
```

We now have a dataset, hansard_count, which includes each word along with the number of times it appears in a given decade.

```
# Inspect the Data
hansard_words_1830_to_1870 %>% group_by(decade) %>%
  sample_n(2) %>%
  ungroup() %>%
  arrange(desc(decade)) %>%
  head()
```

```
## # A tibble: 6 x 3
##   decade word           n
##    <dbl> <chr>      <int>
## 1   1860 possessed   2801
## 2   1860 department  7829
## 3   1850 foregoing     55
## 4   1850 unexpected   188
## 5   1840 mallow       115
## 6   1840 surpassing    32
```

We can now apply tf-idf and visualize our results, in this case, the first 15 words that appear for each decade.

```
# compute tf-idf values and keep top 15 words by tf-idf for each decade
most_dist_hansard_words <- hansard_words_1830_to_1870 %>%
  bind_tf_idf(word, decade, n) %>% # compute tf-idf for each word by decade
  group_by(decade) %>% # group by decade
  slice_max(tf_idf, n = 15) # keep top 15 tf-idf words per group
```

```
library("knitr")
library("kableExtra")

make_kable <- function(df, dec_label) {
  df %>%
    select(word, tf_idf) %>%
    arrange(desc(tf_idf)) %>%
```

```
    mutate(tf_idf = round(as.numeric(tf_idf), 6)) %>%
    kbl(format = "latex",
        booktabs = TRUE,
        col.names = c("Word", "TF--IDF Score"),
        align = c("l", "r"),
        caption = paste0("Top TF--IDF Words — ", dec_label),
        escape = TRUE) %>%
    kable_styling(full_width = FALSE) }

# Optional: pick specific decades; otherwise render all present
# target_decades <- c("1830s","1860s")  # or c(1830,1860) if numeric
# df_in <- most_dist_hansard_words %>% filter(decade %in% target_decades)

df_in <- most_dist_hansard_words

# If you want a specific order, set a factor:
# df_in <- df_in %>%
#   mutate(decade = factor(decade, levels = sort(unique(decade))))

# Print one table per decade (no list() artifact)
df_in %>%
  group_by(decade) %>%
  group_walk(~ print(make_kable(.x, unique(.x$decade))))
```

```
## Warning: Unknown or uninitialised column: `decade`.

## \begin{table}
## \centering
## \caption{\label{tab:unnamed-chunk-17}Top TF--IDF Words — }
## \centering
## \begin{tabular}[t]{lr}
## \toprule
## Word & TF--IDF Score\\
## \midrule
## marquis & 0.000139\\
## miguel & 0.000074\\
## gosford & 0.000072\\
## carlos & 0.000062\\
## appleby & 0.000057\\
## \addlinespace
## terceira & 0.000055\\
## raphael & 0.000050\\
## mulgrave & 0.000041\\
## shew & 0.000040\\
## aldborough & 0.000036\\
## \addlinespace
## vigors & 0.000036\\
```

```
## impropriators & 0.000035\\
## wetherell & 0.000033\\
## plunkett & 0.000033\\
## burthens & 0.000032\\
## \bottomrule
## \end{tabular}
## \end{table}

## Warning: Unknown or uninitialised column: `decade`.

## \begin{table}
## \centering
## \caption{\label{tab:unnamed-chunk-17}Top TF--IDF Words — }
## \centering
## \begin{tabular}[t]{lr}
## \toprule
## Word & TF--IDF Score\\
## \midrule
## ovans & 4.5e-05\\
## warner & 4.4e-05\\
## o'driscoll & 4.3e-05\\
## sliding & 3.8e-05\\
## gauge & 3.7e-05\\
## \addlinespace
## stockdale & 3.7e-05\\
## junta & 3.6e-05\\
## ameers & 3.6e-05\\
## mott & 3.5e-05\\
## keane & 3.3e-05\\
## \addlinespace
## sattara & 3.2e-05\\
## traversers & 3.1e-05\\
## colonisation & 3.0e-05\\
## thorogood & 2.8e-05\\
## differential & 2.8e-05\\
## \bottomrule
## \end{tabular}
## \end{table}

## Warning: Unknown or uninitialised column: `decade`.

## \begin{table}
## \centering
## \caption{\label{tab:unnamed-chunk-17}Top TF--IDF Words — }
## \centering
## \begin{tabular}[t]{lr}
## \toprule
```

```
## Word & TF--IDF Score\\
## \midrule
## crimea & 0.000186\\
## raglan & 0.000133\\
## kars & 0.000106\\
## sebastopol & 0.000100\\
## balaklava & 0.000081\\
## \addlinespace
## sadleir & 0.000047\\
## whiteside & 0.000046\\
## hebdomadal & 0.000044\\
## mather & 0.000039\\
## oude & 0.000038\\
## \addlinespace
## alma & 0.000038\\
## inkerman & 0.000038\\
## lorcha & 0.000037\\
## scutari & 0.000036\\
## menchikoff & 0.000035\\
## \bottomrule
## \end{tabular}
## \end{table}

## Warning: Unknown or uninitialised column: `decade`.

## \begin{table}
## \centering
## \caption{\label{tab:unnamed-chunk-17}Top TF--IDF Words — }
## \centering
## \begin{tabular}[t]{lr}
## \toprule
## Word & TF--IDF Score\\
## \midrule
## disestablishment & 0.000127\\
## fenian & 0.000098\\
## turret & 0.000086\\
## plated & 0.000083\\
## savoy & 0.000080\\
## \addlinespace
## garibaldi & 0.000062\\
## disestablished & 0.000061\\
## abyssinia & 0.000059\\
## cased & 0.000059\\
## churchward & 0.000058\\
## \addlinespace
## armour & 0.000057\\
## fenianism & 0.000056\\
## rifle & 0.000055\\
```

```
## edmunds & 0.000053\\
## coles & 0.000052\\
## \bottomrule
## \end{tabular}
## \end{table}
```

By analyzing the words most characteristic of each decade (the 1830s, 40s, 50s, and 60s), we can easily see a link between those terms and historical events.

Some of the words may key the analyst into rhetorical shifts, like the old-fashioned spelling "shew," discarded after the 1830s. Others attune us to seemingly minor issues that neverthless were discussed frequently in their time. For instance, the distinctive words of the 1830s now include the term "impropriators," a designation of a layperson who held the income rights of a church benefice. The impropriators' rights dated from the Tudor dissolution of the monasteries. Lay impropriators became a target of tithe reform in the 1830s. Other words suggest known turning points of history, for example the governance of railway gauges in the 1840s, the Crimean War in the 1850s, and the prominence of Fenian terrorism and the rifle in the 1860s. Still other words may lead us to new research questions – for instance, asking what is the relevance of "differential" in the 1840s? This question might lead us to discussions of the "differential duty" discussed by the House of Lords in April 1840, when they mooted making foreign exporters pay higher duties, so as to protect domestic interests tied to monopolies.

Cleaning Out the Cache so as to Preserve Memory

At this stage of our analysis, we are working with several datasets. Working with multiple datasets at a given time can provide us with a richer overview of the historical records, however, it can consume a significant amount of memory on our computer. To free memory, we can remove some of the large datasets from our Global Environment before going to our next examples and analyses. One can remove data using `rm()` while passing the variables you wish to clear. For example, `rm()` can be passed a single variable name or multiple variable names.

```
# Remove both datasets from the global environment
rm(hansard_1830, hansard_1840)
```

Or, if instead we wish to remove all variables, `rm()` can be passed the output of `ls()`, a function that lists all the variables from the Global Environment.

```
# List all objects currently in the global environment
ls()
```

```
##  [1] "alice_text"                "alice_url"
##  [3] "book_names"                "books"
##  [5] "d"                         "decades"
##  [7] "df_in"                     "distinctively_carroll"
##  [9] "filtered_tfidf"            "freq_matrix"
## [11] "hansard_1850"              "hansard_1860"
```

```
## [13] "hansard_data"               "hansard_data_1830_to_1870"
## [15] "hansard_tf_idf"             "hansard_words_1830_to_1870"
## [17] "i"                          "j"
## [19] "jsd"                        "jsd_matrix"
## [21] "left_tab"                   "looking_glass_df"
## [23] "looking_glass_text"         "looking_glass_url"
## [25] "make_kable"                 "make_tab"
## [27] "most_dist_hansard_words"    "n_books"
## [29] "new_decade"                 "new_decade_count"
## [31] "peter_df"                   "peter_text"
## [33] "peter_url"                  "right_tab"
## [35] "selected"                   "stop_words"
## [37] "target_decades"             "target_words"
## [39] "tfidf_table"                "tokenized_hansard_data"
## [41] "tokenized_new_decade"       "tokens"
## [43] "top_hansard_tf_idf"         "two_pane_table"
## [45] "word_counts"                "word_counts2"
## [47] "word_freqs"                 "words_per_decade"
```

```
# List all objects currently in the global environment
all_variables <- ls()
# Print the names of all objects (optional, for confirmation)
print(all_variables)
```

```
##  [1] "alice_text"              "alice_url"
##  [3] "book_names"              "books"
##  [5] "d"                       "decades"
##  [7] "df_in"                   "distinctively_carroll"
##  [9] "filtered_tfidf"          "freq_matrix"
## [11] "hansard_1850"            "hansard_1860"
## [13] "hansard_data"            "hansard_data_1830_to_1870"
## [15] "hansard_tf_idf"          "hansard_words_1830_to_1870"
## [17] "i"                       "j"
## [19] "jsd"                     "jsd_matrix"
## [21] "left_tab"                "looking_glass_df"
## [23] "looking_glass_text"      "looking_glass_url"
## [25] "make_kable"              "make_tab"
## [27] "most_dist_hansard_words" "n_books"
## [29] "new_decade"              "new_decade_count"
## [31] "peter_df"                "peter_text"
## [33] "peter_url"               "right_tab"
## [35] "selected"                "stop_words"
## [37] "target_decades"          "target_words"
## [39] "tfidf_table"             "tokenized_hansard_data"
## [41] "tokenized_new_decade"    "tokens"
## [43] "top_hansard_tf_idf"      "two_pane_table"
## [45] "word_counts"             "word_counts2"
## [47] "word_freqs"              "words_per_decade"
```

```r
# Remove all objects from the global environment
rm(list = all_variables)
```

Following `rm()` with `gc()` (which stands for "garbage collection") ensures that memory occupied by removed objects is fully reclaimed. When an object is deleted using `rm()`, R does not immediately free up the associated memory; instead, the memory remains allocated until the garbage collector runs. By calling `gc()`, we force R to release unused memory, preventing excessive memory consumption. This may be particularly important when working with large datasets or performing memory-intensive operations, as it helps reduce the risk of running out of memory.

```r
# Run garbage collection to reclaim memory
gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2747511 146.8   15466953  826.1  19333691 1032.6
## Vcells 13175390 100.6  509849476 3889.9 796639477 6077.9
```

This output from the `gc()` function provides a snapshot of memory usage and triggers garbage collection. The Ncells store internal structures, like lists, while the Vcells store numerical, character, and logical data. The gc trigger values indicate the memory thresholds that, when exceeded, will automatically initiate garbage collection.

Sometimes, due to memory fragmentation, R may not be able to release memory back to the operating system, even after running `gc()`.Memory fragmentation occurs when R allocates memory in multiple small chunks over the course of a session. As objects are created and removed, these chunks become scattered throughout memory. When this happens, R's garbage collector (`gc()`) can free memory, but it cannot always merge these scattered memory blocks into a single large block that the operating system can reclaim. As a result, memory usage can remain high, even if most objects have been removed.

This issue can be seen in RStudio above the Global Environment panel. If a significant amount of memory remains reserved, it may be necessary to restart R to fully release fragmented memory. This can be done by closing out of RStudio and without saving the work space, or by running `q()` with the parameter `save` set to "no".

Important: Running the following code block will quit RStudio.

```r
# Quit the R session without saving the current workspace
q(save = "no")
```

If you quit RStudio, you will need to reload the required libraries:

```r
library("tidyverse")
library("tidytext")
library("hansardr")
```

Distance for Historical Analysis

Distance is used to compare the distributions of words in any two corpora. In general, it is less used to identify the variables that are most characteristic of a corpus, and more frequently used when the analyst needs to position different corpora on a "magic ruler" that ranks them from most similar to least similar to some text.

A historian might use distance for the following applications: * An analyst might choose to identify the speakers whose overall use of political vocabulary most closely resembles Gladstone's. They might then ask whether those speakers were all members of his political party, or whether some came from across the political divide. This kind of analysis could tell us who shared his linguistic style, and whether that style aligned with partisan boundaries or not. * An analyst might use distance to rank the individual speeches that sound the most like any one of a set political passages excerpted from contemporary novels, including those of Charles Dickens, which frequently engage political phenomena. This process might help the analyst to discover the likeliest speeches that Dickens was attending to when he wrote various novels. * An analyst might compare parliamentary speeches and newspaper stories over time to ask questions about when the lines of influence run from parliament to the newspaper, and when questions are first raised in the newspaper and later in parliament * An analyst might apply party distance by decade to inform research on how party identities shifted over time in response to political events or ideological realignments.

In the sections that follow, we will only address the first of these queries – the matter of how to compare two decades of time to highlight what was coming into parliament's attention and what was fading over a twenty-year period.

The method introduced, however, once mastered, can be applied to all other problems of analysis. Indeed, comparing subcorpora with distance underlies a great many possible programs of analysis, each of which can be accomplished by applying distance measurements to different corpora – that is, to comparing one decade against all other decades, one week to all other weeks, one speaker to all other speakers, and one speech to all other speeches. A great many problems of abstract analysis can be rendered into text-mining problems by recognizing the impulse of comparison at their heart, and translating this basic comparison into distance measurements.

Jenson-Shannon Divergence (JSD)

Any algorithm emphasizes certain aspects of a corpus while de-emphasizing others. Investigating the implications of adopting different algorithms is a critical approach to working with data. To illustrate this point, we introduce another metric for measuring a corpus: a modified form of divergence known as Partial Jensen-Shannon Divergence (Partial JSD). Jensen-Shannon Divergence (JSD) is a measure from information theory that quantifies the similarity between probability distributions by calculating the divergence between their averaged distributions. Unlike other distinctiveness measures, which often highlight exceptional words or phrases between two documents, JSD provides a general framework for comparing textual data by treating documents as probability distributions over words. This approach allows us to use Partial JSD to quantify how "close" or "distant" the language between two datasets are.

Distance is a quantitative measure that captures the degree of similarity or difference in the distribution of a word's occurrence across two or more groups. In text analysis, distance measures assess how consistently or variably a word appears in different datasets, helping to determine whether its

usage patterns align or diverge. These measures allow for ranking datasets based on the extent to which words are expressed at comparable rates across them, providing a structured way to compare linguistic patterns, such as by identifying patterns of relative similarity or dissimilarity based on word frequencies, term distributions, or other textual features.

In the following section, we will use the partial Jensen-Shannon Divergence (JSD)—specifically, one-half of the full JSD equation. We have chosen to use partial JSD because we have chosen to focus our analysis on how the distribution of words in one dataset differs from another without averaging both sides of the comparison. This is useful for seeing how much one dataset deviates from a reference point (for example, the decade 1850 against the previous decade, 1840, and vice-versa) rather than measuring a balanced difference. It simplifies the calculation while still highlighting key differences, making it easier to interpret. This approach is especially helpful when comparing texts from different time periods without averaging changes in both directions or masking asymmetrical shifts in word usage.

We hope that readers will become comfortable with the idea that algorithmic approaches to measuring key characteristics of a corpus are varied, and that the analyst can learn from comparing the results of working with different measurements.

Measuring Distance Between Words in a Corpus using JSD

First we will load our data and prepare it for our partial JSD function. The following code tokenizes and cleans both decades.

```
library("tidyverse")
library("hansardr")
library("tidytext")

# Load Hansard data for the 1840s and 1850s
data("hansard_1840")
data("hansard_1850")

# preprocess the 1840s hansard data:
# tokenize the text column into words, remove stop words,
# and filter out tokens that contain digits
hansard_1840 <- hansard_1840 %>% # create a new dataset
  unnest_tokens(word, text) %>% # convert text into one word per row
  anti_join(stop_words) %>% # remove common stop words
  filter(!str_detect(word, "[:digit:]"))  # remove words with digits
```

```
## Joining with `by = join_by(word)`
```

```
# preprocess the 1850s hansard data using the same steps:
# tokenize, remove stop words, and exclude numeric tokens
hansard_1850 <- hansard_1850 %>% # create a new dataset
  unnest_tokens(word, text) %>% # tokenize text
```

```
  anti_join(stop_words) %>% # remove stop words
  filter(!str_detect(word, "[:digit:]"))  # exclude tokens with digits
```

```
## Joining with `by = join_by(word)`
```

We have defined a partial JSD function below. The code defines two functions that measure how much word usage changed between 1840 and 1850. Each function compares the word frequencies from one decade to an average of both decades. One function looks at how much 1840's word usage differs from this average, while the other does the same for 1850. This helps identify words that became more or less common over time without mixing both decades together.

```
# Function to calculate one-sided KL divergence component of JSD
partial_jsd_1840_to_1850 <- function(p, q) {
  m <- 0.5 * (p + q) # Midpoint distribution
  return(p * log2(p / m))} # 1830 deviation from midpoint

partial_jsd_1850_to_1840 <- function(p, q) {
  m <- 0.5 * (p + q) # Midpoint distribution
  return(q * log2(q / m))} # 1860 deviation from midpoint


# define a function for jsd
jsd <- function(p, q) {
  # Ensure p and q are probability distributions
  p <- p / sum(p)
  q <- q / sum(q)
  m <- 0.5 * (p + q)

  # Define a helper function for KL divergence
  kl_div <- function(a, b) {
    a <- ifelse(a == 0, 1, a)  # Avoid log(0)
    b <- ifelse(b == 0, 1, b)
    sum(a * log2(a / b))
  }

  0.5 * kl_div(p, m) + 0.5 * kl_div(q, m)
}
```

The difference in the return statements—one using p and the other using q—ensures that each function measures the directional change in word usage for a specific decade. By using p in one function and q in the other, we avoid blending the two distributions. Instead, we get a clearer, one-sided measure of how word usage changed over time. This allows us to distinguish words that declined after 1840 from those that rose in prominence by 1850.

In both functions, `m = 0.5 * (p + q)` represents the average probability distribution between the two periods, and the return value calculates the information gain when approximating p (or q) with `m`. This approach ensures that we capture asymmetrical shifts in word frequency over time.

At this stage we can calculate word frequencies. This is an essential step because JSD operates on probability distributions. By counting how often each word appears in a given decade (e.g., 1840 and 1850), we can transform these counts into probabilities, which are required to measure divergence. Since Partial JSD specifically focuses on how one decade's word distribution deviates from another without averaging both directions, accurate word frequencies ensure that we capture asymmetrical changes in language use. An asymmetrical change occurs when a word's usage shifts more in one direction than the other. For example, if a word was common in 1840 but became rare in 1850, this is an asymmetrical change because the decrease is not balanced by an equivalent increase elsewhere. Partial JSD captures these one-sided shifts, helping identify words that either faded out or emerged over time rather than averaging both directions.

```
# Count word frequencies for both decades
freq_1840 <- hansard_1840 %>%
  count(word, name = "count_1840")

freq_1850 <- hansard_1850 %>%
  count(word, name = "count_1850")

freq_1840
```

```
##            word count_1840
##          <char>      <int>
##     1:   _cries          1
##     2:      _do          1
##     3:    _hear          1
##     4:   _order          1
##     5: a'becket          1
##    ---
## 62134:    that□          2
## 62135:     the□          1
## 62136:      we□          1
## 62137: laughs□□          1
## 62138:       □w          1
```

In the following code, we take additional steps to focus our analysis on the frequent words from each decade while avoiding rarer terms. This approach will provide us with an overview of both decades.

```
# define how many top words to extract from each decade's corpus.
# this number controls the vocabulary size for comparison.
top_n <- 50  # number of top words to extract

# extract the top n most frequent words from the 1840s word frequency table.
# keep only the word column and return it as a character vector.
top_words_1840 <- freq_1840 %>%
  slice_max(order_by = count_1840, n = top_n) %>%  # select top n by count
  select(word) %>%  # keep only the word column
```

```
  deframe()  # convert to a character vector

# extract the top n most frequent words from the 1850s word frequency table.
# repeat the same steps to prepare for cross-decade comparison.
top_words_1850 <- freq_1850 %>%
  slice_max(order_by = count_1850, n = top_n) %>%  # select top n by count
  select(word) %>%  # keep only the word column
  deframe()  # convert to a character vector
```

Combining the top words from both decades ensures that words that were dominant in either period are included in the analysis, preventing the exclusion of key terms that may have gained or lost prominence.

```
# Combine top words from both decades and keep only unique words
all_top_words <- unique(c(top_words_1840, top_words_1850))
```

We can further prepare and processes the word frequency data to compute Partial JSD, identifying words that changed the most between 1840 and 1850.

The following code does this by: * Merging and Cleaning Data: We create a dataset (`word_dist`) that includes the top words from both decades and their frequencies. We use `left_join()` to merge the word counts from 1840 and 1850, ensuring that every word in either decade is included. Missing values (for words that appear in only one decade) are replaced with zeros to avoid computational issues. * Normalizing Word Frequencies: The word counts are converted into probabilities by dividing each count by the total number of words in that decade. This normalization ensures that word usage is compared proportionally, rather than based on absolute counts. * Handling Logarithmic Calculations: Since logarithms are undefined for zero, words with a probability of 0 are replaced with a small constant (1e-10) to prevent errors in divergence calculations.

```
# Combine the top words from both decades with their frequency tables
# Create a dataset containing the union of all top words
# Join the 1840s word counts
# Join the 1850s word counts
# Replace missing counts with zeros so the two decades can be compared directly
word_dist <- data.frame(word = all_top_words) %>%
  left_join(freq_1840, by = "word") %>%
  left_join(freq_1850, by = "word") %>%
  replace_na(list(count_1840 = 0, count_1850 = 0))

# normalize raw word counts to probabilities by dividing each word's count
# by the total count from its respective decade. this prepares data
# for comparison using divergence or distance metrics.
word_dist <- word_dist %>%
  mutate(p_1840 = count_1840 / sum(count_1840),  # normalize 1840 counts
         p_1850 = count_1850 / sum(count_1850))  # normalize 1850 counts

# Taking the log of zero isn't allowed, so we replace any zeros with a tiny number.
```

```r
# This keeps the math stable without changing the results in any meaningful way.
word_dist$p_1840[word_dist$p_1840 == 0] <- 1e-10
word_dist$p_1850[word_dist$p_1850 == 0] <- 1e-10

head(word_dist)
```

```
##         word count_1840 count_1850     p_1840     p_1850
## 1        hon     135879     125465 0.07734936 0.06871748
## 2      house     115275     118861 0.06562050 0.06510046
## 3 government      75021      95417 0.04270584 0.05226012
## 4      noble      74062      73180 0.04215992 0.04008086
## 5    country      71454      64993 0.04067531 0.03559682
## 6       lord      67357      68942 0.03834309 0.03775970
```

With our word distributions calculated, we are now able to compute Partial JSD and discover the most significant changes between the two decades.

The following code does this by: * Computing Partial JSD: The script applies the Partial JSD functions to compute how much each word's probability in 1840 deviates from the midpoint distribution (1840 to 1850) and vice versa (1850 to 1840). This step captures asymmetrical shifts in word usage. * Assigning Direction of Change: Each word is assigned to only one direction of change, depending on which Partial JSD score is higher. If a word's usage declined after 1840, it is assigned "1840 to 1850" with a negative score. If a word became more common in 1850, it is assigned "1850 to 1840" with a positive score. * Selecting the Most Significant Changes: Finally, we sort words by their absolute Partial JSD score and selects the top 30 most dramatically shifting words.

```r
# Compute Partial Jensen-Shannon Divergence (JSD) for each word
# in both directions:
# - From the 1840 distribution to the 1850 distribution
# - And from 1850 to 1840
# This captures how much each word contributes to the overall
# divergence between decades, depending on the direction of comparison.
word_dist <- word_dist %>%
  rowwise() %>%
  mutate(Partial_JSD_1840 = partial_jsd_1840_to_1850(p_1840, p_1850),
         Partial_JSD_1850 = partial_jsd_1850_to_1840(p_1840, p_1850)) %>%
  ungroup()

# For interpretation, assign each word to a single dominant direction:
# - If its contribution is greater when comparing 1840 to 1850, it is labeled "1840 to 1850
# - Otherwise, it's labeled "1850 to 1840"
# Also, assign a signed Partial JSD value:
# - Negative if the word is more distinctive of 1840 (helps indicate direction visually)
# - Positive if more distinctive of 1850
word_dist <- word_dist %>%
  mutate(Direction = ifelse(Partial_JSD_1840 > Partial_JSD_1850,  # decide dominant directi
                            "1840 to 1850",
```

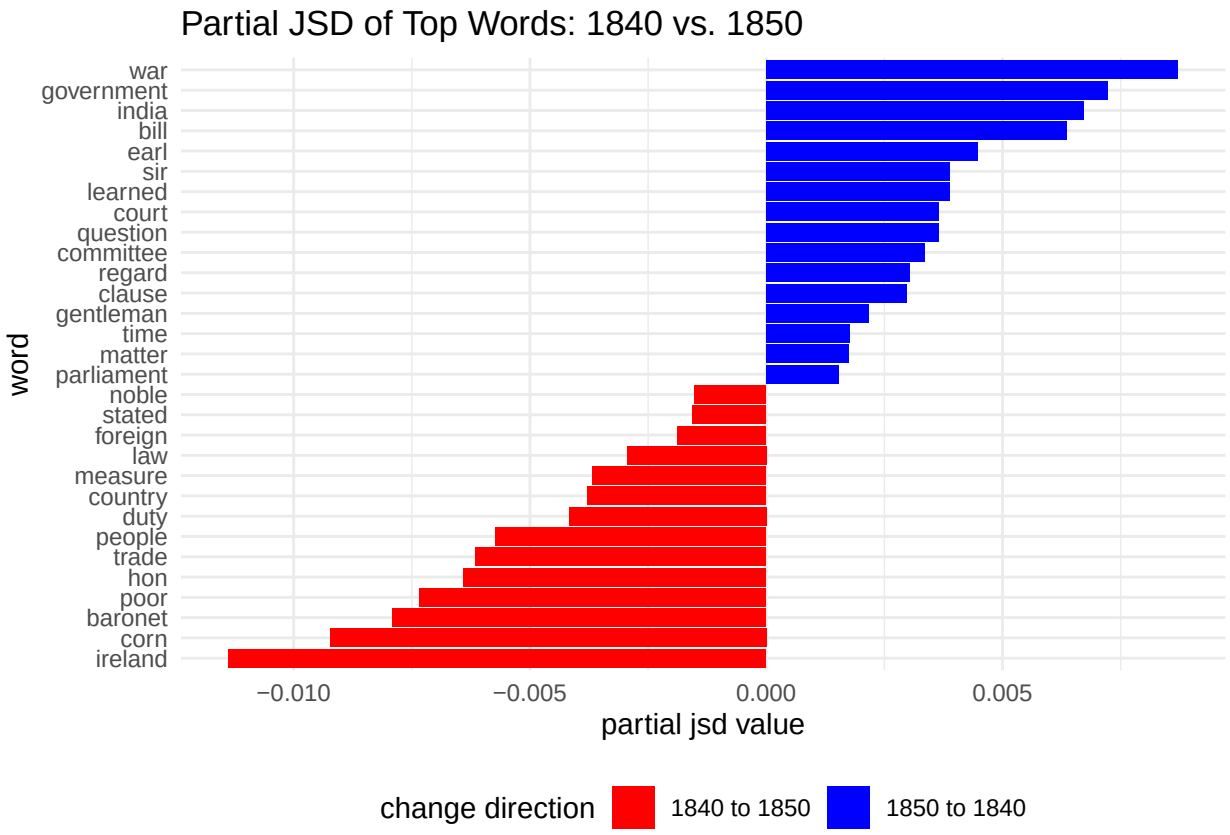
```
                              "1850 to 1840"),
         Partial_JSD = ifelse(Direction == "1840 to 1850",
                              -Partial_JSD_1840,  # negate to show shift from 1840
                               Partial_JSD_1850))  # keep positive for 1850-leaning

# Rank words by the magnitude of their contribution to divergence (i.e., absolute Partial
# and keep the top 30 most distinctive words across the two decades
word_dist <- word_dist %>%
  arrange(desc(abs(Partial_JSD))) %>%  # Sort by highest absolute contribution
  slice(1:30)                          # Keep top 30 words
```

Partial JSD can be interpreted as follows: * Higher Partial JSD: The word's frequency changed significantly over time (either increased or decreased). * Lower Partial JSD: The word remained relatively stable in frequency between the two time periods. * Partial JSD = 0: The word's probability did not change at all between the two time periods. If a word was never used in both periods, its probability is always 0.

One effective way of visualizing the results from Partial JSD is with a diverging bar chart, like in the following visualization. Higher Partial JSD values (e.g., war, india, government) indicate words whose relative importance changed significantly between 1840 and 1850 whereas lower Partial JSD values (e.g., noble, stated, foreign) suggest words that remained relatively stable in their frequency distribution across both decades.

```
# create a mirrored bar chart showing which words contribute most to linguistic shift
# between 1840 and 1850
# bars are flipped to show direction, based on partial jsd
ggplot(word_dist, aes(x = reorder(word, Partial_JSD), # order words by divergence
                      y = Partial_JSD, # y-axis is signed partial jsd
                      fill = Direction)) + # fill bars by direction label
  geom_bar(stat = "identity") + # draw bars with exact values
  coord_flip() + # flip axes for horizontal layout
  labs(title = "Partial JSD of Top Words: 1840 vs. 1850", # chart title
       x = "word", # x-axis label
       y = "partial jsd value", # y-axis label
       fill = "change direction") + # legend title
  theme_minimal() + # use a clean minimal theme
  scale_fill_manual(values = c("1840 to 1850" = "red", # set color for 1840 shift
                               "1850 to 1840" = "blue")) + # set color for 1850 shift
  theme(legend.position = "bottom") # move legend below plot
```



From this analysis, we can start to interpret what was trending and what was going away from the vantage point of an objective observer who had lived through hte years 1840 to 1859. On the up-and-up were Ireland, corn, trade, and duties, and discussions of baronets and the poor.

Less and less in evidence after 1850 were discussions of war, India, and the work of committees. Various formal language were passing, including addressing other members of parliament as "gentleman," talking about the "matter" at hand, or addressing each other as "sir."

The causes and implications of these changes are interesting to contemplate, and, as with much work in text mining, they require deeper reading of context in order to dive deeper. The point is that distance measurement lends us a metric to make concrete phenomena that are otherwise invisible – the pulse of rising and falling trends over time.

Exercises:

1) In chapter 3 we analyzed the top words of the Chancellors of the Exchequers for the 1830s by counting top words and visualizing them in a facet wrapped bar chart. This time use tf-idf to create this visualization, but this time using tf-idf as the measurement instead of count.

2) Take a single debate, such as the 1833 Debate on the Abolition of Slavery, and find the individual speakers. Use JSD to create comparisons between speakers. Remember to go back into the

debates themselves and closely read the text before making definitive claims. Refer to chapter 3 if you need a refresher on how to do this.

3) In chapter 2 and chapter 3, we learned how to use a controlled vocabulary to focus our analyses on subsets of the data that include a particular word of interest. Re-do the comparison of the decades 1830 and 1860, but this time filter the two decades for just debates that include the word "cotton." How does tf-idf obviate the need for stopwords lists? How does working with tf-idf result in a different interpretation than working with stopwords alone? Is one method more accurate than another, or does each algorithm contribute a different perspective on the truth?

4) Up to this point we have analyzed one-word and two-word utterances (e.g. tokens and bigrams). Are there limitations to this approach when it comes to making meaning from historical data?

Practicing with AI

Throughout *Text Mining for Historical Analysis*, we have primarily visualized data using bar charts. This is because bar charts are accessible and allow viewers to quickly apprehend patterns, comparisons, and trends. We can, however, use an AI chatbot to suggest different approaches to visualizing data. Such can be useful for exploratory data analysis and considering alternative forms of representation–such as line graphs for temporal trends or word clouds for lexical frequency. However, AI-generated suggestions are not truly aware of research contexts, so the generated results should be critically explored and vetted by the analyst.

In the following examples, we used suggestions from an AI chatbot to explore alternative ways of visualizing our JSD data. To ensure the AI-generated code was compatible with our existing dataset, we included the code we had been using to create the JSD data frame in our prompt. This helped align the chatbot's responses with the structure and content of our data, so the plots it produced would work correctly. With this context the AI chatbot can generate suggestions that take into account the structure of our data, the libraries we were using (if necessary), and the types of historical patterns we were interested in highlighting. This allowed us to rapidly experiment with new forms of visualization while staying grounded in the context of our existing work.

Analyst Prompt

I calculated partial JSD using the following code. Can you suggest multiple ways of visualizing the results in R? Tell me why you have chosen each visualization.

```
word_dist <- word_dist %>%
  rowwise() %>%
  mutate(
    Partial_JSD_1840 = partial_jsd_1840_to_1850(p_1840, p_1850),
    Partial_JSD_1850 = partial_jsd_1850_to_1840(p_1840, p_1850)) %>%
  ungroup()

word_dist <- word_dist %>%
  mutate(
   Direction = ifelse(Partial_JSD_1840 > Partial_JSD_1850, "1840 to 1850", "1850 to 1840"),
   Partial_JSD = ifelse(Direction == "1840 to 1850", -Partial_JSD_1840, Partial_JSD_1850))

word_dist <- word_dist %>%
  arrange(desc(abs(Partial_JSD))) %>%
  slice(1:30)
```

The chatbot responded with multiple different options for visualizing the data, as shown below. Some of these suggestions are new to us–like a "density ridge plot" or a "sankey diagram"–while others might already be familiar to us, like a "word cloud."

ChatGPT-4o Response

If you have many words and want to see distribution trends, use a Density Ridge Plot. If you want a text-heavy, intuitive view, go for a Word Cloud. If you want a precise, labeled view of shifts, use a Dot Plot. If you want to visually depict flows between decades, use a Sankey Diagram.

We will explore just the density ridge plot, word cloud, and dot plot.

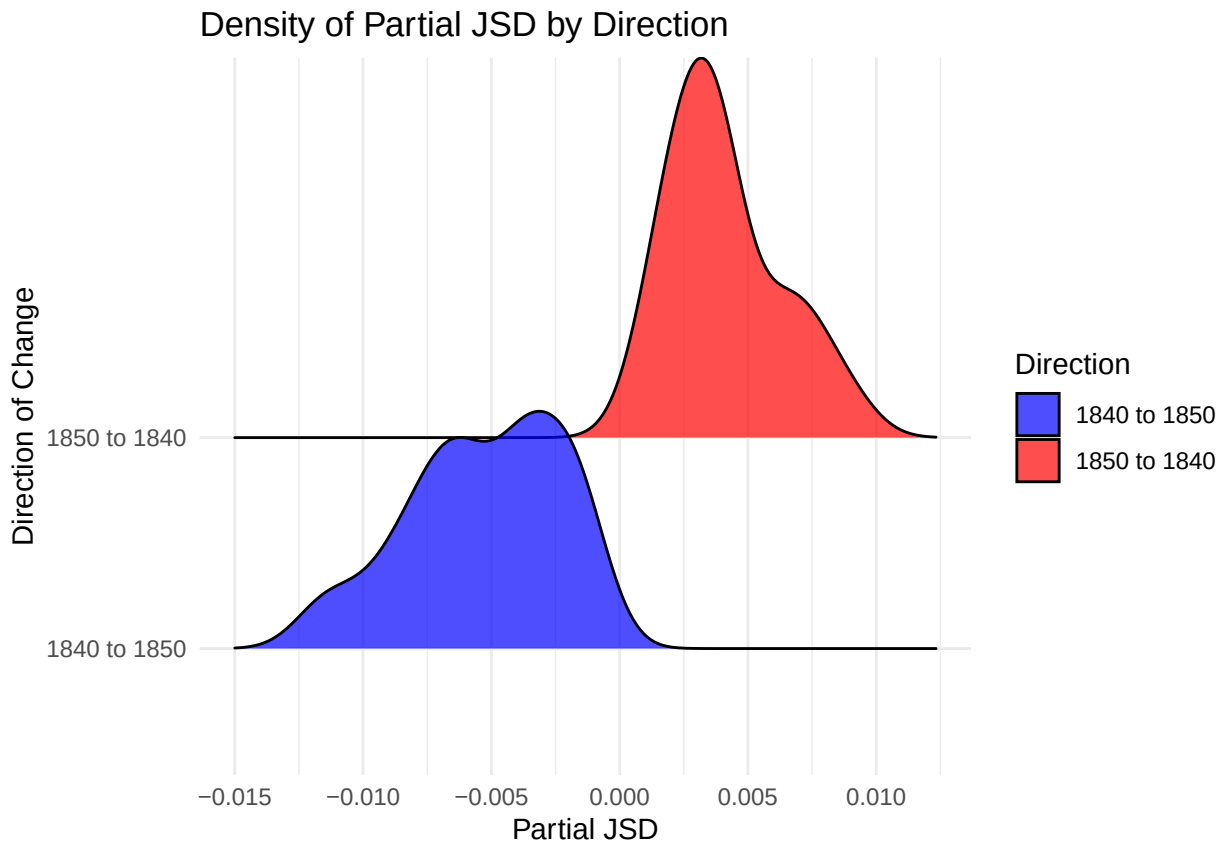
A density ridge plot may be an unfamiliar term, as it has not been previously used in this book. Asking the AI chatbot to explain its suggestion reveals that the plot is used to display the full distribution of Partial JSD values across the dataset, helping to highlight whether the shifts are symmetric or skewed in one direction. While the explanation is not true reasoning but rather a probabilistic output, it happens to align with commonly accepted knowledge—something we confirm by consulting a secondary source (Wilke 2019). However, as we will demonstrate in Chapter 5, AI-generated rationales can often be unreliable in the context of digital history.

The chatbot generated the following code for a density ridge plot.

```
library("ggridges")

ggplot(word_dist, aes(x = Partial_JSD, y = Direction, fill = Direction)) +
  geom_density_ridges(alpha = 0.7) +
  scale_fill_manual(values = c("1840 to 1850" = "blue",
```

```
                                "1850 to 1840" = "red")) +
  labs(title = "Density of Partial JSD by Direction",
       x = "Partial JSD",
       y = "Direction of Change") +
  theme_minimal()
```



The plot is demonstrating how frequently values–such as JSD values–occur across a range , using smooth curves rather than raw counts, as in a bar chart.

Smoothing helps reveal the overall shape of the distribution, making the plot useful for identifying patterns such as skew or symmetry. However, this type of visualization sacrifices detail at the individual word level, making it less effective for examining word-specific differences.

While using a chatbot to brainstorm can be helpful, simply copying and pasting the code it generates —- without understanding how it shapes our analysis —- can lead to misleading results. For example, a density ridge plot assumes that the two distributions being compared (such as word usage from 1840 to 1850 and from 1850 to 1840) have similar shapes and ranges. This assumption holds in our case, because we limited the analysis to the top 50 words from each decade. But in other contexts—especially when looking at all words—this assumption may break down. If one decade has significantly more words than the other, the plot might distort the comparison, making one group appear more prevalent than it really is.

The chatbot generated the following code for a word cloud. Its generated rationale for doing so was to provide a more "intuitive" overview of the words that had the greatest shifts between decades. In this visualization, the red words shifted the most from 1850 to 1840 (indicating a higher Partial JSD in that direction) whereas the blue words shifted the most from 1840 to 1850. Words that appear more prominently had larger changes. In this example, "Ireland" had a greater change than "stated."

```
library("wordcloud")

wordcloud(
  words = word_dist$word,
  freq = abs(word_dist$Partial_JSD),
  colors = ifelse(word_dist$Direction == "1840 to 1850", "blue", "red"),
  scale = c(4, 0.5),
  random.order = FALSE)
```

The advantage of this visualization is that it is understood at a glance. However, it lacks quantitative precision. Unlike the bar charts we created above, the word cloud does not show exact values–only relative importance of words against one another. In addition, the visualization can be misleading as the difference between the sizes of each word may not be proportional to the underlying data. Indeed, word clouds may be highly subjective or misread. The random placement of words, for

example, might have an impact on an analyst's interpretation where a word near the center might look more important than a word at the edge of the cloud, even if this is not true.

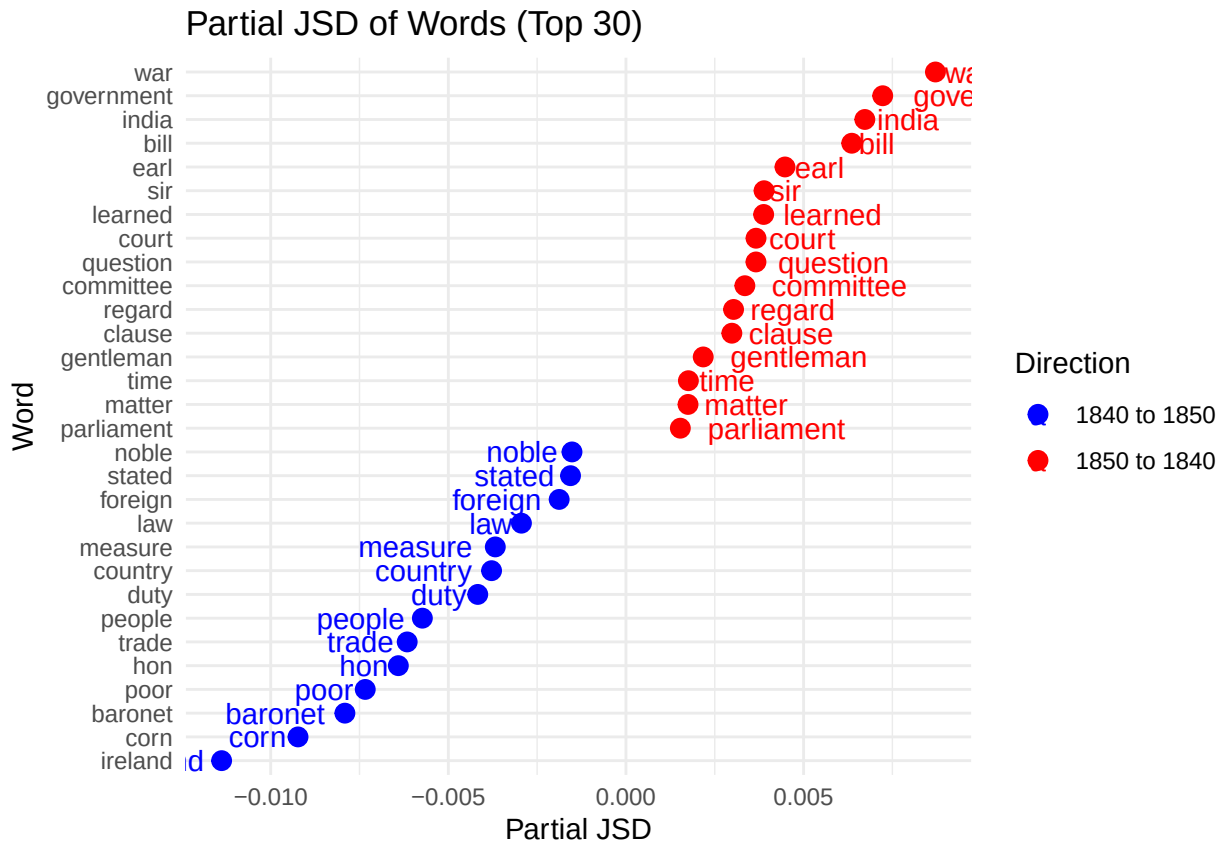
The last option we will explore is the dot plot. The chatbot generated the following code for a dot plot. The visualization appears to communicate the same information as the diverging bar chart we created at the end of the chapter. The left side of the plot shows the words that shifted the most from 1840 to 1850, and the right side shows the words that shifted the most from 1850 to 1840.

In this case, however, the data is represented using a dot plot. Arguably, the dot plot does not provide a significant advantage over the diverging bar chart. Unlike the Density Ridge Plot and the Word Cloud, which offer new perspectives by emphasizing different linguistic features and drawing the viewer's eye to specific patterns, the dot plot does not introduce additional insights beyond what we have already observed at the end of this chapter.

Instead, the dot plot may present more drawbacks than benefits. One potential issue is readability—some labels extend beyond the plot's boundaries, making the visualization harder to interpret at a glance. While this can be adjusted through formatting, it highlights a limitation of the default code provided by the chatbot, which would require additional tweaking by the analyst to improve its presentation.

Perhaps the most critical concern for readability is the absence of a central reference point. As mentioned, Partial JSD relies on a clearly defined midpoint to illustrate distributional shifts. Partial JSD quantifies change by comparing how probability distributions diverge from this central reference point. Unlike the diverging bar chart, which includes a middle bar to visually anchor the comparison, the dot plot lacks this structured reference, making it harder to discern patterns of deviation at a glance. This omission forces the analyst to rely solely on the relative positioning of individual data points, which can be straining to the analyst and may reduce the effectiveness of the visualization for communicating meaningful shifts.

```
ggplot(word_dist, aes(x = Partial_JSD,
                      y = reorder(word, Partial_JSD),
                      color = Direction)) +
  geom_point(size = 3) +
  geom_text(aes(label = word),
            hjust = ifelse(word_dist$Partial_JSD > 0, -0.2, 1.2)) +
  scale_color_manual(values = c("1840 to 1850" = "blue", "1850 to 1840" = "red")) +
  labs(title = "Partial JSD of Words (Top 30)",
       x = "Partial JSD",
       y = "Word") +
  theme_minimal()
```



In sum, brainstorming with chatbots can be a valuable approach to exploratory data analysis. In this example, analysts can quickly generate and explore multiple visualization options, some of which may highlight patterns or approaches to data interpretation that might not have been considered otherwise. However, each suggestion must be evaluated with care. While certain visualizations can be an asset, others may be misleading.