

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

Up until now, we have mostly focused on numeric measurements, such as word counts. In this chapter, we shift our attention to statistical methods. Specifically, we explore two key concepts: 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). A common method for analyzing distinctiveness, which we will explore in this chapter, is “term frequency-inverse document frequency” abbreviated to TF-IDF. Distance measurements are typically used to measure similarity or dissimilarity between data. Just a few of these distance measurements include Jensen-Shannon Divergence (JSD), Euclidean distance, or cosine similarity.

Here is a more detailed description of distinctiveness compared against distance:

Distinctiveness measures score the relationships between words and documents across two or more sets of documents. These measures can identify words that are characteristic of a specific author, document, year, or political group. Take for example Dr. Seuss, a popular American author for children’s books. Dr. Seuss, who trained as a linguist, often uses distinctive words (Seuss 1960). He has a strong preference for onomatopoeic words—words that imitate sounds—such as “plop,” “drizzle,” and “hoot.” He also made up an imaginary lexicon to describe his fictional universe. If we have two sets of documents – one of them consisting books by Dr. Seuss, and one of them consisting of books by a different American author – distinctiveness algorithms will identify the words that are most likely to appear in the Dr. Seuss documents but not other documents. This would result in a list of words that includes “Zizzer-zazzer-zuzz” (a mysterious creature that likely emits a noise that sounds like its name), the “Lerkim” (a beachside shack appropriate for lurking), the “Obsk” (a mountain bird found in a fictional zoo), and the many other creatures that live in Dr. Seuss’s universe. Indeed, most of these words have never been used in publications that are not authored by Dr. Seuss himself. If we wanted a complete list of the words used by Dr. Seuss and no one else, we would use a distinctiveness algorithm such as TF-IDF or log likelihood, which scores unique words with a high value—suggesting they are unique to Dr. Seuss—and shared words with a low value, suggesting they are common across both authors.

Measuring distance, on the other hand, can provide perspective into different types of questions. Suppose that, instead, we want to measure how linguistically different a Dr. Seuss book, *Green Eggs and Ham*, is from a children’s book written by a different author, such as Margaret Wise Brown’s *Goodnight Moon* (Brown 1947). Using distance, we can compare their word distributions. We would do this by first representing each book as a frequency of all the words they contain (e.g., “ham,” “green,” “eggs” for *Green Eggs and Ham* and “moon,” “bowl,” “mouse” for *Goodnight Moon*). To measure the distance between the two books we can “vectorize” the text, a process where each word in a text is transformed into a numerical value corresponding to its frequency. Afterwards, we can compute the distance between word vectors using a metric such as Cosine similarity. A low cosine similarity would indicate that *Green Eggs and Ham* and *Goodnight Moon* are linguistically dissimilar. Now, if we were to compare *Green Eggs and Ham* to another Dr. Seuss book, for example, *The Cat in the Hat*, the calculated distance would be smaller, reflecting the stylistic and lexical similarities between the two books.

Now, say we had a large repository of books and we wanted to discover which ones use vocabularies that are similar to Dr. Seuss's. We could look for books with a similar ratio of sonorous words compared to non-sonorous words. For example, we might want to find the top ten authors who use onomatopoeic language. We might also be interested in how many of their words overlap with Dr. Seuss's and would rely on an algorithm to identify which words are used with similar frequencies by both authors. In this case, we need a measure that captures word-document relationships with nuance. To achieve this, we could measure the "distance" between the word counts in Dr. Seuss's publications and those in other works.

In sum, distance is for quantifying how far apart texts are in terms of their overall patterns, rather than identifying which specific features (like "Zizzer-zazzer-zuzz") make a text unique. While distinctiveness isolates the unique language features of a writer's lexicon, 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 historical thinking by foregrounding the importance of the process in which we produce historical knowledge. The ways in which we decide to process data and measure it—whether through distance or distinctiveness—are guided by our questions. Yet, the choices made during data processing shape the interpretive angles of the corpus that become available to us. In this way, the computational steps used during data processing are as much about the creation of knowledge as is the act of historical analysis, as the method we choose mediates our access to historical records, and its affordances and limitations may require us to shift our analytical focus—for example, from highlighting distinctiveness as a sign of uniqueness in a corpus, to using distance as a measure of relational patterns, or to selecting a different metric better suited to the historical data at hand. These methodological decisions are interpretive choices in their own right as they shape how we understand and conceptualize data.

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 answer 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.

Term Frequency-Inverse Document Frequency (TF-IDF)

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.

On the other hand, an analyst might choose instead 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.

As another example, an analyst could use distinctiveness measures to analyze political parties and their most telling words. By comparing the vocabulary used by members of different parties, the analyst might uncover terms that are 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. Doing this analysis by decade could inform research on how those identities shifted over time in response to political events or ideological realignments.

In sum, 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.

Among the most common algorithms used to understand distinctiveness is "term frequency-inverse document frequency," shortened to "TF-IDF" (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 Spärck-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 using TF-IDF

We will start this chapter by measuring the distinctiveness of language in the 1830s (which saw a total of nearly 8,000 debates) compared with language in the 1860s (which saw a total of nearly 11,000 debates), and vice-versa, using TF-IDF.

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.

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

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

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.

```
# 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)
```

```
head(hansard_data)
```

```
##      sentence_id
##      <char>
## 1: S2V0022P0_0
## 2: S2V0022P0_1
## 3: S2V0022P0_2
## 4: S2V0022P0_3
## 5: S2V0022P0_4
## 6: S2V0022P0_5
##
##
## 1:
## 2:
## 3:
## 4:
## 5: This nation is too much involved in the general interests of Europe not to view with satisfaction
## 6:
##      decade
##      <num>
## 1: 1830
## 2: 1830
## 3: 1830
## 4: 1830
## 5: 1830
## 6: 1830
```

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.

```
# tokenize the 'text' column into individual words,
# remove stop words, and filter out tokens that contain digits
tokenized_hansard_data <- hansard_data %>%
  unnest_tokens(word, text) %>% # break text into one word per row
  anti_join(stop_words) %>% # remove stop words like 'the' or 'and'
  filter(!str_detect(word, "[:digit:]")) # remove words with digits
```

```
head(tokenized_hansard_data)
```

```
##      sentence_id decade      word
##      <char>    <num>    <char>
```

```
## 1: S2V0022P0_0 1830 rose
## 2: S2V0022P0_0 1830 lords
## 3: S2V0022P0_0 1830 rising
## 4: S2V0022P0_0 1830 move
## 5: S2V0022P0_0 1830 humble
## 6: S2V0022P0_0 1830 address
```

Before using the TF-IDF function, `bind_tf_idf()`, we need to do a little more data processing. First, we need to count the individual words per decade and then we need to find the total word count for each decade. This will help us find the term frequency and the inverse document frequency.

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.

```
# count how often each word appears per decade,
# sort by frequency, and remove words that appear only once
hansard_count <- tokenized_hansard_data %>%
  count(decade, word, sort = TRUE) %>% # count word frequency by decade
  filter(n > 1) # keep words that appear more than once

# calculate the total number of word occurrences per decade
total_count <- hansard_count %>%
  group_by(decade) %>% # group by decade
  summarize(total = sum(n)) # sum word counts for each decade
```

At this point we now have two data frames that we will use for our analysis of distinctiveness: `hansard_count` and `total_count`.

`hansard_count` is a data frame with words and their count by each decade, 1830 and 1860.

```
head(hansard_count)
```

```
##   decade      word      n
##   <num>    <char> <int>
## 1:  1860      hon 125781
## 2:  1830     house 125218
## 3:  1830      hon 121494
## 4:  1860     house 113329
## 5:  1860 government 89197
## 6:  1830     noble 86712
```

total_count is a data frame with the total number of words per decade.

```
head(total_count)
```

```
## # A tibble: 2 x 2
##   decade  total
##   <dbl>   <int>
## 1   1830 9039333
## 2   1860 9602786
```

We can use a left join to merge these two dataframes into a single one to be passed to `bind_tf_idf()`.

```
# join total word counts per decade to individual word counts
hansard_words <- left_join(hansard_count, total_count) # add total count to each word entry

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

```
head(hansard_tf_idf)
```

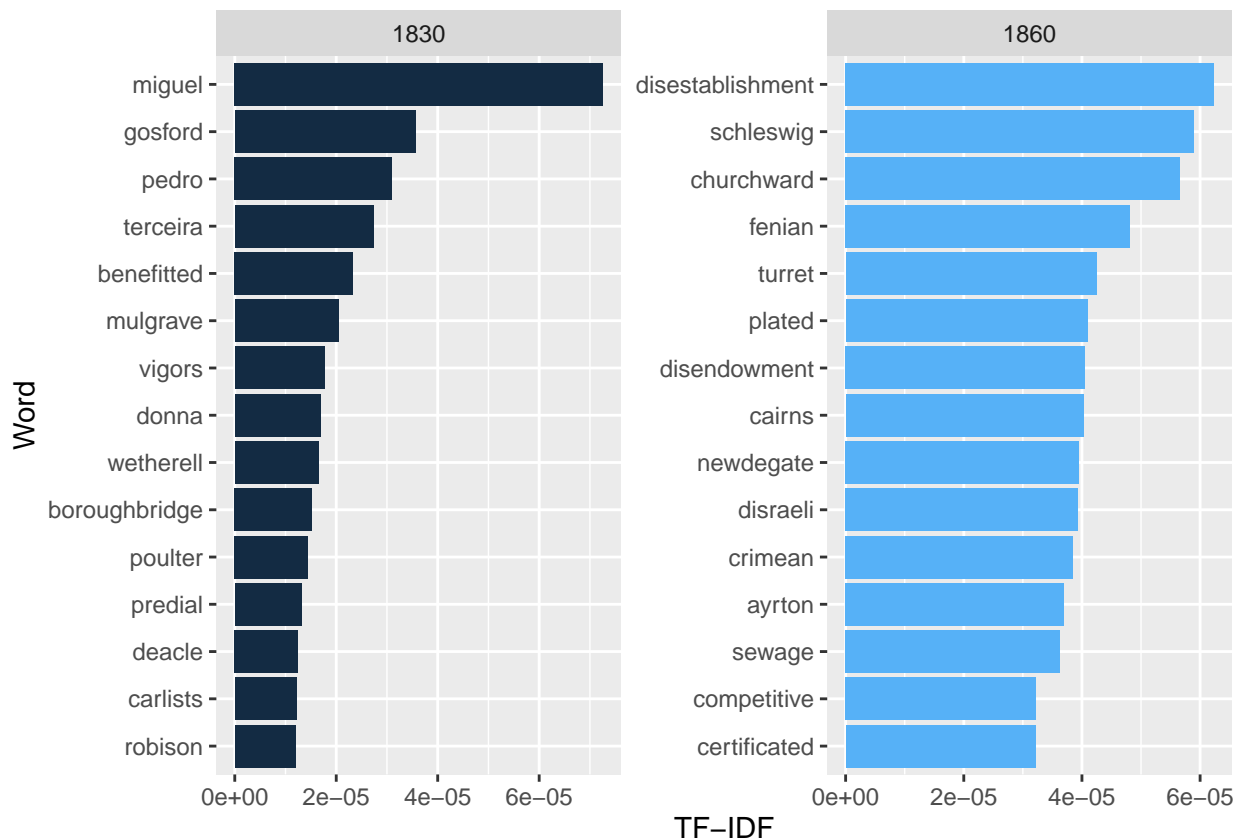
```
##   decade      word      n    total      tf    idf  tf_idf
##   <num>    <char> <int>   <int>    <num> <num>  <num>
## 1:   1860      hon 125781 9602786 0.013098386    0    0
## 2:   1830     house 125218 9039333 0.013852571    0    0
## 3:   1830      hon 121494 9039333 0.013440593    0    0
## 4:   1860     house 113329 9602786 0.011801679    0    0
## 5:   1860 government 89197 9602786 0.009288659    0    0
## 6:   1830     noble 86712 9039333 0.009592743    0    0
```

We can now analyze the results of our TF-IDF function to see which words are distinctive of the 1830s compared to the 1860s (and vice-versa).

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

# plot tf-idf scores by decade
ggplot(hansard_tf_idf, # create plot object
  aes(tf_idf, # x-axis is tf-idf
      reorder(word, tf_idf), # y-axis is word ordered by tf-idf
      fill = decade)) + # fill bars by decade
  geom_col(show.legend = FALSE) + # draw bars, hide legend
  facet_wrap(~decade, # separate panels by decade)
```

```
ncol = 2, # two columns of panels
scales = "free") + # allow free y-scales
labs(x = "TF-IDF", # label x-axis
y = "Word") # label y-axis
```



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 Wales, Australia, is given its name after the 2nd Earl of Gosford, a friend of the governor of New South Wales.

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-Holstein. “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.

In historical research, however, an analyst may be weary of applying a distinctiveness algorithm to two decades at a remove from each other like 1830 and 1860. This is because this approach gives an overall metric of historical difference between the two periods, but not a precise sense of when, between 1830 and 1869, each new term rose to prominence. If we care about specificity with regards to historical dates, we may instead apply TF-IDF to each decade between 1830 and 1860, measuring the distinctiveness of each decade against the previous decade in the century.

The following code loads the 1840 and 1850 Hansard debates and combines them into a single dataframe 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
hansard_1850 <- hansard_1850 %>%
  mutate(decade = 1850)

# Append the 1840s and 1850s data to the existing hansard_data
hansard_data <- 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 dataframe, `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.

```
# 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_count <- 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)
  hansard_decade <- hansard_data %>%
    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_hansard_data <- hansard_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
  hansard_decade_count <- tokenized_hansard_data %>%
    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_count <- bind_rows(hansard_count, hansard_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.

```
head(hansard_count)
```

```

##   decade   word      n
## 1   1830  house 125218
## 2   1830   hon 121494
## 3   1830  noble  86712
## 4   1830  bill  75930
## 5   1830  lord  66550
## 6   1830 country 60592

```

The following code calculates the total word count for each decade in the `hansard_count` dataset. It groups the data by decade and sums the word frequencies (`n`) for each group, creating a new data frame, `total_count`, with one row per decade and the total word count. This summarizes the dataset's text volume across time periods. `total_count` provides the total word count for each decade, which is later used in the TF-IDF calculation.

```
# compute the total number of word occurrences per decade by grouping
# the word count data and summing all frequencies for each decade
total_count <- hansard_count %>%
  group_by(decade) %>% # group by decade
  summarize(total = sum(n)) # sum word counts
```

```
head(total_count)
```

```
## # A tibble: 4 x 2
##   decade  total
##   <dbl>   <int>
## 1   1830 8901882
## 2   1840 9055272
## 3   1850 9674488
## 4   1860 9444854
```

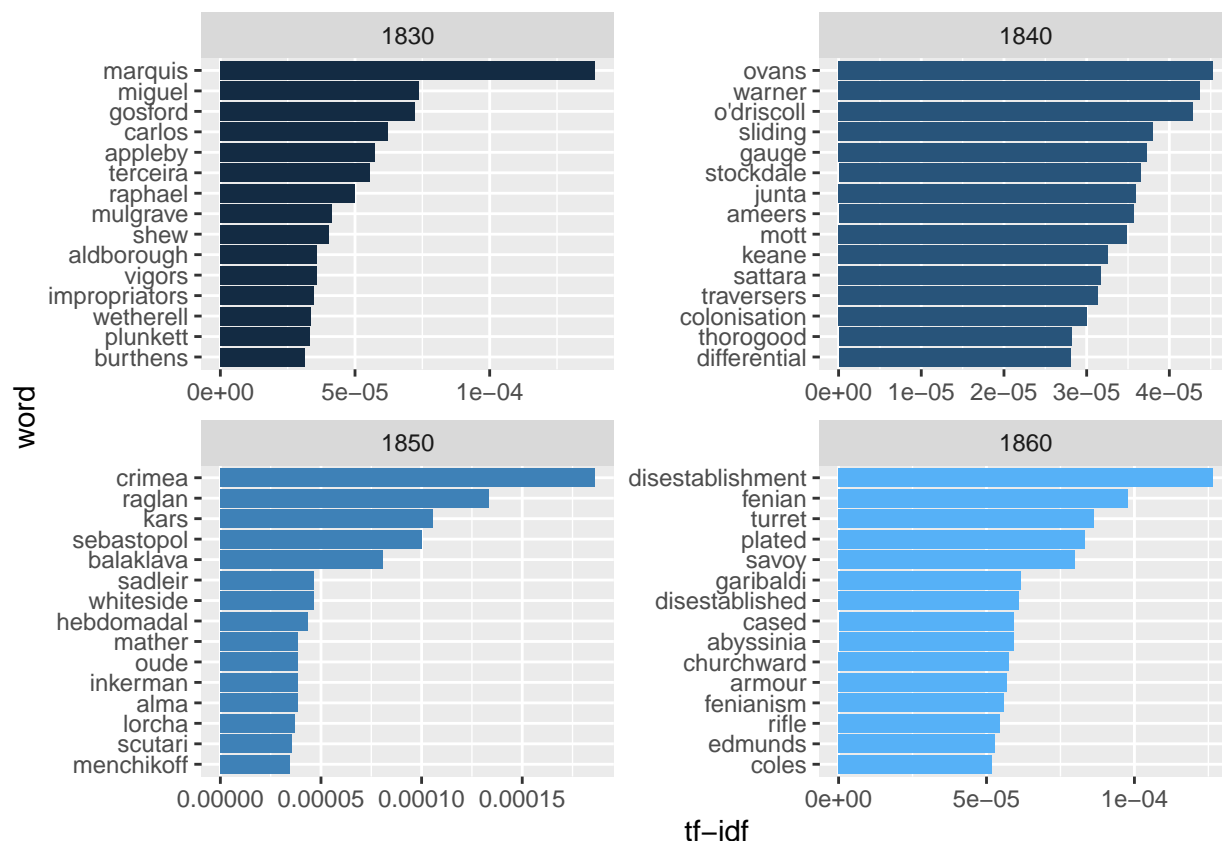
We can now apply TF-IDF and visualize our results, in this case, the first 15 words that appear for each decade.

```
# join total word counts to individual word counts by decade
hansard_words <- left_join(hansard_count, total_count) # merge total count into word-level data
```

```
## Joining with 'by = join_by(decade)'
```

```
# compute tf-idf values and keep top 15 words by tf-idf for each decade
hansard_words <- hansard_words %>% # start with merged word data
  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

# create a faceted bar plot showing the top tf-idf words by decade
ggplot(hansard_words, # build plot from tf-idf data
  aes(tf_idf, # x-axis is tf-idf score
    reorder(word, tf_idf), # order words by tf-idf for clean y-axis
    fill = decade)) + # color bars by decade
  geom_col(show.legend = FALSE) + # draw bars without legend
  facet_wrap(~decade, # create panel per decade
    ncol = 2, # arrange panels in 2 columns
    scales = "free") + # allow free y-axis scaling
  labs(x = "tf-idf", y = "word") # add axis labels
```



By analyzing each decade (the 1830s, 40s, 50s, and 60s), we can see that Benjamin Disraeli rose to prominence in the 1850s, not the 1860s. Several words in the 1850s bar chart are associated with the Crimean War (1853–1856), a conflict in which Britain, France, and the Ottoman Empire fought against Russia. During this period, Disraeli, a leading Tory statesman, was highly critical in Parliamentary debates, particularly regarding the government’s mismanagement of the war effort under Prime Minister Lord Aberdeen. Lord Aberdeen (George Hamilton-Gordon, 4th Earl of Aberdeen) served as Britain’s Prime Minister from 1852 to 1855, leading a coalition government during the Crimean War. His administration faced widespread criticism for failing to provide adequate supplies, including medical care.

We can see Disraeli’s connection to the 1850s through words like Crimea, Sebastopol, Balaklava, Inkerman, and Scutari, all of which are linked to the Crimean War. The Crimean Peninsula was the main battleground, with major conflicts such as the Siege of Sebastopol, a prolonged struggle to capture Russia’s key naval base; the Battle of Balaklava, famous for the Charge of the Light Brigade; and the Battle of Inkerman, fought in dense fog and won by British and French forces. Scutari, in modern-day Turkey, housed a British military hospital. These locations were at the core of British political debates, where Disraeli strongly criticized the government’s mismanagement of the war.

Lord Raglan, the British commander during the Crimean War, was a frequent subject of debate among Disraeli and his conservative peers in the House of Commons. James Sadleir, a politician involved in a financial scandal, and James Whiteside, an Irish Conservative MP and lawyer, may also be referenced here. While Sadleir had no direct ties to Disraeli, Whiteside was part of the broader Tory faction. Additionally, Prince Alexander Menshikov, a Russian general in the Crimean War, played a significant role in Disraeli’s

Parliamentary discussions on Britain's foreign policy and its stance against Russia.

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] "d"                "decades"          "hansard_1850"  
## [4] "hansard_1860"     "hansard_count"    "hansard_data"  
## [7] "hansard_decade"   "hansard_decade_count" "hansard_tf_idf"  
## [10] "hansard_words"    "stop_words"       "tokenized_hansard_data"  
## [13] "total_count"
```

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

```
## [1] "d"                "decades"          "hansard_1850"  
## [4] "hansard_1860"     "hansard_count"    "hansard_data"  
## [7] "hansard_decade"   "hansard_decade_count" "hansard_tf_idf"  
## [10] "hansard_words"    "stop_words"       "tokenized_hansard_data"  
## [13] "total_count"
```

```
# 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.

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

```
##           used (Mb) gc trigger (Mb) max used (Mb)  
## Ncells  2355222 125.8  14742596  787.4 18428244  984.2  
## Vcells 12489079  95.3  406785378 3103.6 633799275 4835.6
```

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".

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

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.

```
# 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
```

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")
```

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 process 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.

```

# merge the top words from both decades with their frequency tables,
# ensuring that missing words are filled with zero counts to enable comparison
word_dist <- data.frame(word = all_top_words) %>% # create dataset with union of top words
  left_join(freq_1840, by = "word") %>% # join 1840s word counts
  left_join(freq_1850, by = "word") %>% # join 1850s word counts
  replace_na(list(count_1840 = 0, count_1850 = 0)) # replace missing counts with 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

# The logarithm of zero (log(0)) is undefined in mathematics—it approaches negative infinity.
# In practice, trying to compute log(0) in R will result in -Inf or NaN, which can break downstream cal
# To avoid this, we replace zero probabilities with a very small constant (1e-10),
# which approximates zero but keeps the log function well-defined.
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 0.07734905 0.06871748
## 2    house    115275    118861 0.06562023 0.06510046
## 3 government    75021     95417 0.04270567 0.05226012
## 4     noble    74063     73180 0.04216032 0.04008086
## 5   country    71454     64993 0.04067515 0.03559682
## 6     lord    67357     68942 0.03834294 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 → 1850) and vice versa (1850 → 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 → 1850" with a negative score. If a word became more common in 1850, it is assigned "1850 → 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.

```
# 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), # Contribution from 1840
         Partial_JSD_1850 = partial_jsd_1850_to_1840(p_1840, p_1850)) %>% # Contribution from 1850
  ungroup()

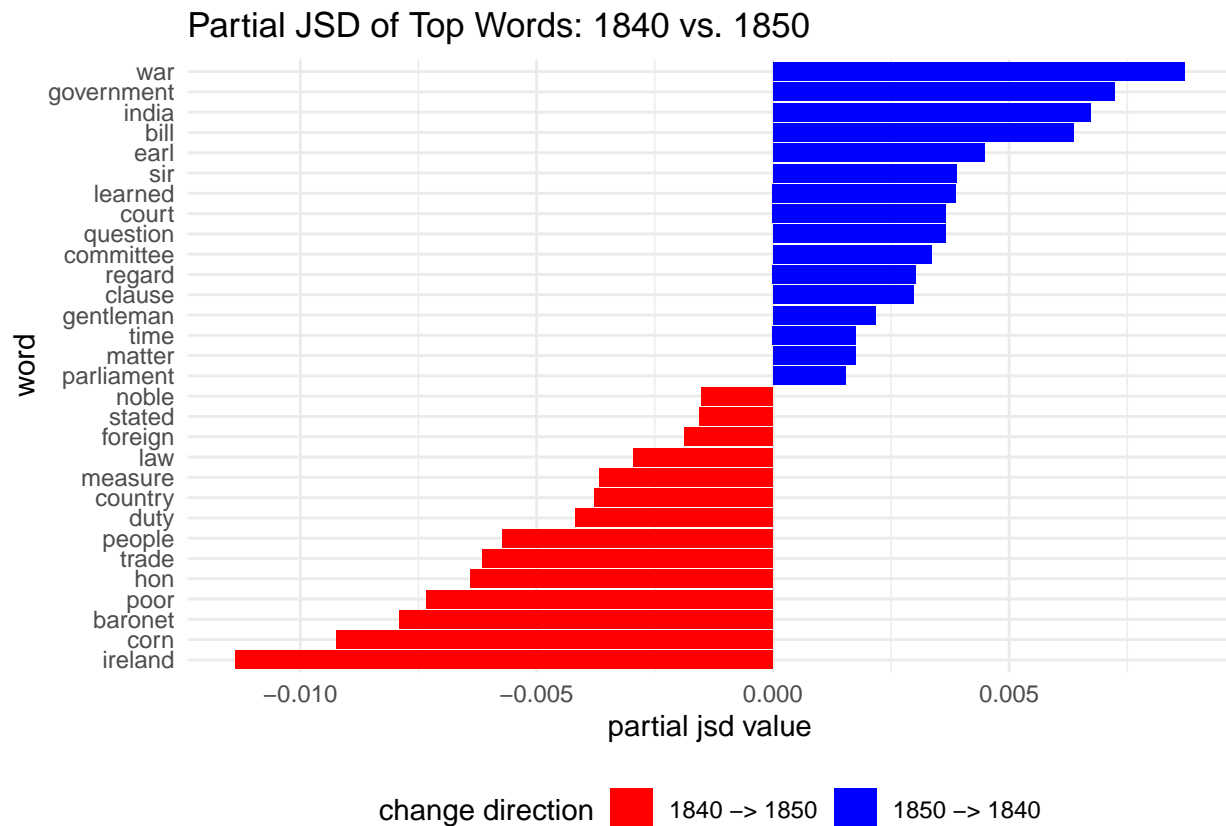
# For interpretation, assign each word to a single dominant direction:
# - If its contribution is greater when comparing 1840 → 1850, it is labeled "1840 → 1850"
# - Otherwise, it's labeled "1850 → 1840"
# Also, assign a signed Partial JSD value:
# - Negative if the word is more distinctive of 1840 (helps indicate direction visually)
# - Positive if more distinctive of 1850
word_dist <- word_dist %>%
  mutate(Direction = ifelse(Partial_JSD_1840 > Partial_JSD_1850, # decide dominant direction
                           "1840 → 1850",
                           "1850 → 1840"),
         Partial_JSD = ifelse(Direction == "1840 → 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 JSD)
# 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 → 1850" = "red", # set color for 1840 shift
    "1850 → 1840" = "blue")) + # set color for 1850 shift
  theme(legend.position = "bottom") # move legend below plot
```



Need a summation of this from Jo.

As discussed in *The Dangerous Art of Text Mining*, techniques such as the application of distinctiveness to find the words most unique to a year or a decade can be used in combination with a controlled vocabulary or limits on the frequency of words to obtain a reasonable timeline of those matters of debate, ideas, and events that marked out each period of time over a century. If the results are unsurprising for nineteenth-century Britain, this lack of surprise only validates the fact that distinctiveness methods may be useful for understanding datasets such as Reddit or Twitter over recent years, when the chronology of what happened is less familiar to historians. Tools of this kind

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. Do you note differences in the words that are most likely to appear in one speaker's lexicon but not another's? 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 focusing the data like this inform our interpretation of it?
- 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 dataframe 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 → 1850", "1850 → 1840"),
    Partial_JSD = ifelse(Direction == "1840 → 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.

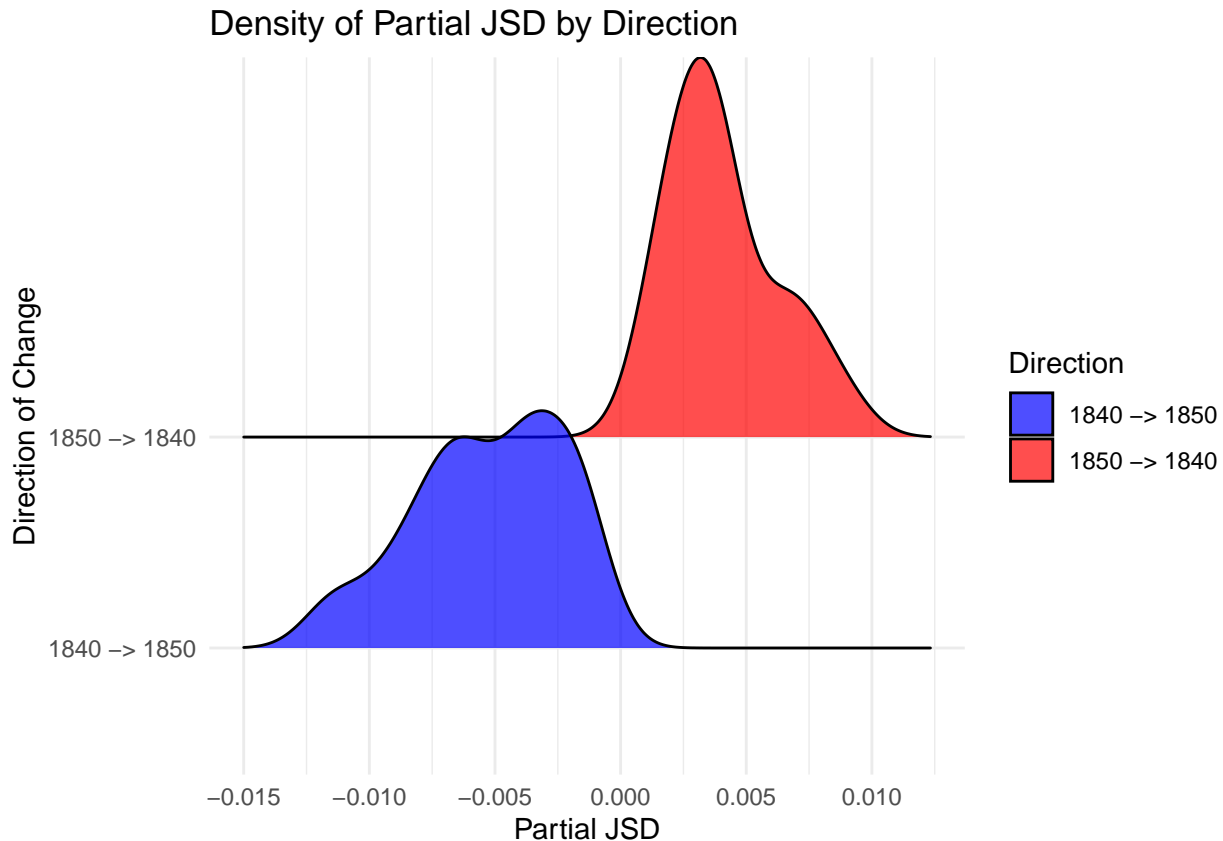
We will explore just the density ridge plot, word cloud, and dot plot.

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. 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. This 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.

```
library(ggbridges)

ggplot(word_dist, aes(x = Partial_JSD, y = Direction, fill = Direction)) +
  geom_density_ridges(alpha = 0.7) +
  scale_fill_manual(values = c("1840 → 1850" = "blue", "1850 → 1840" = "red")) +
  labs(title = "Density of Partial JSD by Direction",
       x = "Partial JSD",
       y = "Direction of Change") +
  theme_minimal()
```



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 -> 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.

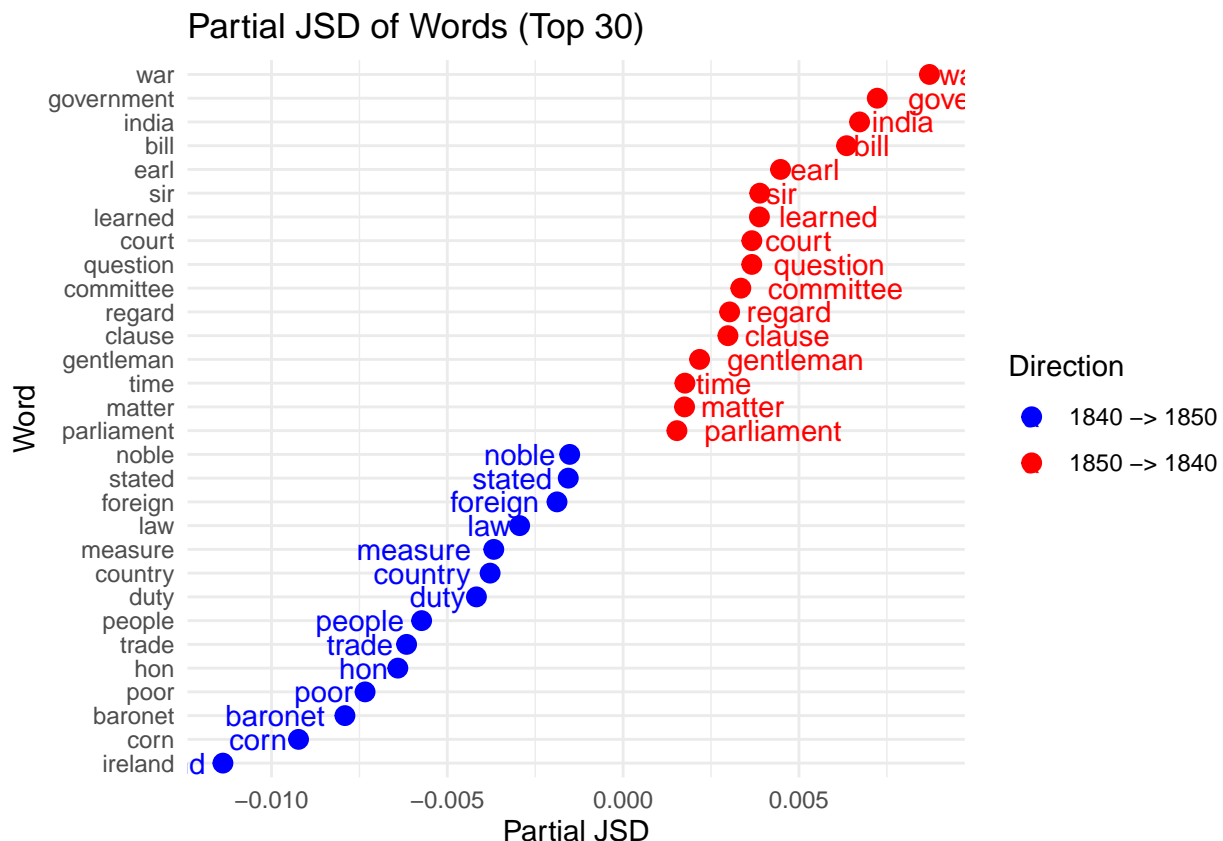
In this case, however, the data is represented using a dot plot. Arguably, the dot plot does not provide a significant advantage over the diverging bar chart. Unlike the Density Ridge Plot and the Word Cloud, which offer new perspectives by emphasizing different linguistic features and drawing the viewer’s 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 → 1850" = "blue", "1850 → 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.