Today We are so pleased to introduce a new package for calculating weighted log odds ratios, **tidylo**.

Often in data analysis, we want to measure how the usage or frequency of some **feature**, such as words, differs across some group or **set**, such as documents. One statistic often used to find these kinds of differences in text data is tf-idf.

tf-idf

A central question in text mining and natural language processing is how to quantify what a document is about. Can we do this by looking at the words that make up the document? One measure of how important a word may be is its term frequency (tf), how frequently a word occurs in a document, as we examined in Chapter 1. There are words in a document, however, that occur many times but may not be important; in English, these are probably words like “the”, “is”, “of”, and so forth. We might take the approach of adding words like these to a list of stop words and removing them before analysis, but it is possible that some of these words might be more important in some documents than others. A list of stop words is not a very sophisticated approach to adjusting term frequency for commonly used words.

Another approach is to look at a term’s inverse document frequency (idf), which decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents. This can be combined with term frequency to calculate a term’s tf-idf (the two quantities multiplied together), the frequency of a term adjusted for how rarely it is used.

The statistic **tf-idf** is intended to measure how important a word is to a document in a collection (or corpus) of documents, for example, to one novel in a collection of novels or to one website in a collection of websites.

It is a rule-of-thumb or heuristic quantity; while it has proved useful in text mining, search engines, etc., its theoretical foundations are considered less than firm by information theory experts. The inverse document frequency for any given term is defined as

idf(term)=ln(ndocumentsndocuments containing term)idf(term)=ln⁡(ndocumentsndocuments containing term)

We can use tidy data principles, as described in Chapter 1, to approach tf-idf analysis and use consistent, effective tools to quantify how important various terms are in a document that is part of a collection.

(Let’s also calculate the total words in each novel here, for later use.)

library(dplyr)

library(janeaustenr)

library(tidytext)

book\_words <- austen\_books() %>%

unnest\_tokens(word, text) %>%

count(book, word, sort = TRUE)

total\_words <- book\_words %>%

group\_by(book) %>%

summarize(total = sum(n))

book\_words <- left\_join(book\_words, total\_words)

book\_words

#> # A tibble: 40,379 × 4

#> book word n total

#> <fct> <chr> <int> <int>

#> 1 Mansfield Park the 6206 160460

#> 2 Mansfield Park to 5475 160460

#> 3 Mansfield Park and 5438 160460

#> 4 Emma to 5239 160996

#> 5 Emma the 5201 160996

#> 6 Emma and 4896 160996

#> 7 Mansfield Park of 4778 160460

#> 8 Pride & Prejudice the 4331 122204

#> 9 Emma of 4291 160996

#> 10 Pride & Prejudice to 4162 122204

#> # … with 40,369 more rows

There is one row in this book\_words data frame for each word-book combination; n is the number of times that word is used in that book and total is the total words in that book. The usual suspects are here with the highest n, “the”, “and”, “to”, and so forth. In Figure 3.1, let’s look at the distribution of n/total for each novel, the number of times a word appears in a novel divided by the total number of terms (words) in that novel. This is exactly what term frequency is.

library(ggplot2)

ggplot(book\_words, aes(n/total, fill = book)) +

geom\_histogram(show.legend = FALSE) +

xlim(NA, 0.0009) +

facet\_wrap(~book, ncol = 2, scales = "free\_y")

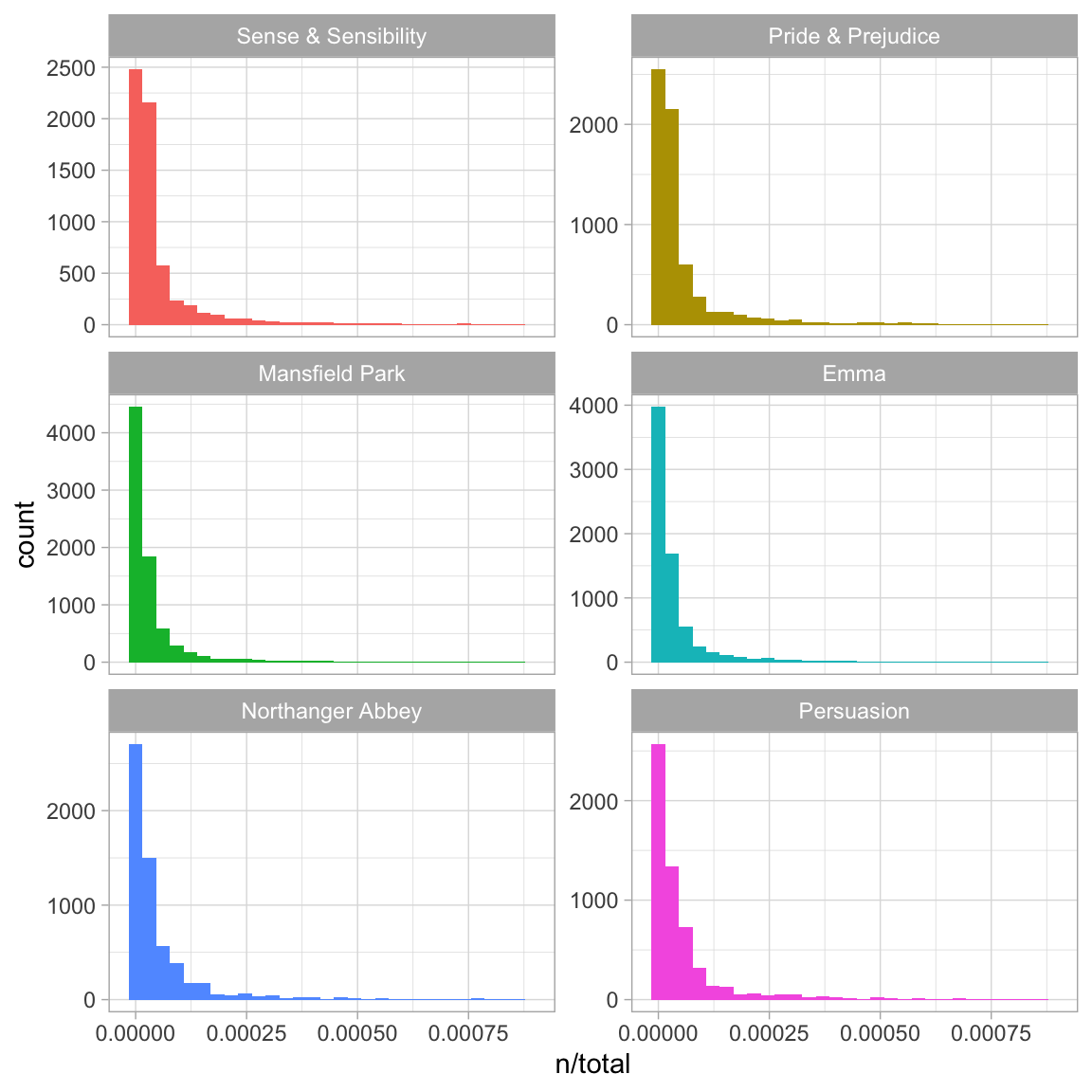


Figure : Term frequency distribution

There are very long tails to the right for these novels (those extremely rare words!) that we have not shown in these plots. These plots exhibit similar distributions for all the novels, with many words that occur rarely and fewer words that occur frequently.

freq\_by\_rank <- book\_words %>%

group\_by(book) %>%

mutate(rank = row\_number(),

`term frequency` = n/total) %>%

ungroup()

freq\_by\_rank

#> # A tibble: 40,379 × 6

#> book word n total rank `term frequency`

#> <fct> <chr> <int> <int> <int> <dbl>

#> 1 Mansfield Park the 6206 160460 1 0.0387

#> 2 Mansfield Park to 5475 160460 2 0.0341

#> 3 Mansfield Park and 5438 160460 3 0.0339

#> 4 Emma to 5239 160996 1 0.0325

#> 5 Emma the 5201 160996 2 0.0323

#> 6 Emma and 4896 160996 3 0.0304

#> 7 Mansfield Park of 4778 160460 4 0.0298

#> 8 Pride & Prejudice the 4331 122204 1 0.0354

#> 9 Emma of 4291 160996 4 0.0267

#> 10 Pride & Prejudice to 4162 122204 2 0.0341

#> # … with 40,369 more rows

The rank column here tells us the rank of each word within the frequency table; the table was already ordered by n so we could use row\_number() to find the rank. Then, we can calculate the term frequency in the same way we did before. Zipf’s law is often visualized by plotting rank on the x-axis and term frequency on the y-axis, on logarithmic scales. Plotting this way, an inversely proportional relationship will have a constant, negative slope.

freq\_by\_rank %>%

ggplot(aes(rank, `term frequency`, color = book)) +

geom\_line(size = 1.1, alpha = 0.8, show.legend = FALSE) +

scale\_x\_log10() +

scale\_y\_log10()

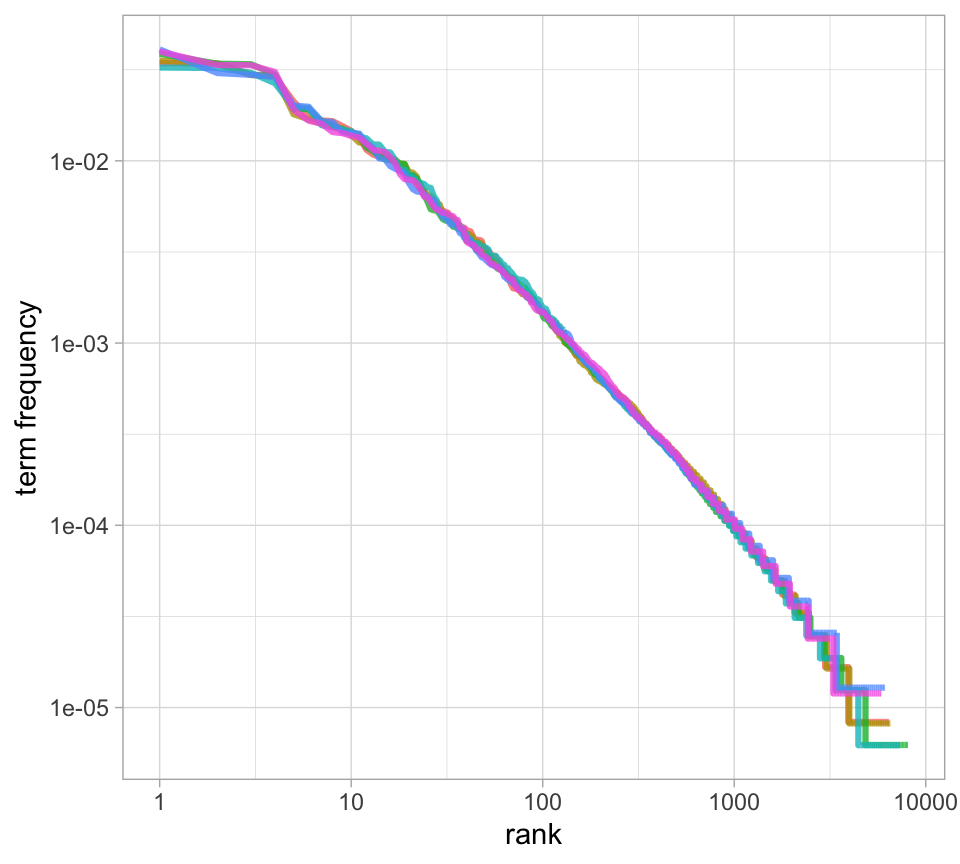


Figure : Zipf’s law (Zipf’s law states that the frequency that a word appears is inversely proportional to its rank.)

Let’s see what the exponent of the power law is for the middle section of the rank range.

rank\_subset <- freq\_by\_rank %>%

filter(rank < 500,

rank > 10)

lm(log10(`term frequency`) ~ log10(rank), data = rank\_subset)

#>

#> Call:

#> lm(formula = log10(`term frequency`) ~ log10(rank), data = rank\_subset)

#>

#> Coefficients:

#> (Intercept) log10(rank)

#> -0.6226 -1.1125

Classic versions of Zipf’s law have frequency∝1rankfrequency∝1rankand we have in fact gotten a slope close to -1 here. Let’s plot this fitted power law with the data in Figure 3.3 to see how it looks.

freq\_by\_rank %>%

ggplot(aes(rank, `term frequency`, color = book)) +

geom\_abline(intercept = -0.62, slope = -1.1,

color = "gray50", linetype = 2) +

geom\_line(size = 1.1, alpha = 0.8, show.legend = FALSE) +

scale\_x\_log10() +

scale\_y\_log10()

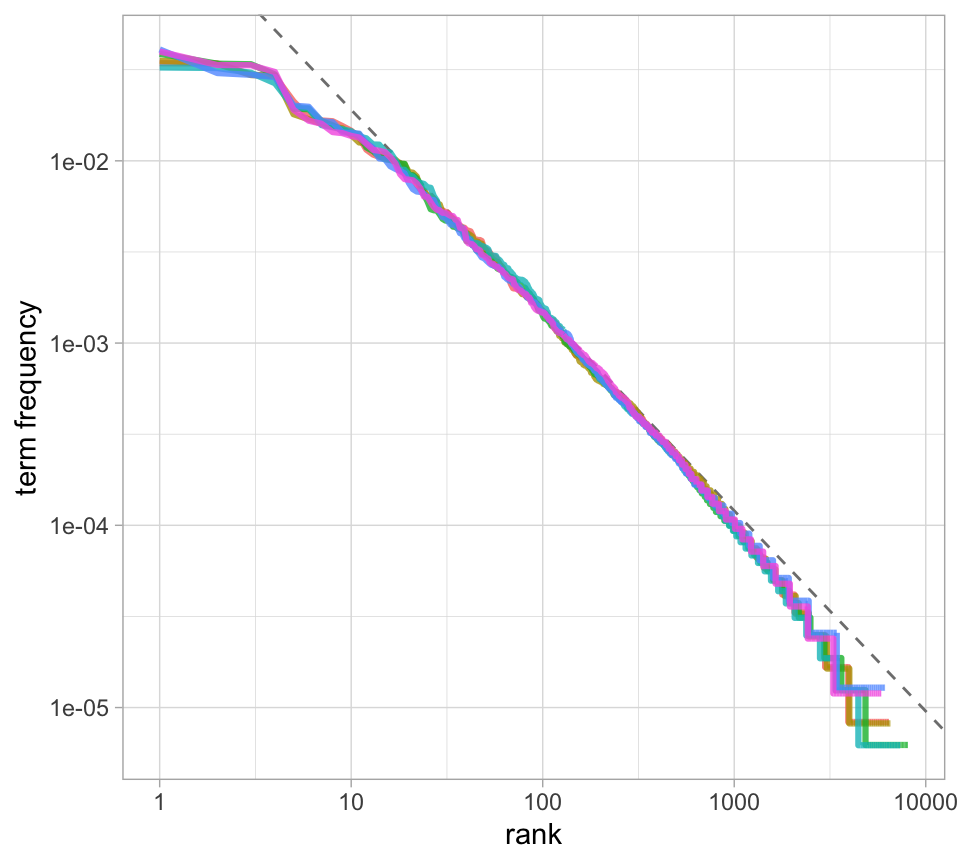


Figure : Fitting an exponent for Zipf’s law

We have found a result close to the classic version of Zipf’s law for the corpus of Jane Austen’s novels. The deviations we see here at high rank are not uncommon for many kinds of language; a corpus of language often contains fewer rare words than predicted by a single power law. The deviations at low rank are more unusual. Jane Austen uses a lower percentage of the most common words than many collections of language. This kind of analysis could be extended to compare authors, or to compare any other collections of text; it can be implemented simply using tidy data principles.

3.3 The bind\_tf\_idf() function

The idea of tf-idf is to find the important words for the content of each document by decreasing the weight for commonly used words and increasing the weight for words that are not used very much in a collection or corpus of documents, in this case, the group of Jane Austen’s novels as a whole. Calculating tf-idf attempts to find the words that are important (i.e., common) in a text, but not *too* common. Let’s do that now.

The bind\_tf\_idf() function in the tidytext package takes a tidy text dataset as input with one row per token (term), per document. One column (word here) contains the terms/tokens, one column contains the documents (book in this case), and the last necessary column contains the counts, how many times each document contains each term (n in this example). We calculated a total for each book for our explorations in previous sections, but it is not necessary for the bind\_tf\_idf() function; the table only needs to contain all the words in each document.

book\_tf\_idf <- book\_words %>%

bind\_tf\_idf(word, book, n)

book\_tf\_idf

#> # A tibble: 40,379 × 7

#> book word n total tf idf tf\_idf

#> <fct> <chr> <int> <int> <dbl> <dbl> <dbl>

#> 1 Mansfield Park the 6206 160460 0.0387 0 0

#> 2 Mansfield Park to 5475 160460 0.0341 0 0

#> 3 Mansfield Park and 5438 160460 0.0339 0 0

#> 4 Emma to 5239 160996 0.0325 0 0

#> 5 Emma the 5201 160996 0.0323 0 0

#> 6 Emma and 4896 160996 0.0304 0 0

#> 7 Mansfield Park of 4778 160460 0.0298 0 0

#> 8 Pride & Prejudice the 4331 122204 0.0354 0 0

#> 9 Emma of 4291 160996 0.0267 0 0

#> 10 Pride & Prejudice to 4162 122204 0.0341 0 0

#> # … with 40,369 more rows

Notice that idf and thus tf-idf are zero for these extremely common words. These are all words that appear in all six of Jane Austen’s novels, so the idf term (which will then be the natural log of 1) is zero. The inverse document frequency (and thus tf-idf) is very low (near zero) for words that occur in many of the documents in a collection; this is how this approach decreases the weight for common words. The inverse document frequency will be a higher number for words that occur in fewer of the documents in the collection.

Let’s look at terms with high tf-idf in Jane Austen’s works.

book\_tf\_idf %>%

select(-total) %>%

arrange(desc(tf\_idf))

#> # A tibble: 40,379 × 6

#> book word n tf idf tf\_idf

#> <fct> <chr> <int> <dbl> <dbl> <dbl>

#> 1 Sense & Sensibility elinor 623 0.00519 1.79 0.00931

#> 2 Sense & Sensibility marianne 492 0.00410 1.79 0.00735

#> 3 Mansfield Park crawford 493 0.00307 1.79 0.00551

#> 4 Pride & Prejudice darcy 373 0.00305 1.79 0.00547

#> 5 Persuasion elliot 254 0.00304 1.79 0.00544

#> 6 Emma emma 786 0.00488 1.10 0.00536

#> 7 Northanger Abbey tilney 196 0.00252 1.79 0.00452

#> 8 Emma weston 389 0.00242 1.79 0.00433

#> 9 Pride & Prejudice bennet 294 0.00241 1.79 0.00431

#> 10 Persuasion wentworth 191 0.00228 1.79 0.00409

#> # … with 40,369 more rows

Here we see all proper nouns, names that are in fact important in these novels. None of them occur in all of novels, and they are important, characteristic words for each text within the corpus of Jane Austen’s novels.

Some of the values for idf are the same for different terms because there are 6 documents in this corpus and we are seeing the numerical value for ln(6/1)ln⁡(6/1), ln(6/2)ln⁡(6/2), etc.

Let’s look at a visualization for these high tf-idf words in Figure 3.4.

library(forcats)

book\_tf\_idf %>%

group\_by(book) %>%

slice\_max(tf\_idf, n = 15) %>%

ungroup() %>%

ggplot(aes(tf\_idf, fct\_reorder(word, tf\_idf), fill = book)) +

geom\_col(show.legend = FALSE) +

facet\_wrap(~book, ncol = 2, scales = "free") +

labs(x = "tf-idf", y = NULL)

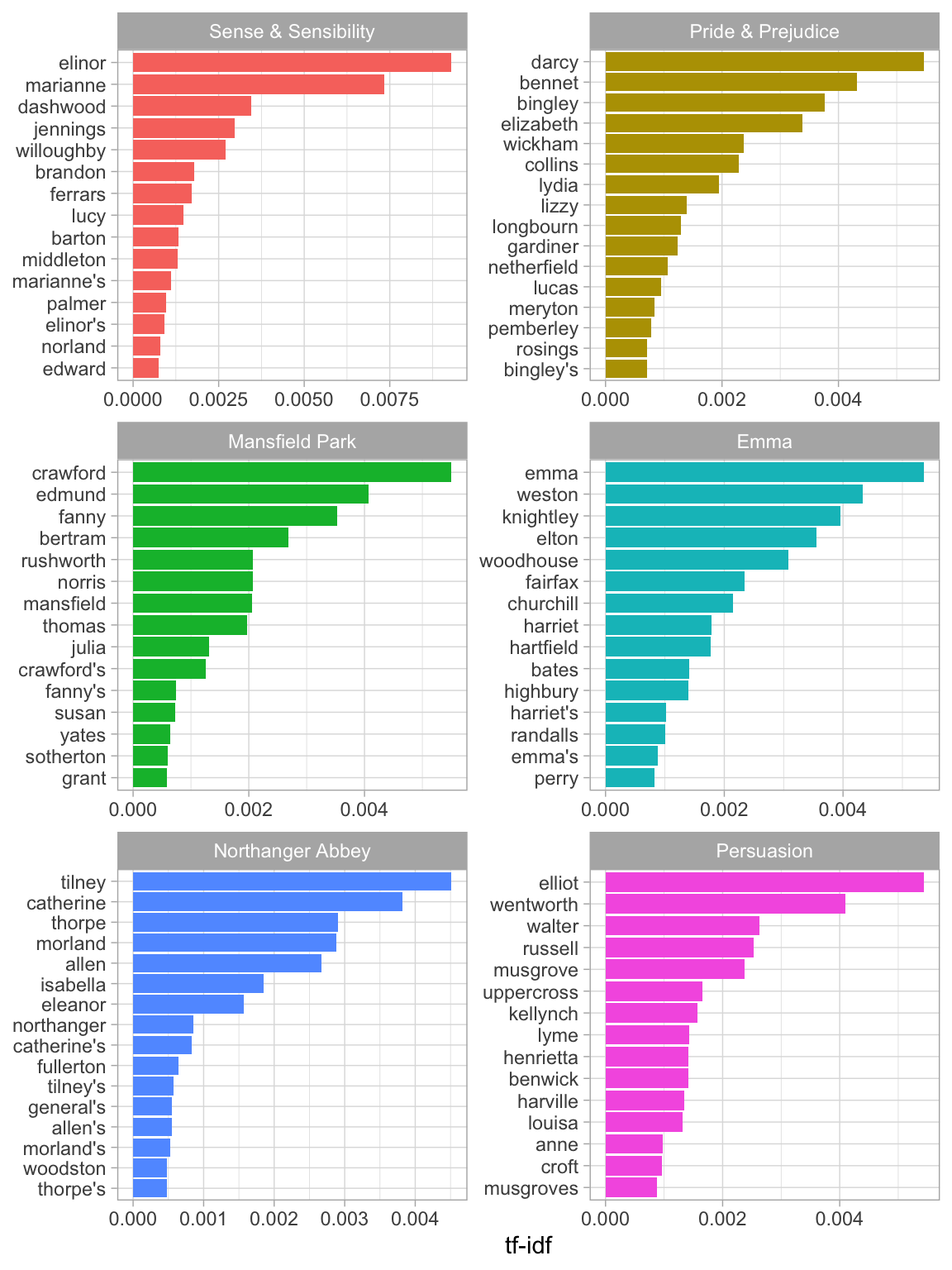


Figure : Highest tf-idf words

Still all proper nouns in Figure 3.4! These words are, as measured by tf-idf, the most important to each novel and most readers would likely agree. What measuring tf-idf has done here is show us that Jane Austen used similar language across her six novels, and what distinguishes one novel from the rest within the collection of her works are the proper nouns, the names of people and places. This is the point of tf-idf; it identifies words that are important to one document within a collection of documents.

3.4 A corpus of physics texts

Let’s work with another corpus of documents, to see what terms are important in a different set of works. In fact, let’s leave the world of fiction and narrative entirely. Let’s download some classic physics texts from Project Gutenberg and see what terms are important in these works, as measured by tf-idf.

This is a pretty diverse bunch. They may all be physics classics, but they were written across a 300-year timespan, and some of them were first written in other languages and then translated to English. Perfectly homogeneous these are not, but that doesn’t stop this from being an interesting exercise!

library(gutenbergr)

physics <- gutenberg\_download(c(37729, 14725, 13476, 30155),

meta\_fields = "author")

Now that we have the texts, let’s use unnest\_tokens() and count() to find out how many times each word was used in each text.

physics\_words <- physics %>%

unnest\_tokens(word, text) %>%

count(author, word, sort = TRUE)

physics\_words

#> # A tibble: 12,671 × 3

#> author word n

#> <chr> <chr> <int>

#> 1 Galilei, Galileo the 3760

#> 2 Tesla, Nikola the 3604

#> 3 Huygens, Christiaan the 3553

#> 4 Einstein, Albert the 2993

#> 5 Galilei, Galileo of 2049

#> 6 Einstein, Albert of 2028

#> 7 Tesla, Nikola of 1737

#> 8 Huygens, Christiaan of 1708

#> 9 Huygens, Christiaan to 1207

#> 10 Tesla, Nikola a 1176

#> # … with 12,661 more rows

Here we see just the raw counts; we need to remember that these documents are all different lengths. Let’s go ahead and calculate tf-idf, then visualize the high tf-idf words in Figure 3.5.

plot\_physics <- physics\_words %>%

bind\_tf\_idf(word, author, n) %>%

mutate(author = factor(author, levels = c("Galilei, Galileo",

"Huygens, Christiaan",

"Tesla, Nikola",

"Einstein, Albert")))

plot\_physics %>%

group\_by(author) %>%

slice\_max(tf\_idf, n = 15) %>%

ungroup() %>%

mutate(word = reorder(word, tf\_idf)) %>%

ggplot(aes(tf\_idf, word, fill = author)) +

geom\_col(show.legend = FALSE) +

labs(x = "tf-idf", y = NULL) +

facet\_wrap(~author, ncol = 2, scales = "free")

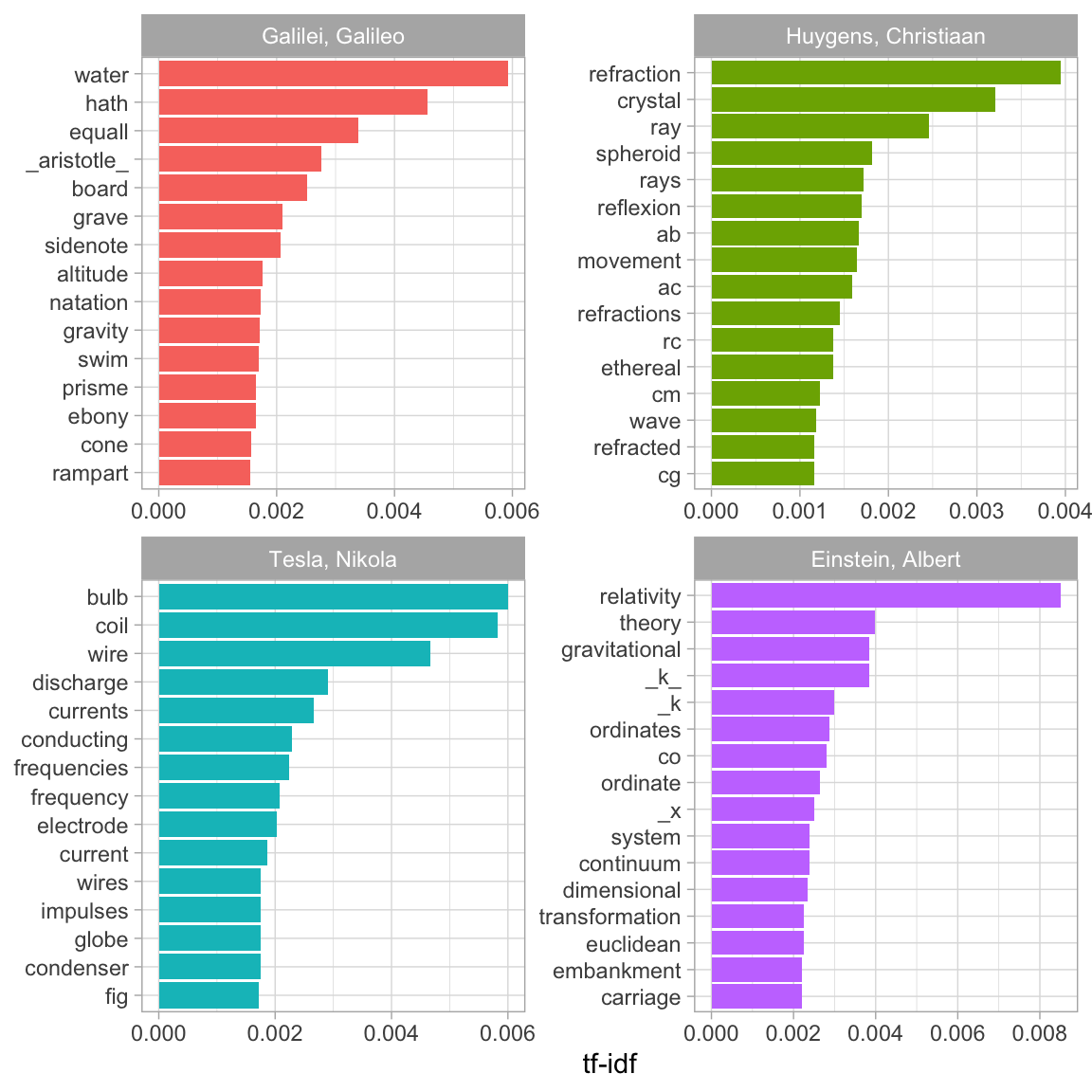


Figure 3.5: Highest tf-idf words in each physics texts

Very interesting indeed. One thing we see here is “*k*” in the Einstein text?!

library(stringr)

physics %>%

filter(str\_detect(text, "\_k\_")) %>%

select(text)

#> # A tibble: 7 × 1

#> text

#> <chr>

#> 1 surface AB at the points AK\_k\_B. Then instead of the hemispherical

#> 2 would needs be that from all the other points K\_k\_B there should

#> 3 necessarily be equal to CD, because C\_k\_ is equal to CK, and C\_g\_ to

#> 4 the crystal at K\_k\_, all the points of the wave CO\_oc\_ will have

#> 5 O\_o\_ has reached K\_k\_. Which is easy to comprehend, since, of these

#> 6 CO\_oc\_ in the crystal, when O\_o\_ has arrived at K\_k\_, because it forms

#> 7 ρ is the average density of the matter and \_k\_ is a constant connected

Some cleaning up of the text may be in order. Also notice that there are separate “co” and “ordinate” items in the high tf-idf words for the Einstein text; the unnest\_tokens() function separates around punctuation like hyphens by default. Notice that the tf-idf scores for “co” and “ordinate” are close to same!

“AB”, “RC”, and so forth are names of rays, circles, angles, and so forth for Huygens.

physics %>%

filter(str\_detect(text, "RC")) %>%

select(text)

#> # A tibble: 44 × 1

#> text

#> <chr>

#> 1 line RC, parallel and equal to AB, to be a portion of a wave of light,

#> 2 represents the partial wave coming from the point A, after the wave RC

#> 3 be the propagation of the wave RC which fell on AB, and would be the

#> 4 transparent body; seeing that the wave RC, having come to the aperture

#> 5 incident rays. Let there be such a ray RC falling upon the surface

#> 6 CK. Make CO perpendicular to RC, and across the angle KCO adjust OK,

#> 7 the required refraction of the ray RC. The demonstration of this is,

#> 8 explaining ordinary refraction. For the refraction of the ray RC is

#> 9 29. Now as we have found CI the refraction of the ray RC, similarly

#> 10 the ray \_r\_C is inclined equally with RC, the line C\_d\_ will

#> # … with 34 more rows

Let’s remove some of these less meaningful words to make a better, more meaningful plot. Notice that we make a custom list of stop words and use anti\_join() to remove them; this is a flexible approach that can be used in many situations. We will need to go back a few steps since we are removing words from the tidy data frame.

mystopwords <- tibble(word = c("eq", "co", "rc", "ac", "ak", "bn",

"fig", "file", "cg", "cb", "cm",

"ab", "\_k", "\_k\_", "\_x"))

physics\_words <- anti\_join(physics\_words, mystopwords,

by = "word")

plot\_physics <- physics\_words %>%

bind\_tf\_idf(word, author, n) %>%

mutate(word = str\_remove\_all(word, "\_")) %>%

group\_by(author) %>%

slice\_max(tf\_idf, n = 15) %>%

ungroup() %>%

mutate(word = fct\_reorder(word, tf\_idf)) %>%

mutate(author = factor(author, levels = c("Galilei, Galileo",

"Huygens, Christiaan",

"Tesla, Nikola",

"Einstein, Albert")))

ggplot(plot\_physics, aes(tf\_idf, word, fill = author)) +

geom\_col(show.legend = FALSE) +

facet\_wrap(~author, ncol = 2, scales = "free") +

labs(x = "tf-idf", y = NULL)

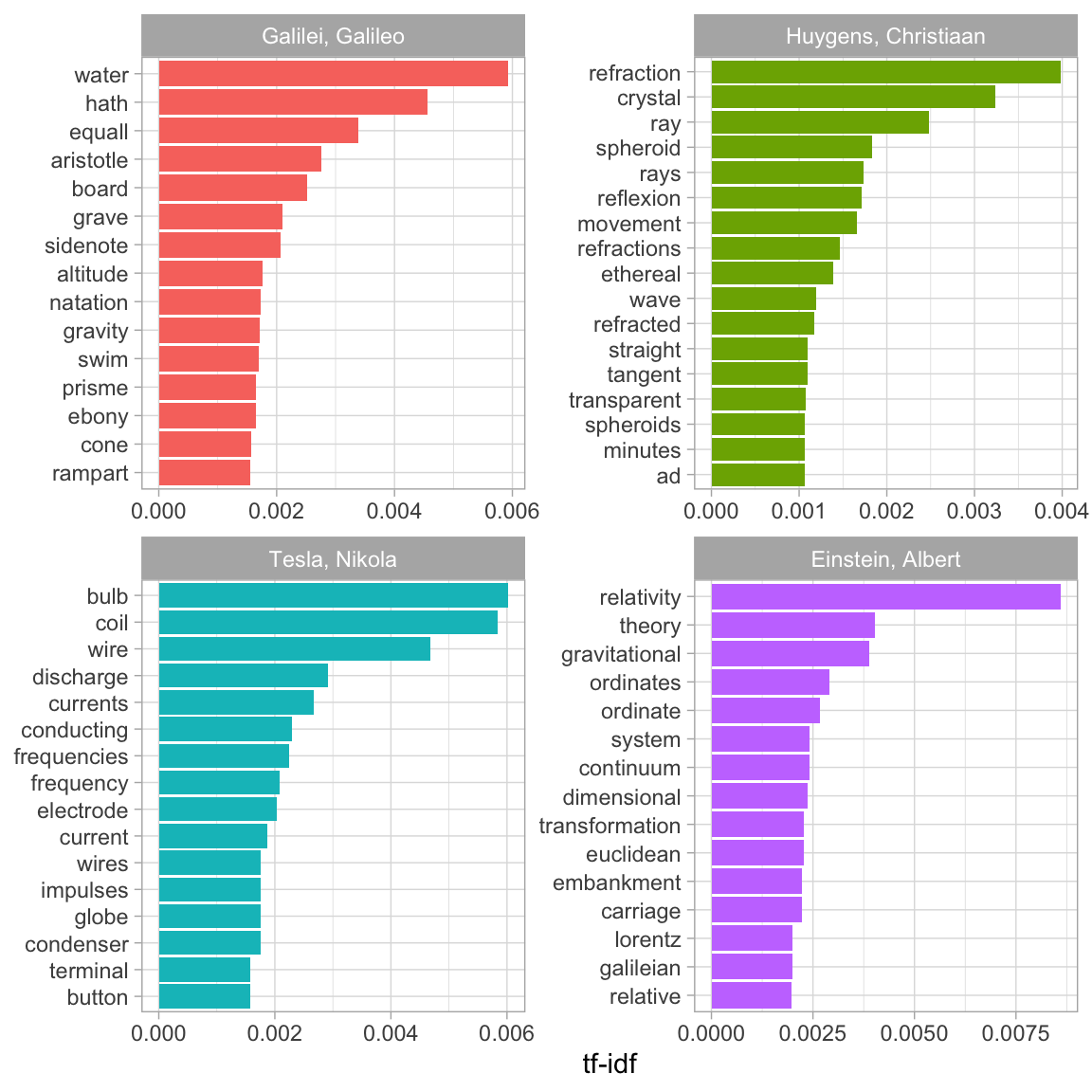


Figure : Highest tf-idf words in classic physics texts

One thing we can conclude from Figure is that we don’t hear enough about ramparts or things being ethereal in physics today.

**Weighty business**

Instead, we can use a **weighted log odds**, which tidylo provides an implementation for using tidy data principles. In particular, this package uses the method outlined in [Monroe, Colaresi, and Quinn (2017)](https://doi.org/10.1093/pan/mpn018) to weight the log odds ratio by an uninformative Dirichlet prior.

It starts:

Julia Silge, astrophysicist, R guru and maker of beautiful charts, a data scientist with what seem by any account to be comfortable and happy cats, united the best blessings of existence; and had lived in the world with very little to distress or vex her.

Let’s just say it gets better from there. **AND** it walks through the benefits of using a weighted log odds ratio for text analysis when the analytical question is focused on differences in frequency across sets, in this particular case, books.

Tyler and I have collaborated on this package implementing this approach, and it is now available on GitHub. You can install it via remotes.

library(remotes)

Tyler and I think that “tidylo” is pronounced “tidy-low”, or maybe, if you prefer, “tidy-el-oh”.

**Some examples**

This weighted log odds approach is useful for text analysis, but not *only* for text analysis. In the weeks since we’ve had this package up and running, I’ve found myself reaching for it in multiple situations, both text and not, in my real-life day job. For this example, let’s look at the [same data](https://juliasilge.com/blog/reorder-within/) , names given to children in the US.

Which names were most common in the 1950s, 1960s, 1970s, and 1980?

library(tidyverse)

library(babynames)

top\_names <- babynames %>%

filter(year >= 1950,

year < 1990) %>%

mutate(decade = (year %/% 10) \* 10,

decade = paste0(decade, "s")) %>%

group\_by(decade) %>%

count(name, wt = n, sort = TRUE) %>%

ungroup

top\_names

## # A tibble: 100,527 x 3

## decade name n

##

## 1 1950s James 846042

## 2 1950s Michael 839459

## 3 1960s Michael 836934

## 4 1950s Robert 832336

## 5 1950s John 799658

## 6 1950s David 771242

## 7 1960s David 736583

## 8 1960s John 716284

## 9 1970s Michael 712722

## 10 1960s James 687905

## # … with 100,517 more rows

Now let’s use the bind\_log\_odds() function from the tidylo package to find the weighted log odds for each name. What are the highest log odds names for these decades?

library(tidylo)

name\_log\_odds <- top\_names %>%

bind\_log\_odds(decade, name, n)

name\_log\_odds %>%

arrange(-log\_odds)

## # A tibble: 100,527 x 4

## decade name n log\_odds

##

## 1 1980s Ashley 357385 477.

## 2 1980s Jessica 471493 457.

## 3 1950s Linda 565481 414.

## 4 1980s Joshua 399131 412.

## 5 1980s Amanda 370873 391.

## 6 1970s Jason 465402 374.

## 7 1980s Justin 291843 363.

## 8 1950s Deborah 431302 348.

## 9 1980s Brandon 234294 331.

## 10 1970s Jennifer 583692 330.

## # … with 100,517 more rows

These are the highest log odds names (names more likely to come from each decade, compared to the decades) when we have used weighting to account for sampling variability.

We can make some visualizations to understand our results better. For example, which names are most characteristic of each decade?

name\_log\_odds %>%

group\_by(decade) %>%

top\_n(10, log\_odds) %>%

ungroup %>%

mutate(decade = as.factor(decade),

name = fct\_reorder(name, log\_odds)) %>%

ggplot(aes(name, log\_odds, fill = decade)) +

geom\_col(show.legend = FALSE) +

facet\_wrap(~decade, scales = "free\_y") +

coord\_flip() +

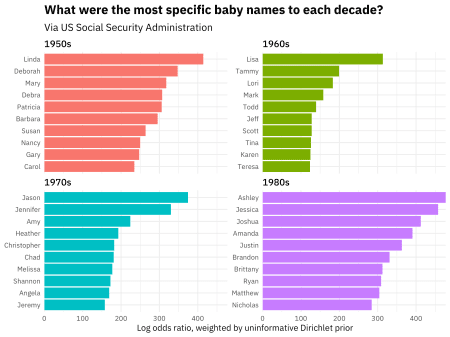
scale\_y\_continuous(expand = c(0,0)) +

labs(y = "Log odds ratio, weighted by uninformative Dirichlet prior",

x = NULL,

title = "What were the most specific baby names to each decade?",

subtitle = "Via US Social Security Administration")



Wow, just reading those lists of names, I can picture how old those people are. I’m a child of the 1970s myself, and my childhood classrooms were in fact filled with Jasons, Jennifers, and Amys. Also notice that the weighted log odds ratios are higher in the 1980s than the other decades; the 1980s names are *more different* than the other decades’ names.

Perhaps we want to understand one decade more deeply. For example, what are the most 1980s names, along with how common they were?

library(ggrepel)

name\_log\_odds %>%

filter(decade == "1980s") %>%

top\_n(50, n) %>%

ggplot(aes(n, log\_odds, label = name)) +

geom\_hline(yintercept = 0, lty = 2,

color = "gray50", alpha = 0.5, size = 1.2) +

geom\_text\_repel(family = "IBMPlexSans") +

geom\_point() +

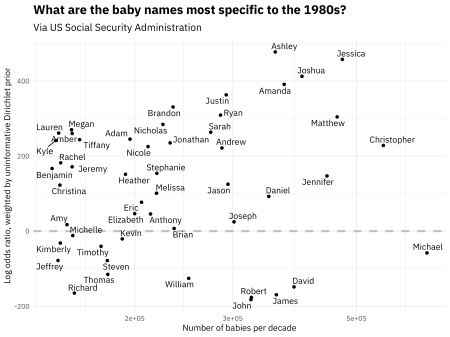
scale\_x\_log10() +

labs(x = "Number of babies per decade",

y = "Log odds ratio, weighted by uninformative Dirichlet prior",

title = "What are the baby names most specific to the 1980s?",

subtitle = "Via US Social Security Administration")



Michael is the most popular name in the 1980s, but it is not particularly likely to be used in the 1980s compared to the other decades (it was common in other decades too). We can see in this visualization the distribution of frequency and log odds ratio.