From the first time I listened to Radiohead’s *The Bends*, the band has been my favorite. I was a grad student in England at the time, and I recall listening to “Fake Plastic Trees” on repeat as I made my way to and from the library each day. By the time *OK Computer* came out, I was hooked. I remain hooked to this day.

As a writer-turned-data-science practitioner, text mining and natural language processing fascinate me. So in this post I’ll combine these two passions—words and Radiohead—to analyze the band’s lyrics.

The first thing to do is create the dataset, using [Josiah Parry’s wonderful geniusR package](https://github.com/JosiahParry/geniusR) to obtain the lyrics. I’ve seen other people get the discography from Wikipedia, but it’s a short enough list that it would be more trouble for me to do it that way than to just create the list from scratch.

library(geniusR)

library(tidytext)

library(tidyverse)

library(wordcloud)

library(dplyr)

library(tidyr)

library(ggplot2)

library(purrr)

library(broom)

theme\_set(theme\_minimal())

# Create dataframe

album <- c("Pablo Honey", "The Bends", "OK Computer", "Kid A",

"Amnesiac", "Hail to the Thief", "In Rainbows",

"The King of Limbs", "A Moon Shaped Pool")

year <- c(1993, 1995, 1997, 2000, 2001, 2003, 2007, 2011, 2016)

Radiohead\_albums <- data.frame(album, year)

# Get lyrics

Radiohead\_lyrics <- Radiohead\_albums %>%

mutate(tracks = map2("Radiohead", album, genius\_album))

Radiohead\_lyrics <- Radiohead\_lyrics %>%

unnest(tracks)

Radiohead\_words <- Radiohead\_lyrics %>%

unnest\_tokens(word, lyric) %>%

anti\_join(stop\_words) %>%

select(album, word, year, track\_title, track\_n, line) # get variables in order for

# convert everything to lower case

Radiohead\_words$word <- sapply(Decemberists\_words$word, tolower)

head(Radiohead\_words)

album word year track\_title track\_n line

1 Pablo Honey sun 1993 You 1 2

2 Pablo Honey moon 1993 You 1 2

3 Pablo Honey stars 1993 You 1 3

4 Pablo Honey run 1993 You 1 4

5 Pablo Honey chaotic 1993 You 1 7

6 Pablo Honey world 1993 You 1 11

So the first thing I want to do is just get a sense of overall word frequency, so I’ll use the wordcloud package.

pal <- brewer.pal(7, "GnBu")

Radiohead\_words %>%

count(word, sort = TRUE) %>%

with(wordcloud(word, n, max.words = 100, rot.per = 0.35, colors = pal))

If you’ve listened to Radiohead for any length of time, you don’t need me to tell you that their songs are often dark. This is evidenced in the word cloud of their lyrics. The word cloud, though, includes all of their albums together, which can’t then give us a real sense of what each album is about or of the progression over time. Even doing a word cloud for each one isn’t going to give us an adequate picture of meaning because often a word that is unimportant to meaning will nonetheless occur frequently. So, for example, the word *there’ll* appears as a frequent word in the above word cloud, but without context that word is virtually meaningless. And while we could add that to our list of stopwords to remove, that’s a pretty inelegant and unsophisticated way to adjust term frequency.

But term frequency (tf) is only one way of discerning the importance of specific terms. Another way is to look at inverse document frequency (idf), a method of weighting infrequent words higher than frequent words within a collection or corpus of documents. In turn, term frequency-inverse document frequency (tf-idf), which multiplies the two scores together, is a widely regarded statistical measure for identifying terms important to a document within a collection of documents. Importance increases in proportion to the number of occurrences in a document, but it is then offset by its frequency in the collection as a whole. Hence, a word that is rare in a collection, but frequent in a document, will have a higher weight.

So let’s run the model and use ggplot2 to visualize the results.

Radiohead\_tf\_idf <- Radiohead\_lyrics %>%

unnest\_tokens(word, lyric) %>%

count(album, word, sort = TRUE) %>%

ungroup()

plot\_radiohead <- Radiohead\_tf\_idf %>%

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

arrange(desc(tf\_idf)) %>%

mutate(word = factor(word, levels = rev(unique(word)))) %>%

mutate(album = factor(album, levels = c("Pablo Honey", "The Bends", "OK Computer", "Kid A",

"Amnesiac", "Hail to the Thief", "In Rainbows",

"The King of Limbs", "A Moon Shaped Pool")))

plot\_radiohead %>%

group\_by(album) %>%

top\_n(5, tf\_idf) %>%

ungroup() %>%

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

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

geom\_col(show.legend = FALSE) +

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

facet\_wrap(~album, ncol = 3, scales = "free") +

coord\_flip()

We can see that the term *raindrops*, which appeared as the most prominent term in the word cloud, is indeed quite important for the *Hail to the Thief* record, with the highest weight of any term in the entire corpus. We can also see that the terms *hurt* and *haunt* from *The King of Limbs* are also highly weighted for that album. Lastly, we can see that the most important words on *A Moon-shaped Pool* are non-English words. So let’s look at one of them to see it’s context.

efil <- Radiohead\_lyrics[grep("efil", Radiohead\_lyrics$lyric),]

efil <- efil %>%

select(album, track\_title, lyric) %>%

distinct()

efil

album track\_title lyric

1 A Moon Shaped Pool Daydreaming ​efil ym fo flaH

Next, I’m curious about how the band has evolved, so I want to look at which words the band has used at a higher or lower rate over time. I’m relying on code that Julia Silge and David Robinson developed [to analyze changes in their own Twitter output](https://www.tidytextmining.com/twitter.html#changes-in-word-use) for the book *Text Mining with R*. As I create the dataset, I’m going to filter() the results so that I’m only keeping words that occur 15 or more times.

Radiohead\_by\_time <- Radiohead\_words %>%

count(year, word) %>%

mutate(time\_total = sum(n)) %>%

group\_by(word) %>%

mutate(word\_total = sum(n)) %>%

ungroup() %>%

rename(count = n) %>%

dplyr::filter(word\_total > 15)

Radiohead\_by\_time

# A tibble: 138 x 5

year word count time\_total word\_total

1 1993 broken 1 4397 28

2 1993 dead 6 4397 16

3 1993 eyes 1 4397 26

4 1993 feel 5 4397 31

5 1993 gonna 1 4397 20

6 1993 love 2 4397 27

7 1993 run 8 4397 17

8 1993 time 7 4397 28

9 1993 wanna 10 4397 17

10 1993 world 7 4397 23

# ... with 128 more rows

Each row in the dataset corresponds to the use of a given word on a particular album (represented for now by the year value). Count represents how many times the word is used on an album. Time\_total represents the total number of words in the album’s lyrics, and word\_total represents the total number of times the word is used in the complete collection of albums.

Now I’ll use nest() from the tidyr package to create a new listed data frame, and then I’ll use map() from the purrr package to apply a regression model, a family = "binomial" glm() model since this is count data.

nested\_data <- Radiohead\_by\_time %>%

nest(-word)

nested\_models <- nested\_data %>%

mutate(models = map(data, ~ glm(cbind(count, time\_total) ~ year, .,

family = "binomial")))

nested\_models

word data models

1 broken

2 dead

3 eyes

4 feel

5 gonna

6 love

7 run

8 time

9 wanna

10 world

# ... with 29 more rows

No I have a models column that holds the glm results, and I want to extract from that the slopes for each word/year. I’ll use map() and tidy() from the broom package and apply an adjustment to the *p*-values to account for multiple comparisons.

slopes <- nested\_models %>%

unnest(map(models, tidy)) %>%

dplyr::filter(term == "year") %>%

mutate(adjusted.p.value = p.adjust(p.value))

top\_slopes <- slopes %>%

dplyr::filter(adjusted.p.value < 0.05)

top\_slopes

# A tibble: 4 x 7

word term estimate std.error statistic p.value adjusted.p.value

1 broken year 0.0878 0.0215 4.09 0.0000428 0.00218

2 light year 0.120 0.0340 3.52 0.000432 0.0216

3 mess year 0.153 0.0349 4.38 0.0000120 0.000623

4 arms year 0.237 0.0510 4.65 0.00000331 0.000176

>

So there are four words that have changed significantly over time: *broken*, *light*, *mess*, and *arms*. Let’s plot the changes to see when they occur.

Radiohead\_by\_time %>%

inner\_join(top\_slopes, by = "word") %>%

ggplot(aes(year, count/time\_total, color = word)) +

geom\_line(size = 1.3) +

geom\_point(size = 2.5) +

labs(y = "Word frequency", x = NULL) +

scale\_x\_continuous(breaks = Radiohead\_words$year,

labels = Radiohead\_words$album) +

theme(axis.text.x = element\_text(angle = 30, hjust = 1),

panel.grid.minor.x = element\_blank(),

panel.grid.major.x = element\_blank())

All four words become more frequent over time. *Broken*, *light*, and *mess* all seem self-explanatory. Personally, I’m curious about *arms* and how that’s used, i.e., does it signify appendages or weapons?

arms <- Radiohead\_lyrics[grep("arms", Radiohead\_lyrics$lyric),]

arms <- arms %>%

select(album, track\_title, lyric) %>%

distinct()

album track\_title lyric

1 OK Computer No Surprises And no alarms and no surprises

2 OK Computer No Surprises No alarms and no surprises

3 OK Computer No Surprises No alarms and no surprises, please

4 Kid A Motion Picture Soundtrack Help me get back to your arms

5 Hail to the Thief Backdrifts You fell into our arms

6 Hail to the Thief Go to Sleep Tip toeing, tying down our arms

7 The King of Limbs Give Up the Ghost Into your arms

8 The King of Limbs Give Up the Ghost (Into your arms)

9 The King of Limbs Give Up the Ghost Into your arms

Well, that’s interesting. Early instances of *arms* in the lyrics actually comes from *alarms*.

There’s a lot one can do with lyrics using NLP. I’ve seen several posts that have done sentiment analysis of Radiohead’s dark, dark lyrics, like [this one here](https://www.rcharlie.com/post/fitter-happier/) that figured out Radiohead’s saddest song. (Spoiler: It’s “True Love Waits.”) I actually started this project using lyrics from The Decemberists, thinking that their involuted lyrics would be interesting to look at, but their songs have so many la-la-las that it was more trouble than it was worth.