Text tokenization and text vectorization

Day 12

Solveig Bjørkholt

Plan for today

What we will learn today:

- Text tokenization
- Text vectorization

Text tokenization

To make text possible to process by a computer, we need to break the text down into units of information. This process is called *tokenization*. Some teaching materials on text analysis treat tokenization as a part of text preprocessing, and in truth, the two are rather overlapping. For example, it is not unusual to remove stopwords after having tokenized the text. Another part of preprocessing is *stemming*, which means taking the root (base) of a word. We'll come back to this, and how to plot text, but first we'll see how to tokenize.

As an example in this lecture, we'll use songs from the musical West Side Story. As a bit of repetition, we'll do some webscraping to fetch the song lyrics from a webpage on the internet.

```
library(rvest)
library(stringr)

song_page <- read_html("https://www.westsidestory.com/lyrics") # Read the html page

songs <- song_page %>%
  html_nodes("h2") %>% # Extract all html-nodes with the tag <h2>
  html_elements("a") %>% # Extract from this node all html-tags with <a>
  html_attr("href") %>% # And from these, the text belonging to the link reference <href>
  str_c("https://www.westsidestory.com", .)  # Paste together the previous webpage string with these

headline <- str_remove_all(songs, "https://www.westsidestory.com/") # Extracting the name of the songs
```

```
for(i in 1:length(songs)) { # For all the links to the different songs...

  download.file(songs[[i]], # Download one html-file after another into the folder specified in destfil
                destfile = str_c("../../datafolder/ws_songs/", headline[i], ".html"))
}
```

```
songfiles <- list.files("../../datafolder/ws_songs/") %>%
  str_c("../../datafolder/ws_songs/", .) # Listing up the html-files in the folder
```

```
songname <- list() # Make an empty list for the songnames
ws_lyrics <- list() # Make an empty list for the lyrics

for(i in 1:length(songfiles)){ # For each file in the folder as listed

  songname[[i]] <- songfiles[i] %>%
    str_remove("../../datafolder/ws_songs/") %>% # Remove the beginning of the file string
    str_remove("\\.html") # Remove the end - leaving us with just the name of the song

  lyric <- read_html(songfiles[i]) # Read the html-page

  lyric <- lyric %>%
    html_node("p") %>% # Extract all nodes with the tag <p>
    html_text2() # Fetch the text from these tags

  ws_lyrics[[i]] <- lyric # Add the lyrics to the empty list

}

ws_lyrics <- str_replace_all(ws_lyrics, "\n", " ") # Replacing double lineshift with space to make the t

ws_lyrics <- str_remove_all(ws_lyrics, "\\b[A-Z]{2,}\\b") # Remove all sequences of big letters followe

westside <- as_tibble(list(songname = unlist(songname), # Put the songnames into a column in a tibble
                           lyrics = unlist(ws_lyrics))) # Put the lyrics into a column in a tibble


glimpse(westside)
```

```
## Rows: 14
## Columns: 2
## $ songname <chr> "a-boy-like-thati-have-a-love", "america", "cool", "finale", ~
## $ lyrics   <chr> " A boy like that who'd kill your brother, Forget that boy an~
```

How to tokenize text

When we tokenize, we split the text up into units. Most often, these units are words[1]. Tokenizing the units into words means that the order of the words disappear. Studies have shown that often, it is not the *order* of the words but the *choice* of words that give us the most information on what the text is about. However, if we speculate that the order of the words might matter, we can also split the text into two and two words, three and three words, or whole sentences. When we tokenize into one word, we refer to these units and **unigrams**. Two words are **bigrams**, three words are **trigrams** and so on. Together, they are called **n-grams**.

To tokenize our group of texts (sometimes referred to as a corpus of text), we'll be using the package `tidytext`.

```
library(tidytext)
```

This package assumes that your data is in a dataframe. So if it's in a list, put it into a dataframe using for example `do.call(rbind, list)` or `enframe`. Once that is done, use the `unnest_tokens` function to

[1]They can also be syllables or groups or words or many other things

tokenize the text. As arguments, add as `input` the name of the old column with the text, as `output` the name of the new column you are creating, and as `token` we specify "words" to get unigrams.

```
westside_tokens <- westside %>%
  unnest_tokens(input = lyrics, # Which variable to get the text from
                output = word, # What the new variable should be called
                token = "words") # What kind of tokens we want to split the text into

westside_tokens
```

```
## # A tibble: 2,624 x 2
##    songname                      word
##    <chr>                         <chr>
##  1 a-boy-like-thati-have-a-love a
##  2 a-boy-like-thati-have-a-love boy
##  3 a-boy-like-thati-have-a-love like
##  4 a-boy-like-thati-have-a-love that
##  5 a-boy-like-thati-have-a-love who'd
##  6 a-boy-like-thati-have-a-love kill
##  7 a-boy-like-thati-have-a-love your
##  8 a-boy-like-thati-have-a-love brother
##  9 a-boy-like-thati-have-a-love forget
## 10 a-boy-like-thati-have-a-love that
## # ... with 2,614 more rows
```

Had we wanted bigrams, we could specify `token = "ngrams"` and `n = 2`.

```
westside %>%
  unnest_tokens(input = lyrics,
                output = word,
                token = "ngrams",
                n = 2)
```

```
## # A tibble: 2,610 x 2
##    songname                      word
##    <chr>                         <chr>
##  1 a-boy-like-thati-have-a-love a boy
##  2 a-boy-like-thati-have-a-love boy like
##  3 a-boy-like-thati-have-a-love like that
##  4 a-boy-like-thati-have-a-love that who'd
##  5 a-boy-like-thati-have-a-love who'd kill
##  6 a-boy-like-thati-have-a-love kill your
##  7 a-boy-like-thati-have-a-love your brother
##  8 a-boy-like-thati-have-a-love brother forget
##  9 a-boy-like-thati-have-a-love forget that
## 10 a-boy-like-thati-have-a-love that boy
## # ... with 2,600 more rows
```

Or we can tokenize into sentences.

```
westside %>%
  unnest_tokens(input = lyrics,
                output = word,
                token = "sentences")
```

```
## # A tibble: 275 x 2
##    songname                      word
##    <chr>                         <chr>
##  1 a-boy-like-thati-have-a-love a boy like that who'd kill your brother, forget~
##  2 a-boy-like-thati-have-a-love stick to your own kind!
##  3 a-boy-like-thati-have-a-love a boy like that will give you sorrow.
##  4 a-boy-like-thati-have-a-love you'll meet another boy tomorrow, one of your o~
##  5 a-boy-like-thati-have-a-love stick to your own kind!
##  6 a-boy-like-thati-have-a-love a boy who kills cannot love, a boy who kills ha~
##  7 a-boy-like-thati-have-a-love and he's the boy who gets your love and gets yo~
##  8 a-boy-like-thati-have-a-love very smart, maria, very smart!
##  9 a-boy-like-thati-have-a-love a boy like that wants one thing only, and when ~
## 10 a-boy-like-thati-have-a-love he'll murder your love; he murdered mine.
## # ... with 265 more rows
```

Stopword removal with tokenized text

If we sort by songname and word and `count` the occurrences of words within each song, the variable `n` shows the number of words for each song from highest to lowest using `sort = TRUE`. Here, we see that stopwords such as "a", "i" and "is" are occurring rather frequently.

```
westside_tokens %>%
  count(songname, word, sort = TRUE) # Counting the number of words per songsame and sorting from highe
```

```
## # A tibble: 1,224 x 3
##    songname                      word        n
##    <chr>                         <chr>   <int>
##  1 gee-officer-krupke            a          32
##  2 maria                         maria      23
##  3 i-feel-pretty                 pretty     19
##  4 tonight-quintet               tonight    19
##  5 a-boy-like-thati-have-a-love i          17
##  6 america                       in         17
##  7 america                       america    16
##  8 gee-officer-krupke            is         16
##  9 jet-song                      you're     16
## 10 i-feel-pretty                 and        15
## # ... with 1,214 more rows
```

To remove the stopwords, we can use `anti_join`. This is an alternative to going back to the character vector in the list and work with *regex*. When we have the data stored as words in a column in a dataframe, we can join against another similar dataframe and remove the cells that have the same word. The `tidytext` package has an inbuilt dataframe called `stop_words` containing stopwords that we might want to remove. Notice that the column in the stop_words dataframe that we want to `anti_join` against is called `word`, which means that to make things easier for ourselves, the column in our dataframe should also be called `word`.

```
tidytext::stop_words # A dataframe with stopwords within the tidytext package
```

```
## # A tibble: 1,149 x 2
##    word        lexicon
##    <chr>       <chr>
```

```
##   1 a           SMART
##   2 a's         SMART
##   3 able        SMART
##   4 about       SMART
##   5 above       SMART
##   6 according   SMART
##   7 accordingly SMART
##   8 across      SMART
##   9 actually    SMART
##  10 after       SMART
## # ... with 1,139 more rows
```

```
westside_tokens <- westside_tokens %>%
  anti_join(stop_words, by = "word") # Joining against the stopwords dataframe to get rid of cells with

westside_tokens %>%
  count(songname, word, sort = TRUE)
```

```
## # A tibble: 568 x 3
##    songname                     word        n
##    <chr>                        <chr>   <int>
##  1 maria                        maria      23
##  2 i-feel-pretty                pretty     19
##  3 tonight-quintet              tonight    19
##  4 america                      america    16
##  5 tonight-duet                 tonight    13
##  6 i-feel-pretty                feel       12
##  7 cool                         boy        11
##  8 a-boy-like-thati-have-a-love love       10
##  9 gee-officer-krupke           krupke      9
## 10 a-boy-like-thati-have-a-love boy         8
## # ... with 558 more rows
```

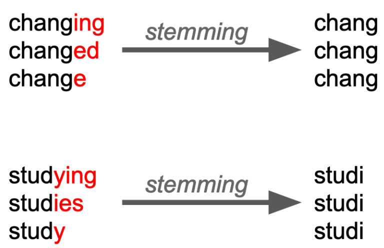
That looks better! If you have bigrams and want to remove stopwords using this method, have a look at chapter 4 in Tidytext.

Stemming the word

Consider the words "have", "having" and "had". Would you say that they are more or less different than "have" and "run"? Inflections of words usually do more to the noise in our dataset than give useful information. Because once the word is spelled differently, computer interprets the words as completely distinct words. That is why we often choose to stem. In the word "wait", for example, stemming involves recuding the word to "wait" for all its different inflections:

- **wait** (infinitive)
- **wait** (imperative)
- **wait**s (present, 3rd person, singular)
- **wait** (present, other persons and/or plural)
- **wait**ed (simple past)
- **wait**ed (past participle)
- **wait**ing (progressive)

So stemming means taking the *stem* of the word, also called the *root* or the *base*. This goes for verbs and nouns alike, and it might involve chopping off parts of the words such as affixes and suffixes[2].



	original_word	stemmed_words
0	connect	connect
1	connected	connect
2	connection	connect
3	connections	connect
4	connects	connect

	original_word	stemmed_word
0	trouble	troubl
1	troubled	troubl
2	troubles	troubl
3	troublemsome	troublemsom

To stem tokens in R, load a wordstemming package such as `SnowballC` and use the function `wordStem` on the variable with the tokens.

```
library(SnowballC)

westside_tokens <- westside_tokens %>%
  mutate(stem = wordStem(word))
```

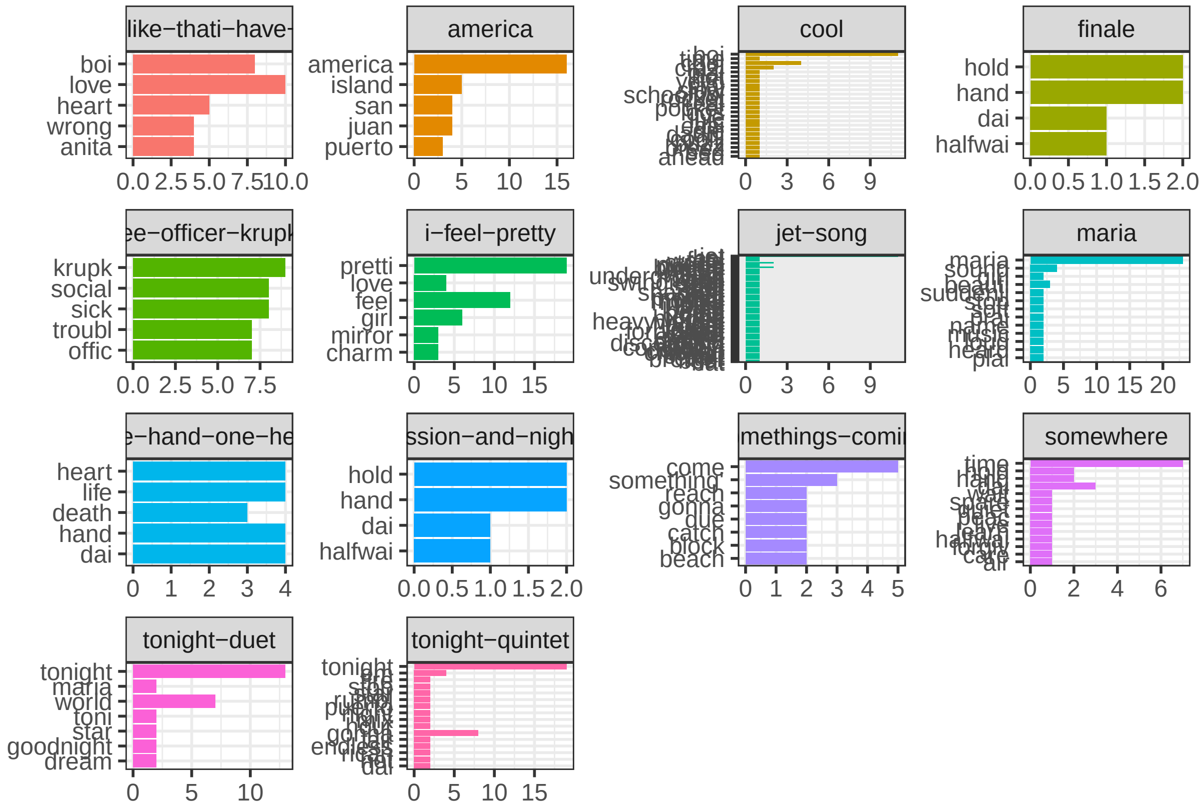
Plotting tokenized text

One way of extracting insight from text is to tokenize it and then plot it.

```
plot_df <- westside_tokens %>% # Make a new dataframe to plot
  count(songname, stem) %>% # Counting the number of words per songname
  group_by(songname) %>% # Grouping by songname and...
  slice_max(n, n = 5) %>% # Taking out the five top most used words per songname
  ungroup() # Ungrouping to get the dataframe back to normal

plot_df %>%
  ggplot(aes(n, fct_reorder(stem, n), # Placing number of occurences (n) on the x-axis and word on the
             fill = songname)) + # Adding colors to the bars after the songname
  geom_bar(stat = "identity") + # Making a barplot where the y-axis is the measure (stat = "identity")
  facet_wrap(~ songname, ncol = 4, scales = "free") + # Making one small plot per songname, arranging t
  labs(x = "", y = "") + # Removing all labels on x- and y-axis
  theme_bw() + # Make the background white
  theme(legend.position = "none") # Removing the legend for what colors mean
```

[2] A more nuanced way of recuing a word to its root is to use *lemmatization*. Here, you also take into account the context in which the word appears. It is a more complex method and often not necessary, but it can be fruitful in some analyses.

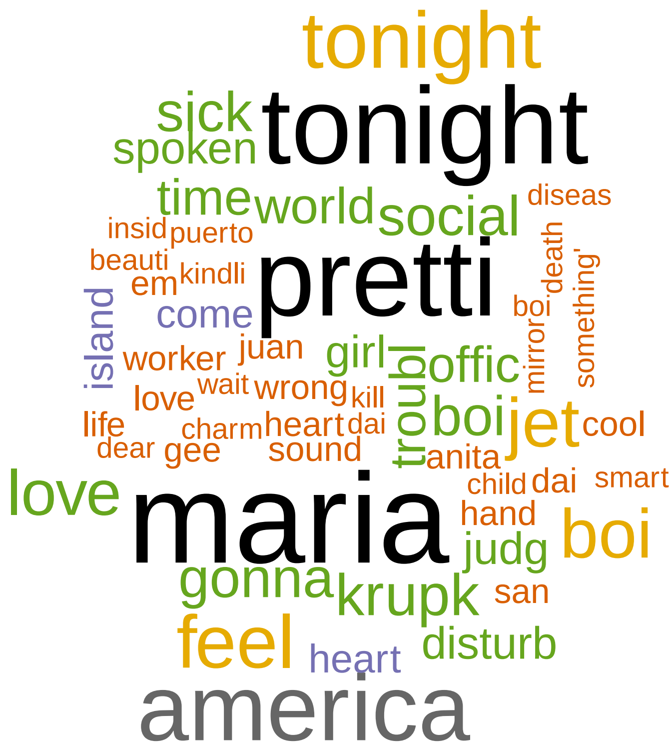


Or, we can make a wordcloud. There are many packages available to make wordclouds, in this case, I use the `wordcloud` package.

```
library(wordcloud) # Loading one package that we can use to make wordclouds

wordcloud_df <- westside_tokens %>%
  count(songname, stem) # Counting the number of words per songname

wordcloud_df %>%
  with(wordcloud(stem, n, # Making a wordcloud with the words weighed by how frequent they are
                 max.words = 100, # Plotting maximum 100 words to not make the plot too big
                 colors = brewer.pal(8, "Dark2")[factor(westside_tokens$songname)])) # Coloring the wor
```



Text vectorization

Once we have tokenized the text and done the last bit of preprocessing, we are ready to vectorize the text. Text vectorization involves converting the text into a numerical representation. Once again, we have several options, but in this class we'll focus on two of the easiest; *bag of words* and *TF-IDF*.

Bag of words

Imagine that you take all the documents that you're analyzing, count how many times every single word shows up in the document, and put it into a matrix. This is *bag of words*. It's a matrix with documents as the rows and words as the columns. In our case, the rows would have been songnames. This means that when we create a matrix with bag of words, the matrices become very big (wide). We have 14 songs, but hundreds of words.

	the	cat	sat	on	hat	dog	ate	and
Document 1	2	1	1	1	1	0	0	0
Document 2	3	1	0	0	1	1	1	1

To exemplify the process, let's pick three sentences from our songs and put them into a matrix. As you can see, the matrix quickly expands as we add text. Here, we can also see the benefits of preprocessing the text – we could avoid quite a bit of noise if we remove the stopwords before vectorizing.

Sentence	you	were	in	love	or	so	said	i	know	boat	a	can	get	on	she	isn't	merely	insane
you were in love or so you said	2	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
i know a boat you can get on	1	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0
she isn't in love she's merely insane	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1

How to create a bag of words in R? One possibility is to use the `quanteda` package and the function `cast_dfm`. To do this, we first need to `count` the number of times each words shows up in each song, as we've done before. Here, I also show how to use `name = "count"` to call the new variable "count". Then use `cast_dfm`, and as arguments, we specify (1) the documents to be the rows, (2) the tokens to be the columns and (3) the number to fill the cells. `dfm` stands for document feature matrix.

```
library(quanteda)

westside_tokens %>%
  count(songname, stem, name = "count") %>% # By default, the count-variable is called "n". Use name =
  cast_dfm(songname, # Specify the douments in your analysis (becoming the rows)
           stem, # Specify the tokens in your analysis (becoming the columns)
           count) # Specify the number of times each token shows up in each document (becoming the cell
```

```
## Document-feature matrix of: 14 documents, 444 features (91.09% sparse) and 0 docvars.
##                                 features
## docs                             anita belong boi brother care forev forget head
##   a-boy-like-thati-have-a-love       4      1   8       1    1     1      1    1
##   america                            0      0   0       0    0     0      0    0
##   cool                               0      0  11       0    0     0      0    0
##   finale                             0      0   0       0    0     0      0    0
##   gee-officer-krupke                 0      0   3       1    1     0      0    1
##   i-feel-pretty                      0      0   2       0    0     0      0    0
##                                 features
## docs                             hear heart
##   a-boy-like-thati-have-a-love      1     5
##   america                           0     0
##   cool                              0     0
##   finale                            0     0
##   gee-officer-krupke                0     0
##   i-feel-pretty                     0     0
## [ reached max_ndoc ... 8 more documents, reached max_nfeat ... 434 more features ]
```

TF-IDF

Bag of words work surprisingly well for being such a crude way of vectorizing, but it definitely has its limits. For example, it does nothing to try to convey how important a specific word is at representing the text. In the sentence: "Democracy is under attack", most of us would agree that the words "democracy" and "attack" are more important to the meaning of the sentence than "is" and "under". However, a bag of words technique would weigh all words equally, and this technique might give too much weight to unimportant words, since these tend to show up more often than important words.

To account for this, we introduce a way of weighing the words. This technique gives more weight to words that occur rather frequently in a text compared to how often the word occurs in other texts in our selection. Thus, we assume that words occurring frequently in a text compared to other texts will carry more meaning as to what the text is about. Words occurring frequently in all texts – such as "under" and "before" – are given less weight. The weighing mechanism is called *term frequency - inverse document frequency*, shortened *TF-IDF*.

Basically, the TF-IDF measure counts the number of times a given term shows up in a document (term frequency), and divides it by a measure showing how frequent the term is in the selection of documents overall (inverse document frequency)[3].

$TF - IDF = \frac{TermFrequency}{InverseDocumentFrequency}$

To do calculate the TF-IDF in R, we can use the function `bind_tf_idf` from the `tidytext` package. This gives us both the term frequency (tf), the inverse document frequency (idf) and the term frequency inverse document frequency (tf-idf) for every token in the texts.

```
westside_tokens <- westside_tokens %>%
  count(songname, stem, name = "count") %>%
  bind_tf_idf(stem, songname, count) # Making the tf-idf measure

westside_tokens
```

```
## # A tibble: 554 x 6
##    songname                     stem    count     tf   idf tf_idf
##    <chr>                        <chr>   <int>  <dbl> <dbl>  <dbl>
##  1 a-boy-like-thati-have-a-love anita       4 0.0571  2.64 0.151
##  2 a-boy-like-thati-have-a-love belong      1 0.0143  2.64 0.0377
##  3 a-boy-like-thati-have-a-love boi         8 0.114   1.03 0.118
##  4 a-boy-like-thati-have-a-love brother     1 0.0143  1.54 0.0220
##  5 a-boy-like-thati-have-a-love care        1 0.0143  1.54 0.0220
##  6 a-boy-like-thati-have-a-love forev       1 0.0143  1.95 0.0278
##  7 a-boy-like-thati-have-a-love forget      1 0.0143  1.95 0.0278
##  8 a-boy-like-thati-have-a-love head        1 0.0143  1.95 0.0278
##  9 a-boy-like-thati-have-a-love hear        1 0.0143  2.64 0.0377
## 10 a-boy-like-thati-have-a-love heart       5 0.0714  1.95 0.139
## # ... with 544 more rows
```

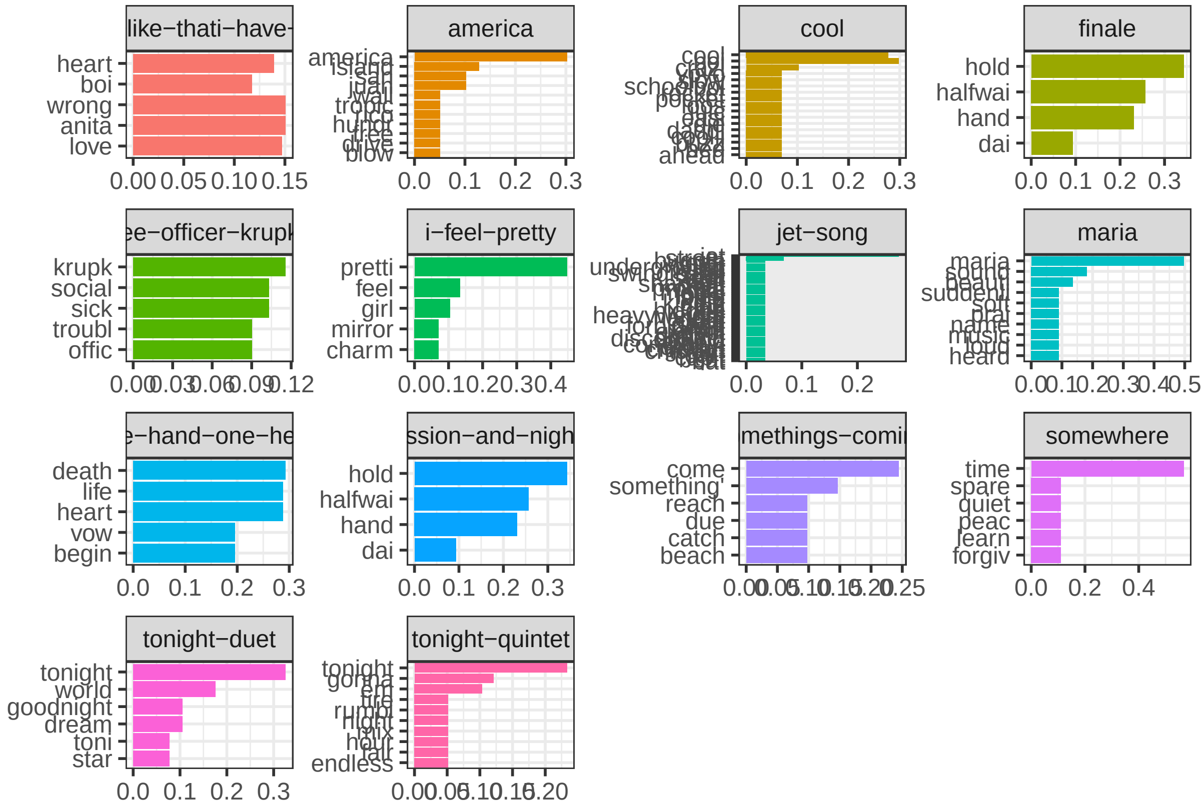
We can also plot these tf-idf weighed tokens as we did above. Although the difference is not glaring in this case because we do not have that much text to work with, we can for example notice that the word "boi" ("boy") has gotten a lower weight since it shows up frequently in several different songs.

```
plot_df_tfidf <- westside_tokens %>%
  group_by(songname) %>%
  slice_max(tf_idf, n = 5) %>% # Get the five highest units on the tf-idf measure
  ungroup()

plot_df_tfidf %>%
  ggplot(aes(tf_idf, fct_reorder(stem, tf_idf), # Plotting words based on their tf-idf score
             fill = songname)) +
  geom_bar(stat = "identity") +
  facet_wrap(~ songname, ncol = 4, scales = "free") +
  labs(x = "", y = "") +
```

[3] The inverse document frequency measure for any term is calculated as $idf(term) = ln(\frac{n_{Documents}}{n_{DocumentsContainingTerm}})$

```r
  theme_bw() +
  theme(legend.position = "none")
```



And we can create a document feature matrix (dfm) using tf-idf instead of bag of words.

```r
westside_tokens %>%
  cast_dfm(songname,
           stem,
           tf_idf) # Specify tf-idf
```

```
## Document-feature matrix of: 14 documents, 444 features (91.09% sparse) and 0 docvars.
##                                 features
## docs                                 anita     belong        boi     brother
##   a-boy-like-thati-have-a-love 0.1508033 0.03770082 0.11767079 0.022006358
##   america                      0         0          0          0
##   cool                         0         0          0.29804773 0
##   finale                       0         0          0          0
##   gee-officer-krupke           0         0          0.01506760 0.007514366
##   i-feel-pretty                0         0          0.01838606 0
##                                 features
## docs                                   care      forev     forget        head
##   a-boy-like-thati-have-a-love 0.022006358 0.02779872 0.02779872 0.027798716
##   america                      0           0          0          0
##   cool                         0           0          0          0
##   finale                       0           0          0          0
```

```
##   gee-officer-krupke           0.007514366 0          0          0.009492245
##   i-feel-pretty                0           0          0          0
##                               features
## docs                                 hear     heart
##   a-boy-like-thati-have-a-love 0.03770082 0.1389936
##   america                      0          0
##   cool                         0          0
##   finale                       0          0
##   gee-officer-krupke           0          0
##   i-feel-pretty                0          0
## [ reached max_ndoc ... 8 more documents, reached max_nfeat ... 434 more features ]
```

```
saveRDS(westside_tokens, file = "../../datafolder/westside_tokens.rds")
```