Combining `perl`, `c` and `R`

Bruna Wundervald

May, 2019

1 Introduction

Music information retrieval is a recent area is a science, which has been growing fastly. It combines computational tools and music theory to amplify our knowledge and utility of music data, in the most different formats. There are many ways of representing music data, such as chords, lyrics, audio, and so on. Each one of those carries different levels of information about the songs, and they will be used for distinct tasks.

In `R`, the `chorrrds` package is responsible for extracting music chords of a given artist. The chords come in a character format, meaning that some treatment needs to be done if we want to use this data properly. For lyrics data, the `vagalumeR` package provides access to the Vagalume API (https://www.vagalume.com.br/), a music lyrics website. The lyrics also come in a text format, giving us plenty of data to explore and evaluate. These two packages are a great source of data for music analysis in general.

In this report, we will present how to extract data using those two packages in `R`. In addition to that, we will manipulate the data using `perl` and `c`, but interfacing the languages with `R`. We do that because `R` is the most well-known language in the statistics community. This means that even when we write code in more efficient languages, like `c` or `fortran`, there is always a port to `R`. This makes it easy for the users, who in general are not programmers, to actually use it.

2 `perl`

`Perl` is a general-purpose programming language, originally developed for text manipulation. Now, `perl` is used for any type of task, including system administration, web development, GUI development, and more.

The biggest potential of `perl` is definitely its capability with handling strings. For such reasons, `perl` is commonly called by other languages when a function needs to handle strings. In `R`, for instance, there are a few functions that allows the user to set `perl = TRUE`, to make it easier to use *perl-like* regular expressions.

The script above shows the `perl` code that receives an argument and searches for some regular expressions in it using perl. The goal of this is to receive, in `R`, a character that represents a chord in a song and return meaningful features about this chord. In this case, we have an interest in finding out if the chord is:

- minor,
- diminished,
- augmented,
- sus,
- chord with the 7th,
- chord with the major 7th,
- chord with the 6th,
- chord with the 4th,
- chord with the augmented 5th,
- chord with the diminished 5th,
- chord with the 9th,
- chord with varying bass.

What happens is that the expression of a chord, like "Cm" or "D#", does not really means anything for the computer. This way, it's not easy to extract any kind of information using only the raw version of the chord. The extracted features will represent useful information regarding the harmonic information of the chords.

```
use constant FALSE => 1==0;
use constant TRUE => not FALSE;

# introducing the chord to do the regex
my $chord = $ARGV[0];
# calculating all 'regexes'
$min =  $chord =~ /m/;
$dim =  $chord =~ /dim|\u00b0/;
$aug =  $chord =~ /aug|\+/;
$sus =  $chord =~ /(sus)/;
$with7 =  $chord =~ /7/;
$maj7 =  $chord =~ /7(M|\+)/;
$with6 =  $chord =~ /(6|13)/;
$with4 =  $chord =~ /(4|11)/;
$aug5 =  $chord =~ /5(#|\\+)/;
$dim5 =  $chord =~ /5(b|-)/;
$with9 =  $chord =~ /(9|2)/;
$bass = $chord =~ /(?<=\/).*/;
# returning results as 0 and 1 for true and false
@array1 = ($min*1, $dim*1, $aug*1, $sus*1, $with7*1, $maj7*1, $with6*1,
$with4*1, $aug5*1, $dim5*1, $with9*1, $bass*1);
print @array1;
```

3 c

C, on the other hand, is a high-level and general-purpose programming language. It is famous for being super fast, especially when we need to deal with matrices. For algorithms that need to do extensive countings, for example, c is an option that will give a good performance.

In this report, we use c to count how many words there are in a song, using its lyrics. The code below shows how to receive a text file from the command line and count how many words it has.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {

  FILE* fptr = fopen(argv[1],"r"); /* opening file passed by the cmd */
  if(fptr == NULL){ /* checking if file is something */
    perror("fopen");
    exit(EXIT_FAILURE);
  }

  char ch;
  int w = 0;

  fptr = fopen(argv[1], "r"); /* actually opening it */
  if (fptr == NULL) { /* checking if it worked */
    printf("can't open file");
  }
  else {
    ch = fgetc(fptr);
    while (ch != EOF) {  /* counting the words (separated by space or \n) */
```

```
      if (ch == ' ' || ch == '\n') {
        w++;
      }
      ch = fgetc(fptr);
    }
    printf("Words in song are = %d", w); /* returning the results */
  }
  fclose(fptr);
}
```

4 Using it R

Finally, we use R to obtain and manipulate the data. The chords are extracted with the chorrrds package, and the lyrics with the vagalumeR package. Note that, for the lyrics, we need the API credential, which will not be shown here since that is confidential information for each API user. We will be using the band Muse in this case.

Both the connections to perl and c are made by the system() function, which pretty much just invokes the OS command. For c, since this is a famous language, there are more options than this one, using the Rcpp or inline packages, that even allow the user to save c code as objects in R. For now, we will stay with the system() option.

```
library(ggridges)
library(tidyverse)

songs <- chorrrds::get_songs("muse")
chords <- chorrrds::get_chords(songs$url)
```

```
# dimensions of our data
dim(chords) # 3577 rows and 3 columns
```

```
## [1] 3623    3
```

```
glimpse(chords)
```

```
## Observations: 3,623
## Variables: 3
## $ chord <fct> B, C, B, B, C, F#m, E, Bm, D, A5, E, F#m, E, Bm, D, A5, ...
## $ key   <fct> G, G, G, G, G, A, A, A, A, A, A, A, A, A, A, A, A, A, A,...
## $ music <fct> muse a crying shame, muse a crying shame, muse a crying ...
```

```
# saving only the unique chords found
chords_unique <- chords %>%
  pull(chord) %>%
  unique()

# calculating the results for each chord using perl function
regex <- chords_unique %>%
  stringr::str_remove_all(., pattern = "\\(|\\)") %>%
  map(~{
    arg <- .x
```

```
    cmd <- paste("perl", "assignment/regex.pl", arg)
    system(cmd, intern = TRUE)
  })

# obtaining the formatted results
results <- regex  %>% map(~{
  stringr::str_split_fixed(.x, n = 12, pattern = "") %>% c()
}) %>%
  setNames(1:126) %>%
  bind_rows() %>%
  t() %>%
  as.data.frame() %>%
  setNames(
    c("minor", "dimi", "augm", "sus",
      "seventh", "seventh_M", "sixth",
      "fourth", "fifth_aug", "fifth_dim",
      "ninth", "bass")
  ) %>%
  mutate_if(is.factor, as.character) %>%
  mutate_if(is.character, as.numeric) %>%
  mutate(chord = chords_unique) %>%
  full_join(chords, by = ('chord' = 'chord'))

glimpse(results) # 3577 rows and 15 columns now
```

```
## Observations: 3,623
## Variables: 15
## $ minor     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ dimi      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ augm      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ sus       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ seventh   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ seventh_M <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ sixth     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ fourth    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ fifth_aug <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ fifth_dim <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ ninth     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ bass      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ chord     <fct> B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B...
## $ key       <fct> G, G, G, F#, F#, F#, F#, F#, F#, D, D, D, G, G, G, G...
## $ music     <fct> muse a crying shame, muse a crying shame, muse a cry...
```

```
# creating a plot to visualize the densities of the extracted variables
df <- results %>%
  select(-chord, -key) %>%
  group_by(music) %>%
  summarise_all(mean) %>%
  tidyr::gather(group, vars, minor, seventh, sus,
                seventh_M, sixth, fifth_dim, fifth_aug,
                fourth, ninth, bass, dimi, augm) %>%
  mutate(group = as.factor(group))
```

```r
df$group <- forcats::lvls_revalue(
  df$group,
  c("Augmented", "Bass", "Diminished",
    "Augm. Fifth", "Dimi. Fifth",
    "Fourth", "Minor", "Ninth", "Seventh",
    "Major Seventh", "Sixth", "Sus"))

# visualizing it!
df %>%
ggplot(aes(vars, group, fill = group)) +
  geom_density_ridges(alpha = 0.6) +
  scale_fill_cyclical(values =  "darksalmon") +
  guides(fill = FALSE) +
  xlim(0, 1) +
  labs(x = "Densities", y = "Extracted variables") +
  theme_bw(14)
```

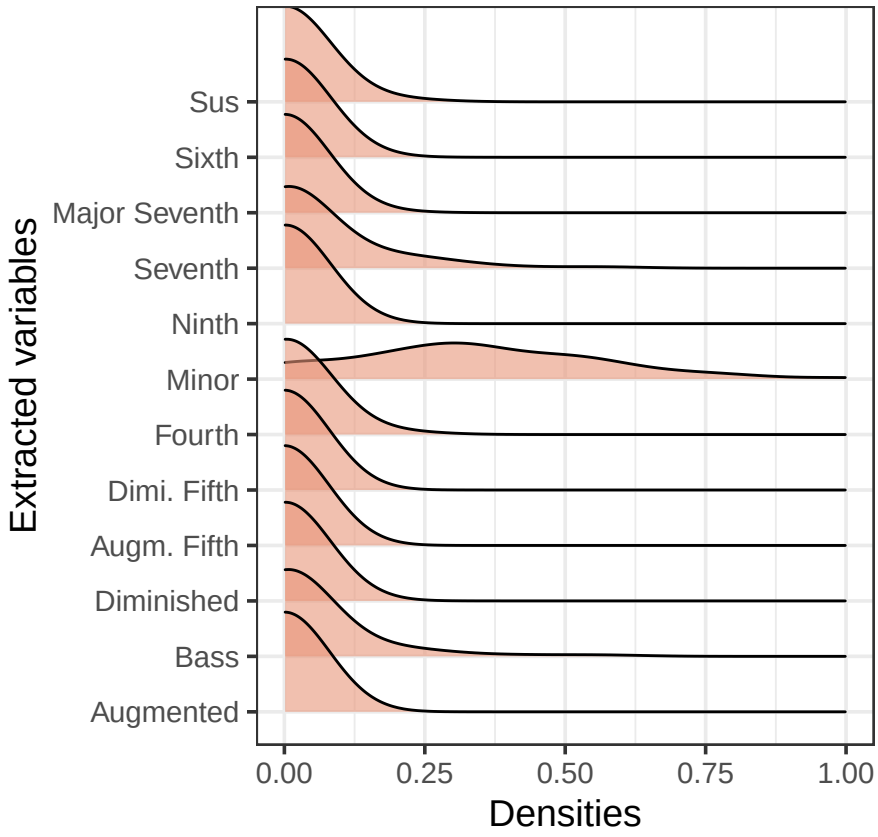

Figure 1: Densities of the extracted features for Muse songs

The Figure 1 shows us the densities of the features extracted. We can see that there are not many differences for most of the features. The only one that stands out a little is the proportion of minor chords in Muse songs, indicating that they use it a lot when composing.

```r
source("assignment/credentials.R")

# obtaining lyrics from Muse songs with the Vagalume API
songs <- "muse" %>%
  purrr::map_dfr(vagalumeR::songNames)

# formatted results for lyrics
lyrics <-  songs %>%
  dplyr::pull(song.id) %>%
  purrr::map(vagalumeR::lyrics,
             artist = "muse",
             type = "id",
             key = vg) %>%
  purrr::map_dfr(data.frame) %>%
  dplyr::select(-song) %>%
  dplyr::right_join(songs %>%
                      dplyr::select(song, song.id), by = "song.id")

dim(lyrics) # 143 rows and 8 columns
# saving it!
write.table(lyrics, "lyrics.txt")

# saving lyrics separately to use with c
purrr::map2(
  .x = lyrics$text %>% as.character(),
  .y = lyrics$id,
  .f =
    ~{
      write.table(.x,
                  file = paste0("lyrics/file_", .y, ".txt",
                                collapse = ""))

    })
```

```r
# applying c function to each lyric
fs <- list.files("assignment/lyrics/")

res <- fs %>% map_chr(~{
  paste(getwd(),"/count ", paste0("assignment/lyrics/", .x) ,sep="")}) %>%
  purrr::map_chr(~{system(.x, intern = TRUE)})

# obtaining formatted results
df <- data.frame(song = lyrics$song,
                 n_words = str_extract(res, "[0-9]{1,5}") %>% as.numeric())
```

```r
# visualizing it!
df %>%
  ggplot(aes(reorder(song, n_words), n_words)) +
  geom_bar(alpha = 0.6, stat = "identity", fill = "darksalmon") +
  guides(fill = FALSE) +
  coord_flip() +
  labs(x = "Songs", y = "Number of words") +
  theme_bw(10)
```

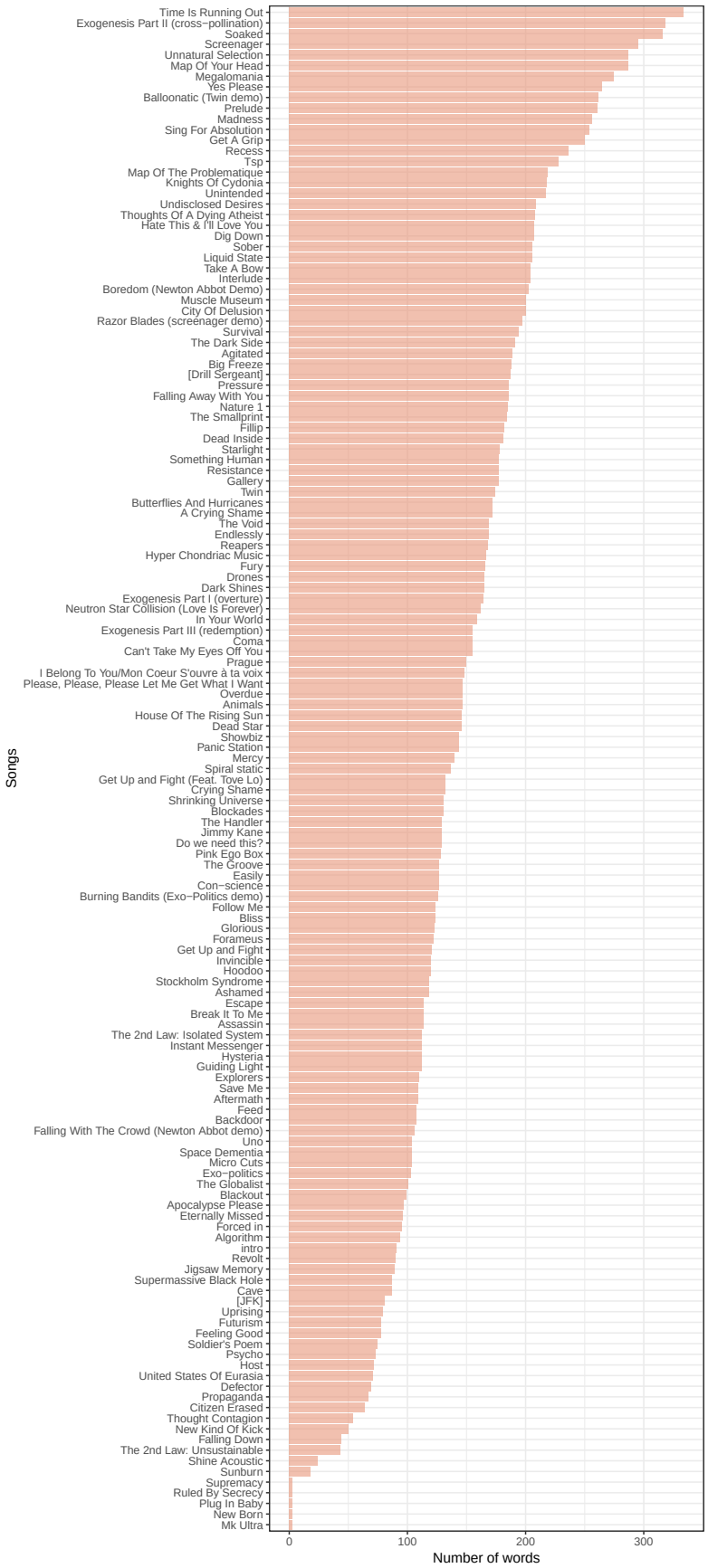

Figure 2: Word counts for each Muse song found in the Vagalume website

Figure 2, on the other hand, shows us the word counts for each song. We can see that some songs do not have many words, which probably means that they are mostly instrumental. Other songs have many words, showing that they are more focused on the lyrics, since that is a strong characteristic of the Muse band.