UBMI-IFC, UNAM
Coyoacan, CDMX

Sequence Characterization Test 2

Using Genomic-Benchmarks Data

Author: Fuentes David
Tutors:
- PhD. Poot Augusto
- MSc. Pedraza Carlos

Table of contents

1 Data Preparation

1.1 Downloading data

Python Code

```
# To inspect each dataset to select two
from genomic_benchmarks.data_check import info as \
    info_gb

# To download each dataset
from genomic_benchmarks.loc2seq import \
    download_dataset

import os
```

```
# To load the 'custom-functions' module
os.chdir("/path/to/Project/scripts")
from custom_functions import *
```

```
info_gb("human_ensembl_regulatory", version=0)
```

```
download_dataset("human_ensembl_regulatory", version=0)
```

1.2 Formatting data

Concatenate sequences in '.txt' files into a single FASTA (this is a temporary solution since the FASTA has "ugly" headers (counter doesn't work)) 'find' + 'xargs' approach used since awk couldn't handle >80K files at once

Bash Code

```
cd /path/to/Project/datasets
mkdir GenomicBenchmarks

# Move sequences to our project datasets directory
mv ~/.genomic_benchmarks/human_ensembl_regulatory \
   ./GenomicBenchmarks

elements=("enhancer" "promoter" "OCR")
data_path="GenomicBenchmarks/human_ensembl_regulatory/train"

# Concatenates sequence 'txt' files into single FASTA
# NOTE: '${variable,,}' changes string 'variable' to lowercase
for element in "${elements[@]}"; do
  find "${data_path}/${element,,}/" -type f -name '*.txt' | \
  xargs -I {} awk -v prefix="${element,,}" \
        'BEGIN{counter=0} {print ">"prefix"_"counter"|training";
         print $0; counter+=1}' {} \
         > "GB-Test/${element}s_training.fasta"
done
```

2 Data Characterization

2.1 Libraries required

Not many changes had been currently done to the libraries required, however many are expected to be included in order to perform some statistical tests. Their list, and relevant descriptions are below:

R Code

```
# For genome-functions.R
library(stringr)
library(stringi)
library(primes)
# Data handling
library(data.table)  # Large Data Processing
library(dplyr)
library(plyr)
# For parallel computing
library(doParallel)
library(foreach)
# For biological functions:
library(Biostrings)  # Global Alignments
library(DNAshapeR)   # DNA Shape Features
# For plotting
library(paletteer)   # Colot Palettes
library(cowplot)     # Plot Grids
library(ggplot2)
library(see)         # Half-Violing Plots
# For pretty tables
library(knitr)       # PDF rendering options
library(kableExtra)  # Prettier tables
# For statistic analysis
library(stats)
library(irlba)       # Memory-efficient PCA
# library(nortest)
```

Then we load both our "pretty-display" functions and our genomic functions (which we've changed to reduce the biases identified in the previous test).

```
project_path <- "/path/to/Project/"
source(paste0(project_path, "scripts/genome-functions.R"))
source(paste0(project_path, "scripts/custom-functions.R"))
```

2.2 Sequence Characterization

Note that we did not use the full extent of both 'promoters' and 'OCRs' sequence sets, since this allowed us to divide their elements evenly.

```
# Note: Further modifications were made to corresponding Rscript
elements <- c("enhancers", "promoters", "OCRs")
Ns_sequences <- c(85512, 75930, 69900)
# Total numbers of sequences are 85512, 75934 and 69902
Ns_elements_per_csv <- c(3563, 2531, 2330)
Ns_clusters <- c(12, 10, 10)

for (n in seq_along(elements)) {
  element_n <- elements[n]
  n_sequences <- Ns_sequences[n]
  n_elements_per_csv <- Ns_elements_per_csv[n]

  n_cycles <- n_sequences / n_elements_per_csv
  corescluster <- makeCluster(Ns_clusters[n])
  registerDoParallel(corescluster)

  element_path <- paste0(project_path,
                         "datasets/GB-Test/", element_n)
  fasta_path <- paste0(element_path, "_training.fasta")
  seqs <- scan(fasta_path, character(),
```

```
              quote = "")[seq(2, n_sequences * 2, 2)]

  libs <- c("stringr", "stringi", "primes")
  foreach(i = 1:n) %dopar% {
    required_libs(libs)
    i_start <- ((i - 1) * n_elements_per_csv) + 1
    i_final <- i * n_elements_per_csv
    if (i > 1) {
      write.table(sequences_characterizer(seqs[i_start:i_final],
                                          Ks = c(2, 3, 5)),
                  paste0(element_path, "-training_", i, ".csv"),
                  sep = ",", row.names = FALSE, col.names = FALSE)
    } else {
      write.csv(sequences_characterizer(seqs[i_start:i_final],
                                        Ks = c(2, 3, 5)),
                paste0(element_path, "-training_", i, ".csv"),
                row.names = FALSE)
    }
  }
  stopCluster(corescluster)
}
```

This characterization process took around 3 hrs for each element (the longest set –of enhancers–, took 3.034 hours to finish), which came to an approximate of 9 hrs of processing time in total.

2.3 CSV Concatenation

Bash Code

```
cd /path/to/Project/datasets
mkdir ./GB-Test/tabulator-generated

elements=("enhancer" "promoter" "OCR")

# Concatenate all CSVs into their respective file
for element in "${elements[@]}"; do
  ls GB-Test/"${element}"*.csv | sort -V | \
  xargs cat > GB-Test/features_"${element}"s.csv
  # Move CSVs to different directory for better readability
  mv GB-Test/"${element}"*.csv GB-Test/tabulator-generated
done
```

2.4 Sequence Lengths

Due to the uniformity of sequence lengths (251 bp and 500 bp for promoters and enhancers repectively) of the previous dataset, it was decided to take account of sequence length distribution as a feature of the current dataset. However, since this was not one of the features with highest priority, I forgot to add it to '`sequence_characterizer`' before the generation of the already described CSVs. The following AWK script takes care of that:

```
cd /path/to/Project/datasets/GB-Test
elements=("enhancer" "promoter" "OCR")

for element in "${elements[@]}"; do
  awk '!/^>/{print length}' "${element}"s_training.fasta \
  > seqlens_"${element}"s.txt
done
```

3 Exploration analysis

3.1 Sequence Length Distribution

Due to the uniformity of sequence lengths (251 bp and 500 bp for promoters and enhancers repectively) of the previous dataset, it was decided to take account of sequence length distribution as a feature of the current dataset. However, since this was not one of the features with highest priority, I forgot to add it to '`sequence_characterizer`' before the generation of the already described CSVs. The following AWK script takes care of that:

R Code

```
setwd(paste0(project_path, "datasets/GB-Test"))

seq_lengths <- list(
  Enhancer = scan("seqlens_enhancers.txt", sep = "\n"),
  Promoter = scan("seqlens_promoters.txt", sep = "\n"),
  OCR = scan("seqlens_OCRs.txt", sep = "\n"))

list_lengths <- sapply(seq_lengths, length)

seq_lengths_df <- data.frame(
  Type = rep(names(seq_lengths), list_lengths),
  Length = unlist(seq_lengths)
)

seq_ranges_df <- seq_lengths_df %>%
  group_by(Type) %>%
  dplyr::summarise(Min = min(Length), Max = max(Length)) %>%
  dplyr::mutate(Position = row_number())

bw <- 20
max_len <- ((max(seq_lengths_df$Length) %/% bw) + 1) * bw
x_breaks <- seq(100, max_len, 100)
```

With this histogram we can notice that OCR sequences follow a more "*natural*" length distribution, while promoters and enhancers seem to distribute along semi-defined lengths (mainly 200, 400, and 600 bp). Since sequence length distribution is fairly different in each set of sequences, I will opt to momentarily leave the current 'sequence length' feature out of the principal table.

3.2 Kmer Features' Distributions

Since we only computed **kmers** of **size 2**, **3** and **5**, the number of per kmer feautures will be significantly less. This decision was taken after noticing in the previous test that kmers up to size 5 were still able to separate data after PCA. Now we'll make a visualization of the computed kmers.

```
source("../scripts/genome-functions.R")
```

```
Loading required package: MASS
```

```
Loading required package: survival
```

```
source("../scripts/custom-functions.R")
```

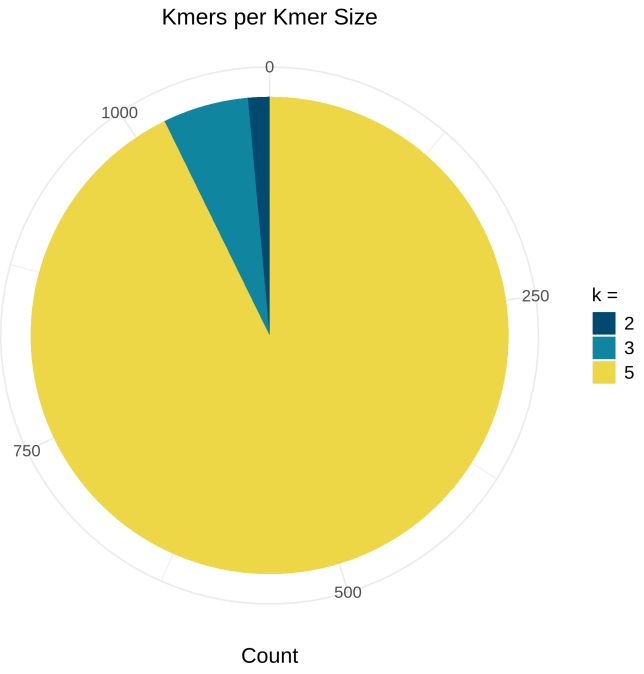
```
Attaching package: 'dplyr'
```

```
The following object is masked from 'package:kableExtra':

    group_rows
```

```
The following object is masked from 'package:MASS':
```

```
    select

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

Loading required package: Matrix
```

```r
k_Ns <- c(n_ki(2), n_ki(3), n_ki(5))
k_stt <- c(0); k_fin <- c(0)

for (k_n in k_Ns) {
  k_stt <- c(k_stt, tail(k_fin,1) + 1)
  k_fin <- c(k_fin, tail(k_fin,1) + k_n)
}

k_inds <- array(data = c(k_stt[-1], k_fin[-1]), dim = c(3, 2),
                dimnames = list(1:3, c("start","final")))

kmer_set_sizes <- data.frame(sizes = k_Ns,
                             k_sets = c("2", "3", "5"))
```



3.3 Feature Means and Standard Deviations

Given the amount of data generated, '`knitr`' seemed to have difficulties loading the generated CSVs, therefore in order to create the "Summary Table" of Means and Std. Deviations, an AWK approach was tried in order to exploit the line-by-line processing nature of said language. Since the initial problem with '`knitr`' was solved after disabling *LazyLoading* through the '`cache-lazy: false`' chunk-execution Quarto option, further comparisons with R should be made to justify its' use instead of an all-R approach. The following AWK code was saved as get_summary.awk:

AWK Code

```awk
BEGIN {
    FS = ","      # Input field separator (CSV format)
    OFS = ","     # Output field separator (CSV format)
    OFMT = "%.7g" # Same number of decimals as R
}
FNR == 1 {
    if (ARGIND == 1) {
        # Capture column headers from the first file
        for (i = 1; i <= NF; i++) {
            headers[i] = $i
        }
        # Print output headers
        print "Type", "Field", "Means", "StDevs"
    }
    next  # Skip the header row of each file
} {       # ELSE, process each lines values:
    for (i = 1; i <= NF; i++) {
        sum[i] += $i         # Sum of each column
        sumsq[i] += $i * $i  # Squares' sum for each column
    }
}
ENDFILE {
    # Determine the type based on the filename
    type = (FILENAME ~ /promoter/) ? "Promoter" : \
           (FILENAME ~ /enhancer/) ? "Enhancer" : \
           (FILENAME ~ /OCR/) ? "OCR" : "Unknown"
    # FNR as in File Number of Records (i.e. rows or lines)
    count = FNR - 1
    # Output summaries for each column
    for (i = 1; i <= NF; i++) {
        mean = sum[i] / count
        stddev = sqrt((sumsq[i] / count) - (mean * mean))
        print type, headers[i], mean, stddev
    }
    # Reset data for the next file
    delete sum
    delete sumsq
}
```

Assuming we're already have our CSVs inside "`/path/to/Project/`", we have to run the last AWK script as follows:

Bash Code

```bash
cd /path/to/Project/
awk -f scripts/get_summary.awk datasets/GB-Testing/*.csv \
    > datasets/GB-Test/summary_features.csv
```

R Code

```r
# setwd(project_path)
setwd("/home/davidfm/Projects/UBMI-IFC/EnhaProm/")
CRE_summary <- read.csv("datasets/summary_features.csv",
                        check.names = FALSE)
# CRE_summary2 <- read.csv("datasets/summary_CREs.csv",
#                          check.names = FALSE)
```

```r
nrows <- length(which(CRE_summary$Type == "Enhancer"))
nnrows <- length(which(CRE_summary$Type == "OCR"))
coffeetable(CRE_summary[c(1:13, nrows + 1:13,
```

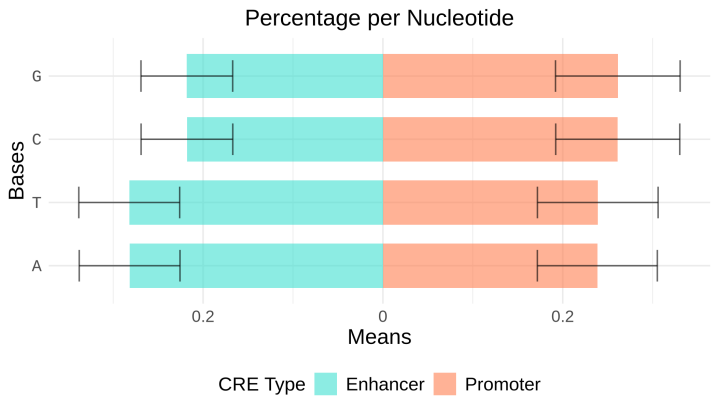
```
                    nrows + nnrows + 1:13), ])
```

	Type	Field	Means	StDevs
1		A	0.2817380	0.0560363
2		T	0.2821757	0.0560942
3		C	0.2179892	0.0510012
4		G	0.2180972	0.0510040
5		temp	80.5650100	3.1597480
6		shan	1.9548420	0.0407624
7	Enhancer	r_id	47.5854500	2.7667170
8		r_sc	-1045.7680000	547.1141000
9		rc_id	47.2004300	3.0255740
10		rc_sc	-1053.5820000	541.5453000
11		k2_shan	3.8237590	0.0861816
12		k2_adiv	0.9910934	0.0231475
13		k2_rdiv	0.0522719	0.0242676
3332		A	0.2869270	0.0628862
3333		T	0.2882950	0.0632890
3334		C	0.2124306	0.0516076
3335		G	0.2123474	0.0515038
3336		temp	79.9878900	3.4470800
3337		shan	1.9453030	0.0673908
3338	OCR	r_id	48.0614400	4.0694370
3339		r_sc	-858.9832000	301.8847000
3340		rc_id	47.5415600	4.4363470
3341		rc_sc	-868.4117000	297.0601000
3342		k2_shan	3.7984470	0.1779310
3343		k2_adiv	0.9852396	0.0505896
3344		k2_rdiv	0.0544983	0.0200769
6663		A	0.2385224	0.0667081
6664		T	0.2389874	0.0671027
6665		C	0.2611998	0.0689944
6666		G	0.2612903	0.0693120
6667		temp	85.0170100	4.8250460
6668		shan	1.9443910	0.0585401
6669	Promoter	r_id	47.4953600	2.7523480
6670		r_sc	-1488.8020000	338.5834000
6671		rc_id	47.0913000	3.1133880
6672		rc_sc	-1506.2620000	343.0558000
6673		k2_shan	3.8255740	0.1195835
6674		k2_adiv	0.9986764	0.0111762
6675		k2_rdiv	0.0310538	0.0134805

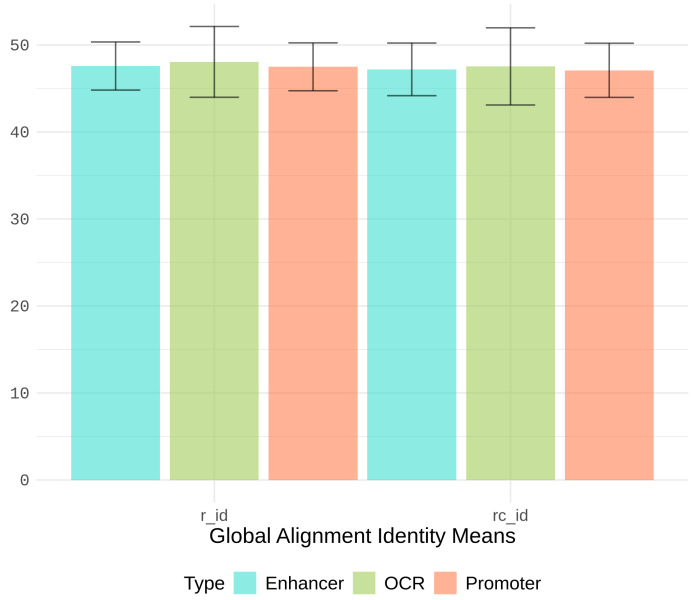
Now, in order to work with our summary table in a cleaner way, it's better if get a better hold of the column indexes of our "per-kmer" features. This will help us access the columns of KSG-product, barcode with an *exponent-based* metric or barcode with a *prime number-based* metric for either "promoters", "enhancers" or "OCRs".

```
pe_arr <- function(seq_data)
  return(array(data = seq_data, dim = c(rev(k_inds)[1], 3),
    dimnames = list(1:rev(k_inds)[1], c("enha","ocr","prom"))))

borders <- c(0, 3331, 6662)
inds_feats_per_kmers <- function(n, n_ws = 19,
                                 n_per_k = 3, total = 3331) {
  k_seq <- seq(n_ws + n, total - n_per_k + n, n_per_k)
  inds <- numeric(0)
  for (i in 0:(n_per_k - 1))
    inds <- c(inds, i * total + k_seq)
  return(inds)
}

indxs <- list(prod = pe_arr(inds_feats_per_kmers(n = 1)),
              bcds = pe_arr(inds_feats_per_kmers(n = 2)),
              bclp = pe_arr(inds_feats_per_kmers(n = 3)))
```

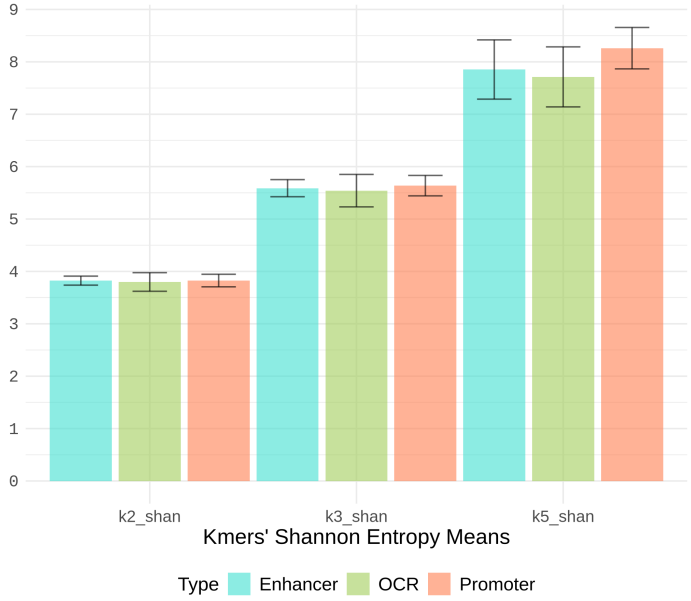
```
nucl_percs <- c(1:4 + borders[1], 1:4 + borders[3])

pyrplot_(CRE_summary[nucl_percs,],
         x_label = "Bases", y_breaks = seq(-1, 1, 0.2),
         title = "Percentage per Nucleotide")
```



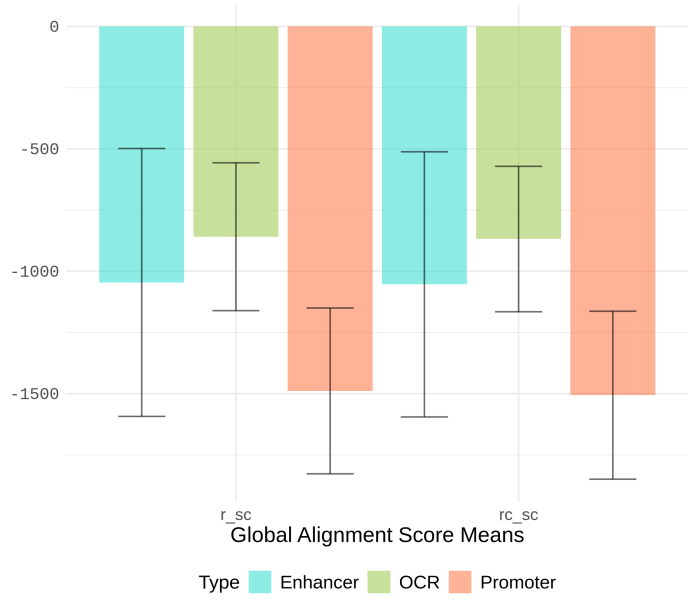

```
generate_inds <- function(base_values, borders)
  unlist(lapply(borders, function(b) base_values + b))

id_vals <- generate_inds(c(7,9), borders)
sc_vals <- generate_inds(c(8,10), borders)
kshan_vals <- generate_inds(c(11,14,17), borders)
kadiv_vals <- generate_inds(c(12,15,18), borders)
krdiv_vals <- generate_inds(c(13,16,19), borders)

ibarplot_(CRE_summary[id_vals,],
          legend_title = "Type",
          x_label = "Global Alignment Identity Means")
```



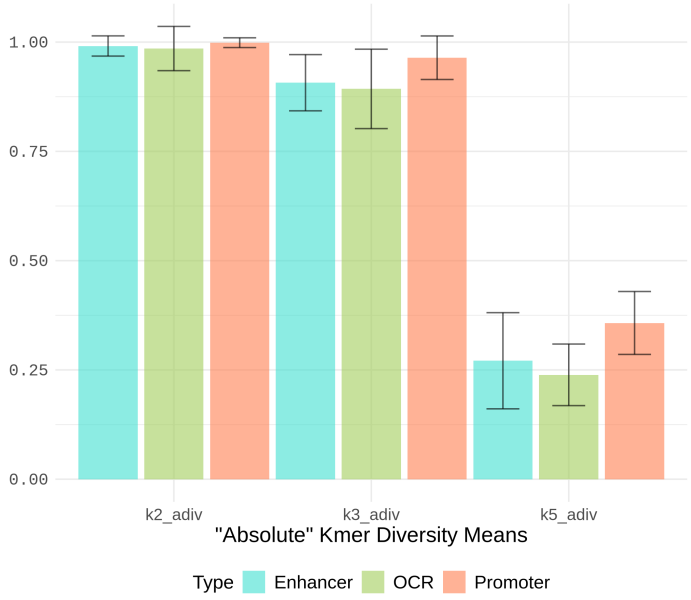

```
ibarplot_(CRE_summary[sc_vals,],
          legend_title = "Type",
          x_label = "Global Alignment Score Means")
```



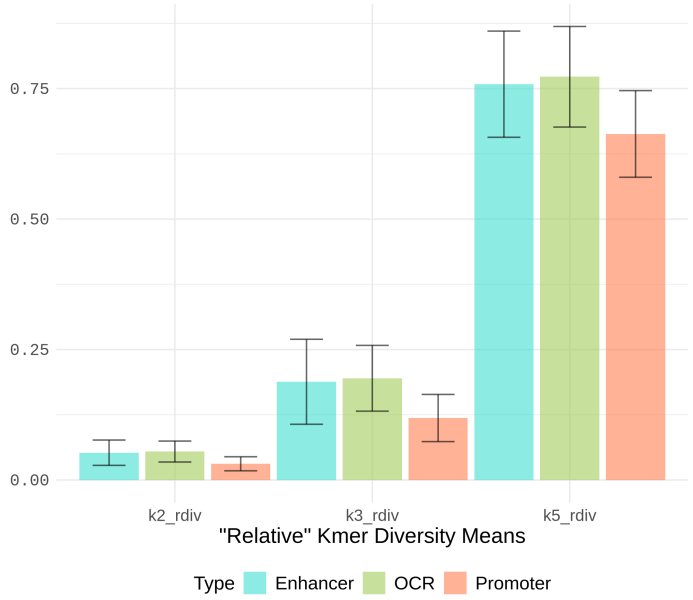

```
ibarplot_(CRE_summary[kadiv_vals,],
          legend_title = "Type",
          x_label = '"Absolute" Kmer Diversity Means')
```



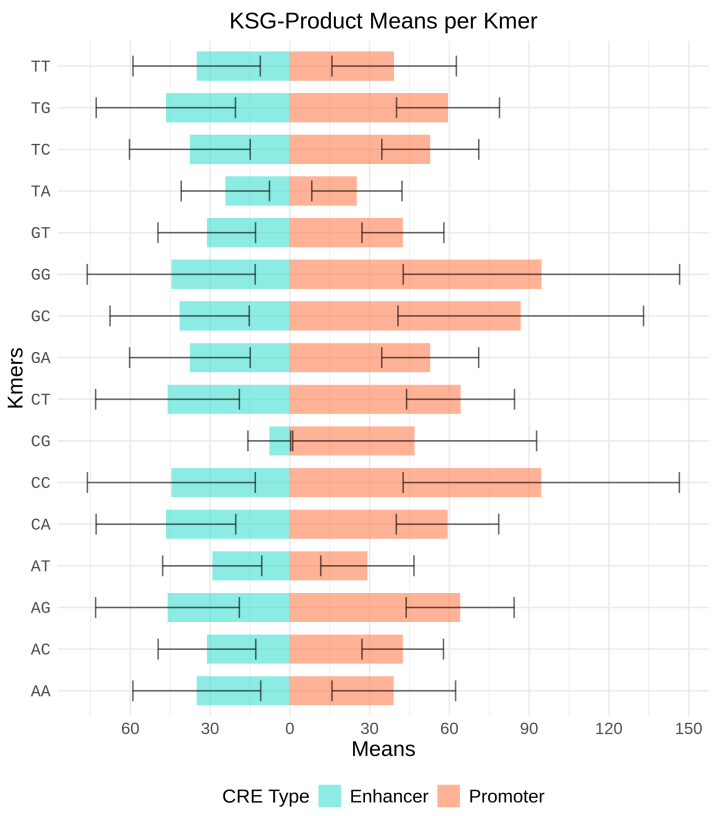

```
ibarplot_(CRE_summary[kshan_vals,],
          y_breaks = seq(0, 9, 1), legend_title = "Type",
          x_label = "Kmers' Shannon Entropy Means")
```



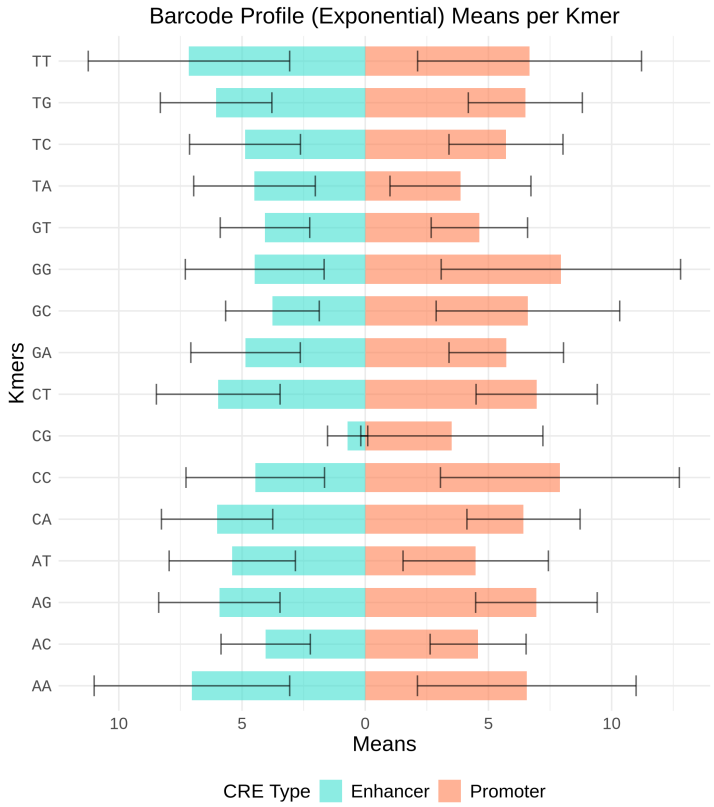

```
ibarplot_(CRE_summary[krdiv_vals,],
          legend_title = "Type",
          x_label = '"Relative" Kmer Diversity Means')
```



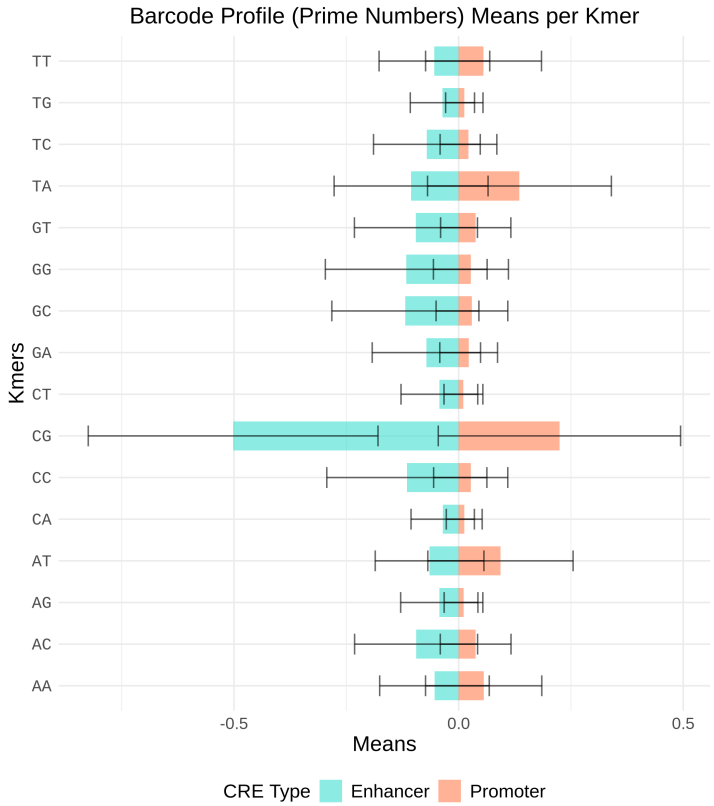

```
aCREs <- c("enha", "prom")
kmer_names <- combi_kmers(k=2)
# kmer_names2 <- combi_kmers(k=3)[1:48]

pyrplot_(CRE_summary[indxs$prod[1:16, aCREs],], kmer_names,
         x_label = "Kmers", y_breaks = seq(-150, 150, 30),
         title = "KSG-Product Means per Kmer")
```




```
pyrplot_(CRE_summary[indxs$bcds[1:16, aCREs],], kmer_names,
         x_label = "Kmers", y_breaks = seq(-30, 30, 5),
         title = "Barcode Profile (Exponential) Means per Kmer")
```




```
# pyrplot_(CRE_summary[indxs$bclp[17:64, aCREs],], kmer_names2,
pyrplot_(CRE_summary[indxs$bclp[1:16, aCREs],], kmer_names,
         x_label = "Kmers", #y_breaks = seq(-30, 30, 5),
         title = "Barcode Profile (Prime Numbers) Means per Kmer")
```

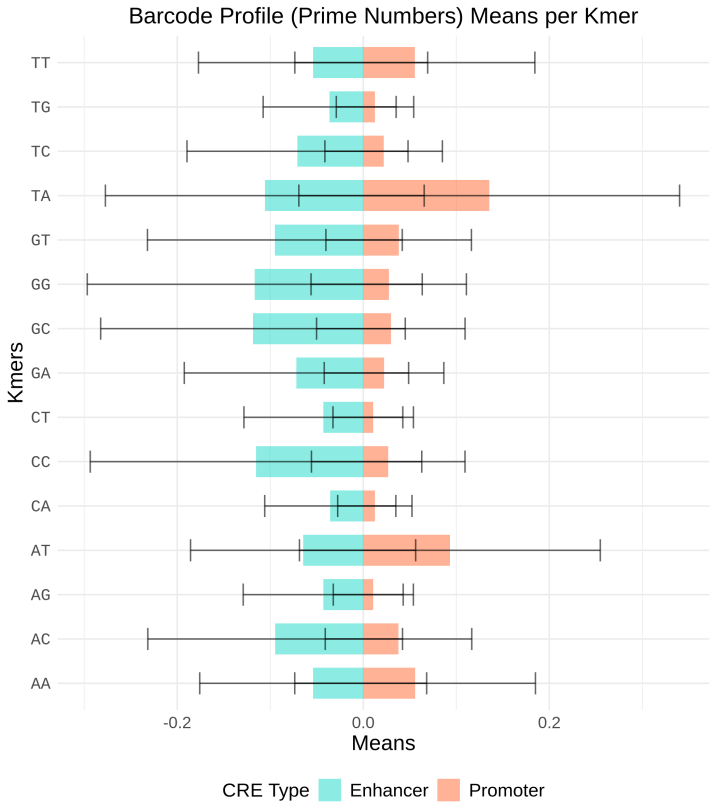



```
# summ_bclp <- CRE_summary2[indxs$bclp[17:64, aCREs],]
# summ_bclp <- CRE_summary2[indxs$bclp[1:16, aCREs],]
```

```
# summ_bclp$Means <- log(summ_bclp$Means)
# summ_bclp$StDevs <- log(summ_bclp$StDevs)

# pyrplot_(summ_bclp, kmer_names2,
# pyrplot_(summ_bclp, kmer_names,
#          x_label = "Kmers", #y_breaks = seq(-30, 30, 5),
#          title = "Barcode Profile (Prime Numbers) Means per Kmer")

# coffeetable(CRE_summary[indxs$prod[1:5,],])
# coffeetable(CRE_summary[indxs$bcds[1:5,],])
# coffeetable(CRE_summary[indxs$bclp[1:5,],])
```

With the last table we can observe a clear difference between mean values of CG kmers, however this partially disrupts the scale in the plot since none of the rest of kmers seem to distribute as widely. If we leave 'CG' kmers aside, the plot looks like this:

```
pyrplot_(CRE_summary[indxs$bclp[(1:16)[-7], aCREs],], kmer_names[-7],
         x_label = "Kmers", #y_breaks = seq(-30, 30, 5),
         title = "Barcode Profile (Prime Numbers) Means per Kmer")
```



3.4 Primary analysis

Firstly we read our CSV tables into our conviniently named dataframes.

```
setwd(project_path)

CRE_data <- list(
Enhancers = read.csv("datasets/GB-Test/gb_enhancers_training.csv",
                     check.names = FALSE),
Promoters = read.csv("datasets/GB-Test/gb_promoters_training.csv",
                     check.names = FALSE),
OCRs = read.csv("datasets/GB-Test/gb_OCRs_training.csv",
                check.names = FALSE))
```