Motif import, export, and manipulation

Benjamin Jean-Marie Tremblay*

14 March 2020

Abstract

The universalmotif package offers a number of functions to handle motifs. These are introduced and explored here, including those relating to: import, export, motif modification, creation, visualization, and other miscellaneous utilities.

Contents

*b2tremblay@uwaterloo.ca

1 Introduction

This vignette will introduce the `universalmotif` class and its structure, the import and export of motifs in R, basic motif manipulation, creation, and visualization. For an introduction to sequence motifs, see the introductory vignette. For sequence-related utilities, see the sequences vignette. For motif comparisons and P-values, see the motif comparisons and P-values vignette.

2 The universalmotif class and conversion utilities

2.1 The universalmotif class

The `universalmotif` package stores motifs using the `universalmotif` class. The most basic `universalmotif` object exposes the `name`, `alphabet`, `type`, `type`, `strand`, `icscore`, `consensus`, and `motif` slots; furthermore, the `pseudocount` and `bkg` slots are also stored but not shown. `universalmotif` class motifs can be PCM, PPM, PWM, or ICM type.

```
library(universalmotif)
data(examplemotif)
examplemotif
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   11.54
#>         Consensus:   TATAWAW
#>
#>   T A T A   W A   W
#> A 0 1 0 1 0.5 1 0.5
#> C 0 0 0 0 0.0 0 0.0
#> G 0 0 0 0 0.0 0 0.0
#> T 1 0 1 0 0.5 0 0.5
```

A brief description of all the available slots:

- `name`: motif name
- `altname`: (optional) alternative motif name
- `family`: (optional) a word representing the transcription factor or matrix family
- `organism`: (optional) organism of origin
- `motif`: the actual motif matrix
- `alphabet`: motif alphabet
- `type`: motif 'type', one of PCM, PPM, PWM, ICM; see the introductory vignette
- `icscore`: (generated automatically) Sum of information content for the motif
- `nsites`: (optional) number of sites the motif was created from
- `pseudocount`: this value to added to the motif matrix during certain type conversions; this is necessary to avoid `-Inf` values from appearing in PWM type motifs
- `bkg`: a named vector of probabilities which represent the background letter frequencies
- `bkgsites`: (optional) total number of background sequences from motif creation

- `consensus`: (generated automatically) for DNA/RNA/AA motifs, the motif consensus
- `strand`: strand motif can be found on
- `pval`: (optional) P-value from *de novo* motif search
- `qval`: (optional) Q-value from *de novo* motif search
- `eval`: (optional) E-value from *de novo* motif search
- `multifreq`: (optional) higher-order motif representations.
- `extrainfo`: (optional) any extra motif information that cannot fit in the existing slots

The other slots will be shown as they are filled.

```
library(universalmotif)
data(examplemotif)

## The various slots can be accessed individually using `[`

examplemotif["consensus"]
#> [1] "TATAWAW"

## To change a slot, use `[<-`

examplemotif["family"] <- "My motif family"
examplemotif
#>
#>        Motif name:   motif
#>            Family:   My motif family
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   11.54
#>         Consensus:   TATAWAW
#>
#>   T A T A   W A   W
#> A 0 1 0 1 0.5 1 0.5
#> C 0 0 0 0 0.0 0 0.0
#> G 0 0 0 0 0.0 0 0.0
#> T 1 0 1 0 0.5 0 0.5
```

Though the slots can easily be changed manually with `[<-`, a number of safeguards have been put in place for some of the slots which will prevent incorrect values from being introduced.

```
library(universalmotif)
data(examplemotif)

## The consensus slot is dependent on the motif matrix

examplemotif["consensus"]
#> [1] "TATAWAW"

## Changing this would mean it no longer matches the motif

examplemotif["consensus"] <- "GGGAGAG"
#> Error in .local(x, i, ..., value): this slot is unmodifiable with [<-

## Another example of trying to change a protected slot:

examplemotif["strand"] <- "x"
```

```
#> Error in validObject_universalmotif(x):
#> * strand must be one of +, -, +-
```

Below the exposed metadata slots, the actual 'motif' matrix is shown. Each position is its own column: row names showing the alphabet letters, and the column names showing the consensus letter at each position.

2.2 Converting to and from another package's class

The `universalmotif` package aims to unify most of the motif-related Bioconductor packages by providing the `convert_motifs()` function. This allows for easy transition between supported packages (see `?convert_motifs` for a complete list of supported packages).

The `convert_motifs` function is embedded in most of the `universalmotif` functions, meaning that compatible motif classes from other packages can be used without needed to manually convert them first. However keep in mind some conversions are terminal. Furthermore, internally, all motifs regardless of class are handled as `universalmotif` objects, even if the returning class is not. This will result in at times slightly different objects (though usually no information should be lost).

```
library(universalmotif)
library(MotifDb)
data(examplemotif)
data(MA0003.2)

## convert from a `universalmotif` motif to another class

convert_motifs(examplemotif, "TFBSTools-PWMatrix")
#> An object of class PWMatrix
#> ID:
#> Name: motif
#> Matrix Class: Unknown
#> strand: *
#> Pseudocounts: 1
#> Tags:
#> list()
#> Background:
#>    A    C    G    T
#> 0.25 0.25 0.25 0.25
#> Matrix:
#>           T         A         T         A          W         A          W
#> A -6.658211  1.989247 -6.658211  1.989247  0.9928402  1.989247  0.9928402
#> C -6.658211 -6.658211 -6.658211 -6.658211 -6.6582115 -6.658211 -6.6582115
#> G -6.658211 -6.658211 -6.658211 -6.658211 -6.6582115 -6.658211 -6.6582115
#> T  1.989247 -6.658211  1.989247 -6.658211  0.9928402 -6.658211  0.9928402

## convert to universalmotif

convert_motifs(MA0003.2)
#>
#>        Motif name:   TFAP2A
#>    Alternate name:   MA0003.2
#>            Family:   Helix-Loop-Helix
#>          Organism:   9606
#>          Alphabet:   DNA
#>              Type:   PCM
#>           Strands:   +
```

```
#>          Total IC:   12.9
#>         Consensus:   NNNNGCCYSAGGSCA
#>      Target sites:   5098
#>        Extra info:   [centrality_logp] -4343
#>                      [family] Helix-Loop-Helix
#>                      [medline] 10497269
#>                      ...
#>
#>      N    N    N    N    G    C    C    Y    S    A    G    G    S    C    A
#> A 1387 2141  727 1517   56    0    0   62  346 3738  460    0  116  451 3146
#> C 1630 1060 1506  519 1199 5098 4762 1736 2729  236    0    0 1443 3672  690
#> G  851  792  884  985 3712    0    0   85 1715  920 4638 5098 3455  465  168
#> T 1230 1105 1981 2077  131    0  336 3215  308  204    0    0   84  510 1094

## convert between two packages

convert_motifs(MotifDb[1], "TFBSTools-ICMatrix")
#> [[1]]
#> An object of class ICMatrix
#> ID: badis.ABF2
#> Name: ABF2
#> Matrix Class: Unknown
#> strand: *
#> Pseudocounts: 1
#> Schneider correction: FALSE
#> Tags:
#> $dataSource
#> [1] "ScerTF"
#>
#> Background:
#>    A    C    G    T
#> 0.25 0.25 0.25 0.25
#> Matrix:
#>            T          C          T          A          G          A
#> A 0.08997357 0.02119039 0.02119039 1.64861232 0.02119039 1.43716039
#> C 0.08997357 1.64861232 0.02119039 0.02119039 0.02119039 0.03430887
#> G 0.02188546 0.02119039 0.02119039 0.02119039 1.64861232 0.03430887
#> T 0.78058151 0.02119039 1.64861232 0.02119039 0.02119039 0.03430887
```

3 Importing and exporting motifs

3.1 Importing

The `universalmotif` package offers a number of `read_*()` functions to allow for easy import of various motif formats. These include:

- `read_cisbp()`: CIS-BP (Weirauch et al. 2014)
- `read_homer()`: `HOMER` (Heinz et al. 2010)
- `read_jaspar()`: JASPAR (Khan et al. 2018)
- `read_matrix()`: generic reader for simply formatted motifs
- `read_meme()`: `MEME` (Bailey et al. 2009)
- `read_motifs()`: native `universalmotif` format
- `read_transfac()`: TRANSFAC (Wingender et al. 1996)

- `read_uniprobe()`: UniPROBE (Hume et al. 2015)

These functions should work natively with these formats, but if you are generating your own motifs in one of these formats than it must adhere quite strictly to the format. An example of each of these is included in this package (see `system.file("extdata", package="universalmotif")`).

3.2 Exporting

Compatible motif classes can be written to disk using:

- `write_homer()`
- `write_jaspar()`
- `write_matrix()`
- `write_meme()`
- `write_motifs()`
- `write_transfac()`

The `write_matrix()` function, similar to its `read_matrix()` counterpart, can write motifs as simple matrices with an optional header. Additionally, please keep in mind format limitations. For example, multiple `MEME` motifs written to a single file will all share the same alphabet, with identical background letter frequencies.

4 Modifying motifs and related functions

4.1 Converting motif type

Any `universalmotif` object can transition between PCM, PPM, PWM, and ICM types seamlessly using the `convert_type()` function. The only exception to this is if the ICM calculation is performed with sample correction, or as relative entropy. If this occurs, then back conversion to another type will be inaccurate (and `convert_type()` would not warn you, since it can't know this has taken place).

```
library(universalmotif)
data(examplemotif)

## This motif is currently a PPM:

examplemotif["type"]
#> [1] "PPM"
```

When converting to PCM, the `nsites` slot is needed to tell it how many sequences it originated from. If empty, 100 is used.

```
convert_type(examplemotif, "PCM")
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PCM
#>           Strands:   +-
#>          Total IC:   11.54
#>         Consensus:   TATAWAW
#>
#>     T   A   T   A  W   A  W
#> A   0 100   0 100 50 100 50
#> C   0   0   0   0  0   0  0
#> G   0   0   0   0  0   0  0
#> T 100   0 100   0 50   0 50
```

For converting to PWM, the `pseudocount` slot is used to determine if any correction should be applied:

```
examplemotif["pseudocount"]
#> [1] 1
convert_type(examplemotif, "PWM")
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PWM
#>           Strands:   +-
#>          Total IC:   11.54
#>         Consensus:   TATAWAW
#>
#>       T     A     T     A     W     A     W
#> A -6.66  1.99 -6.66  1.99  0.99  1.99  0.99
#> C -6.66 -6.66 -6.66 -6.66 -6.66 -6.66 -6.66
#> G -6.66 -6.66 -6.66 -6.66 -6.66 -6.66 -6.66
#> T  1.99 -6.66  1.99 -6.66  0.99 -6.66  0.99
```

You can either change the `pseudocount` slot manually beforehand, or pass one to `convert_type()`.

```
convert_type(examplemotif, "PWM", pseudocount = 1)
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PWM
#>           Strands:   +-
#>          Total IC:   11.54
#>         Consensus:   TATAWAW
#>
#>       T     A     T     A     W     A     W
#> A -6.66  1.99 -6.66  1.99  0.99  1.99  0.99
#> C -6.66 -6.66 -6.66 -6.66 -6.66 -6.66 -6.66
#> G -6.66 -6.66 -6.66 -6.66 -6.66 -6.66 -6.66
#> T  1.99 -6.66  1.99 -6.66  0.99 -6.66  0.99
```

There are a couple of additional options for ICM conversion: `nsize_correction` and `relative_entropy`. The former uses the `TFBSTools:::schneider_correction()` function (and thus requires that the `TFBSTools` package be installed) for sample size correction. The latter uses the `bkg` slot to calculate information content.

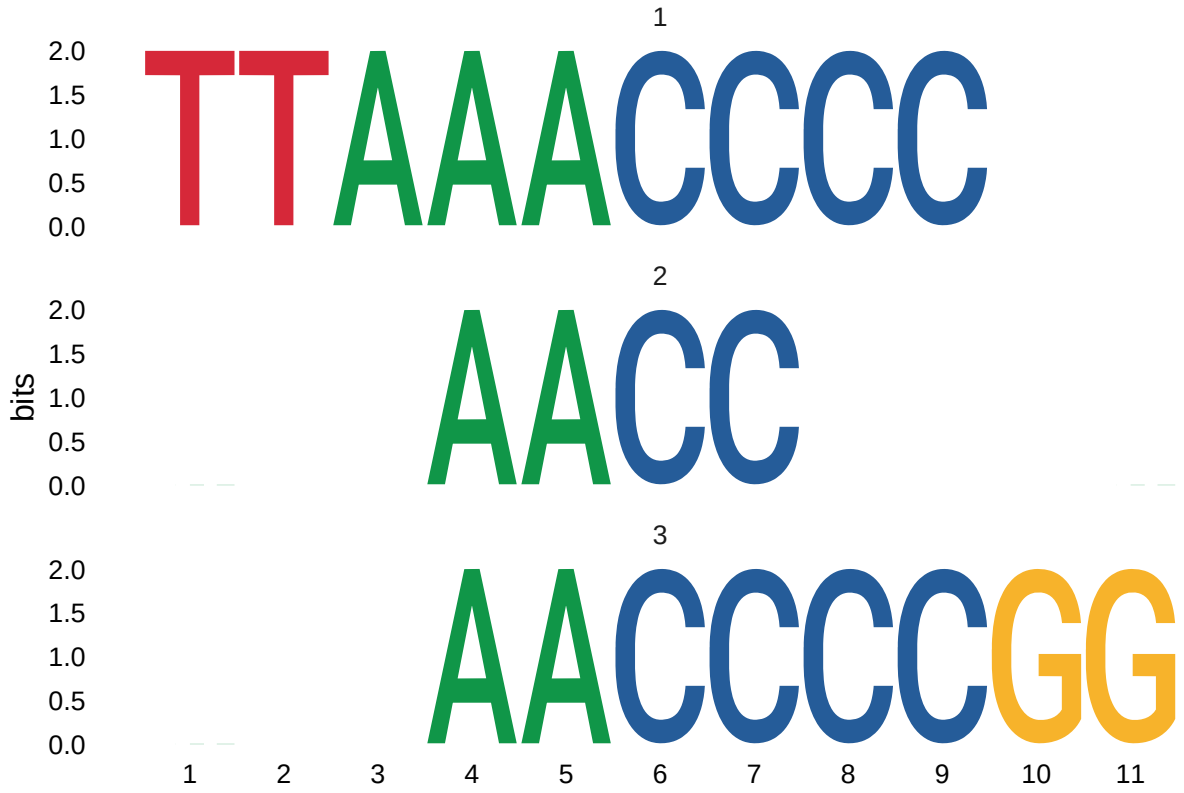
```
examplemotif["nsites"] <- 10
convert_type(examplemotif, "ICM", nsize_correction = FALSE)
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   ICM
#>           Strands:   +-
#>          Total IC:   11.54
#>         Consensus:   TATAWAW
#>      Target sites:   10
#>
#>   T A T A   W A   W
#> A 0 2 0 2 0.5 2 0.5
#> C 0 0 0 0 0.0 0 0.0
#> G 0 0 0 0 0.0 0 0.0
#> T 2 0 2 0 0.5 0 0.5
```

```
convert_type(examplemotif, "ICM", nsize_correction = TRUE)
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   ICM
#>           Strands:   +-
#>          Total IC:   11.54
#>         Consensus:   TATAWAW
#>      Target sites:   10
#>
#>      T    A    T    A    W    A    W
#> A 0.00 1.75 0.00 1.75 0.38 1.75 0.38
#> C 0.00 0.00 0.00 0.00 0.00 0.00 0.00
#> G 0.00 0.00 0.00 0.00 0.00 0.00 0.00
#> T 1.75 0.00 1.75 0.00 0.38 0.00 0.38

examplemotif["bkg"] <- c(A = 0.4, C = 0.1, G = 0.1, T = 0.4)
convert_type(examplemotif, "ICM", relative_entropy = TRUE)
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   ICM
#>           Strands:   +-
#>          Total IC:   11.54
#>         Consensus:   TATAWAW
#>      Target sites:   10
#>
#>      T    A    T    A    W    A    W
#> A 0.00 1.32 0.00 1.32 0.16 1.32 0.16
#> C 0.00 0.00 0.00 0.00 0.00 0.00 0.00
#> G 0.00 0.00 0.00 0.00 0.00 0.00 0.00
#> T 1.32 0.00 1.32 0.00 0.16 0.00 0.16
```

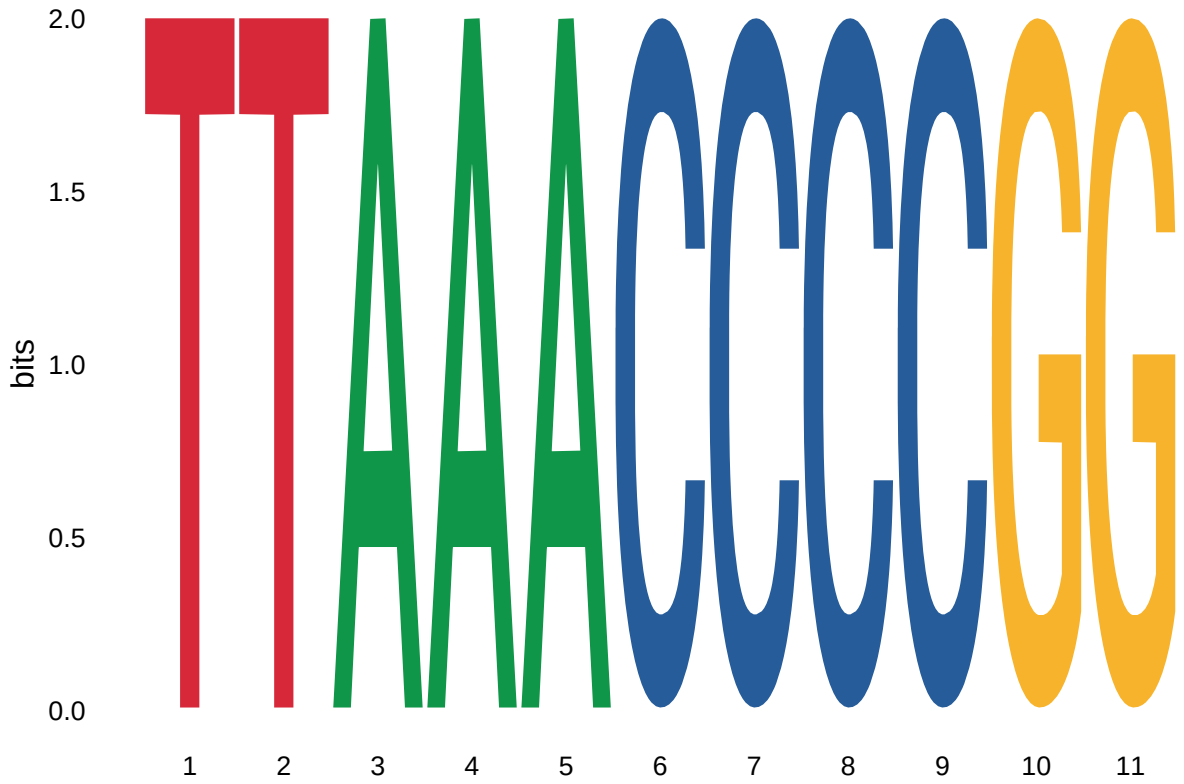
4.2 Merging motifs

The `universalmotif` package includes the `merge_motifs()` function to combine motifs. Motifs are first aligned, and the best match found before the motif matrices are averaged. The implementation for this is identical to that used by `compare_motifs()` (see the motif comparisons vignette for more information).

```
library(universalmotif)

m1 <- create_motif("TTAAACCCC", name = "1")
m2 <- create_motif("AACC", name = "2")
m3 <- create_motif("AACCCCGG", name = "3")

view_motifs(c(m1, m2, m3))
```




```
view_motifs(merge_motifs(c(m1, m2, m3), method = "PCC"))
```



4.3 Motif reverse complement

Get the reverse complement of a motif.

```
library(universalmotif)
data(examplemotif)

## Quickly switch to the reverse complement of a motif

## Original:

examplemotif
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   11.54
#>         Consensus:   TATAWAW
#>
#>   T A T A   W A   W
#> A 0 1 0 1 0.5 1 0.5
#> C 0 0 0 0 0.0 0 0.0
#> G 0 0 0 0 0.0 0 0.0
#> T 1 0 1 0 0.5 0 0.5

## Reverse complement:

motif_rc(examplemotif)
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   11.54
#>         Consensus:   WTWTATA
#>
#>     W T   W T A T A
#> A 0.5 0 0.5 0 1 0 1
#> C 0.0 0 0.0 0 0 0 0
#> G 0.0 0 0.0 0 0 0 0
#> T 0.5 1 0.5 1 0 1 0
```

4.4 Switching between DNA and RNA alphabets

Since not all motif formats or programs support RNA alphabets by default, the `switch_alph()` function can quickly go between DNA and RNA motifs.

```
library(universalmotif)
data(examplemotif)

## DNA --> RNA

switch_alph(examplemotif)
#>
```

```
#>        Motif name:   motif
#>          Alphabet:   RNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   11.54
#>         Consensus:   UAUAWAW
#>
#>   U A U A   W A   W
#> A 0 1 0 1 0.5 1 0.5
#> C 0 0 0 0 0.0 0 0.0
#> G 0 0 0 0 0.0 0 0.0
#> U 1 0 1 0 0.5 0 0.5

## RNA --> DNA

motif <- create_motif(alphabet = "RNA")
motif
#>
#>        Motif name:   motif
#>          Alphabet:   RNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   12.21
#>         Consensus:   UGKGMYUCMA
#>
#>      U    G    K    G    M    Y    U    C    M    A
#> A 0.01 0.04 0.00 0.01 0.30 0.00 0.00 0.04 0.60 0.93
#> C 0.13 0.00 0.01 0.03 0.45 0.52 0.02 0.95 0.25 0.00
#> G 0.07 0.95 0.25 0.85 0.24 0.00 0.02 0.00 0.01 0.07
#> U 0.79 0.01 0.74 0.11 0.00 0.47 0.96 0.00 0.14 0.00

switch_alph(motif)
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   12.21
#>         Consensus:   TGKGMYTCMA
#>
#>      T    G    K    G    M    Y    T    C    M    A
#> A 0.01 0.04 0.00 0.01 0.30 0.00 0.00 0.04 0.60 0.93
#> C 0.13 0.00 0.01 0.03 0.45 0.52 0.02 0.95 0.25 0.00
#> G 0.07 0.95 0.25 0.85 0.24 0.00 0.02 0.00 0.01 0.07
#> T 0.79 0.01 0.74 0.11 0.00 0.47 0.96 0.00 0.14 0.00
```

4.5 Motif trimming

Get rid of low information content edges on motifs, such as `NNCGGGCNN` to `CGGGC`. The 'amount' of trimming can also be controlled by setting a minimum required information content.

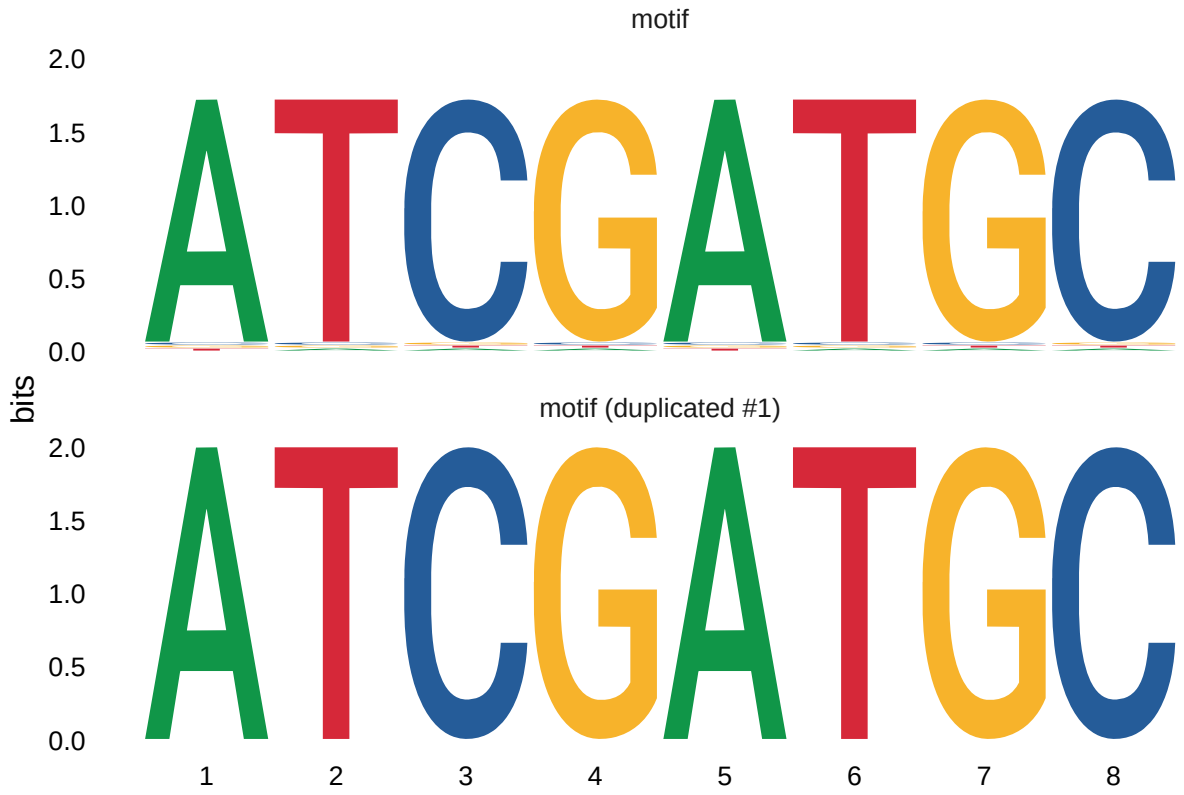
```
library(universalmotif)

motif <- create_motif("NNGCSGCGGNN")
```

```
motif
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   13
#>         Consensus:   NNGCSGCGGNN
#>
#>      N    N G C   S G C G G    N    N
#> A 0.25 0.25 0 0 0.0 0 0 0 0 0.25 0.25
#> C 0.25 0.25 0 1 0.5 0 1 0 0 0.25 0.25
#> G 0.25 0.25 1 0 0.5 1 0 1 1 0.25 0.25
#> T 0.25 0.25 0 0 0.0 0 0 0 0 0.25 0.25

trim_motifs(motif)
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   13
#>         Consensus:   GCSGCGG
#>      Target sites:   100
#>
#>   G C   S G C G G
#> A 0 0 0.0 0 0 0 0
#> C 0 1 0.5 0 1 0 0
#> G 1 0 0.5 1 0 1 1
#> T 0 0 0.0 0 0 0 0
```

4.6 Rounding motifs

Round off near-zero probabilities.

```
motif1 <- create_motif("ATCGATGC", pseudocount = 5, type = "PPM", nsites = 100)
motif2 <- round_motif(motif1)
view_motifs(c(motif1, motif2), dedup.names =  TRUE)
```



5 Motif creation

Though `universalmotif` class motifs can be created using the `new` constructor, the `universalmotif` package provides the `create_motif()` function which aims to provide a simpler interface to motif creation. The `universalmotif` class was initially designed to work natively with DNA, RNA, and amino acid motifs. Currently though, it can handle any custom alphabet just as easily. The only downsides to custom alphabets is the lack of support for certain slots such as the `consensus` and `strand` slots.

The `create_motif()` function will be introduced here only briefly; see `?create_motif` for details.

5.1 From a PCM/PPM/PWM/ICM matrix

Should you wish to make use of the `universalmotif` functions starting from a unsupported motif class, you can instead create `universalmotif` class motifs using the `create_motif()` function.

```
motif.matrix <- matrix(c(0.7, 0.1, 0.1, 0.1,
                         0.7, 0.1, 0.1, 0.1,
                         0.1, 0.7, 0.1, 0.1,
                         0.1, 0.7, 0.1, 0.1,
                         0.1, 0.1, 0.7, 0.1,
                         0.1, 0.1, 0.7, 0.1,
                         0.1, 0.1, 0.1, 0.7,
                         0.1, 0.1, 0.1, 0.7), nrow = 4)

motif <- create_motif(motif.matrix, alphabet = "RNA", name = "My motif",
                      pseudocount = 1, nsites = 20, strand = "+")

## The 'type', 'icscore' and 'consensus' slots will be filled for you
```

```
motif
#>
#>        Motif name:   My motif
#>          Alphabet:   RNA
#>              Type:   PPM
#>           Strands:   +
#>          Total IC:   4.68
#>         Consensus:   AACCGGUU
#>      Target sites:   20
#>
#>     A   A   C   C   G   G   U   U
#> A 0.7 0.7 0.1 0.1 0.1 0.1 0.1 0.1
#> C 0.1 0.1 0.7 0.7 0.1 0.1 0.1 0.1
#> G 0.1 0.1 0.1 0.1 0.7 0.7 0.1 0.1
#> U 0.1 0.1 0.1 0.1 0.1 0.1 0.7 0.7
```

As a short aside: if you have a motif formatted simply as a matrix, you can still use it with the `universalmotif` package functions natively without creating a motif with `create_motif()`, as `convert_motifs()` also has the ability to handle motifs formatted as matrices. However it is much safer to first specify the motif beforehand with `create_motif()`.

5.2 From sequences or character strings

If all you have is a particular consensus sequence in mind, you can easily create a full motif using `create_motif()`. This can be convenient if you'd like to create a quick motif to use with an external program such as from the `MEME` suite or `HOMER`.

```
motif <- create_motif("CCNSNGG", nsites = 50, pseudocount = 1)

## Now to disk:
## write_meme(motif, "meme_motif.txt")

motif
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   8.39
#>         Consensus:   CCNSNGG
#>      Target sites:   50
#>
#>      C    C    N   S    N    G    G
#> A 0.00 0.00 0.22 0.0 0.22 0.00 0.00
#> C 0.99 0.99 0.26 0.5 0.26 0.00 0.00
#> G 0.00 0.00 0.26 0.5 0.26 0.99 0.99
#> T 0.00 0.00 0.26 0.0 0.26 0.00 0.00
```

5.3 Generating random motifs

If you wish, it's easy to create random motifs. The values within the motif are generated using `rgamma()` to avoid creating low information content motifs. If background probabilities are not provided, then they are generated with `rpois()`.

```
create_motif()
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   12.11
#>         Consensus:   CGTMRGWTCT
#>
#>      C    G    T    M    R    G    W    T    C    T
#> A 0.13 0.01 0.01 0.65 0.44 0.01 0.73 0.06 0.02 0.08
#> C 0.85 0.24 0.23 0.35 0.00 0.01 0.00 0.00 0.94 0.03
#> G 0.01 0.75 0.04 0.00 0.56 0.93 0.00 0.05 0.01 0.05
#> T 0.01 0.00 0.73 0.00 0.01 0.05 0.27 0.89 0.04 0.85
```

You can change the probabilities used to generate the values within the motif matrix:

```
create_motif(bkg = c(A = 0.2, C = 0.4, G = 0.2, T = 0.2))
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   11.43
#>         Consensus:   KTCYCTYCRT
#>
#>      K    T    C    Y    C    T    Y C    R    T
#> A 0.01 0.00 0.22 0.00 0.00 0.19 0.00 0 0.70 0.04
#> C 0.17 0.01 0.70 0.52 0.99 0.01 0.48 1 0.00 0.03
#> G 0.29 0.24 0.01 0.00 0.01 0.00 0.00 0 0.27 0.21
#> T 0.53 0.76 0.08 0.48 0.00 0.80 0.52 0 0.02 0.72
```

With a custom alphabet:

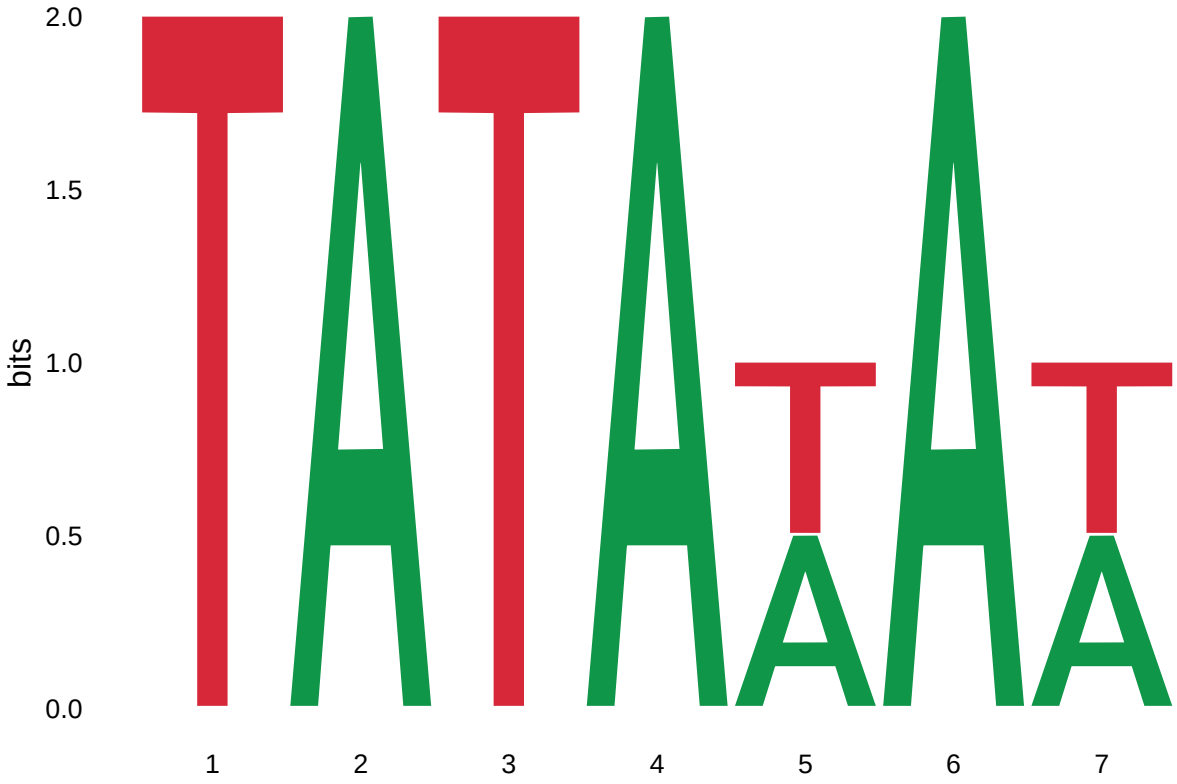
```
create_motif(alphabet = "QWERTY")
#>
#>        Motif name:   motif
#>          Alphabet:   EQRTWY
#>              Type:   PPM
#>          Total IC:   15.08
#>
#>   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
#> E 0.00 0.00 0.69 0.00 0.00 0.96 0.01 0.40 0.22  0.03
#> Q 0.00 0.04 0.00 0.00 0.00 0.00 0.02 0.14 0.53  0.00
#> R 0.01 0.00 0.00 0.00 0.63 0.01 0.81 0.00 0.00  0.18
#> T 0.00 0.33 0.18 0.03 0.27 0.00 0.01 0.00 0.09  0.23
#> W 0.99 0.22 0.00 0.02 0.00 0.00 0.00 0.07 0.00  0.55
#> Y 0.00 0.40 0.13 0.95 0.10 0.03 0.15 0.39 0.17  0.00
```

6 Motif visualization

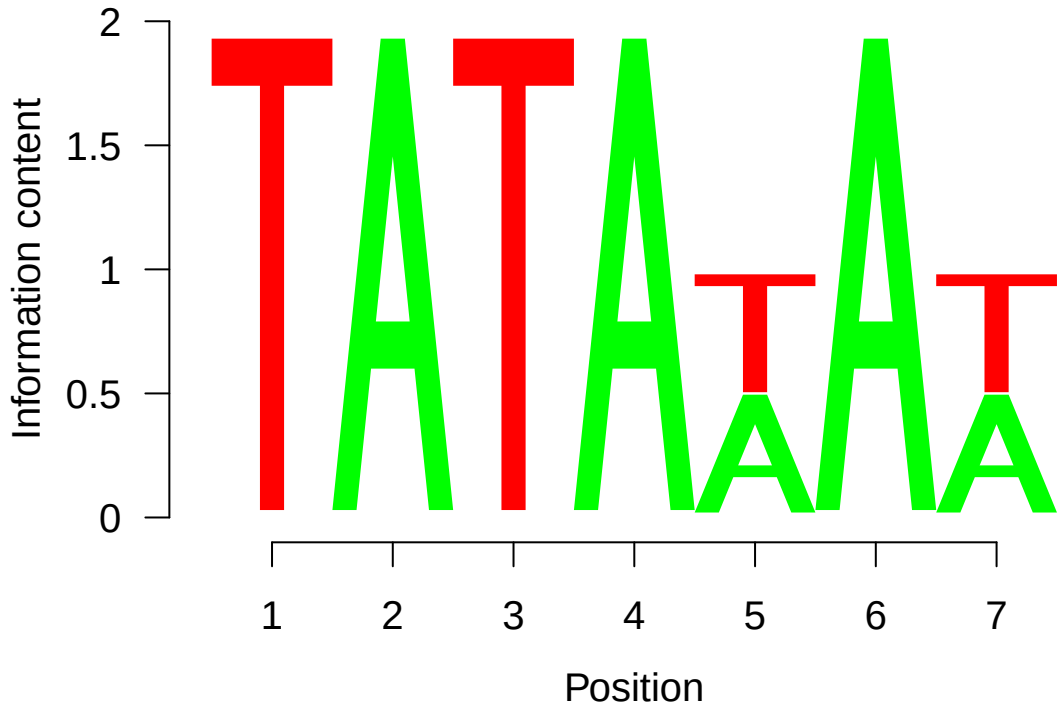
6.1 Motif logos

There are several packages which offer motif visualization capabilities, such as `seqLogo`, `Logolas`, `motifStack`, and `ggseqlogo`. The `universalmotif` package has chosen `ggseqlogo` as the default implementation, and used to drive the `universalmotif` package function `view_motifs()`. Here I will briefly show how to use these to visualize `universalmotif` class motifs.

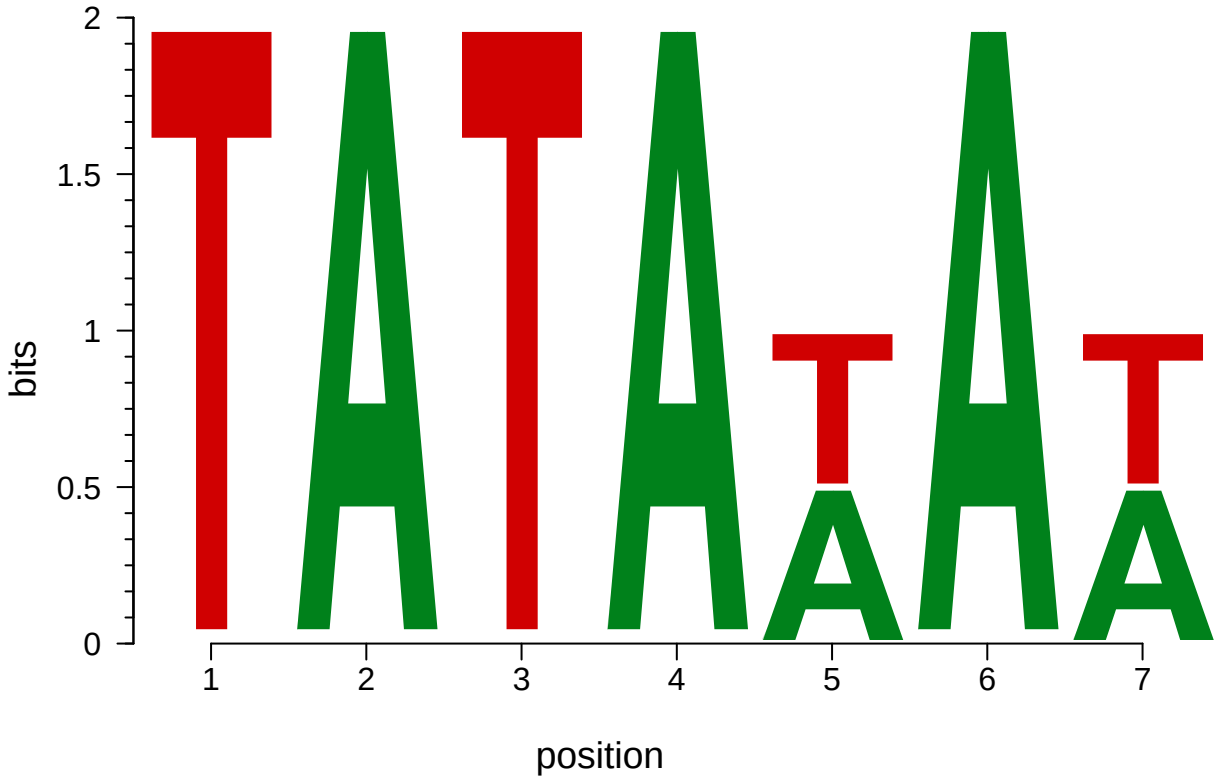
```
library(universalmotif)
data(examplemotif)

## With the native `view_motifs` function:
view_motifs(examplemotif)
```




```
## For all the following examples, simply passing the functions a PPM is
## sufficient
motif <- convert_type(examplemotif, "PPM")
## Only need the matrix itself
motif <- motif["motif"]

## seqLogo:
seqLogo::seqLogo(motif)
#> Warning in if (class(pwm) == "pwm") {: the condition has length > 1 and only the
#> first element will be used
#> Warning in if (class(pwm) == "data.frame") {: the condition has length > 1 and
#> only the first element will be used
#> Warning in if (class(pwm) != "matrix") {: the condition has length > 1 and only
#> the first element will be used
```

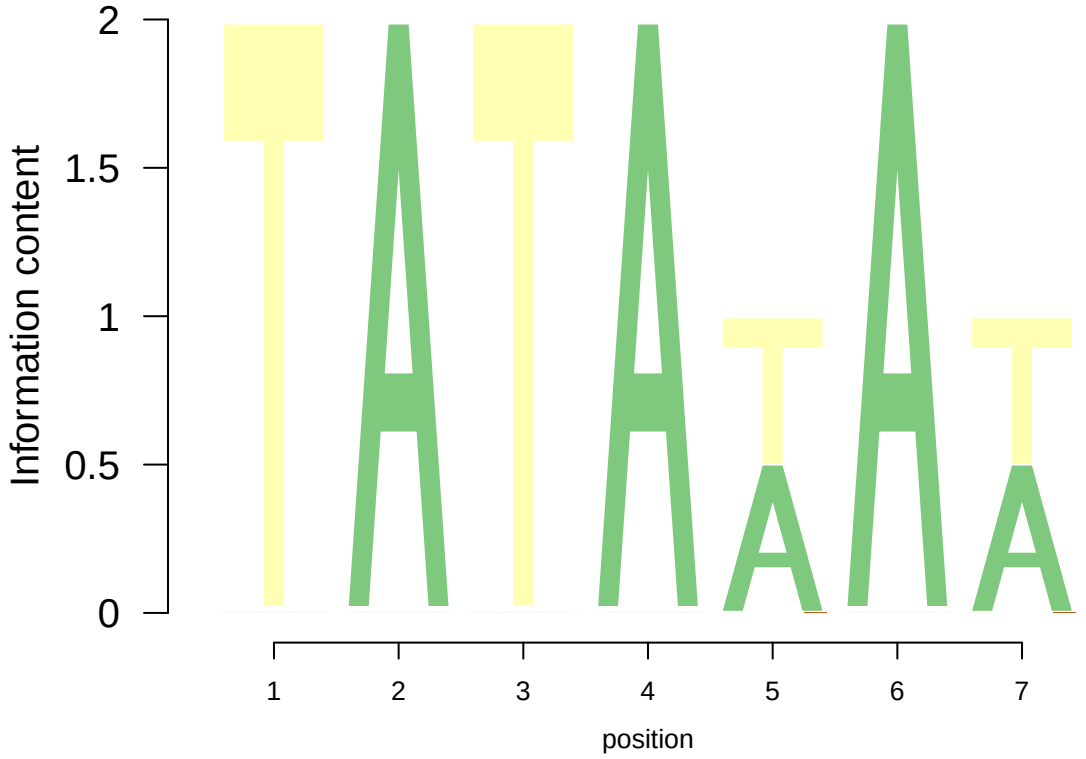



```
## motifStack:
motifStack::plotMotifLogo(motif)
```

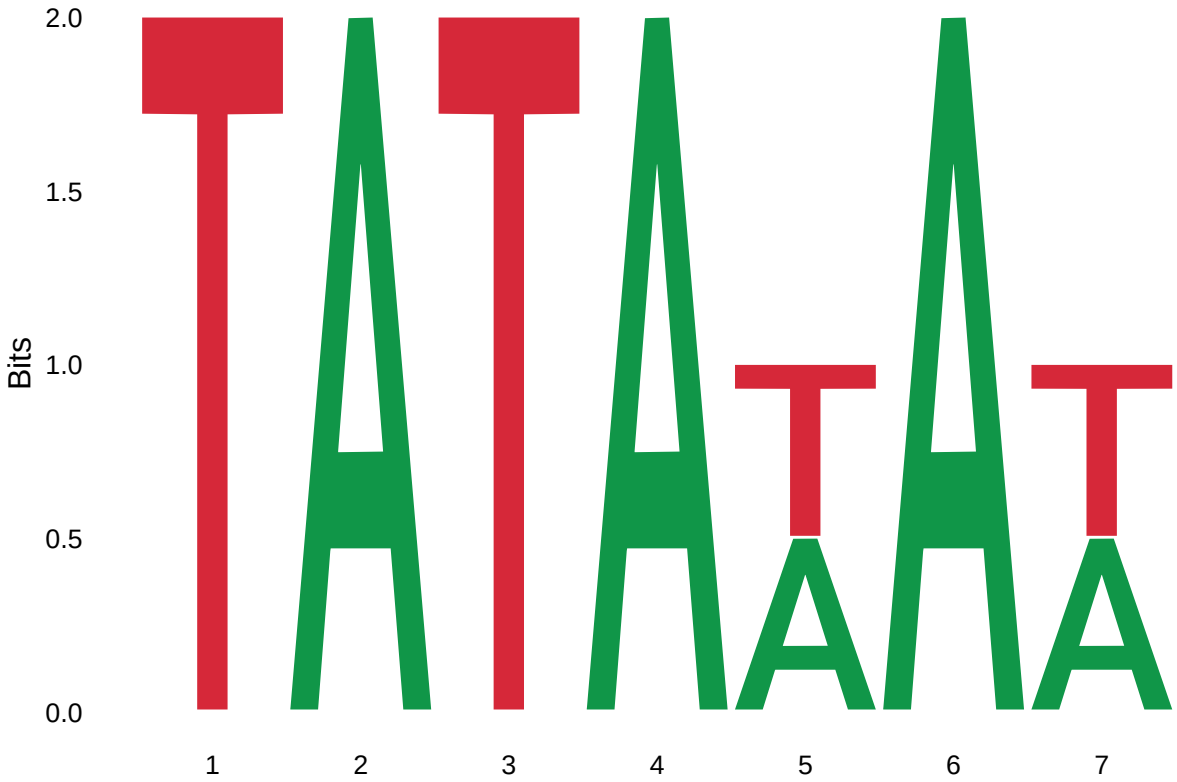



```
## Logolas:
colnames(motif) <- seq_len(ncol(motif))
Logolas::logomaker(motif, type = "Logo")
```

```
#> color_type not provided, so switching to per_row option for
#>                color_type
#> frame width not provided, taken to be 1
#> Warning in if (class(table) == "data.frame") {: the condition has length > 1 and
#> only the first element will be used
#> Warning in if (class(table) != "matrix") {: the condition has length > 1 and
#> only the first element will be used
#> using a background with equal probability for all symbols
```




```
## ggseqlogo:
ggseqlogo::ggseqlogo(motif)
```

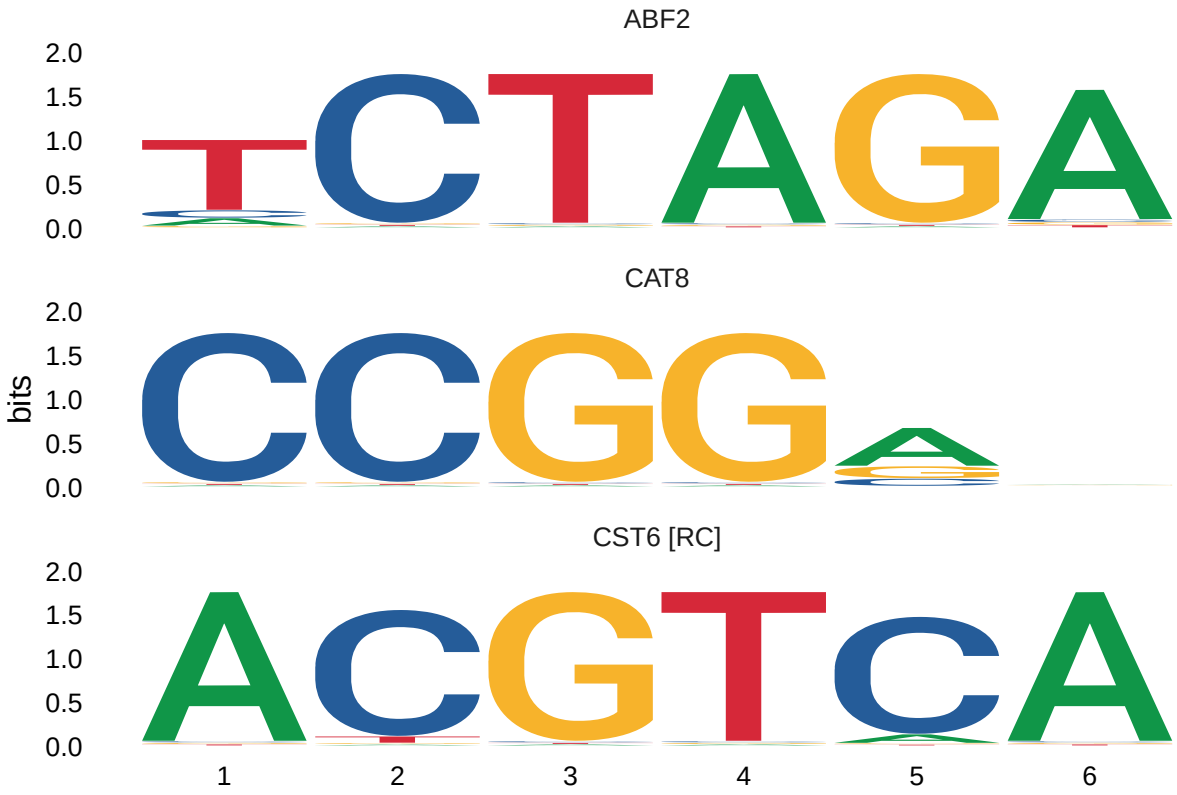


The `Logolas` and `ggseqlogo` offer many additional options for logo customization, including custom alphabets as well as manually determining the heights of each letter, via the `grid` and `ggplot2` packages respectively.

6.2 Stacked motif logos

The `motifStack` package allows for a number of different motif stacking visualizations. The `universalmotif` package, while not capable of emulating these, still offers basic stacking via `view_motifs()`. The motifs are aligned using `compare_motifs()`.

```
library(universalmotif)
library(MotifDb)

motifs <- convert_motifs(MotifDb[1:3])
view_motifs(motifs)
```



7 Higher-order motifs

Though PCM, PPM, PWM, and ICM type motifs are still widely used today, a few 'next generation' motif formats have been proposed. These wish to add another layer of information to motifs: positional interdependence. To illustrate this, consider the following sequences:

Table 1: Example sequences.

#	Sequence
1	CAAAACC
2	CAAAACC
3	CAAAACC
4	CTTTTCC
5	CTTTTCC
6	CTTTTCC

This becomes the following PPM:

Table 2: Position Probability Matrix.

Position	1	2	3	4	5	6	7
A	0.0	0.5	0.5	0.5	0.5	0.0	0.0
C	1.0	0.0	0.0	0.0	0.0	1.0	1.0
G	0.0	0.0	0.0	0.0	0.0	0.0	0.0
T	0.0	0.5	0.5	0.5	0.5	0.0	0.0

Based on the PPM representation, all three of CAAAACC, CTTTTCC, and CTATACC are equally likely. Though looking at the starting sequences, should CTATACC really be considered so? For transcription factor binding sites, it is not always so. By incorporating this type of information into the motif, it can allow for increased accuracy in motif searching. A few implementations of this include: TFFM by Mathelier and Wasserman (2013), BaMM by Siebert and Soding (2016), and KSM by Guo et al. (2018).

The `universalmotif` package implements its own, rather simplified, version of this concept. Plainly, the standard PPM has been extended to include `k`-letter frequencies, with `k` being any number higher than 1. For example, the 2-letter version of the table 2 motif would be:

Table 3: 2-letter probability matrix.

Position	1	2	3	4	5	6
AA	0.0	0.5	0.5	0.5	0.0	0.0
AC	0.0	0.0	0.0	0.0	0.5	0.0
AG	0.0	0.0	0.0	0.0	0.0	0.0
AT	0.0	0.0	0.0	0.0	0.0	0.0
CA	0.5	0.0	0.0	0.0	0.0	0.0
CC	0.0	0.0	0.0	0.0	0.0	1.0
CG	0.0	0.0	0.0	0.0	0.0	0.0
CT	0.5	0.0	0.0	0.0	0.0	0.0
GA	0.0	0.0	0.0	0.0	0.0	0.0
GC	0.0	0.0	0.0	0.0	0.0	0.0
GG	0.0	0.0	0.0	0.0	0.0	0.0
GT	0.0	0.0	0.0	0.0	0.0	0.0
TA	0.0	0.0	0.0	0.0	0.0	0.0
TC	0.0	0.0	0.0	0.0	0.5	0.0
TG	0.0	0.0	0.0	0.0	0.0	0.0
TT	0.0	0.5	0.5	0.5	0.0	0.0

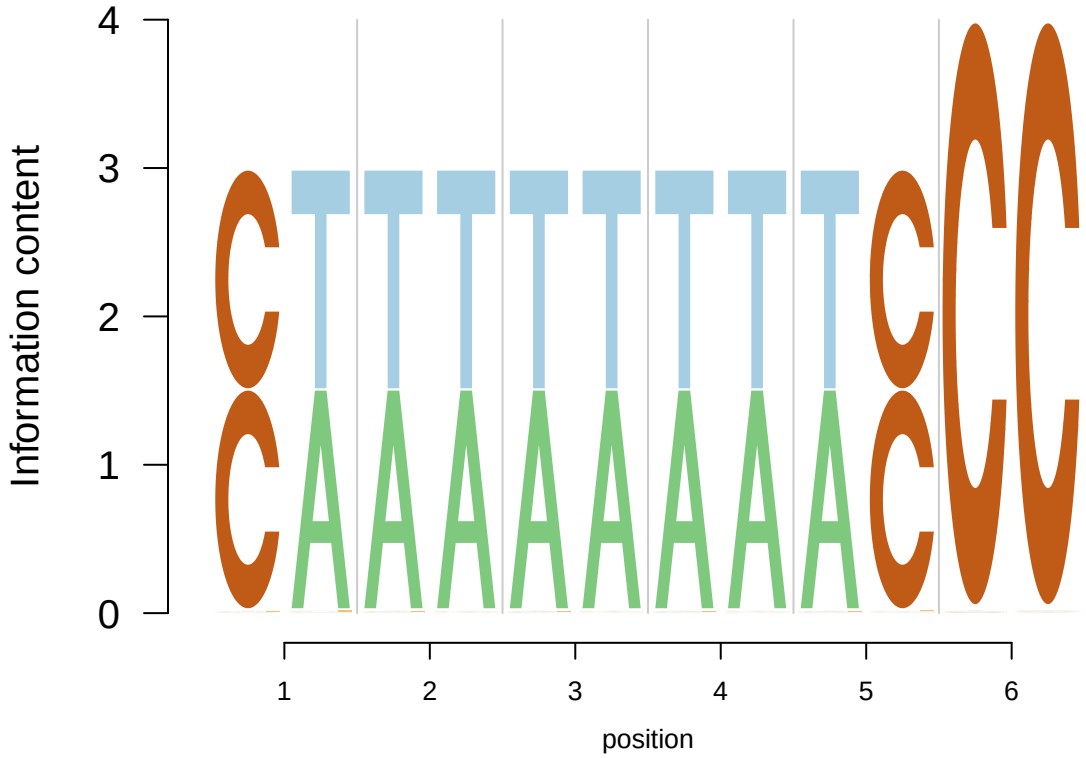
This format shows the probability of each letter combined with the probability of the letter in the next position. The seventh column has been dropped, since it is not needed: the information in the sixth column is sufficient, and there is no eighth position to draw 2-letter probabilities from. Now, the probability of getting CTATACC is no longer equal to CTTTTCC and CAAAACC. This information is kept in the `multifreq` slot of `universalmotif` class motifs. To add this information, use the `add_multifreq()` function.

```
library(universalmotif)

motif <- create_motif("CWWWWCC", nsites = 6)
sequences <- DNAStringSet(rep(c("CAAAACC", "CTTTTCC"), 3))
motif.k2 <- add_multifreq(motif, sequences, add.k = 2)

## Alternatively:
# motif.k2 <- create_motif(sequences, add.multifreq = 2)

motif.k2
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   10
#>         Consensus:   CWWWWCC
#>      Target sites:   6
```

```
#>    k-letter freqs:   2
#>
#>   C   W   W   W   W C C
#> A 0 0.5 0.5 0.5 0.5 0 0
#> C 1 0.0 0.0 0.0 0.0 1 1
#> G 0 0.0 0.0 0.0 0.0 0 0
#> T 0 0.5 0.5 0.5 0.5 0 0
```

Unfortunately `view_motifs()` cannot be used to visualize this higher order motif representation. However, this can be done via the `Logolas` package:

```
library(Logolas)
logomaker(motif.k2["multifreq"][["2"]], type = "Logo",
          color_type = "per_symbol")
#> frame width not provided, taken to be 1
#> Warning in if (class(table) == "data.frame") {: the condition has length > 1 and
#> only the first element will be used
#> Warning in if (class(table) != "matrix") {: the condition has length > 1 and
#> only the first element will be used
#> using a background with equal probability for all symbols
```



This information is most useful with functions such as `scan_sequences()` and `enrich_motifs()`. Though other tools in the `universalmotif` can work with `multifreq` motifs (such as `motif_pvalue()`, `compare_motifs()`), keep in mind they are not as well supported as regular motifs (getting P-values from `multifreq` motifs is exponentially slower, and P-values from using `compare_motifs()` for `multifreq` motifs are not available by default). See the sequences vignette for using `scan_sequences()` with the `multifreq` slot.

8 Miscellaneous motif utilities

A number of convenience functions are included for manipulating motifs.

8.1 DNA/RNA/AA consensus functions

For DNA, RNA and AA motifs, the `universalmotif` will automatically generate a `consensus` string slot. Furthermore, `create_motif()` can generate motifs from consensus strings. The internal functions for these have been made available:

- `consensus_to_ppm()`
- `consensus_to_ppmAA()`
- `get_consensus()`
- `get_consensusAA()`

```
library(universalmotif)

get_consensus(c(A = 0.7, C = 0.1, G = 0.1, T = 0.1))
#> [1] "A"

consensus_to_ppm("G")
#> [1] 0.001 0.001 0.997 0.001
```

8.2 Filter through lists of motifs

Filter a list of motifs, using the `universalmotif` slots with `filter_motifs()`.

```
library(universalmotif)
library(MotifDb)

## Let us extract all of the Arabidopsis and C. elegans motifs (note that
## conversion from the MotifDb format is terminal)

motifs <- filter_motifs(MotifDb, organism = c("Athaliana", "Celegans"))
#> motifs converted to class 'universalmotif'

## Only keeping motifs with sufficient information content and length:

motifs <- filter_motifs(motifs, icscore = 10, width = 10)

head(summarise_motifs(motifs))
#>        name family  organism     consensus alphabet strand  icscore nsites
#> 1      ERF1    AP2 Athaliana    NMGCCGCCRN      DNA      +- 12.40700     NA
#> 2    ATERF6    AP2 Athaliana    NTGCCGGCGB      DNA      +- 11.77649     NA
#> 3    ATCBF3    AP2 Athaliana    ATGTCGGYNN      DNA      +- 10.66970     NA
#> 4 AT2G18300   bHLH Athaliana  NNNGCACGTGNN      DNA      +- 11.50133     NA
#> 5   bHLH104   bHLH Athaliana    GGCACGTGCC      DNA      +- 16.05350     NA
#> 6    hlh-16   bHLH  Celegans NNNCAATATKGNN      DNA      +- 10.32432     NA
```

8.3 Generate random motif matches

Get a random set of sequences which are created using the probabilities of the motif matrix, in effect generating motif sites, with `sample_sites()`.

```
library(universalmotif)
data(examplemotif)
```

```
sample_sites(examplemotif)
#> DNAStringSet object of length 100:
#>       width seq
#>   [1]     7 TATATAT
#>   [2]     7 TATATAA
#>   [3]     7 TATAAAT
#>   [4]     7 TATAAAT
#>   [5]     7 TATAAAA
#>   ...   ... ...
#>  [96]     7 TATAAAT
#>  [97]     7 TATAAAT
#>  [98]     7 TATATAT
#>  [99]     7 TATAAAA
#> [100]     7 TATAAAA
```

8.4 Motif shuffling

Shuffle a set of motifs with `shuffle_motifs()`. The original shuffling implementation is taken from `shuffle_sequences()`, described in the sequences vignette.

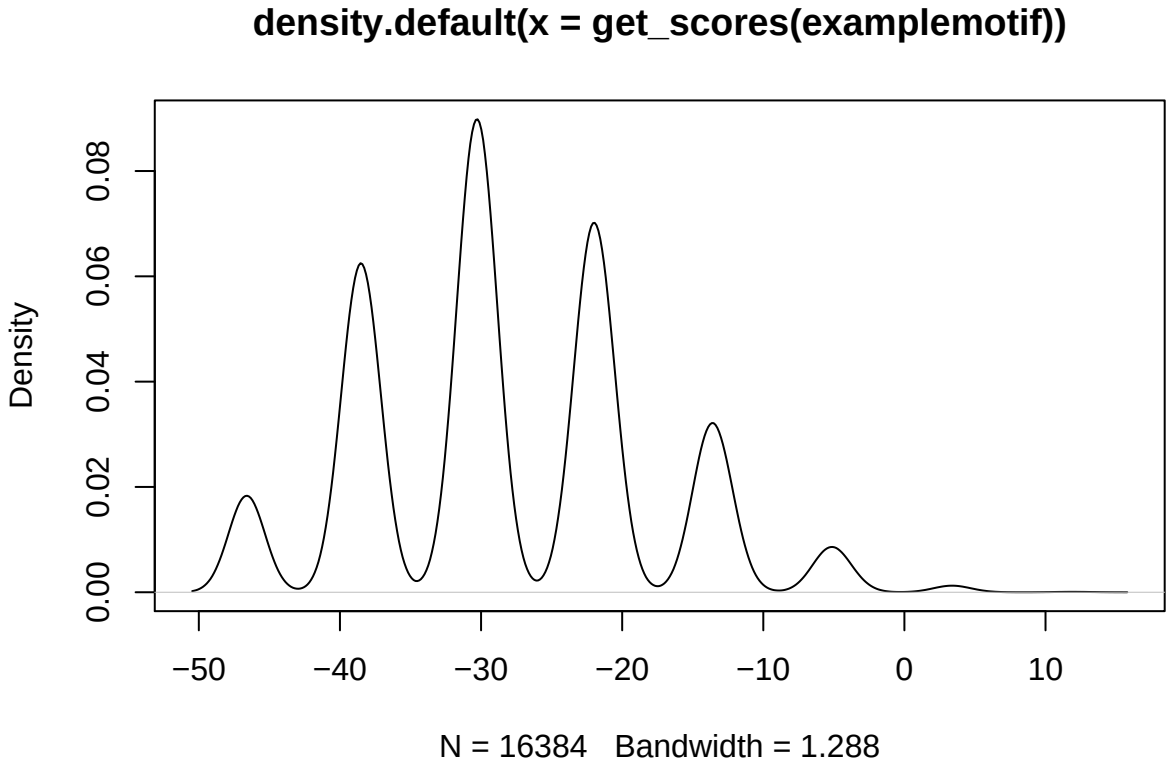
```
library(universalmotif)
library(MotifDb)

motifs <- convert_motifs(MotifDb[1:50])
head(summarise_motifs(motifs))
#>    name    organism consensus alphabet strand   icscore
#> 1  ABF2 Scerevisiae    TCTAGA      DNA     +-  9.371235
#> 2  CAT8 Scerevisiae    CCGGAN      DNA     +-  7.538740
#> 3  CST6 Scerevisiae    TGACGT      DNA     +-  9.801864
#> 4 ECM23 Scerevisiae     AGATC      DNA     +-  6.567494
#> 5  EDS1 Scerevisiae   GGAANAA      DNA     +-  9.314287
#> 6  FKH2 Scerevisiae   GTAAACA      DNA     +- 11.525400

motifs.shuffled <- shuffle_motifs(motifs, k = 3)
head(summarise_motifs(motifs.shuffled))
#>               name consensus alphabet strand   icscore
#> 1  ABF2 [shuffled]    TTGTCT      DNA     +-  8.549073
#> 2  CAT8 [shuffled]    WCCTCG      DNA     +-  7.738366
#> 3  CST6 [shuffled]    GCGGAC      DNA     +-  8.573465
#> 4 ECM23 [shuffled]     GCCTT      DNA     +-  7.492615
#> 5  EDS1 [shuffled]   CCAGGAC      DNA     +- 10.102520
#> 6  FKH2 [shuffled]   AMTCGCR      DNA     +-  8.159721
```

8.5 Scoring and match functions

Motif matches in a set of sequences are typically obtained using logodds scores. Several functions are exposed to reveal some of the internal work that goes on.

- `get_matches()`: show all possible sequence matches above a certain score
- `get_scores()`: obtain all possible scores from all possible sequence matches
- `motif_score()`: translate score thresholds to logodds scores
- `score_match()`: return logodds scores for sequence matches

```
library(universalmotif)
data(examplemotif)
examplemotif
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   11.54
#>         Consensus:   TATAWAW
#>
#>   T A T A   W A   W
#> A 0 1 0 1 0.5 1 0.5
#> C 0 0 0 0 0.0 0 0.0
#> G 0 0 0 0 0.0 0 0.0
#> T 1 0 1 0 0.5 0 0.5

## Get the min and max possible scores:
motif_score(examplemotif)
#>      0%    100%
#> -46.606  11.929

## Show matches above a score of 10:
get_matches(examplemotif, 10)
#> [1] "TATAAAA" "TATATAA" "TATAAAT" "TATATAT"

## Score a specific sequence:
score_match(examplemotif, "TTTTTTT")
#> [1] -14.012

## Take a look at the distribution of scores:
plot(density(get_scores(examplemotif)))
```



8.6 Type conversion functions

While `convert_type()` will take care of switching the current type for `universalmotif` objects, the individual type conversion functions are also available for personal use. These are:

- `icm_to_ppm()`
- `pcm_to_ppm()`
- `ppm_to_icm()`
- `ppm_to_pcm()`
- `ppm_to_pwm()`
- `pwm_to_ppm()`

These functions take a one dimensional vector. To use these for matrices:

```
library(universalmotif)

m <- create_motif(type = "PCM")["motif"]
m
#>    W  M  T  S  T  W  W  A  T  W
#> A 49 65  0  0  8 29 61 94  8 32
#> C  5 34 14 44  1  0  0  0  2  0
#> G  3  0  0 56  5  8 10  0  0 20
#> T 43  1 86  0 86 63 29  6 90 48

apply(m, 2, pcm_to_ppm)
#>         W    M    T    S    T    W    W    A    T    W
#> [1,] 0.49 0.65 0.00 0.00 0.08 0.29 0.61 0.94 0.08 0.32
#> [2,] 0.05 0.34 0.14 0.44 0.01 0.00 0.00 0.00 0.02 0.00
#> [3,] 0.03 0.00 0.00 0.56 0.05 0.08 0.10 0.00 0.00 0.20
#> [4,] 0.43 0.01 0.86 0.00 0.86 0.63 0.29 0.06 0.90 0.48
```

Additionally, the `position_icscore()` can be used to get the total information content per position:

```
library(universalmotif)

position_icscore(c(0.7, 0.1, 0.1, 0.1))
#> [1] 0.6307803
```

Session info

```
#> R version 4.0.0 (2020-04-24)
#> Platform: x86_64-apple-darwin17.0 (64-bit)
#> Running under: macOS Mojave 10.14.6
#>
#> Matrix products: default
#> BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
#>
#> locale:
#> [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
#> attached base packages:
#> [1] stats4    parallel  stats     graphics  grDevices utils     datasets
#> [8] methods   base
#>
#> other attached packages:
#>  [1] TFBSTools_1.26.0     Logolas_1.12.0       dplyr_0.8.5
#>  [4] ggtree_2.2.0         ggplot2_3.3.0        MotifDb_1.30.0
#>  [7] GenomicRanges_1.40.0 GenomeInfoDb_1.24.0  Biostrings_2.56.0
#> [10] XVector_0.28.0       IRanges_2.22.0       S4Vectors_0.26.0
#> [13] BiocGenerics_0.34.0  universalmotif_1.6.0
#>
#> loaded via a namespace (and not attached):
#>   [1] colorspace_1.4-1            grImport2_0.2-0
#>   [3] ellipsis_0.3.0              base64enc_0.1-3
#>   [5] aplot_0.0.4                 rGADEM_2.36.0
#>   [7] farver_2.0.3                bit64_0.9-7
#>   [9] AnnotationDbi_1.50.0        R.methodsS3_1.8.0
#>  [11] motifStack_1.32.0           knitr_1.28
#>  [13] ade4_1.7-15                 jsonlite_1.6.1
#>  [15] splitstackshape_1.4.8       Rsamtools_2.4.0
#>  [17] seqLogo_1.54.0              gridBase_0.4-7
#>  [19] annotate_1.66.0             GO.db_3.10.0
#>  [21] png_0.1-7                   R.oo_1.23.0
#>  [23] BiocManager_1.30.10         readr_1.3.1
#>  [25] compiler_4.0.0              httr_1.4.1
#>  [27] rvcheck_0.1.8               assertthat_0.2.1
#>  [29] Matrix_1.2-18               lazyeval_0.2.2
#>  [31] htmltools_0.4.0             tools_4.0.0
#>  [33] gtable_0.3.0                glue_1.4.0
#>  [35] TFMPvalue_0.0.8             GenomeInfoDbData_1.2.3
#>  [37] reshape2_1.4.4              tinytex_0.22
#>  [39] Rcpp_1.0.4.6                Biobase_2.48.0
#>  [41] vctrs_0.2.4                 ape_5.3
#>  [43] nlme_3.1-147                rtracklayer_1.47.0
```

```
#>  [45] ggseqlogo_0.1              gbRd_0.4-11
#>  [47] xfun_0.13                  CNEr_1.24.0
#>  [49] stringr_1.4.0              ps_1.3.2
#>  [51] lifecycle_0.2.0            poweRlaw_0.70.6
#>  [53] gtools_3.8.2               XML_3.99-0.3
#>  [55] zlibbioc_1.34.0            MASS_7.3-51.6
#>  [57] scales_1.1.0               BSgenome_1.56.0
#>  [59] hms_0.5.3                  SummarizedExperiment_1.18.0
#>  [61] RColorBrewer_1.1-2         yaml_2.2.1
#>  [63] memoise_1.1.0              MotIV_1.44.0
#>  [65] stringi_1.4.6              RSQLite_2.2.0
#>  [67] SQUAREM_2020.2             highr_0.8
#>  [69] tidytree_0.3.3             caTools_1.18.0
#>  [71] BiocParallel_1.22.0        bibtex_0.4.2.2
#>  [73] Rdpack_0.11-1              rlang_0.4.5
#>  [75] pkgconfig_2.0.3            matrixStats_0.56.0
#>  [77] bitops_1.0-6               pracma_2.2.9
#>  [79] evaluate_0.14              lattice_0.20-41
#>  [81] purrr_0.3.4                htmlwidgets_1.5.1
#>  [83] GenomicAlignments_1.24.0   treeio_1.12.0
#>  [85] patchwork_1.0.0            labeling_0.3
#>  [87] bit_1.1-15.2               processx_3.4.2
#>  [89] tidyselect_1.0.0           plyr_1.8.6
#>  [91] magrittr_1.5               bookdown_0.18
#>  [93] R6_2.4.1                   DelayedArray_0.14.0
#>  [95] DBI_1.1.0                  pillar_1.4.3
#>  [97] withr_2.2.0                KEGGREST_1.28.0
#>  [99] RCurl_1.98-1.2             tibble_3.0.1
#> [101] crayon_1.3.4               rmarkdown_2.1
#> [103] jpeg_0.1-8.1               grid_4.0.0
#> [105] data.table_1.12.8          blob_1.2.1
#> [107] digest_0.6.25              xtable_1.8-4
#> [109] tidyr_1.0.2                R.utils_2.9.2
#> [111] munsell_0.5.0              DirichletMultinomial_1.30.0
```

References

Bailey, T. L., M. Boden, F. A. Buske, M. Frith, C. E. Grant, L. Clementi, J. Ren, W. W. Li, and W. S. Noble. 2009. "MEME Suite: Tools for Motif Discovery and Searching." *Nucleic Acids Research* 37: W202–W208.

Guo, Y., K. Tian, H. Zeng, X. Guo, and D. K. Gifford. 2018. "A Novel K-Mer Set Memory (KSM) Motif Representation Improves Regulatory Variant Prediction." *Genome Research* 28: 891–900.

Heinz, S., C. Benner, N. Spann, E. Bertolino, Y. C. Lin, P. Laslo, J. X. Cheng, C. Murre, H. Singh, and C. K. Glass. 2010. "Simple Combinations of Lineage-Determining Transcription Factors Prime Cis-Regulatory Elements Required for Macrophage and B Cell Identities." *Molecular Cell* 38 (4): 576–89.

Hume, M. A., L. A. Barrera, S. S. Gisselbrecht, and M. L. Bulyk. 2015. "UniPROBE, Update 2015: New Tools and Content for the Online Database of Protein-Binding Microarray Data on Protein-Dna Interactions." *Nucleic Acids Research* 43: D117–D122.

Khan, A., O. Fornes, A. Stigliani, M. Gheorghe, J. A. Castro-Mondragon, R. van der Lee, A. Bessy, et al. 2018. "JASPAR 2018: Update of the Open-Access Database of Transcription Factor Binding Profiles and Its Web Framework." *Nucleic Acids Research* 46 (D1): D260–D266.

Mathelier, A., and W. W. Wasserman. 2013. "The Next Generation of Transcription Factor Binding Site

Prediction." *PLoS Computational Biology* 9 (9): e1003214.

Siebert, M., and J. Soding. 2016. "Bayesian Markov Models Consistently Outperform PWMs at Predicting Motifs in Nucleotide Sequences." *Nucleic Acids Research* 44 (13): 6055–69.

Weirauch, M. T., A. Yang, M. Albu, A. G. Cote, A. Montenegro-Montero, P. Drewe, H. S. Najafabadi, et al. 2014. "Determination and Inference of Eukaryotic Transcription Factor Sequence Specificity." *Cell* 158 (6): 1431–43.

Wingender, E., P. Dietze, H. Karas, and R. Knuppel. 1996. "TRANSFAC: A Database on Transcription Factors and Their Dna Binding Sites." *Nucleic Acids Research* 24 (1): 238–41.