Motif import, export, and manipulation

Benjamin Jean-Marie Tremblay*

17 October 2021

Abstract

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

Contents

---

*benjamin.tremblay@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
#>       Pseudocount:   1
#>         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
#>       Pseudocount:   1
#>         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 eval(expr, envir, enclos): * 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). Should you ever come across a motif class from another Bioconductor package which is not supported by the `universalmotif` package, but believe it should be, then feel free to bring it up with me.

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 final. 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")
#> Note: motif [motif] has an empty nsites slot, using 100.
#> 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
#>         Pseudocount:   1
#>           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")
#> Note: motif [ABF2] has an empty nsites slot, using 100.
#> [[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 (not recommended; use `saveRDS()` instead)
- `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")`). If you know of additional motif formats which are not supported in the `universalmotif` package that you believe should be, or of any mistakes in the way the `universalmotif` package parses supported formats, then please let me know.

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 won't know this has taken place).

```r
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")
#> Note: motif [motif] has an empty nsites slot, using 100.
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PCM
#>           Strands:   +-
#>          Total IC:   11.54
#>       Pseudocount:   1
#>         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")
#> Note: motif [motif] has an empty nsites slot, using 100.
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PWM
#>           Strands:   +-
#>          Total IC:   11.54
#>       Pseudocount:   1
#>         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)
#> Note: motif [motif] has an empty nsites slot, using 100.
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PWM
#>           Strands:   +-
#>          Total IC:   11.54
#>       Pseudocount:   1
#>         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. See the IntroductionToSequenceMotifs vignette for an overview on the various types of ICM calculations.

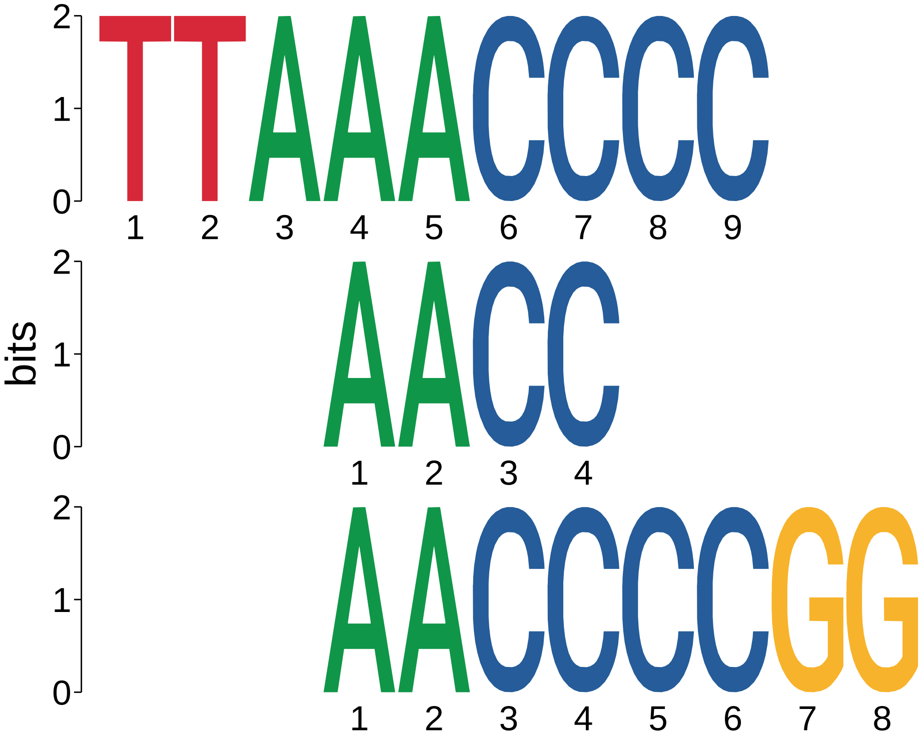
```
examplemotif["nsites"] <- 10
convert_type(examplemotif, "ICM", nsize_correction = FALSE)
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   ICM
#>           Strands:   +-
#>          Total IC:   11.54
#>       Pseudocount:   1
#>         Consensus:   TATAWAW
#>      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
#>       Pseudocount:   1
#>         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
#>       Pseudocount:   1
#>         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),
  show.positions.once = FALSE, show.names = FALSE)
```



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

This functionality can also be automated to reduce the number of overly similar motifs in larger datasets via the `merge_similar()` function.

```r
library(universalmotif)
library(MotifDb)

motifs <- filter_motifs(MotifDb, family = "bHLH")[1:100]
#> motifs converted to class 'universalmotif'
length(motifs)
#> [1] 100

motifs <- merge_similar(motifs)
length(motifs)
#> [1] 63
```

Comparison and merging parameters can be fine-tuned as users wish. See the `compare_motifs()` and `merge_motifs()` documentation for more details, as well as the “Motif comparison and P-values” vignette.

4.3 Motif reverse complement

Get the reverse complement of a motif.

```r
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
#>       Pseudocount:   1
#>         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
#>       Pseudocount:   1
#>         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
#>       Pseudocount:   1
#>         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:   8.82
#>       Pseudocount:   0
#>         Consensus:   NUKUSUKCGC
#>
#>      N    U    K    U    S    U    K    C    G    C
#> A 0.44 0.02 0.00 0.00 0.15 0.11 0.06 0.17 0.21 0.01
#> C 0.06 0.03 0.08 0.03 0.45 0.00 0.00 0.64 0.00 0.69
#> G 0.20 0.02 0.35 0.00 0.40 0.25 0.52 0.10 0.79 0.07
#> U 0.30 0.93 0.57 0.97 0.01 0.64 0.42 0.09 0.00 0.23

switch_alph(motif)
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   8.82
#>       Pseudocount:   0
#>         Consensus:   NTKTSTKCGC
#>
#>      N    T    K    T    S    T    K    C    G    C
#> A 0.44 0.02 0.00 0.00 0.15 0.11 0.06 0.17 0.21 0.01
#> C 0.06 0.03 0.08 0.03 0.45 0.00 0.00 0.64 0.00 0.69
#> G 0.20 0.02 0.35 0.00 0.40 0.25 0.52 0.10 0.79 0.07
#> T 0.30 0.93 0.57 0.97 0.01 0.64 0.42 0.09 0.00 0.23
```

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, as well as the direction of trimming (by default both edges are trimmed).

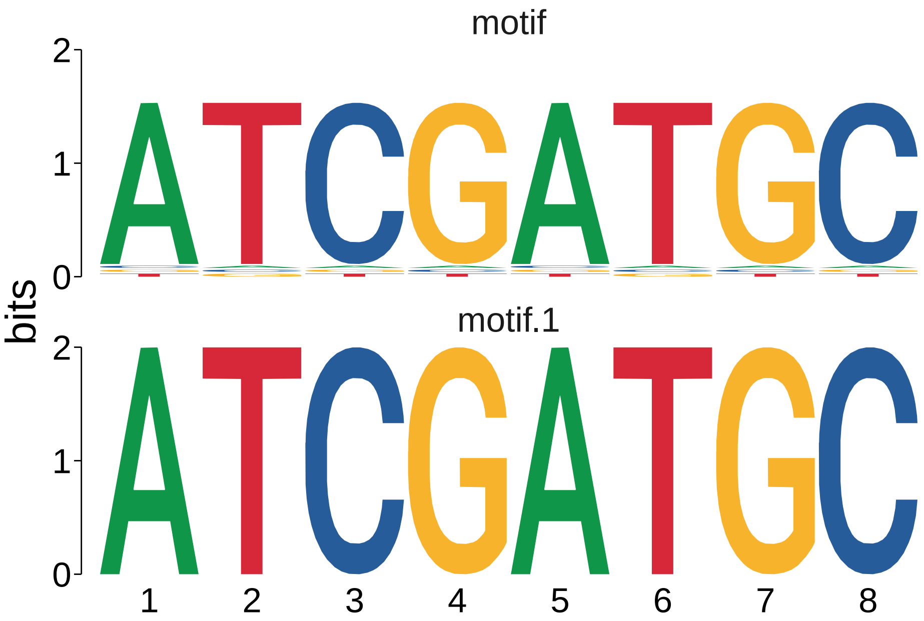
```
library(universalmotif)

motif <- create_motif("NNGCSGCGGNN")
motif
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   13
#>       Pseudocount:   0
#>         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
#>       Pseudocount:   0
#>         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
trim_motifs(motif, trim.from = "right")
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   13
#>       Pseudocount:   0
#>         Consensus:   NNGCSGCGG
#>      Target sites:   100
#>
#>      N    N G C   S G C G G
#> A 0.25 0.25 0 0 0.0 0 0 0 0
#> C 0.25 0.25 0 1 0.5 0 1 0 0
#> G 0.25 0.25 1 0 0.5 1 0 1 1
#> T 0.25 0.25 0 0 0.0 0 0 0 0
```

4.6 Rounding motifs

Round off near-zero probabilities.

```
motif1 <- create_motif("ATCGATGC", pseudocount = 10, type = "PPM", nsites = 100)
motif2 <- round_motif(motif1)
view_motifs(c(motif1, motif2))
```



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 motif class unsupported by `convert_motifs()`, you can instead manually create `universalmotif` class motifs using the `create_motif()` function and the motif matrix.

```
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
#>        Pseudocount:   1
#>          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 brief 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 simply as matrices. However it is much safer to first format 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`. Note that ambiguity letters can be used with single strings.

```
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
#>        Pseudocount:   1
#>          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 to, 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:   8.59
#>       Pseudocount:   0
#>         Consensus:   AWANYKWMWY
#>
#>      A    W A    N    Y    K    W    M    W    Y
#> A 0.97 0.39 1 0.13 0.01 0.04 0.29 0.43 0.61 0.14
#> C 0.01 0.00 0 0.12 0.29 0.00 0.10 0.37 0.00 0.43
#> G 0.01 0.09 0 0.34 0.00 0.29 0.12 0.20 0.00 0.10
#> T 0.01 0.51 0 0.40 0.70 0.67 0.49 0.00 0.39 0.34
```

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:   9.65
#>       Pseudocount:   0
#>         Consensus:   YAYMCMYSNC
#>
#>      Y    A    Y    M    C    M    Y    S    N    C
#> A 0.01 0.90 0.00 0.55 0.04 0.41 0.00 0.00 0.23 0.00
#> C 0.50 0.08 0.53 0.44 0.82 0.59 0.51 0.48 0.23 0.99
#> G 0.04 0.00 0.07 0.01 0.00 0.00 0.08 0.46 0.48 0.00
#> T 0.46 0.01 0.40 0.01 0.14 0.00 0.41 0.07 0.05 0.01
```

With a custom alphabet:

```
create_motif(alphabet = "QWERTY")
#>
#>        Motif name:   motif
#>          Alphabet:   EQRTWY
#>              Type:   PPM
#>          Total IC:   12.6
#>       Pseudocount:   0
#>
#>   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
#> E 0.20 0.34 0.06 0.08 0.10 0.00 0.08 0.04 0.00  0.00
#> Q 0.00 0.66 0.22 0.31 0.13 0.62 0.01 0.70 0.23  0.00
#> R 0.37 0.00 0.15 0.00 0.60 0.00 0.03 0.26 0.04  0.02
#> T 0.00 0.00 0.00 0.00 0.07 0.00 0.02 0.00 0.44  0.91
#> W 0.08 0.00 0.56 0.17 0.08 0.37 0.06 0.00 0.28  0.06
#> Y 0.34 0.00 0.00 0.44 0.01 0.00 0.80 0.00 0.01  0.00
```

6 Motif visualization

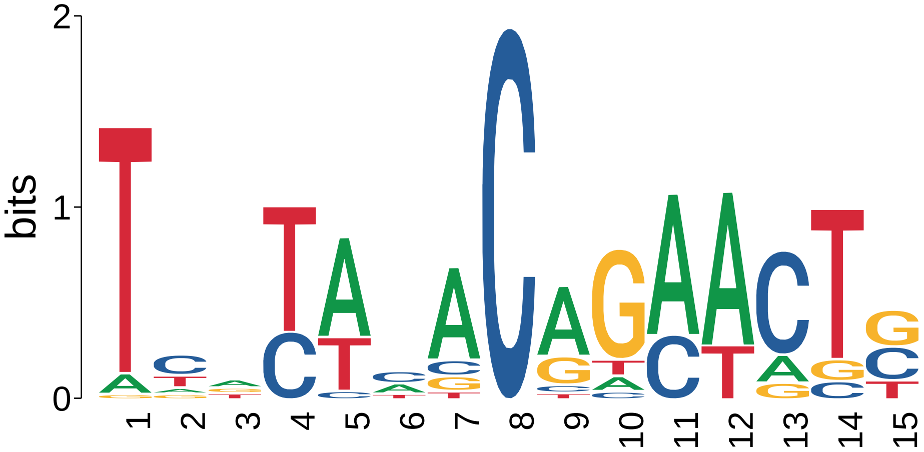
6.1 Motif logos

There are several packages which offer motif visualization capabilities, such as `seqLogo`, `motifStack`, and `ggseqlogo`. The `universalmotif` package has its own implementation via the function `view_motifs()`, which renders motifs using the `ggplot2` package (similar to `ggseqlogo`). 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)
```

The `view_motifs()` function generates `ggplot` objects; feel free to manipulate them as such. For example, flipping the position numbers for larger motifs (where the text spacing can become tight):

```
view_motifs(create_motif(15)) +
  ggplot2::theme(
    axis.text.x = ggplot2::element_text(angle = 90, hjust = 1)
  )
```

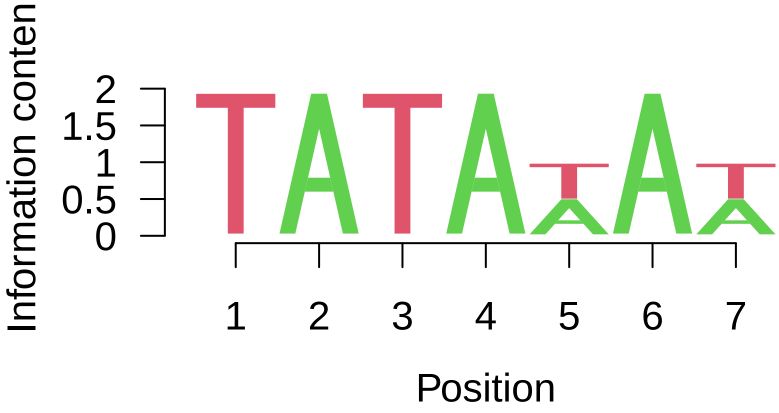


A large number of options are available for tuning the way motifs are plotted in `view_motifs()`. Visit the documentation for more information.

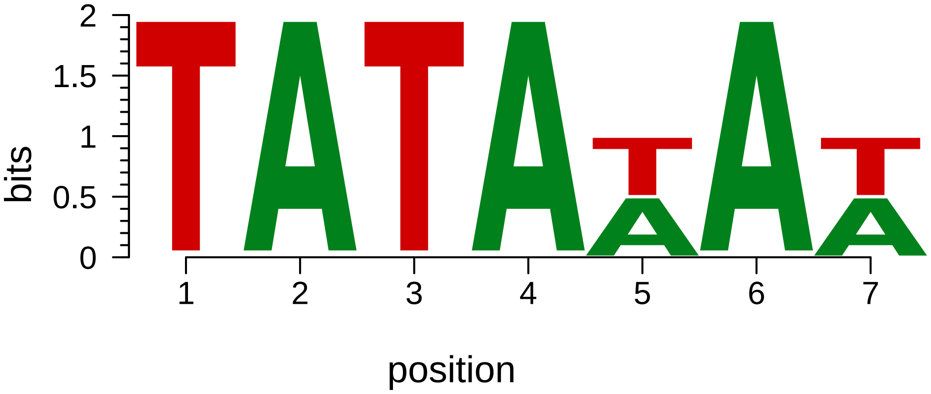
Using the other Bioconductor packages to view `universalmotif` motifs is fairly easy as well:

```
## 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)
```



```
## motifStack:
motifStack::plotMotifLogo(motif)
#> Loading required namespace: Cairo
```

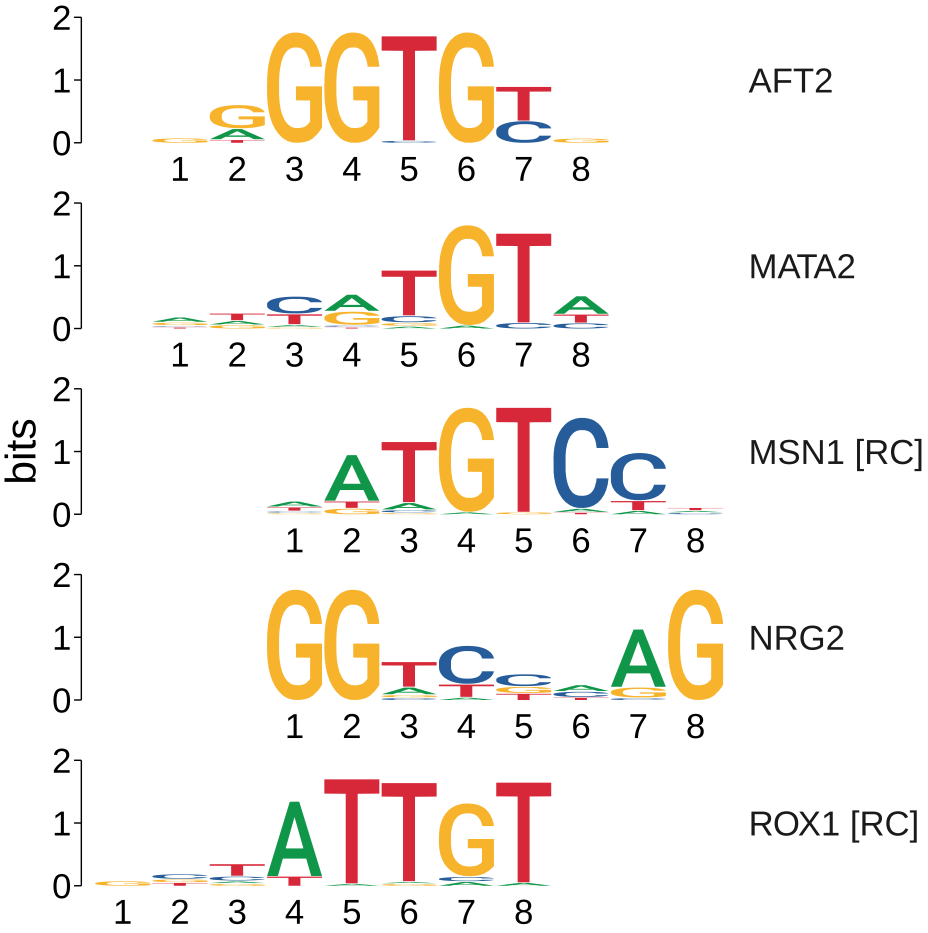


```
## ggseqlogo:
ggseqlogo::ggseqlogo(motif)
#> Warning: The `<scale>` argument of `guides()` cannot be `FALSE`. Use "none" instead as
#> of ggplot2 3.3.4.
#> i The deprecated feature was likely used in the ggseqlogo package.
#>   Please report the issue at <https://github.com/omarwagih/ggseqlogo/issues>.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
#> generated.
```

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 most of these, still offers basic stacking via `view_motifs()`. The motifs are aligned using `compare_motifs()`.

```r
library(universalmotif)
library(MotifDb)

motifs <- convert_motifs(MotifDb[50:54])
view_motifs(motifs, show.positions.once = FALSE, names.pos = "right")
```



6.3 Plot arbitrary text logos

The logo plotting capabilities of `view_motifs()` can be used for any kind of arbitrary text logo. All you need is a numeric matrix (the heights of the characters), with the desired characters as row names. The following example is taken from the `view_logo()` documentation.

```
library(universalmotif)
data(examplemotif)

## Start from a numeric matrix:
toplot <- examplemotif["motif"]

# Adjust the character heights as you wish (negative values are possible):
toplot[4] <- 2
toplot[20] <- -0.5

# Mix and match the number of characters per letter/position:
rownames(toplot)[1] <- "AA"

toplot <- toplot[c(1, 4), ]

toplot
#>    T A T A    W A   W
#> AA 0 1 0 1  0.5 1 0.5
#> T  2 0 1 0 -0.5 0 0.5

view_logo(toplot)
```

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

#	Sequence
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, this sometimes is not the case. By incorporating this type of information into the motif, it can allow for increased accuracy in motif searching. A few example 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
#>       Pseudocount:   0
#>         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
```

To plot these motifs, use `view_motifs()`:

```
view_motifs(motif.k2, use.freq = 2)
```

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 Tidy motif manipulation with the `universalmotif_df` data structure

For those who enjoy using the tidyverse functions for data handling, motifs can additionally represented as the modified `data.frame` format: `universalmotif_df`. This format allows one to modify motif slots for multiples motifs simultaneously using the `universalmotif_df` columns, and then return to a list of motifs afterwards to resume use with `universalmotif` package functions. A few key functions have been provided in relation to this format:

- `to_df()`: Generate a `universalmotif_df` object from a list of motifs.
- `update_motifs()`: After modifying the `universalmotif_df` object, apply these modifications to the actual `universalmotif` objects (contained within the `motif` column).
- `to_list()`: Return to a list of `universalmotif` objects for use with `universalmotif` package functions. Note that it is not required to use `update_motifs()` before using `to_list()`, as modifications will be checked for and applied if found.
- `requires_update()`: Boolean check as to whether the `universalmotif` objects and the `universalmotif_df` columns differ and require either a `update_motifs()` or `to_list()` call to re-sync them.

```
library(universalmotif)
library(MotifDb)

## Obtain a `universalmotif_df` object
motifs <- to_df(MotifDb)
head(motifs)
#>         motif  name     altname    organism consensus alphabet strand   icscore
#> 1  <mot:ABF2>  ABF2  badis.ABF2 Scerevisiae    TCTAGA      DNA     +-  9.371235
#> 2  <mot:CAT8>  CAT8  badis.CAT8 Scerevisiae    CCGGAN      DNA     +-  7.538740
#> 3  <mot:CST6>  CST6  badis.CST6 Scerevisiae    TGACGT      DNA     +-  9.801864
#> 4 <mot:ECM23> ECM23 badis.ECM23 Scerevisiae     AGATC      DNA     +-  6.567494
#> 5  <mot:EDS1>  EDS1  badis.EDS1 Scerevisiae   GGAANAA      DNA     +-  9.314287
#> 6  <mot:FKH2>  FKH2  badis.FKH2 Scerevisiae   GTAAACA      DNA     +- 11.525400
#>    type pseudocount          bkg dataSource
#> 1  PPM           1 0.25, 0.....     ScerTF
#> 2  PPM           1 0.25, 0.....     ScerTF
#> 3  PPM           1 0.25, 0.....     ScerTF
#> 4  PPM           1 0.25, 0.....     ScerTF
#> 5  PPM           1 0.25, 0.....     ScerTF
#> 6  PPM           1 0.25, 0.....     ScerTF
#>
#> [Hidden empty columns: family, nsites, bkgsites, pval, qval, eval.]
```

Some tidy manipulation:

```
library(dplyr)

motifs <- motifs %>%
  mutate(bkg = case_when(
    organism == "Athaliana" ~ list(c(A = 0.32, C = 0.18, G = 0.18, T = 0.32)),
    TRUE ~ list(c(A = 0.25, C = 0.25, G = 0.25, T = 0.25))
  ))
head(filter(motifs, organism == "Athaliana"))
#>             motif      name    altname family   organism  consensus alphabet
#> 1 *  <mot:ORA59>     ORA59 M0005_1.02    AP2 Athaliana   MGCCGCCN      DNA
#> 2 *   <mot:WIN1>      WIN1 M0006_1.02    AP2 Athaliana NCRCCGCNNN      DNA
```

```
#> 3 * <mot:AT1G..> AT1G22985 M0007_1.02    AP2 Athaliana  NNGCCGNNN      DNA
#> 4 *   <mot:TEM1>      TEM1 M0008_1.02    AP2 Athaliana   WATGTTGC      DNA
#> 5 *  <mot:ERF11>     ERF11 M0009_1.02    AP2 Athaliana NNGCCGNNNN      DNA
#> 6 * <mot:RAP2.6>    RAP2.6 M0010_1.02    AP2 Athaliana   NNGCCGNN      DNA
#>   strand   icscore type pseudocount          bkg dataSource
#> 1     +- 11.351632  PPM           1 0.32, 0..... cisbp_1.02
#> 2     +-  6.509679  PPM           1 0.32, 0..... cisbp_1.02
#> 3     +-  5.155725  PPM           1 0.32, 0..... cisbp_1.02
#> 4     +- 11.182383  PPM           1 0.32, 0..... cisbp_1.02
#> 5     +-  5.148803  PPM           1 0.32, 0..... cisbp_1.02
#> 6     +-  4.227144  PPM           1 0.32, 0..... cisbp_1.02
#>
#> [Hidden empty columns: nsites, bkgsites, pval, qval, eval.]
#> [Rows marked with * are changed. Run update_motifs() or to_list() to
#>   apply changes.]
```

Feel free to add columns as well. You can add 1d vectors which will be added to the `extrainfo` slots of motifs. (Note that they will be coerced to character vectors!)

```
motifs <- motifs %>%
  mutate(MotifIndex = 1:n())
head(motifs)
#>           motif  name     altname    organism consensus alphabet strand
#> 1 *  <mot:ABF2>  ABF2  badis.ABF2 Scerevisiae    TCTAGA      DNA     +-
#> 2 *  <mot:CAT8>  CAT8  badis.CAT8 Scerevisiae    CCGGAN      DNA     +-
#> 3 *  <mot:CST6>  CST6  badis.CST6 Scerevisiae    TGACGT      DNA     +-
#> 4 * <mot:ECM23> ECM23 badis.ECM23 Scerevisiae     AGATC      DNA     +-
#> 5 *  <mot:EDS1>  EDS1  badis.EDS1 Scerevisiae   GGAANAA      DNA     +-
#> 6 *  <mot:FKH2>  FKH2  badis.FKH2 Scerevisiae   GTAAACA      DNA     +-
#>     icscore type pseudocount          bkg dataSource MotifIndex
#> 1  9.371235  PPM           1 0.25, 0.....     ScerTF          1
#> 2  7.538740  PPM           1 0.25, 0.....     ScerTF          2
#> 3  9.801864  PPM           1 0.25, 0.....     ScerTF          3
#> 4  6.567494  PPM           1 0.25, 0.....     ScerTF          4
#> 5  9.314287  PPM           1 0.25, 0.....     ScerTF          5
#> 6 11.525400  PPM           1 0.25, 0.....     ScerTF          6
#>
#> [Hidden empty columns: family, nsites, bkgsites, pval, qval, eval.]
#> [Rows marked with * are changed. Run update_motifs() or to_list() to
#>   apply changes.]

to_list(motifs)[[1]]
#>
#>        Motif name:   ABF2
#>    Alternate name:   badis.ABF2
#>          Organism:   Scerevisiae
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   9.37
#>       Pseudocount:   1
#>         Consensus:   TCTAGA
#>        Extra info:   [dataSource] ScerTF
#>                      [MotifIndex] 1
```

```
#>
#>      T    C    T    A    G    A
#> A 0.09 0.01 0.01 0.97 0.01 0.94
#> C 0.09 0.97 0.01 0.01 0.01 0.02
#> G 0.02 0.01 0.01 0.01 0.97 0.02
#> T 0.80 0.01 0.97 0.01 0.01 0.02
```

If during the course of your manipulation you've generated temporary columns which you wish to drop, you can set `extrainfo = FALSE` to discard all extra columns. Be careful though, this will discard any previously existing `extrainfo` data as well.

```
to_list(motifs, extrainfo = FALSE)[[1]]
#> Discarding unknown slot(s) 'dataSource', 'MotifIndex' (set
#>   `extrainfo=TRUE` to preserve these).
#>
#>        Motif name:   ABF2
#>    Alternate name:   badis.ABF2
#>          Organism:   Scerevisiae
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   9.37
#>       Pseudocount:   1
#>         Consensus:   TCTAGA
#>
#>      T    C    T    A    G    A
#> A 0.09 0.01 0.01 0.97 0.01 0.94
#> C 0.09 0.97 0.01 0.01 0.01 0.02
#> G 0.02 0.01 0.01 0.01 0.97 0.02
#> T 0.80 0.01 0.97 0.01 0.01 0.02
```

9 Miscellaneous motif utilities

A number of convenience functions are included for manipulating motifs.

9.1 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
```

9.2 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

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    altname family  organism     consensus alphabet strand  icscore
#> 1      ERF1 M0025_1.02    AP2 Athaliana    NMGCCGCCRN      DNA     +- 12.40700
#> 2    ATERF6 M0027_1.02    AP2 Athaliana    NTGCCGGCGB      DNA     +- 11.77649
#> 3    ATCBF3 M0032_1.02    AP2 Athaliana    ATGTCGGYNN      DNA     +- 10.66970
#> 4 AT2G18300 M0155_1.02   bHLH Athaliana  NNNGCACGTGNN      DNA     +- 11.50133
#> 5   bHLH104 M0159_1.02   bHLH Athaliana    GGCACGTGCC      DNA     +- 16.05350
#> 6    hlh-16 M0173_1.02   bHLH  Celegans NNNCAATATKGNN      DNA     +- 10.32432
#>   nsites
#> 1     NA
#> 2     NA
#> 3     NA
#> 4     NA
#> 5     NA
#> 6     NA
```

9.3 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 TATAAAA
#>   [3]     7 TATAAAA
#>   [4]     7 TATATAA
#>   [5]     7 TATAAAA
#>   ...   ... ...
#>  [96]     7 TATAAAT
#>  [97]     7 TATAAAT
#>  [98]     7 TATATAT
#>  [99]     7 TATAAAA
#> [100]     7 TATAAAT
```

9.4 Motif shuffling

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

```
library(universalmotif)
library(MotifDb)

motifs <- convert_motifs(MotifDb[1:50])
head(summarise_motifs(motifs))
#>    name     altname    organism consensus alphabet strand   icscore
#> 1  ABF2  badis.ABF2 Scerevisiae    TCTAGA      DNA     +-  9.371235
#> 2  CAT8  badis.CAT8 Scerevisiae    CCGGAN      DNA     +-  7.538740
#> 3  CST6  badis.CST6 Scerevisiae    TGACGT      DNA     +-  9.801864
#> 4 ECM23 badis.ECM23 Scerevisiae     AGATC      DNA     +-  6.567494
#> 5  EDS1  badis.EDS1 Scerevisiae   GGAANAA      DNA     +-  9.314287
#> 6  FKH2  badis.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]    TRWKCC      DNA     +-  5.856591
#> 2  CAT8 [shuffled]    GGGMAA      DNA     +-  7.261442
#> 3  CST6 [shuffled]    SCSNCA      DNA     +-  5.214041
#> 4 ECM23 [shuffled]     CGRSS      DNA     +-  5.597233
#> 5  EDS1 [shuffled]   AATGAGG      DNA     +- 10.638115
#> 6  FKH2 [shuffled]   GBCMTGT      DNA     +-  8.392421
```

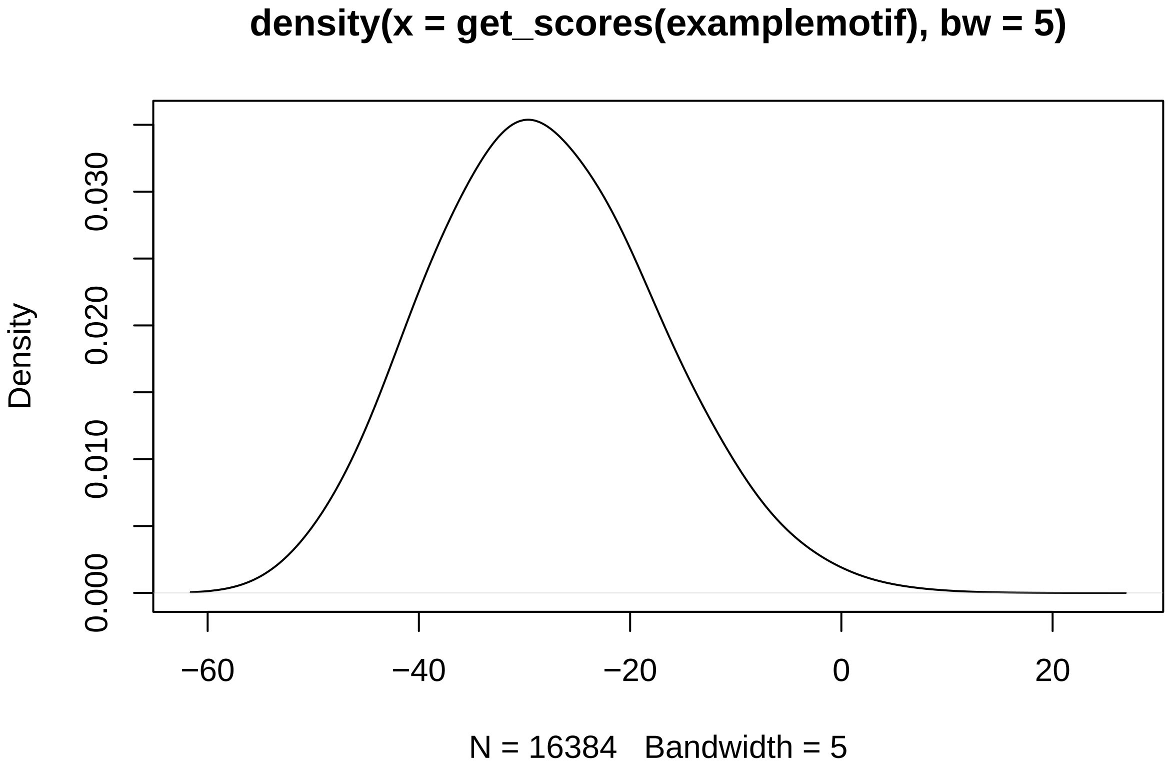
9.5 Scoring and match functions

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
- `prob_match()`: return probabilities for sequence matches
- `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
#>       Pseudocount:   1
#>         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"

## Get the probability of a match:
prob_match(examplemotif, "TTTTTTT", allow.zero = FALSE)
#> [1] 6.103516e-05

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

## Take a look at the distribution of scores:
plot(density(get_scores(examplemotif), bw = 5))
```



9.6 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
#>    T  K  G  T  R  A  W  Y   T  C
#> A  0  1  0  0 66 78 65  4   0  7
#> C  2  1 14  0  3 22  8 53   0 86
#> G  8 32 86  7 29  0  0  4   0  7
#> T 90 66  0 93  2  0 27 39 100  0

apply(m, 2, pcm_to_ppm)
#>         T    K    G    T    R    A    W    Y T    C
#> [1,] 0.00 0.01 0.00 0.00 0.66 0.78 0.65 0.04 0 0.07
#> [2,] 0.02 0.01 0.14 0.00 0.03 0.22 0.08 0.53 0 0.86
#> [3,] 0.08 0.32 0.86 0.07 0.29 0.00 0.00 0.04 0 0.07
#> [4,] 0.90 0.66 0.00 0.93 0.02 0.00 0.27 0.39 1 0.00
```

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 4.3.1 (2023-06-16)
#> Platform: x86_64-pc-linux-gnu (64-bit)
#> Running under: Ubuntu 22.04.3 LTS
#> 
#> Matrix products: default
#> BLAS:   /home/biocbuild/bbs-3.18-bioc/R/lib/libRblas.so 
#> LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0
#> 
#> locale:
#>  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C              
#>  [3] LC_TIME=en_GB              LC_COLLATE=C              
#>  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8   
#>  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                 
#>  [9] LC_ADDRESS=C               LC_TELEPHONE=C            
#> [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C       
#> 
#> time zone: America/New_York
#> tzcode source: system (glibc)
#> 
#> attached base packages:
#> [1] stats4    stats     graphics  grDevices utils     datasets  methods  
#> [8] base     
#> 
#> other attached packages:
```

```
#>  [1] TFBSTools_1.40.0     cowplot_1.1.1         dplyr_1.1.3
#>  [4] ggtree_3.10.0        ggplot2_3.4.4         MotifDb_1.44.0
#>  [7] GenomicRanges_1.54.0 Biostrings_2.70.0     GenomeInfoDb_1.38.0
#> [10] IRanges_2.36.0       S4Vectors_0.40.0      BiocGenerics_0.48.0
#> [13] XVector_0.42.0       universalmotif_1.20.0
#> 
#> loaded via a namespace (and not attached):
#>   [1] jsonlite_1.8.7              magrittr_2.0.3
#>   [3] farver_2.1.1                rmarkdown_2.25
#>   [5] fs_1.6.3                    BiocIO_1.12.0
#>   [7] zlibbioc_1.48.0             vctrs_0.6.4
#>   [9] memoise_2.0.1               Cairo_1.6-1
#>  [11] Rsamtools_2.18.0            RCurl_1.98-1.12
#>  [13] base64enc_0.1-3             tinytex_0.48
#>  [15] htmltools_0.5.6.1           S4Arrays_1.2.0
#>  [17] CNEr_1.38.0                 SparseArray_1.2.0
#>  [19] gridGraphics_0.5-1          pracma_2.4.2
#>  [21] htmlwidgets_1.6.2           plyr_1.8.9
#>  [23] cachem_1.0.8                GenomicAlignments_1.38.0
#>  [25] lifecycle_1.0.3             pkgconfig_2.0.3
#>  [27] Matrix_1.6-1.1              R6_2.5.1
#>  [29] fastmap_1.1.1               GenomeInfoDbData_1.2.11
#>  [31] MatrixGenerics_1.14.0       digest_0.6.33
#>  [33] aplot_0.2.2                 colorspace_2.1-0
#>  [35] TFMPvalue_0.0.9             patchwork_1.1.3
#>  [37] AnnotationDbi_1.64.0        RSQLite_2.3.1
#>  [39] seqLogo_1.68.0              labeling_0.4.3
#>  [41] fansi_1.0.5                 httr_1.4.7
#>  [43] abind_1.4-5                 compiler_4.3.1
#>  [45] bit64_4.0.5                 withr_2.5.1
#>  [47] BiocParallel_1.36.0         DBI_1.1.3
#>  [49] R.utils_2.12.2              MASS_7.3-60
#>  [51] poweRlaw_0.70.6             DelayedArray_0.28.0
#>  [53] rjson_0.2.21                gtools_3.9.4
#>  [55] caTools_1.18.2              tools_4.3.1
#>  [57] splitstackshape_1.4.8       ape_5.7-1
#>  [59] ggseqlogo_0.1               R.oo_1.25.0
#>  [61] glue_1.6.2                  restfulr_0.0.15
#>  [63] nlme_3.1-163                grid_4.3.1
#>  [65] reshape2_1.4.4              ade4_1.7-22
#>  [67] generics_0.1.3              gtable_0.3.4
#>  [69] BSgenome_1.70.0             tzdb_0.4.0
#>  [71] R.methodsS3_1.8.2           tidyr_1.3.0
#>  [73] data.table_1.14.8           hms_1.1.3
#>  [75] utf8_1.2.4                  pillar_1.9.0
#>  [77] stringr_1.5.0               yulab.utils_0.1.0
#>  [79] treeio_1.26.0               lattice_0.22-5
#>  [81] rtracklayer_1.62.0          bit_4.0.5
#>  [83] annotate_1.80.0             tidyselect_1.2.0
#>  [85] DirichletMultinomial_1.44.0 GO.db_3.18.0
#>  [87] knitr_1.44                  grImport2_0.3-0
#>  [89] bookdown_0.36               SummarizedExperiment_1.32.0
#>  [91] xfun_0.40                   Biobase_2.62.0
#>  [93] matrixStats_1.0.0           stringi_1.7.12
```

```
#>  [95] lazyeval_0.2.2            ggfun_0.1.3
#>  [97] yaml_2.3.7                evaluate_0.22
#>  [99] codetools_0.2-19          tibble_3.2.1
#> [101] ggplotify_0.1.2           cli_3.6.1
#> [103] xtable_1.8-4              munsell_0.5.0
#> [105] Rcpp_1.0.11               png_0.1-8
#> [107] XML_3.99-0.14             parallel_4.3.1
#> [109] readr_2.1.4               blob_1.2.4
#> [111] jpeg_0.1-10               bitops_1.0-7
#> [113] tidytree_0.4.5            scales_1.2.1
#> [115] motifStack_1.46.0         purrr_1.0.2
#> [117] crayon_1.5.2              rlang_1.1.1
#> [119] KEGGREST_1.42.0
```

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.