# Motif import, export, and manipulation

***Benjamin Jean-Marie Tremblay***[*1]

[1]University of Waterloo, Waterloo, Canada

[*]b2tremblay@uwaterloo.ca

**3 May 2019**

## Contents

# 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 advanced usage and analyses, see the advanced usage 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:   14
#>         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; see the *Multifreq* section concerning `add_multifreq()` in the advanced usage vignette
- `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:   14
#>         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_motif` function. This allows for easy transition between supported packages (see `?convert_motif` 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 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).

```
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:   14
#>         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] 0
convert_type(examplemotif, "PWM")
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PWM
#>           Strands:   +-
#>          Total IC:   14
#>         Consensus:   TATAWAW
#>
#>      T    A    T    A    W    A    W
#> A -Inf    2 -Inf    2    1    2    1
#> C -Inf -Inf -Inf -Inf -Inf -Inf -Inf
#> G -Inf -Inf -Inf -Inf -Inf -Inf -Inf
#> T    2 -Inf    2 -Inf    1 -Inf    1
```

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:   14
#>         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:   14
#>         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:   14
#>         Consensus:   TATAWAW
#>      Target sites:   10
#>
#>       T     A     T     A    W     A    W
#> A  0.00 17.53  0.00 17.53 3.77 17.53 3.77
#> 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 17.53  0.00 17.53  0.00 3.77  0.00 3.77

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:   14
#>         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 Comparing and merging motifs

There a few functions available in other Bioconductor packages which allow for motif comparison. These include `PWMSimlarity()` (*TFBSTools*), `motifDistances()` (*MotIV*), and `motifSimilarity()` (*PWMEnrich*). Unfortunately these functions are not designed for comparing large numbers of motifs, and can result in long run times. Furthermore they are restrictive in their option range. The *universalmotif* package aims to fix this by providing the `compare_motifs()` function.

This function has been written to allow comparisons using the following metrics: Pearson correlation coefficient, Euclidean distance, Sandelin-Wasserman similarity, and Kullback-Leibler divergence. A large number of options to tune the comparison algorithm are also available, including normalisation, preventing comparison of regions with low information content, controlling possible overhang length, and checking the reverse complement of each motif.

```
library(universalmotif)
library(MotifDb)

## No need to convert class, all universalmotif functions will do it
## automatically

motifs.dist <- compare_motifs(MotifDb[1:5], progress = FALSE)
as.dist(motifs.dist)
#>              ABF2        CAT8        CST6       ECM23
#> CAT8   0.11657663
#> CST6   0.11330732  0.30922803
#> ECM23 -0.07800426 -0.19591797  0.45224482
#> EDS1   0.14826927  0.02107960  0.04629048  0.31556997

## Additionally P-value calculations can be performed for requested
## comparisons

compare_motifs(MotifDb[1:5], compare.to = 1:5, progress = FALSE)
#> No significant hits
```

The `compare_motifs()` functionality is revisited in the advanced usage vignette.

Additionally, *universalmotif* provides the `merge_motifs()` function. This uses the same algorithm from `compare_motifs()` to find the higher scoring alignments before averaging the motifs as PPM type.

```
library(universalmotif)
library(MotifDb)

motifs <- convert_motifs(MotifDb[1:5])

## Let us peek at the motifs before merging:

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

## Now merge:

merge_motifs(motifs)
#>
#>        Motif name:   ABF2/CAT8/CST6/ECM23/EDS1
#>    Alternate name:   badis.ABF2/badis.CAT8/badis.CST6/badis.E...
#>          Organism:   Scerevisiae
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   3.74
#>         Consensus:   TGANGNA
#>        Extra info:   dataSource: ScerTF
#>                      dataSource: ScerTF
#>                      dataSource: ScerTF
#>                      : ...
#>
#>      T    G    A    N    G    N    A
#> A 0.09 0.02 0.58 0.35 0.11 0.40 0.83
#> C 0.10 0.25 0.20 0.39 0.23 0.03 0.01
#> G 0.25 0.58 0.01 0.01 0.60 0.22 0.01
#> T 0.56 0.15 0.21 0.25 0.06 0.35 0.15
```

## 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:   14
#>         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:   12
#>         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:   14
#>         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:   WUAGUYGRMG
#>
#>      W    U    A    G U    Y    G    R    M    G
#> A 0.30 0.09 0.75 0.00 0 0.00 0.02 0.42 0.51 0.23
#> C 0.07 0.00 0.24 0.00 0 0.49 0.02 0.02 0.47 0.00
#> G 0.00 0.00 0.00 0.97 0 0.08 0.96 0.52 0.00 0.73
#> U 0.63 0.90 0.01 0.03 1 0.43 0.00 0.05 0.02 0.04

switch_alph(motif)
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   12.21
#>         Consensus:   WTAGTYGRMG
#>
#>      W    T    A    G T    Y    G    R    M    G
#> A 0.30 0.09 0.75 0.00 0 0.00 0.02 0.42 0.51 0.23
#> C 0.07 0.00 0.24 0.00 0 0.49 0.02 0.02 0.47 0.00
#> G 0.00 0.00 0.00 0.97 0 0.08 0.96 0.52 0.00 0.73
#> T 0.63 0.90 0.01 0.03 1 0.43 0.00 0.05 0.02 0.04
```

## 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
#>      Target sites:   4
#>
#>      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:   4
#>
#>   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
```

# 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 `rdirichlet()` (from *gtools*) to avoid creating low information content motifs.

```
create_motif()
#>
#>        Motif name:   motif
#>          Alphabet:   DNA
#>              Type:   PPM
#>           Strands:   +-
#>          Total IC:   10.06
#>         Consensus:   NMGCTTKRRG
#>
#>      N    M    G    C    T    T    K    R    R G
#> A 0.43 0.32 0.04 0.00 0.00 0.04 0.00 0.56 0.49 0
#> C 0.03 0.57 0.01 0.78 0.06 0.01 0.02 0.00 0.00 0
#> G 0.22 0.11 0.76 0.03 0.18 0.11 0.54 0.41 0.51 1
#> T 0.32 0.00 0.20 0.18 0.76 0.83 0.45 0.02 0.00 0
```

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:   10.83
#>         Consensus:   TYGCAGAYCC
#>
#>      T    Y    G    C    A    G    A    Y    C    C
#> A 0.11 0.02 0.00 0.06 0.71 0.17 0.68 0.01 0.12 0.13
#> C 0.06 0.47 0.23 0.94 0.15 0.01 0.25 0.37 0.77 0.87
```

```
#> G 0.06 0.00 0.75 0.00 0.00 0.82 0.07 0.00 0.11 0.00
#> T 0.77 0.50 0.03 0.00 0.14 0.00 0.00 0.62 0.00 0.00
```

With a custom alphabet:

```
create_motif(alphabet = "QWERTY")
#>
#>        Motif name:   motif
#>          Alphabet:   EQRTWY
#>              Type:   PPM
#>          Total IC:   15.94
#>
#>   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
#> E 0.00 0.70 0.20 0.02 0.00 0.03 0.00 0.18 0.00  0.13
#> Q 0.00 0.00 0.07 0.11 0.04 0.00 0.00 0.00 0.84  0.05
#> R 0.64 0.13 0.30 0.01 0.01 0.03 0.65 0.00 0.00  0.00
#> T 0.00 0.14 0.00 0.83 0.00 0.95 0.00 0.45 0.03  0.81
#> W 0.36 0.04 0.42 0.03 0.94 0.00 0.32 0.37 0.12  0.00
#> Y 0.00 0.00 0.00 0.00 0.00 0.00 0.02 0.00 0.01  0.01
```

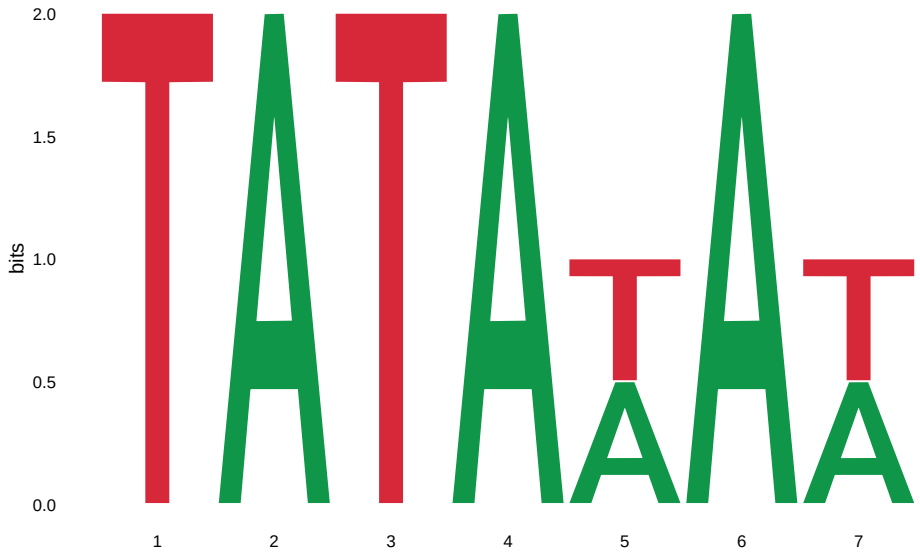
# 6 Motif visualization

## 6.1 Motif logos

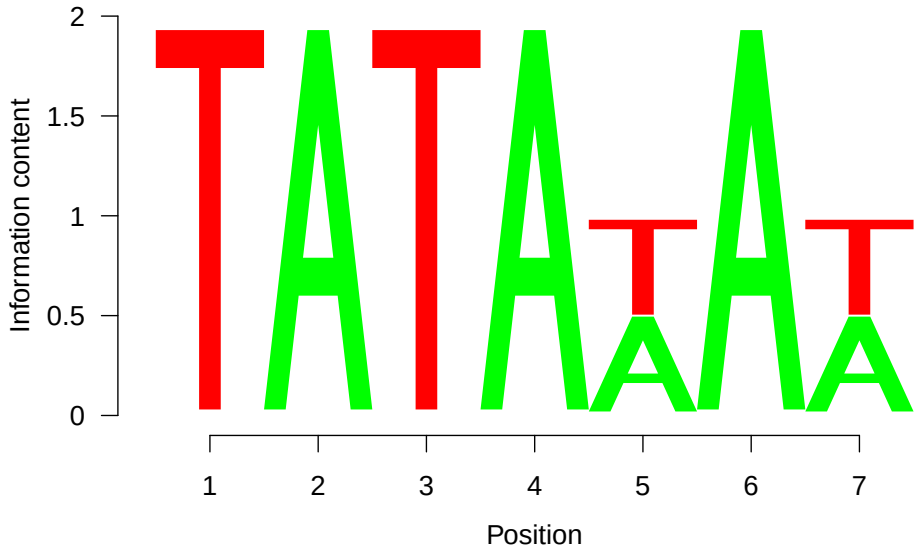
There are several packages which offer motif visualization capabilities, such as *seqLogo*, *Logolas*, *motifStack*, and *ggseqlogo*. The *universalmotif* package has chosen *ggseqlogo* as the default implementation, and used to drive the *universalmotif* package function `view_motifs()`. Here I will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

```
library(universalmotif)
data(examplemotif)

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

**Motif manipulation**

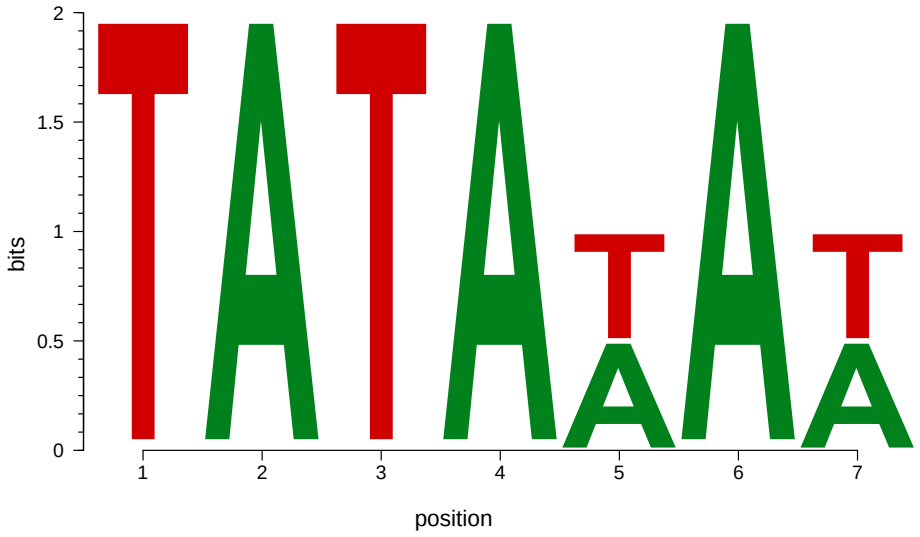



```
## For all the following examples, simply passing the functions a PPM is
## sufficient
motif <- convert_type(examplemotif, "PPM")
## Only need the matrix itself
motif <- motif["motif"]

## seqLogo:
seqLogo::seqLogo(motif)
```

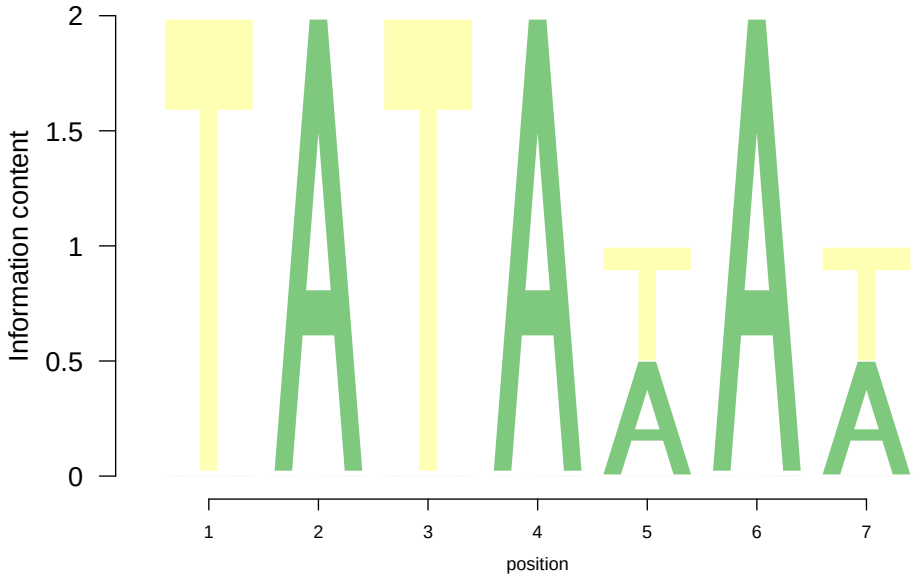



```
## motifStack:
motifStack::plotMotifLogo(motif)
```

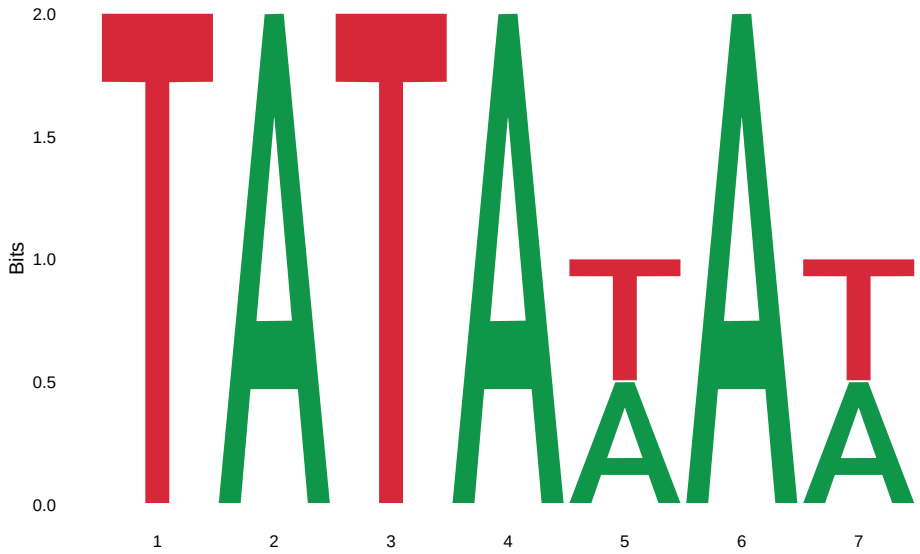
**Motif manipulation**




```
## 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
#> using a background with equal probability for all symbols
```




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

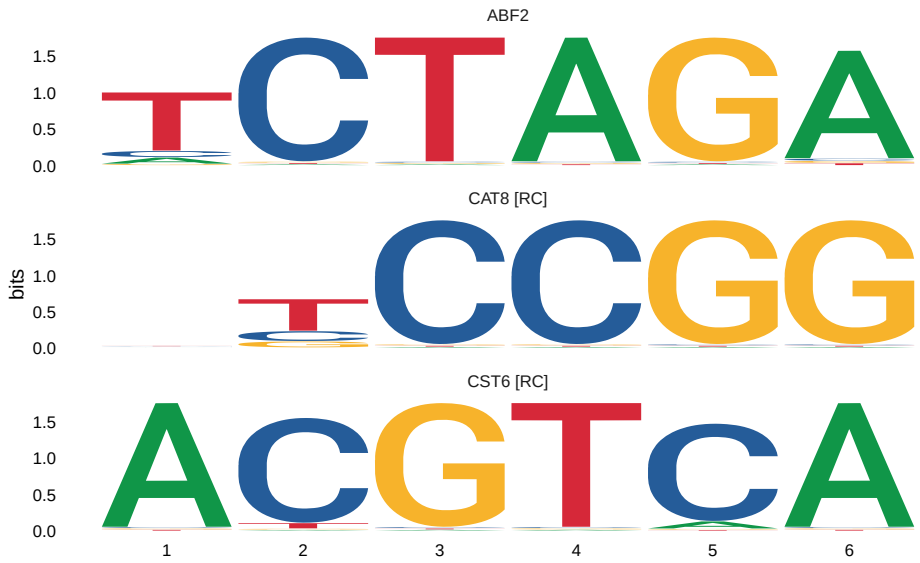



The *Logolas* and *ggseqlogo* offer many additional options for logo customization, including custom alphabets as well as manually determining the heights of each letter, via the *grid* and *ggplot2* packages respectively.

## 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 Miscellaneous motif utilities

A number of convenience functions are included for manipulating motifs.

## 7.1 `filter_motifs()`

Filter a list of motifs, using the `universalmotif` slots.

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

## 7.2 `sample_sites()`

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

```
library(universalmotif)
data(examplemotif)

sample_sites(examplemotif)
#>   A DNAStringSet instance of length 100
#>       width seq
```

```
#>   [1]     7 TATATAA
#>   [2]     7 TATATAA
#>   [3]     7 TATATAA
#>   [4]     7 TATAAAT
#>   [5]     7 TATATAA
#>   ...   ... ...
#>  [96]     7 TATATAT
#>  [97]     7 TATAAAA
#>  [98]     7 TATATAA
#>  [99]     7 TATAAAT
#> [100]     7 TATATAA
```

## 7.3 `shuffle_motifs()`

Shuffle a set of 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     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]    TCCCGA      DNA     +- 9.034091
#> 2  CAT8 [shuffled]    ABCMGA      DNA     +- 6.034019
#> 3  CST6 [shuffled]    CTGTAG      DNA     +- 8.559627
#> 4 ECM23 [shuffled]     GAACA      DNA     +- 7.493224
#> 5  EDS1 [shuffled]   ASCGAGY      DNA     +- 9.236069
#> 6  FKH2 [shuffled]   HTAGCKG      DNA     +- 8.339249
```

# Session info

```
#> R version 3.6.0 (2019-04-26)
#> Platform: x86_64-w64-mingw32/x64 (64-bit)
#> Running under: Windows Server 2012 R2 x64 (build 9600)
#>
#> Matrix products: default
```

```
#>
#> locale:
#> [1] LC_COLLATE=C
#> [2] LC_CTYPE=English_United States.1252
#> [3] LC_MONETARY=English_United States.1252
#> [4] LC_NUMERIC=C
#> [5] LC_TIME=English_United States.1252
#>
#> attached base packages:
#> [1] stats4    parallel  stats     graphics  grDevices utils     datasets
#> [8] methods   base
#>
#> other attached packages:
#>  [1] TFBSTools_1.22.0     ggplot2_3.1.1        ggtree_1.16.0
#>  [4] MotifDb_1.26.0       Biostrings_2.52.0    XVector_0.24.0
#>  [7] IRanges_2.18.0       S4Vectors_0.22.0     BiocGenerics_0.30.0
#> [10] universalmotif_1.2.0 BiocStyle_2.12.0
#>
#> loaded via a namespace (and not attached):
#>   [1] VGAM_1.1-1                  colorspace_1.4-1
#>   [3] grImport2_0.1-4             GenomicRanges_1.36.0
#>   [5] base64enc_0.1-3             rGADEM_2.32.0
#>   [7] bit64_0.9-7                 AnnotationDbi_1.46.0
#>   [9] splines_3.6.0               R.methodsS3_1.7.1
#>  [11] motifStack_1.28.0           knitr_1.22
#>  [13] ade4_1.7-13                 jsonlite_1.6
#>  [15] splitstackshape_1.4.8       Rsamtools_2.0.0
#>  [17] seqLogo_1.50.0              gridBase_0.4-7
#>  [19] annotate_1.62.0             GO.db_3.8.2
#>  [21] png_0.1-7                   R.oo_1.22.0
#>  [23] BiocManager_1.30.4          readr_1.3.1
#>  [25] compiler_3.6.0              httr_1.4.0
#>  [27] rvcheck_0.1.3               assertthat_0.2.1
#>  [29] Matrix_1.2-17               lazyeval_0.2.2
#>  [31] htmltools_0.3.6             tools_3.6.0
#>  [33] gtable_0.3.0                glue_1.3.1
#>  [35] TFMPvalue_0.0.8             GenomeInfoDbData_1.2.1
#>  [37] reshape2_1.4.3              dplyr_0.8.0.1
#>  [39] tinytex_0.12                Rcpp_1.0.1
#>  [41] Biobase_2.44.0              Logolas_1.8.0
#>  [43] ape_5.3                     nlme_3.1-139
#>  [45] rtracklayer_1.44.0          ggseqlogo_0.1
#>  [47] gbRd_0.4-11                 xfun_0.6
#>  [49] CNEr_1.20.0                 stringr_1.4.0
#>  [51] ps_1.3.0                    poweRlaw_0.70.2
#>  [53] gtools_3.8.1                XML_3.98-1.19
#>  [55] zlibbioc_1.30.0             MASS_7.3-51.4
#>  [57] scales_1.0.0                BSgenome_1.52.0
#>  [59] hms_0.4.2                   SummarizedExperiment_1.14.0
#>  [61] RColorBrewer_1.1-2          yaml_2.2.0
#>  [63] memoise_1.1.0               MotIV_1.40.0
```

```
#>  [65] SQUAREM_2017.10-1           stringi_1.4.3
#>  [67] RSQLite_2.1.1               highr_0.8
#>  [69] tidytree_0.2.4              caTools_1.17.1.2
#>  [71] BiocParallel_1.18.0         bibtex_0.4.2
#>  [73] GenomeInfoDb_1.20.0         Rdpack_0.11-0
#>  [75] rlang_0.3.4                 pkgconfig_2.0.2
#>  [77] matrixStats_0.54.0          bitops_1.0-6
#>  [79] evaluate_0.13               lattice_0.20-38
#>  [81] purrr_0.3.2                 htmlwidgets_1.3
#>  [83] GenomicAlignments_1.20.0    treeio_1.8.0
#>  [85] labeling_0.3                bit_1.1-14
#>  [87] processx_3.3.0              tidyselect_0.2.5
#>  [89] plyr_1.8.4                  magrittr_1.5
#>  [91] bookdown_0.9                R6_2.4.0
#>  [93] DelayedArray_0.10.0         DBI_1.0.0
#>  [95] pillar_1.3.1                withr_2.1.2
#>  [97] KEGGREST_1.24.0             RCurl_1.95-4.12
#>  [99] tibble_2.1.1                crayon_1.3.4
#> [101] rmarkdown_1.12              jpeg_0.1-8
#> [103] grid_3.6.0                  data.table_1.12.2
#> [105] blob_1.1.1                  digest_0.6.18
#> [107] xtable_1.8-4                tidyr_0.8.3
#> [109] R.utils_2.8.0               munsell_0.5.0
#> [111] DirichletMultinomial_1.26.0
```

# 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.

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.

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.