TF-COMB

Release 0.1.0

Loosolab

Aug 11, 2023

CONTENTS

1 Contents of this documentation **3**
- 1.1 Install . . . 3
- 1.2 Examples . . . 3
- 1.3 Methods . . . 72
- 1.4 API reference . . . 76
- 1.5 Changelog . . . 112

Python Module Index **115**

Index **117**

TF-COMB stands for "**T**ranscription **F**actor **C**o-**O**ccurrence using **M**arket **B**asket analysis" and is a python module for identifying co-occurring TFs in regulatory regions. Additionally, TF-COMB applies various downstream methods such as distance, orientation and network analysis.

The source code is located at: https://github.com/loosolab/TF-COMB

CHAPTER

ONE

CONTENTS OF THIS DOCUMENTATION

1.1 Install

Install from the github repository using:

```
$ pip install git+git://github.com/loosolab/TF-COMB
```

1.2 Examples

The usage of TF-comb is split into three main parts:

1. Setting up the Transcription Factor Binding Sites (TFBS) to use in the analysis
2. Counting co-occurrences and performing market basket analysis
3. Visualization of results and downstream analysis such as network analysis

List of example notebooks:

1.2.1 ChIP-seq analysis

This notebook shows how to analyze the TF-co-occurrences of predefined TFBS e.g. from ChIP-seq peaks. The data used here is obtained from the ENCODE project (https://pubmed.ncbi.nlm.nih.gov/29126249/) and consists of all TF ChIP-seq experiments for the celltype GM12878, and subset to chr4 in the human genome.

Setup a CombObj

First, we load the tfcomb package and set up an empty CombObj:

[1]:
```
from tfcomb import CombObj
C = CombObj()
```

[2]:
```
C
```

[2]:
```
<CombObj>
```

Read TFBS from .bed-file

Next, we are going to fill the CombObj 'C' with binding sites from GM12878 ChIP-seq experiments:

```
[3]: C.TFBS_from_bed("../data/GM12878_hg38_chr4_TF_chipseq.bed")
```

```
INFO: Reading sites from '../data/GM12878_hg38_chr4_TF_chipseq.bed'...
INFO: Processing sites
INFO: Read 112109 sites (151 unique names)
```

Now, the CombObj contains the .TFBS variable holding all TFBS to use for analysis:

```
[4]: C.TFBS[:10]
```

```
[4]: [chr4   11875   11876   ZBTB33  1000    .,
 chr4   116639  116640  JUNB    974     .,
 chr4   116678  116679  RUNX3   1000    .,
 chr4   121620  121621  RUNX3   1000    .,
 chr4   124050  124051  ZNF217  948     .,
 chr4   124052  124053  SMARCA5 802     .,
 chr4   124169  124170  NR2F1   634     .,
 chr4   124289  124290  CBX5    906     .,
 chr4   124363  124364  E4F1    1000    .,
 chr4   124365  124366  PKNOX1  1000    .]
```

The CombObj will now reflect that TFBS were added:

```
[5]: C
```

```
[5]: <CombObj: 112109 TFBS (151 unique names)>
```

Perform market basket analysis

Next, the function .market_basket() is used to perform the co-occurrence analysis of the sites just added to .TFBS:

```
[6]: C.market_basket()
```

```
Internal counts for 'TF_counts' were not set. Please run .count_within() to obtain TF-TF␣
↪co-occurrence counts.
WARNING: No counts found in <CombObj>. Running <CombObj>.count_within() with standard␣
↪parameters.
INFO: Setting up binding sites for counting
INFO: Counting co-occurrences within sites
INFO: Counting co-occurrence within background
INFO: Running with multiprocessing threads == 1. To change this, give 'threads' in the␣
↪parameter of the function.
INFO: Progress: 10%
INFO: Progress: 20%
INFO: Progress: 30%
INFO: Progress: 40%
INFO: Progress: 50%
INFO: Progress: 60%
INFO: Progress: 70%
INFO: Progress: 80%
INFO: Progress: 90%
```

(continues on next page)

(continued from previous page)

```
INFO: Done finding co-occurrences! Run .market_basket() to estimate significant pairs
INFO: Market basket analysis is done! Results are found in <CombObj>.rules
```

As is shown in the info messages, this also runs the .count_within() function of 'C' (if no counts were found yet). If you want to set specific parameters for count_within, you can split these calculations such as seen here:

```
C.count_within(max_distance=200)
C.market_basket()
```

In any case, running .market_basket() will fill out the .rules variable of the CombObj:

[7]:
```
C.rules.head()
```

[7]:
```
              TF1    TF2  TF1_TF2_count  TF1_count  TF2_count    cosine  \
CTCF-RAD21   CTCF  RAD21           1751       2432       2241  0.750038
RAD21-CTCF  RAD21   CTCF           1751       2241       2432  0.750038
RAD21-SMC3  RAD21   SMC3           1376       2241       1638  0.718192
SMC3-RAD21   SMC3  RAD21           1376       1638       2241  0.718192
CTCF-SMC3    CTCF   SMC3           1361       2432       1638  0.681898

               zscore
CTCF-RAD21  18.643056
RAD21-CTCF  18.643056
RAD21-SMC3  20.314026
SMC3-RAD21  20.314026
CTCF-SMC3   20.245177
```

Printing the CombObj again will also tell you how many rules were found in the market basket analysis:

[8]:
```
C
```

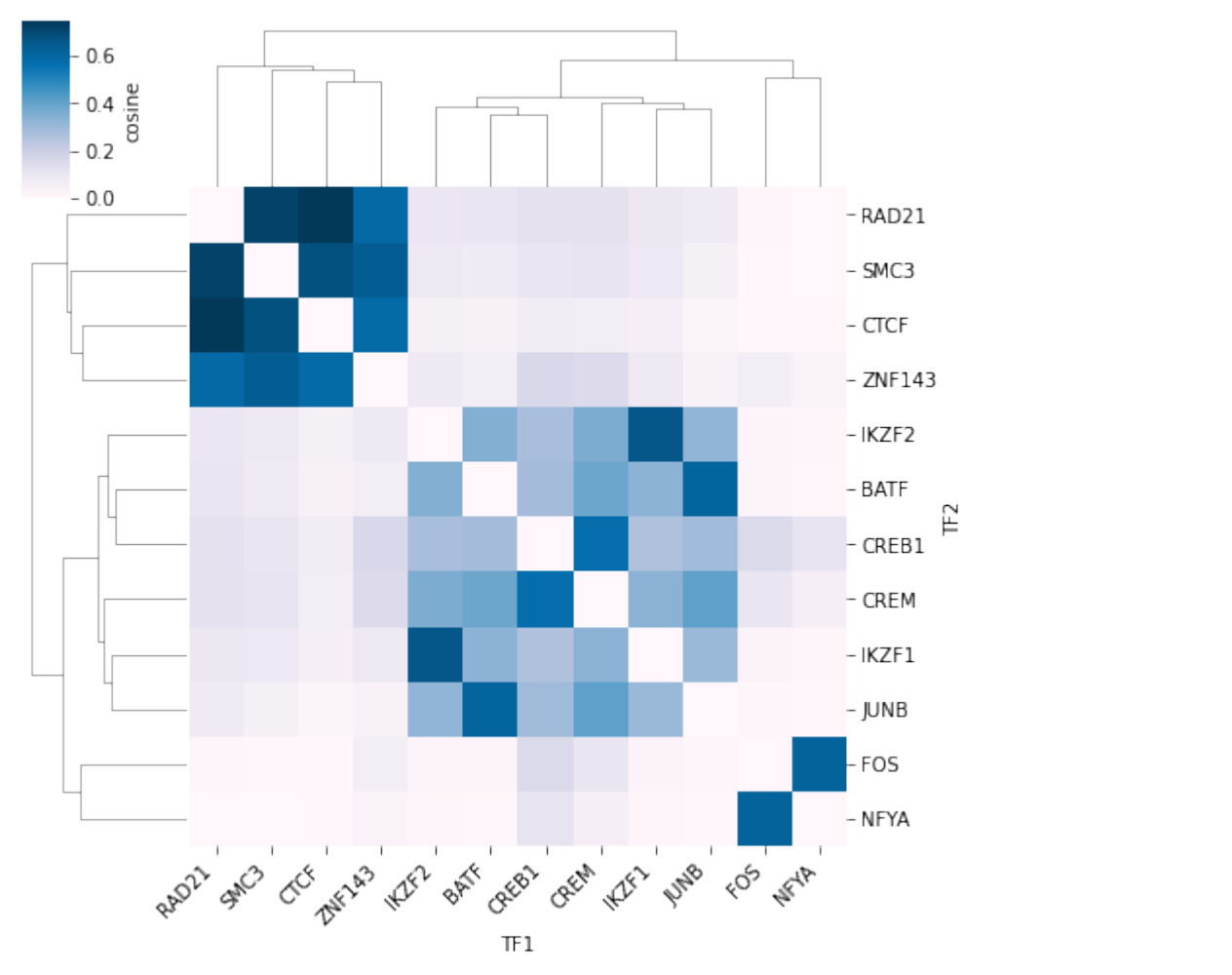
[8]:
```
<CombObj: 112109 TFBS (151 unique names) | Market basket analysis: 21284 rules>
```

Visualize results

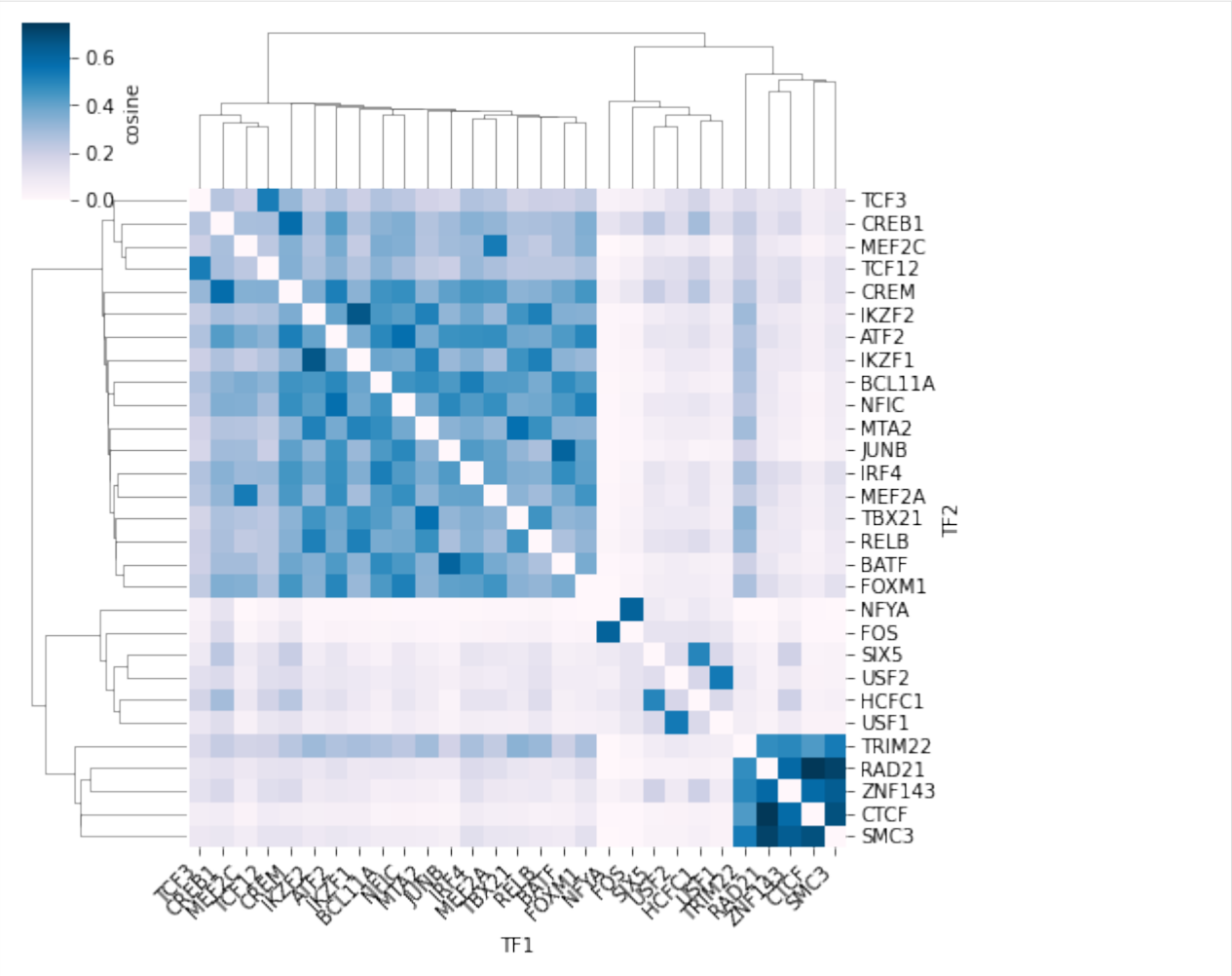
The CombObj 'C' contains a number of different visualizations for the identified TF-TF co-occurrence pairs. The default measure plotted is 'cosine', but many of the options of the plots can be changed as seen in the examples below:

Heatmap

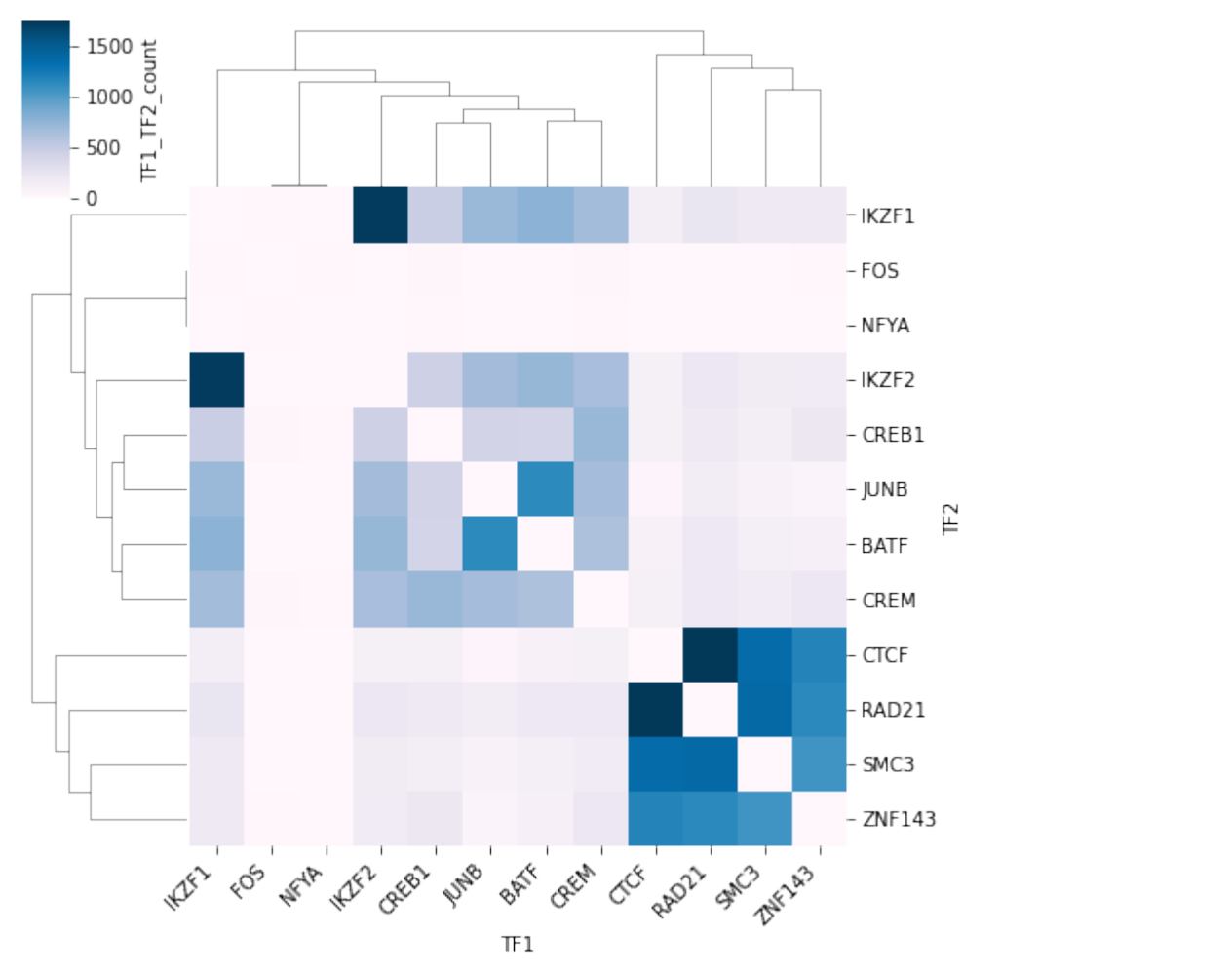
[9]:
```
_ = C.plot_heatmap()
```



[10]:
```
#Showing more rules
_ = C.plot_heatmap(n_rules=50)
```

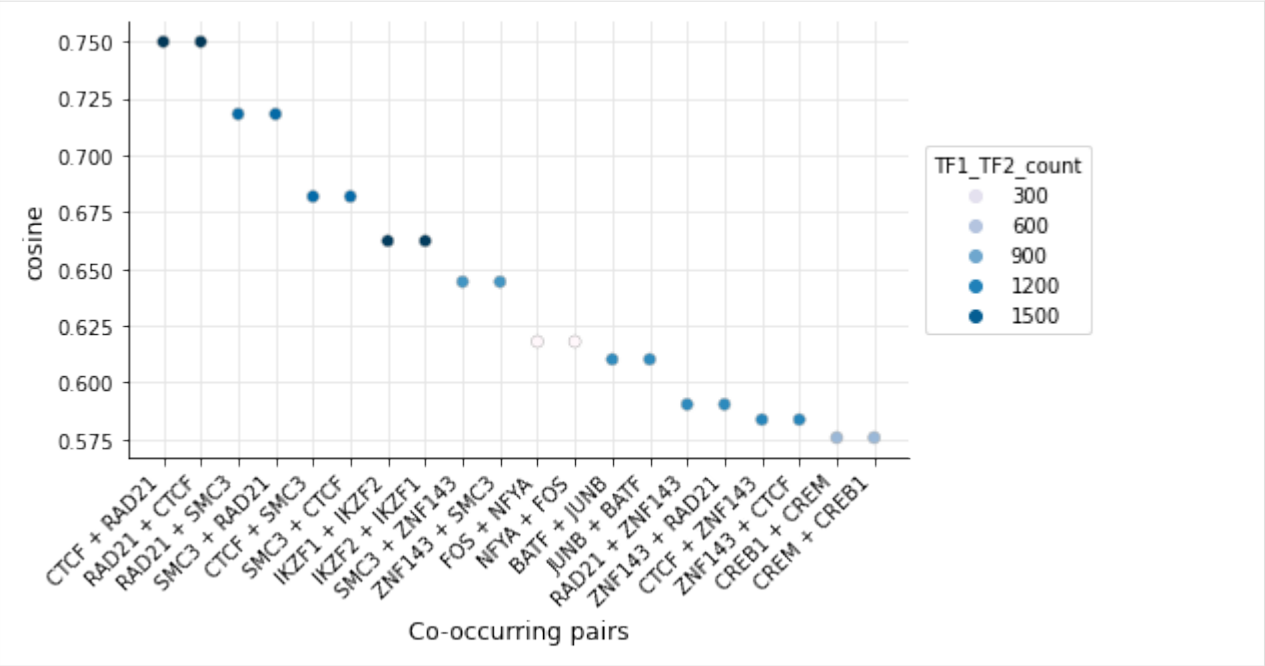


[11]:
```
#Coloring the heatmap by TF1_TF2_count
_ = C.plot_heatmap(color_by="TF1_TF2_count")
```

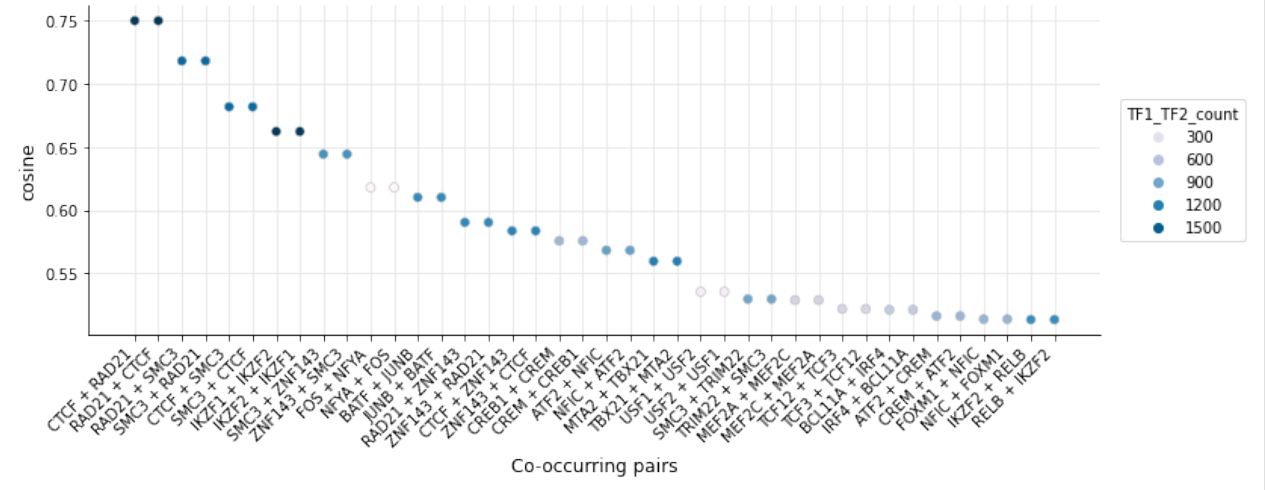


Bubble plot

[12]:
```
_ = C.plot_bubble()
```



[13]:
```
#Show more rules
_ = C.plot_bubble(n_rules=40, figsize=(12,4))
```



Handling duplicate rules

As seen in the bubble plot above, the rules are duplicated due to the cosine measure, which scores TF1-TF2 the same way as TF2-TF1. In order to simplify the results, you can use .simplify_rules() to remove the duplicates from the .rules:

[14]:
```
C.simplify_rules()
```

[15]:
```
C.rules
```

[15]:

	TF1	TF2	TF1_TF2_count	TF1_count	TF2_count	cosine	\
CTCF-RAD21	CTCF	RAD21	1751	2432	2241	0.750038	

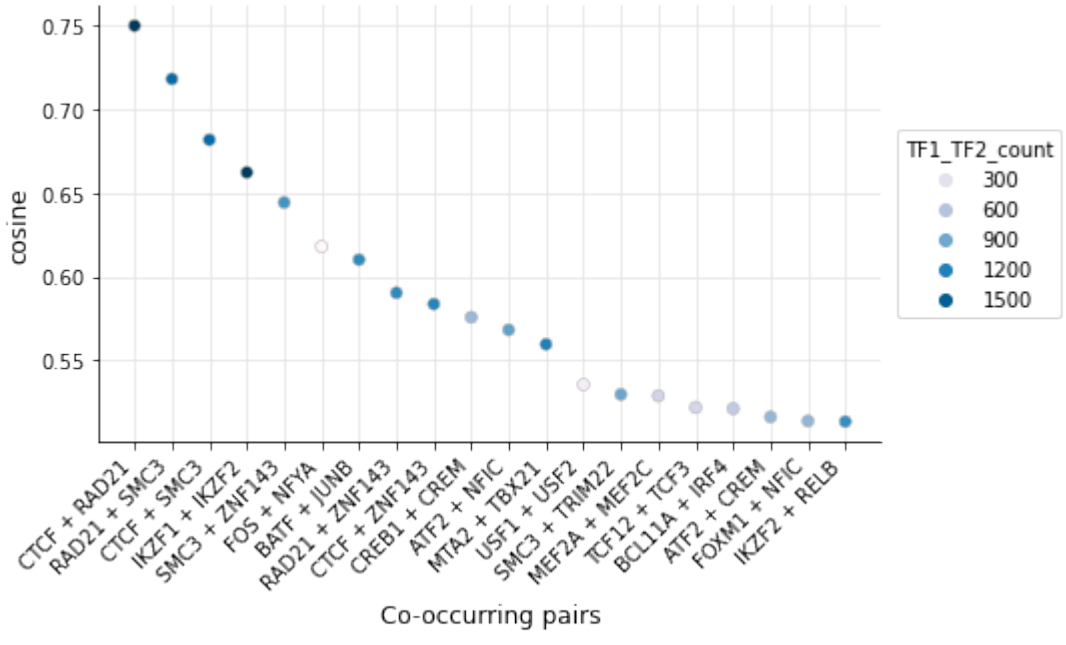
(continues on next page)

(continued from previous page)

```
RAD21-SMC3    RAD21    SMC3           1376       2241       1638  0.718192
CTCF-SMC3      CTCF    SMC3           1361       2432       1638  0.681898
IKZF1-IKZF2  IKZF1   IKZF2           1726       2922       2324  0.662343
SMC3-ZNF143   SMC3  ZNF143           1060       1638       1652  0.644383
...            ...     ...            ...        ...        ...       ...
HDAC6-JUNB   HDAC6    JUNB              1        172       1866  0.001765
RELB-SUZ12    RELB   SUZ12              1       2011        172  0.001700
ATF7-SUZ12    ATF7   SUZ12              1       2061        172  0.001680
RAD21-SUZ12  RAD21   SUZ12              1       2241        172  0.001611
IKZF2-SUZ12  IKZF2   SUZ12              1       2324        172  0.001582

                zscore
CTCF-RAD21   18.643056
RAD21-SMC3   20.314026
CTCF-SMC3    20.245177
IKZF1-IKZF2  11.215960
SMC3-ZNF143  21.838431
...                ...
HDAC6-JUNB   -6.905016
RELB-SUZ12   -6.793962
ATF7-SUZ12   -6.119294
RAD21-SUZ12  -7.500216
IKZF2-SUZ12  -6.649424

[10642 rows x 7 columns]
```

Now, you can again use plot_bubble() to visualize the highest-scoring co-occurring pairs:

[16]:
```
_ = C.plot_bubble()
```



Save CombObj to a pickle object

We can now save the CombObj to a pickle object to be used in other analysis:

[17]:
```
C.to_pickle("../data/GM12878.pkl")
```

1.2.2 TFBS from motifs

In this notebook, we show how a CombObj can be set up using motifs and a set of target regions.

Setup a CombObj

[1]:
```
from tfcomb import CombObj
C = CombObj()
```

Search for TFBS within peak regions

We are using a subset of ATAC-seq peaks from GM12878 as the target regions. The .TFBS variable can then be filled as seen here:

[2]:
```
C.TFBS_from_motifs(regions="../data/GM12878_hg38_chr4_ATAC_peaks.bed",
                   motifs="../data/HOCOMOCOv11_HUMAN_motifs.txt",
                   genome="../data/hg38_chr4.fa.gz",
                   threads=4)
```

```
INFO: Scanning for TFBS with 4 thread(s)...
INFO: Progress: 11%
INFO: Progress: 20%
INFO: Progress: 30%
INFO: Progress: 41%
INFO: Progress: 50%
INFO: Progress: 60%
INFO: Progress: 70%
INFO: Progress: 81%
INFO: Progress: 90%
INFO: Finished!
INFO: Processing scanned TFBS
INFO: Identified 165810 TFBS (401 unique names) within given regions
```

[3]:
```
C.TFBS[:10]
```

[3]:
```
[chr4   11750   11772   THAP1   10.09272        +,
 chr4   11806   11823   CTCFL   12.12093        -,
 chr4   11806   11825   CTCF    13.68703        -,
 chr4   11864   11881   CTCFL   12.23068        -,
 chr4   11864   11883   CTCF    12.56734        -,
 chr4   11938   11945   MYF6    8.75866 -,
 chr4   11997   12010   MYOG    9.91354 -,
 chr4   11999   12009   TCF12   9.23823 +,
 chr4   11999   12009   TCF4    9.99356 +,
 chr4   12000   12007   MYF6    8.75866 -]
```

Perform market basket analysis

As shown in previous notebooks, we can now perform the market basket analysis on the sites within .TFBS:

[4]:
```
C.market_basket(threads=10)
```

```
Internal counts for 'TF_counts' were not set. Please run .count_within() to obtain TF-TF␣
↪co-occurrence counts.
WARNING: No counts found in <CombObj>. Running <CombObj>.count_within() with standard␣
↪parameters.
INFO: Setting up binding sites for counting
INFO: Counting co-occurrences within sites
INFO: Counting co-occurrence within background
INFO: Progress: 14%
INFO: Progress: 28%
INFO: Progress: 40%
INFO: Progress: 52%
INFO: Progress: 64%
INFO: Progress: 76%
INFO: Progress: 88%
INFO: Finished!
INFO: Done finding co-occurrences! Run .market_basket() to estimate significant pairs
INFO: Market basket analysis is done! Results are found in <CombObj>.rules
```

[5]:
```
C
```

[5]:
```
<CombObj: 165810 TFBS (401 unique names) | Market basket analysis: 126997 rules>
```

[6]:
```
C.rules.head()
```

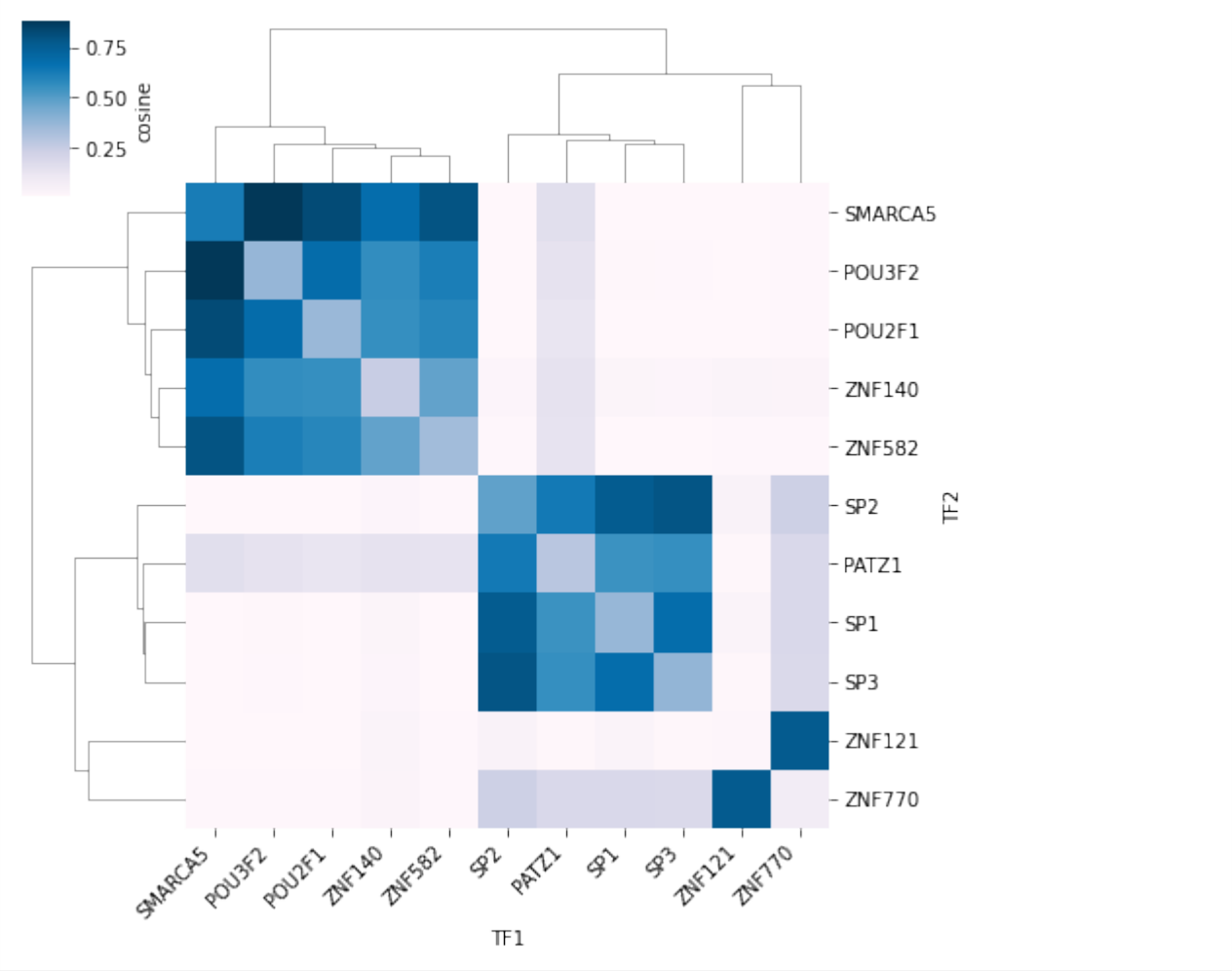
[6]:

	TF1	TF2	TF1_TF2_count	TF1_count	TF2_count	cosine	zscore
POU3F2-SMARCA5	POU3F2	SMARCA5	239	302	241	0.885902	129.586528
SMARCA5-POU3F2	SMARCA5	POU3F2	239	241	302	0.885902	129.586528
POU2F1-SMARCA5	POU2F1	SMARCA5	263	426	241	0.820810	135.355691
SMARCA5-POU2F1	SMARCA5	POU2F1	263	241	426	0.820810	135.355691
SMARCA5-ZNF582	SMARCA5	ZNF582	172	241	195	0.793419	117.370387

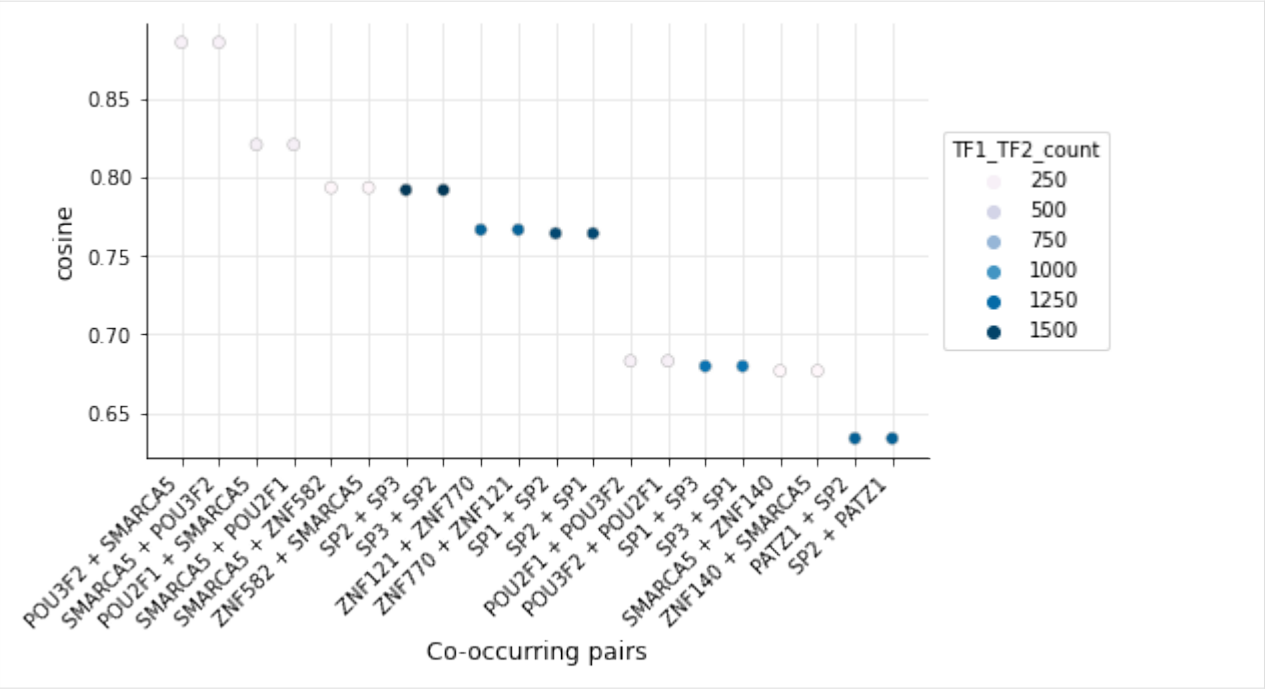
Visualize rules

With the market basket analysis done, we have the option to visualize the identified co-occurring TFs:

[7]:
```
_ = C.plot_heatmap()
```



[8]:
```
_ = C.plot_bubble()
```



The effect of count parameters

In this example, the first rules contain more *TF1_TF2_counts* than the individual counts of TF1 and TF2:

[9]:
```
C.rules.head(1)
```

[9]:
```
                    TF1      TF2  TF1_TF2_count  TF1_count  TF2_count  \
POU3F2-SMARCA5  POU3F2  SMARCA5            239        302        241

                   cosine      zscore
POU3F2-SMARCA5  0.885902  129.586528
```

How can that be? If there are multiple combinations of TF1-TF2 within the same window, these combinations can add up to more than the number of individual TF positions. This effect can be controlled by setting `binarize=True` in *count_within*. This will ensure that each co-occurrence is only counted once per window:

[10]:
```
C.count_within(binarize=True, threads=8)
C.market_basket()
```

```
INFO: Counting co-occurrences within sites
INFO: Counting co-occurrence within background
INFO: Progress: 16%
INFO: Progress: 28%
INFO: Progress: 32%
INFO: Progress: 44%
INFO: Progress: 54%
INFO: Progress: 64%
INFO: Progress: 72%
INFO: Progress: 80%
INFO: Progress: 92%
INFO: Finished!
```

(continues on next page)

(continued from previous page)

```
INFO: Done finding co-occurrences! Run .market_basket() to estimate significant pairs
INFO: Market basket analysis is done! Results are found in <CombObj>.rules
```

[11]:
```
C.rules.head()
```

[11]:
```
                 TF1     TF2  TF1_TF2_count  TF1_count  TF2_count    cosine  \
ZNF121-ZNF770  ZNF121  ZNF770           1249       1242       2394  0.724335
ZNF770-ZNF121  ZNF770  ZNF121           1249       2394       1242  0.724335
PAX5-ZNF770      PAX5  ZNF770            863        968       2394  0.566906
ZNF770-PAX5    ZNF770    PAX5            863       2394        968  0.566906
SP1-SP2           SP1     SP2            570       1718       2150  0.296581

                  zscore
ZNF121-ZNF770  103.405506
ZNF770-ZNF121  103.405506
PAX5-ZNF770     82.606752
ZNF770-PAX5     82.606752
SP1-SP2         35.851669
```

It is also possible to play around with the maximum overlap allowed. The default is no overlap allowed, but setting *max_overlap* to 1 (all overlaps allowed), highlights some of the TFs which are highly overlapping:

[12]:
```
C.count_within(binarize=True, max_overlap=1, threads=8)
C.market_basket()
```

```
INFO: Counting co-occurrences within sites
INFO: Counting co-occurrence within background
INFO: Progress: 12%
INFO: Progress: 24%
INFO: Progress: 32%
INFO: Progress: 46%
INFO: Progress: 58%
INFO: Progress: 64%
INFO: Progress: 78%
INFO: Progress: 80%
INFO: Progress: 90%
INFO: Finished!
INFO: Done finding co-occurrences! Run .market_basket() to estimate significant pairs
INFO: Market basket analysis is done! Results are found in <CombObj>.rules
```

[13]:
```
C.rules.head()
```

[13]:
```
               TF1    TF2  TF1_TF2_count  TF1_count  TF2_count    cosine  \
ARNT-EPAS1    ARNT  EPAS1             18         18         18  1.000000
EPAS1-ARNT   EPAS1   ARNT             18         18         18  1.000000
NR4A1-NR4A2  NR4A1  NR4A2            141        141        141  1.000000
NR4A2-NR4A1  NR4A2  NR4A1            141        141        141  1.000000
MITF-TFE3     MITF   TFE3             82         92         87  0.916559

                 zscore
ARNT-EPAS1   128.428571
EPAS1-ARNT   128.428571
NR4A1-NR4A2  127.000000
```

(continues on next page)

(continued from previous page)

```
NR4A2-NR4A1  127.000000
MITF-TFE3    116.772609
```

1.2.3 Selection of rules

With increasing number of transcription factors, co-occurrence analysis generally yields thousand of rules, which makes interpretation difficult. In this notebook, we will show how some different methods to select rules of interest from the original analysis. We will use the same data as within the 'chipseq analysis' notebook:

[1]:
```
import tfcomb.objects
C = tfcomb.objects.CombObj().from_pickle("../data/GM12878.pkl")
```

[2]:
```
C
```

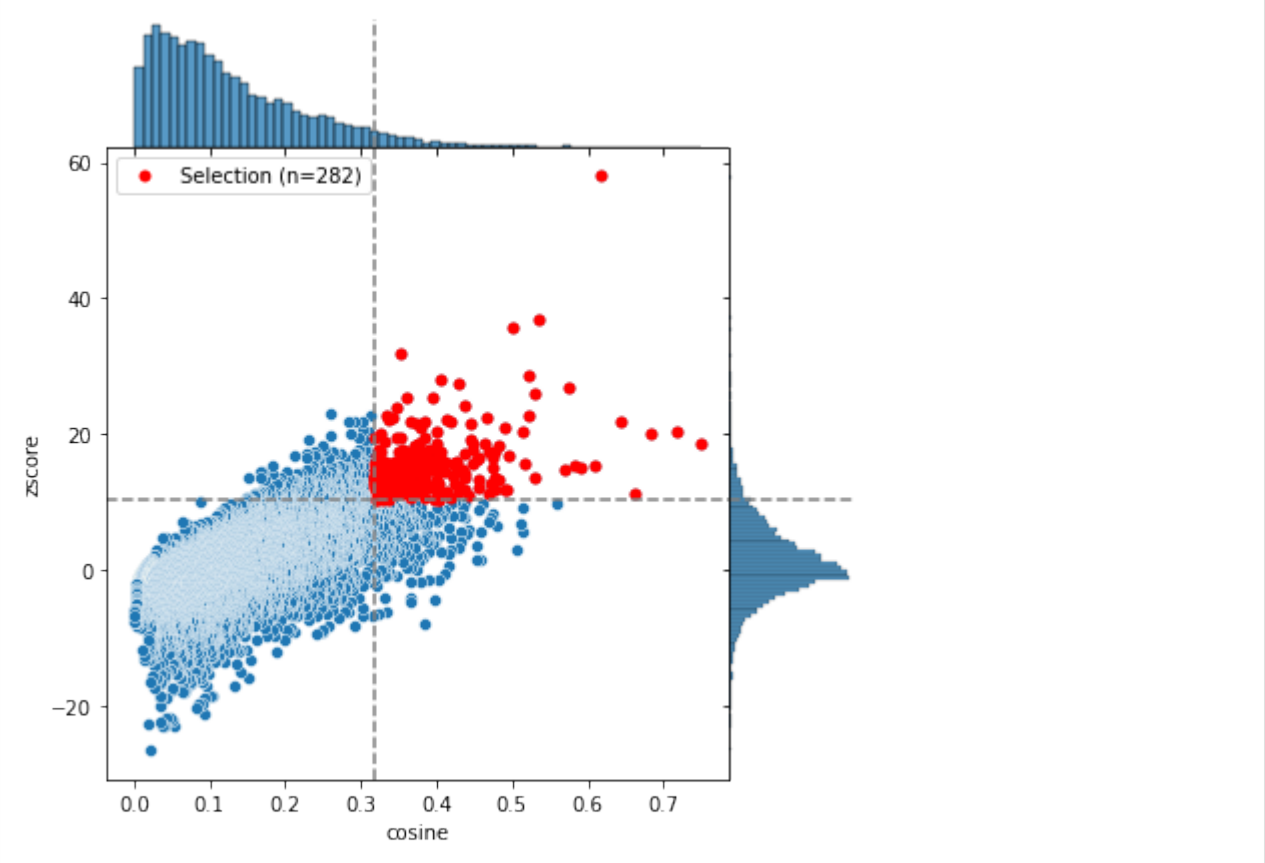
[2]:
```
<CombObj: 112109 TFBS (151 unique names) | Market basket analysis: 10642 rules>
```

Select rules using co-occurrence measures

In this example, we will use the function .select_significant_rules() to select rules based on two measures. Default for this function are the "zscore" and "cosine" measures.

[3]:
```
selected = C.select_significant_rules()
```

```
INFO: x_threshold is None; trying to calculate optimal threshold
INFO: y_threshold is None; trying to calculate optimal threshold
INFO: Creating subset of TFBS and rules using thresholds
```



The function returns an instance of a CombObj containing only the selected rules and TFBS for the TFs within .rules:

[4]:
```
selected
```

[4]:
```
<CombObj: 83705 TFBS (86 unique names) | Market basket analysis: 282 rules>
```

[5]:
```
selected.rules
```

[5]:
```
                TF1     TF2  TF1_TF2_count  TF1_count  TF2_count    cosine  \
CTCF-RAD21     CTCF   RAD21           1751       2432       2241  0.750038
RAD21-SMC3    RAD21    SMC3           1376       2241       1638  0.718192
CTCF-SMC3      CTCF    SMC3           1361       2432       1638  0.681898
IKZF1-IKZF2   IKZF1   IKZF2           1726       2922       2324  0.662343
SMC3-ZNF143    SMC3  ZNF143           1060       1638       1652  0.644383
...             ...     ...            ...        ...        ...       ...
GABPA-ZBED1   GABPA   ZBED1            211        666        659  0.318495
GABPA-MTA3    GABPA    MTA3            216        666        692  0.318173
CBFB-STAT5A    CBFB  STAT5A            195        829        454  0.317855
GABPA-NFATC1  GABPA  NFATC1            214        666        681  0.317763
ETS1-TAF1      ETS1    TAF1            166        424        644  0.317674

                 zscore
CTCF-RAD21    18.643056
RAD21-SMC3    20.314026
CTCF-SMC3     20.245177
IKZF1-IKZF2   11.215960
```

(continues on next page)

(continued from previous page)

```
SMC3-ZNF143   21.838431
...                ...
GABPA-ZBED1   14.915274
GABPA-MTA3    12.609150
CBFB-STAT5A   15.633269
GABPA-NFATC1  13.977084
ETS1-TAF1     13.702745

[282 rows x 7 columns]
```

We will save this object for use in other notebooks:

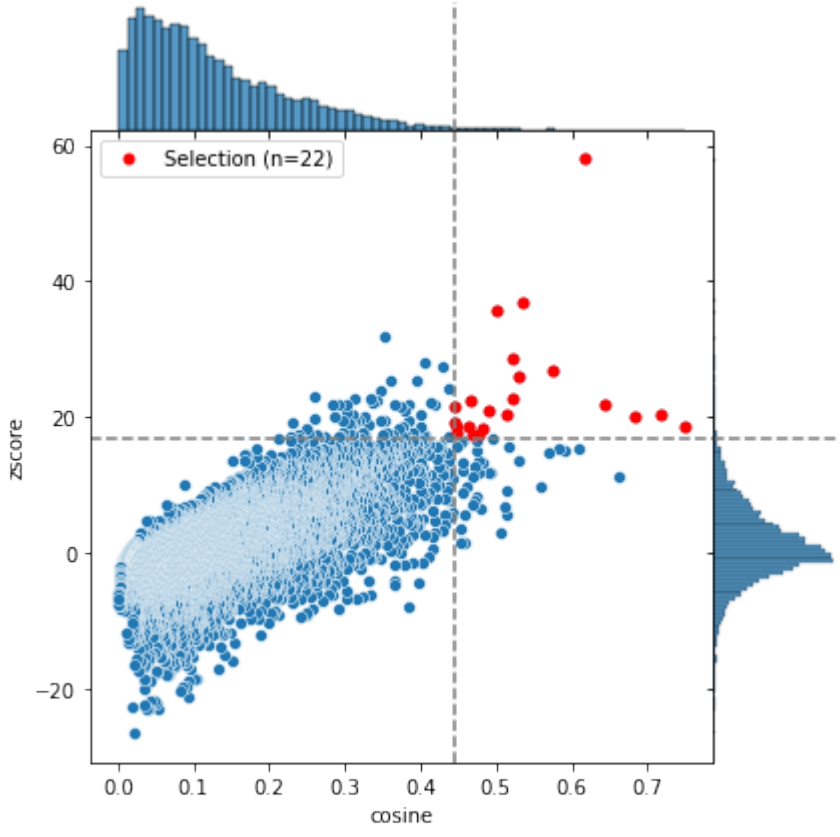
[6]:
```
selected.to_pickle("../data/GM12878_selected.pkl")
```

The strictness of the selection can be controlled via *x_threshold_percent* and *y_threshold_percent* as seen here:

[7]:
```
C.select_significant_rules(x_threshold_percent=0.01, y_threshold_percent=0.01)
```

```
INFO: x_threshold is None; trying to calculate optimal threshold
INFO: y_threshold is None; trying to calculate optimal threshold
INFO: Creating subset of TFBS and rules using thresholds
```

[7]:
```
<CombObj: 30229 TFBS (29 unique names) | Market basket analysis: 22 rules>
```



Select top rules

We can also use the function `.select_top_rules()` to select the first *n* rules from .rules:

[8]:
```
selected2 = C.select_top_rules(n=100)
```

[9]:
```
selected2.rules
```

[9]:
```
                TF1     TF2  TF1_TF2_count  TF1_count  TF2_count    cosine  \
CTCF-RAD21     CTCF   RAD21           1751       2432       2241  0.750038
RAD21-SMC3    RAD21    SMC3           1376       2241       1638  0.718192
CTCF-SMC3      CTCF    SMC3           1361       2432       1638  0.681898
IKZF1-IKZF2   IKZF1   IKZF2           1726       2922       2324  0.662343
SMC3-ZNF143    SMC3  ZNF143           1060       1638       1652  0.644383
...             ...     ...            ...        ...        ...       ...
IKZF1-NFATC3  IKZF1  NFATC3            770       2922       1167  0.416980
ATF7-IKZF2     ATF7   IKZF2            908       2061       2324  0.414886
DPF2-IKZF1     DPF2   IKZF1            963       1844       2922  0.414864
ATF2-EP300     ATF2   EP300            484       1484        922  0.413774
CHD2-SP1       CHD2     SP1            219        611        460  0.413090

                 zscore
CTCF-RAD21    18.643056
RAD21-SMC3    20.314026
CTCF-SMC3     20.245177
IKZF1-IKZF2   11.215960
SMC3-ZNF143   21.838431
...                 ...
IKZF1-NFATC3   6.271326
ATF7-IKZF2     2.376816
DPF2-IKZF1     0.597690
ATF2-EP300    11.601051
CHD2-SP1      22.192709

[100 rows x 7 columns]
```

Select rules based on a list of TF names

If you are only interested in the rules concerning a list of TFs, it is possible to use `.select_TF_rules` to subset the rules to a subset of TF names. Here, we create a subset of rules containing the TFs in TF_list:

[10]:
```
TF_list = ["ELK1", "CTCF", "ZNF143", "YY1"]
```

[11]:
```
selected3 = C.select_TF_rules(TF_list)
```

```
INFO: Selected 6 rules
INFO: Creating subset of object
```

[12]:
```
selected3
```

[12]:
```
<CombObj: 5686 TFBS (4 unique names) | Market basket analysis: 6 rules>
```

[13]:
```
selected3.rules
```

[13]:
```
              TF1     TF2  TF1_TF2_count  TF1_count  TF2_count    cosine  \
CTCF-ZNF143  CTCF  ZNF143           1170       2432       1652  0.583713
YY1-ZNF143    YY1  ZNF143            555       1365       1652  0.369591
CTCF-YY1     CTCF     YY1            481       2432       1365  0.263996
ELK1-YY1     ELK1     YY1             88        237       1365  0.154718
ELK1-ZNF143  ELK1  ZNF143             95        237       1652  0.151825
CTCF-ELK1    CTCF    ELK1             28       2432        237  0.036881

                zscore
CTCF-ZNF143  15.459923
YY1-ZNF143    6.797111
CTCF-YY1     -3.079479
ELK1-YY1      4.946988
ELK1-ZNF143   3.295755
CTCF-ELK1    -6.360275
```

It is also possible to specify that the rules should only be selected from matches to TF1 (by setting TF2=False):

[14]:
```
selected3 = C.select_TF_rules(TF_list, TF2=False)
```

```
INFO: Selected 257 rules
INFO: Creating subset of object
```

[15]:
```
selected3.rules
```

[15]:
```
              TF1     TF2  TF1_TF2_count  TF1_count  TF2_count    cosine  \
CTCF-RAD21   CTCF   RAD21           1751       2432       2241  0.750038
CTCF-SMC3    CTCF    SMC3           1361       2432       1638  0.681898
CTCF-ZNF143  CTCF  ZNF143           1170       2432       1652  0.583713
CTCF-TRIM22  CTCF  TRIM22            900       2432       1786  0.431837
YY1-ZNF143    YY1  ZNF143            555       1365       1652  0.369591
...           ...     ...            ...        ...        ...       ...
CTCF-SUZ12   CTCF   SUZ12              4       2432        172  0.006185
CTCF-PAX8    CTCF    PAX8              3       2432        165  0.004736
CTCF-NFYA    CTCF    NFYA              1       2432         21  0.004425
CTCF-ZZZ3    CTCF    ZZZ3              1       2432         32  0.003585
CTCF-FOS     CTCF     FOS              1       2432         45  0.003023

                zscore
CTCF-RAD21   18.643056
CTCF-SMC3    20.245177
CTCF-ZNF143  15.459923
CTCF-TRIM22   4.237106
YY1-ZNF143    6.797111
...                ...
CTCF-SUZ12   -6.526432
CTCF-PAX8    -7.317208
CTCF-NFYA    -2.072913
CTCF-ZZZ3    -2.896546
CTCF-FOS     -3.586322

[257 rows x 7 columns]
```

In case a TF is not found in rules, `.select_TF_rules` will write a warning and raise an error:

[16]:
```
TF_list = ["NOTFOUND"]
selected_warning = C.select_TF_rules(TF_list)
```

```
WARNING: 1/1 names in 'TF_list' were not found within 'TF1' names: ['NOTFOUND']
WARNING: 1/1 names in 'TF_list' were not found within 'TF2' names: ['NOTFOUND']
```

```
InputError: No rules could be selected - please adjust TF_list and/or TF1/TF2 parameters.
```

1.2.4 Network analysis

This notebook shows examples of how to perform network analysis on TF-COMB co-occurrence results.

Load CombObj containing rules

First, we will load the data as created in the "select rules"-notebook.

[1]:
```
import tfcomb
from tfcomb import CombObj
C = CombObj().from_pickle("../data/GM12878_selected.pkl")
```

[2]:
```
C
```

[2]:
```
<CombObj: 83705 TFBS (86 unique names) | Market basket analysis: 282 rules>
```

Visualize network

We can now use the CombObj to directly visualize the network. The first time the function is run, the .network attribute will be created within the CombObj:

[3]:
```
C.plot_network()
```

```
WARNING: The .network attribute is not set yet - running build_network().
INFO: Finished! The network is found within <CombObj>.network.
```

[3]:

You can also change the colors of the nodes and edges via `edge_cmap` and `node_cmap`:

[4]:
```
C.plot_network(edge_cmap="Blues", node_cmap="viridis")
```

[4]:

By default, the nodes are colored by count of binding sites, and edges are colored by the 'cosine' score. However, this is easily adjusted via the 'color__by' parameters:

[5]:
```
C.plot_network(color_node_by="TF1", color_edge_by="zscore")
```

[5]:

Different network layouts

It is possible to change the network layout via the "engine" parameter of plot_network:

[6]:
```
C.plot_network(engine="circo", legend_size=70)
```

[6]:

[7]:
```
C.plot_network(engine="fdp", size_node_by="TF1_count")
```

[7]:

[8]:
```
C.plot_network(engine="dot")
```

[8]:

[9]:
```
#The available layouts are:
import graphviz; graphviz.ENGINES
```

[9]:
```
{'circo', 'dot', 'fdp', 'neato', 'osage', 'patchwork', 'sfdp', 'twopi'}
```

Cluster network nodes

It is often of interest to partition individual TFs into groups of highly connected TFs (highly co-occurring). This is possible using the *cluster_network()* function. Below we show different settings for performing TF clustering based on co-occurring networks.

Using louvain clustering (default)

The default partition_network uses louvain clustering of the python-louvain package (https://github.com/taynaud/python-louvain):

[10]:
```
C.cluster_network()
```
```
INFO: Added 'cluster' attribute to the network attributes
```

The partition was added to the network attributes as well as to the internal .TF_table of the CombObj:

[11]:
```
C.TF_table
```

[11]:

	TF1	TF1_count	TF2	TF2_count	cluster
CTCF	CTCF	2432	CTCF	2432	1
RAD21	RAD21	2241	RAD21	2241	1
SMC3	SMC3	1638	SMC3	1638	1
IKZF1	IKZF1	2922	IKZF1	2922	2
IKZF2	IKZF2	2324	IKZF2	2324	2
...	...	...	...	...	...
ELK1	ELK1	237	ELK1	237	9
CEBPB	CEBPB	273	CEBPB	273	6
RCOR1	RCOR1	415	RCOR1	415	6
TCF7	TCF7	512	TCF7	512	6
CBX5	CBX5	887	CBX5	887	6

[86 rows x 5 columns]

This enables plotting of the partition using plot_network:

[12]:
```
C.plot_network(color_node_by="cluster")
```
[12]:

Louvain clustering with weights

Louvain clustering can also work with weighted edges, as seen here for cosine:

[13]:
```
C.cluster_network(weight="cosine")
```
```
INFO: Added 'cluster' attribute to the network attributes
```

[14]:
```
C.plot_network(color_node_by="cluster")
```
[14]:

[15]:
```
#It is also possible to plot the network without label by overwriting labels via node_
↪attributes:
C.plot_network(color_node_by="cluster", legend_size=0, node_attributes={"label": ""})
```
[15]:

Using block model

The third option is using the graph-tool 'minimize.minimize_blockmodel_dl'-function (https://graph-tool.skewed.de/static/doc/inference.html#graph_tool.inference.minimize.minimize_blockmodel_dl):

[16]:
```
C.cluster_network(method="blockmodel")
```

[17]:
```
C.plot_network(color_node_by="cluster")
```
[17]:

1.2.5 Orientation analysis

This is an example of a orientation analysis using .TFBS from motif sites.

Create CombObj and fill it with .TFBS from motif scanning

[1]:
```
import tfcomb
C = tfcomb.CombObj(verbosity=0)
```

[2]:
```
C.TFBS_from_motifs(regions="../data/GM12878_hg38_chr4_ATAC_peaks.bed",
                   motifs="../data/HOCOMOCOv11_HUMAN_motifs.txt",
                   genome="../data/hg38_chr4_masked.fa.gz",
                   threads=8)
```

For this analysis, we will run count_within() with the stranded option turned on:

[3]:
```
C.count_within(stranded=True, threads=8)
C.market_basket()
```

[4]:
```
C.rules
```

[4]:
```
                        TF1        TF2  TF1_TF2_count  TF1_count  TF2_count  \
KLF1(-)-KLF9(-)     KLF1(-)    KLF9(-)            145        291        209
KLF9(-)-KLF1(-)     KLF9(-)    KLF1(-)            145        209        291
KLF1(-)-KLF12(-)    KLF1(-)   KLF12(-)            152        291        240
KLF12(-)-KLF1(-)   KLF12(-)    KLF1(-)            152        240        291
KLF12(-)-KLF9(-)   KLF12(-)    KLF9(-)            127        240        209
...                    ...        ...            ...        ...        ...
STAT2(+)-SP1(+)    STAT2(+)     SP1(+)              1        556        636
NFE2L1(+)-SP2(+)  NFE2L1(+)     SP2(+)              1        451        798
SP2(+)-NFE2L1(+)     SP2(+)  NFE2L1(+)              1        798        451
BCL11A(+)-SP1(-)  BCL11A(+)     SP1(-)              1        562        653
SP1(-)-BCL11A(+)     SP1(-)  BCL11A(+)              1        653        562

                      cosine     zscore
KLF1(-)-KLF9(-)     0.587961  66.330168
KLF9(-)-KLF1(-)     0.587961  66.330168
KLF1(-)-KLF12(-)    0.575164  61.565308
KLF12(-)-KLF1(-)    0.575164  61.565308
KLF12(-)-KLF9(-)    0.567055  62.215597
...                      ...        ...
STAT2(+)-SP1(+)     0.001682  -5.135968
NFE2L1(+)-SP2(+)    0.001667  -4.294191
SP2(+)-NFE2L1(+)    0.001667  -4.294191
BCL11A(+)-SP1(-)    0.001651  -4.399691
SP1(-)-BCL11A(+)    0.001651  -4.399691

[226676 rows x 7 columns]
```

Analyze preferential orientation of motifs

First, we create a directionality analysis for the rules found:

[5]:
```
df = C.analyze_orientation()
```

```
INFO: Rules are symmetric - scenarios counted are: ['same', 'opposite']
```

[6]:
```
df.head()
```

[6]:
```
             TF1   TF2  TF1_TF2_count      same  opposite       std  \
SP3-SP4      SP3   SP4            534  0.758427  0.241573  0.365471
PATZ1-SP1  PATZ1   SP1            631  0.730586  0.269414  0.326098
PATZ1-SP3  PATZ1   SP3            642  0.725857  0.274143  0.319410
SP1-SP3      SP1   SP3            756  0.705026  0.294974  0.289951
KLF1-KLF9   KLF1  KLF9            243  0.851852  0.148148  0.497594

                pvalue
SP3-SP4    7.004689e-33
PATZ1-SP1  4.936837e-31
PATZ1-SP3  2.479952e-30
SP1-SP3    1.751909e-29
KLF1-KLF9  5.347598e-28
```

We can subset these on pvalue and number of sites:

[7]:
```
selected = df[(df["pvalue"] < 0.01) & (df["TF1_TF2_count"] > 50)]
```

[8]:
```
#Number of TF pairs with significant differences in orientation
selected.shape[0]
```

[8]:
```
476
```

We can also use the .loc-operator of the pandas dataframe to show the results of a subset of TF1-TF2-pairs:

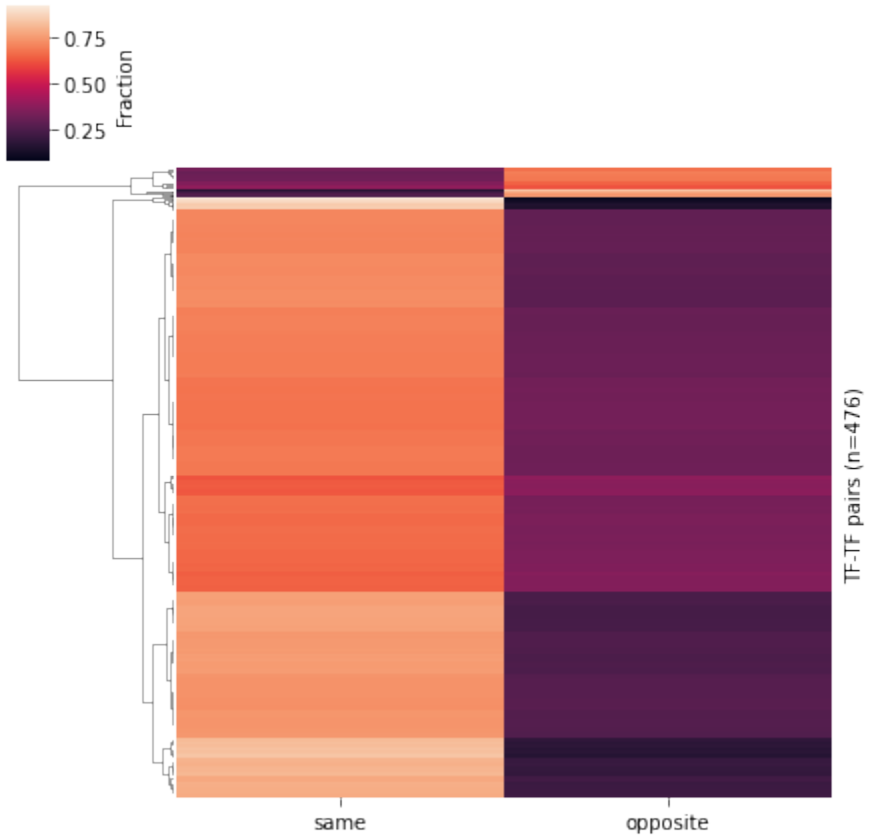
[9]:
```
df.loc[["EGR1-IRF4", "SP1-TAF1"]]
```

[9]:

	TF1	TF2	TF1_TF2_count	same	opposite	std	pvalue
EGR1-IRF4	EGR1	IRF4	15	0.866667	0.133333	0.518545	0.004509
SP1-TAF1	SP1	TAF1	153	0.679739	0.320261	0.254189	0.000009

Visualization of orientation preference

[10]:
```
_ = selected.plot_heatmap()
```



We can select the subsets by investigating the selected pairs:

[11]:
```
selected.sort_values("same").head(5)
```

[11]:
```
             TF1   TF2  TF1_TF2_count      same  opposite       std  \
KLF3-ZIC3   KLF3  ZIC3             66  0.166667  0.833333  0.471405
SP1-ZFX      SP1   ZFX             81  0.222222  0.777778  0.392837
PATZ1-ZFX  PATZ1   ZFX             74  0.229730  0.770270  0.382220
SP4-ZFX      SP4   ZFX             52  0.250000  0.750000  0.353553
ASCL1-WT1  ASCL1   WT1             61  0.262295  0.737705  0.336166

                pvalue
KLF3-ZIC3   6.093838e-08
SP1-ZFX     5.733031e-07
PATZ1-ZFX   3.320871e-06
SP4-ZFX     3.114910e-04
ASCL1-WT1   2.047606e-04
```

[12]:
```
selected.sort_values("opposite").head(5)
```

[12]:
```
            TF1   TF2  TF1_TF2_count      same  opposite       std  \
KLF4-KLF5  KLF4  KLF5             63  0.920635  0.079365  0.594868
ETV4-KLF3  ETV4  KLF3             57  0.912281  0.087719  0.583053
KLF9-KLF9  KLF9  KLF9            123  0.886179  0.113821  0.546139
KLF4-MAZ   KLF4   MAZ             70  0.871429  0.128571  0.525279
KLF4-KLF9  KLF4  KLF9             74  0.864865  0.135135  0.515997

                pvalue
KLF4-KLF5   2.432643e-11
ETV4-KLF3   4.806291e-10
KLF9-KLF9   1.072707e-17
KLF4-MAZ    5.126299e-10
KLF4-KLF9   3.443424e-10
```

Extended analysis with directional=True

The first analysis presented does not take into account the relative order of TF1-TF2, e.g. if the orientation "same" represents "TF1-TF2" or

[13]:
```
C.count_within(directional=True, stranded=True, threads=8)
C.market_basket()
```

[14]:
```
df = C.analyze_orientation()
```

```
INFO: Rules are directional - scenarios counted are: ['TF1-TF2', 'TF2-TF1', 'convergent',
↪ 'divergent']
```

[15]:
```
df.head()
```

[15]:
```
              TF1    TF2  TF1_TF2_count   TF1-TF2   TF2-TF1  convergent  \
SP2-SP2       SP2    SP2           1077  0.395543  0.395543    0.102136
SP1-SP1       SP1    SP1            687  0.417758  0.417758    0.075691
SP3-SP3       SP3    SP3            718  0.412256  0.412256    0.094708
PATZ1-PATZ1 PATZ1  PATZ1            547  0.422303  0.422303    0.078611
SP4-SP4       SP4    SP4            371  0.444744  0.444744    0.070081
```

(continues on next page)

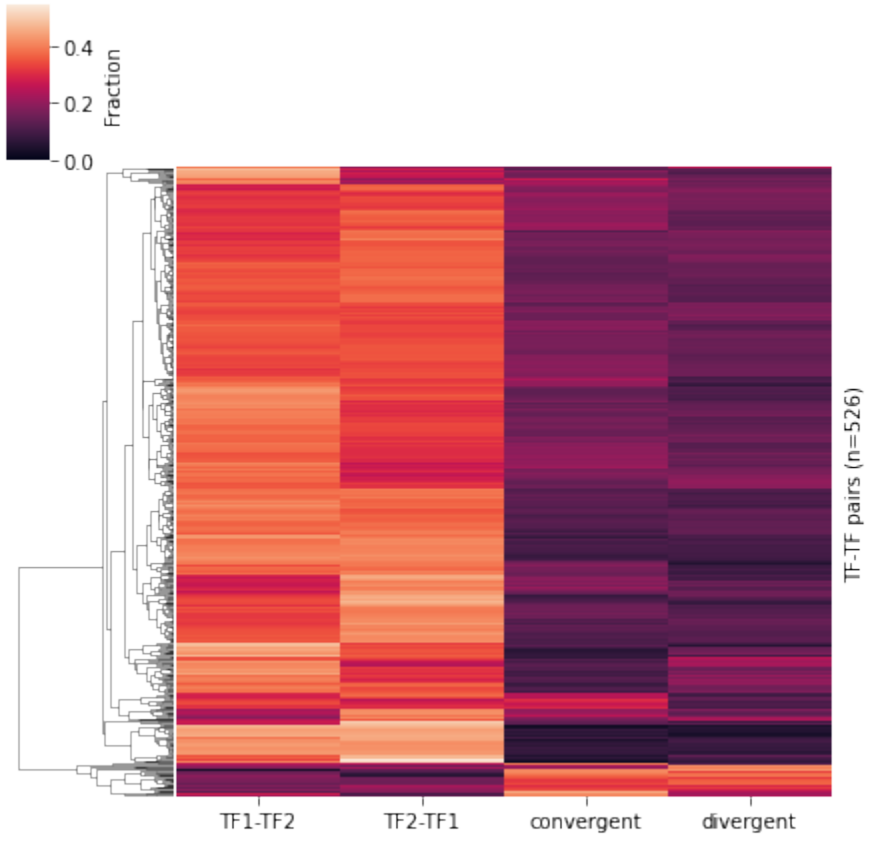
(continued from previous page)

```
              divergent       std        pvalue
SP2-SP2        0.106778  0.168069  8.140464e-79
SP1-SP1        0.088792  0.193784  8.390630e-67
SP3-SP3        0.080780  0.187444  2.559523e-65
PATZ1-PATZ1    0.076782  0.198960  4.875297e-56
SP4-SP4        0.040431  0.225196  1.132384e-48
```

Similarly to the first analysis, we can select the significant pairs and visualize the preferences for orientation:

[16]:
```
selected = df[(df["pvalue"] < 0.05) & (df["TF1_TF2_count"] > 50)]
```

[17]:
```
_ = selected.plot_heatmap()
```



In-depth look at preferential orientation

By sorting the selected co-occurring TF pairs, it is also possible to visualize the top pairs within each scenario as seen below.

TFs specific in TF1-TF2 orientation

[18]:
```
selected.sort_values("TF1-TF2", ascending=False).head()
```

[18]:
```
               TF1     TF2  TF1_TF2_count   TF1-TF2   TF2-TF1  convergent  \
KLF9-ZNF341    KLF9  ZNF341             80  0.500000  0.337500    0.100000
KLF1-KLF4      KLF1    KLF4             97  0.494845  0.360825    0.041237
KLF4-KLF4      KLF4    KLF4             56  0.482143  0.482143    0.017857
BCL11A-SPIB  BCL11A    SPIB             56  0.482143  0.321429    0.107143
KLF5-ZNF341    KLF5  ZNF341             61  0.475410  0.278689    0.114754

             divergent       std        pvalue
KLF9-ZNF341   0.062500  0.206408  6.866468e-09
KLF1-KLF4     0.103093  0.214005  1.575098e-11
KLF4-KLF4     0.017857  0.268055  1.851271e-10
BCL11A-SPIB   0.089286  0.187287  3.069277e-05
KLF5-ZNF341   0.131148  0.167382  1.331723e-04
```

TFs specific in TF2-TF2 orientation

[19]:
```
selected.sort_values("TF2-TF1", ascending=False).head()
```

[19]:
```
            TF1   TF2  TF1_TF2_count   TF1-TF2   TF2-TF1  convergent  \
KLF4-MAZ   KLF4   MAZ             70  0.328571  0.542857    0.042857
EGR1-KLF9  EGR1  KLF9             72  0.277778  0.541667    0.138889
KLF4-KLF5  KLF4  KLF5             63  0.380952  0.539683    0.000000
ETV4-KLF3  ETV4  KLF3             57  0.403509  0.508772    0.017544
E2F6-SP4   E2F6   SP4             80  0.275000  0.500000    0.137500

           divergent       std        pvalue
KLF4-MAZ    0.085714  0.232262  7.933608e-10
EGR1-KLF9   0.041667  0.217248  7.288757e-09
KLF4-KLF5   0.079365  0.253430  1.621951e-10
ETV4-KLF3   0.070175  0.242831  9.055097e-09
E2F6-SP4    0.087500  0.184560  3.725792e-07
```

TFs specific in convergent orientation

[20]:
```
selected.sort_values("convergent", ascending=False).head()
```

[20]:
```
             TF1   TF2  TF1_TF2_count   TF1-TF2   TF2-TF1  convergent  \
ASCL1-SP3  ASCL1   SP3             55  0.127273  0.181818    0.454545
MAZ-ZFX      MAZ   ZFX             54  0.203704  0.111111    0.444444
SP1-ZFX      SP1   ZFX             81  0.111111  0.111111    0.419753
AR-IRF1       AR  IRF1             51  0.274510  0.098039    0.411765
```

(continues on next page)

(continued from previous page)

```
ASCL1-WT1  ASCL1   WT1             61  0.147541  0.114754    0.409836

           divergent       std    pvalue
ASCL1-SP3   0.236364  0.143452  0.003533
MAZ-ZFX     0.240741  0.140627  0.005055
SP1-ZFX     0.358025  0.162343  0.000011
AR-IRF1     0.215686  0.130433  0.015372
ASCL1-WT1   0.327869  0.141892  0.002055
```

TFs specific in divergent orientation

[21]:
```
selected.sort_values("divergent", ascending=False).head()
```

[21]:
```
                TF1    TF2  TF1_TF2_count   TF1-TF2   TF2-TF1  convergent  \
STAT2-ZFP28    STAT2  ZFP28             55  0.127273  0.254545    0.181818
KLF3-ZIC3       KLF3   ZIC3             66  0.030303  0.136364    0.409091
IRF3-ZFP28      IRF3  ZFP28             58  0.206897  0.155172    0.224138
NFE2L1-STAT2  NFE2L1  STAT2             68  0.176471  0.220588    0.205882
SP4-ZFX         SP4    ZFX             52  0.192308  0.057692    0.365385

              divergent       std        pvalue
STAT2-ZFP28    0.436364  0.134738  7.445704e-03
KLF3-ZIC3      0.424242  0.197358  9.148367e-07
IRF3-ZFP28     0.413793  0.113059  3.069838e-02
NFE2L1-STAT2   0.397059  0.099740  4.364185e-02
SP4-ZFX        0.384615  0.154645  1.883579e-03
```

[1]:
```
import tfcomb.annotation
import pandas as pd
pd.set_option("display.max_columns", 50)
```

1.2.6 Annotate binding sites to genes

We will start by reading in known co-occurring rules:

[2]:
```
C = tfcomb.CombObj().from_pickle("../data/GM12878_selected.pkl")
```

[3]:
```
C.rules.head(20)
```

[3]:
```
                TF1     TF2  TF1_TF2_count  TF1_count  TF2_count    cosine  \
CTCF-RAD21     CTCF   RAD21           1751       2432       2241  0.750038
RAD21-SMC3    RAD21    SMC3           1376       2241       1638  0.718192
CTCF-SMC3      CTCF    SMC3           1361       2432       1638  0.681898
IKZF1-IKZF2   IKZF1   IKZF2           1726       2922       2324  0.662343
SMC3-ZNF143    SMC3  ZNF143           1060       1638       1652  0.644383
FOS-NFYA        FOS    NFYA             19         45         21  0.618070
BATF-JUNB      BATF    JUNB           1135       1854       1866  0.610218
RAD21-ZNF143  RAD21  ZNF143           1136       2241       1652  0.590408
CTCF-ZNF143    CTCF  ZNF143           1170       2432       1652  0.583713
```

(continues on next page)

(continued from previous page)

```
CREB1-CREM     CREB1    CREM            717       1114       1392  0.575780
ATF2-NFIC       ATF2    NFIC            957       1484       1911  0.568283
USF1-USF2       USF1    USF2            136        259        249  0.535537
SMC3-TRIM22     SMC3  TRIM22            906       1638       1786  0.529701
MEF2A-MEF2C    MEF2A   MEF2C            425       1129        572  0.528864
TCF12-TCF3     TCF12    TCF3            407        884        688  0.521884
BCL11A-IRF4   BCL11A    IRF4            522       1009        994  0.521233
ATF2-CREM       ATF2    CREM            742       1484       1392  0.516259
FOXM1-NFIC     FOXM1    NFIC            750       1115       1911  0.513799
HCFC1-SIX5     HCFC1    SIX5            111        287        172  0.499595
ATF2-FOXM1      ATF2   FOXM1            638       1484       1115  0.495982

                 zscore
CTCF-RAD21    18.643056
RAD21-SMC3    20.314026
CTCF-SMC3     20.245177
IKZF1-IKZF2   11.215960
SMC3-ZNF143   21.838431
FOS-NFYA      58.099184
BATF-JUNB     15.351205
RAD21-ZNF143  15.168180
CTCF-ZNF143   15.459923
CREB1-CREM    26.936982
ATF2-NFIC     14.789330
USF1-USF2     36.979279
SMC3-TRIM22   13.521569
MEF2A-MEF2C   26.006588
TCF12-TCF3    28.784395
BCL11A-IRF4   22.614173
ATF2-CREM     15.723393
FOXM1-NFIC    20.462765
HCFC1-SIX5    35.818992
ATF2-FOXM1    16.925349
```

We then use *get_pair_locations* to get the locations of the co-occurring sites:

[4]:
```
locations = C.get_pair_locations(("JUNB", "BATF"))
```
```
INFO: Setting up binding sites for counting
```

[5]:
```
type(locations)
```
[5]:
```
tfcomb.utils.TFBSPairList
```

[6]:
```
len(locations)
```
[6]:
```
1135
```

We can show these locations as a table using `.as_table`:

[7]:
```
location_table = locations.as_table()
location_table
```

[7]:
```
     site1_chrom  site1_start  site1_end site1_name  site1_score site1_strand  \
0           chr4       699544     699545       BATF         1000            .
1           chr4       799162     799163       BATF         1000            .
2           chr4       924255     924256       JUNB         1000            .
3           chr4      1218967    1218968       BATF         1000            .
4           chr4      1710533    1710534       BATF         1000            .
...          ...          ...        ...        ...          ...          ...
1130        chr4    185788438  185788439       BATF         1000            .
1131        chr4    186134462  186134463       BATF         1000            .
1132        chr4    186839676  186839677       JUNB         1000            .
1133        chr4    189503100  189503101       JUNB         1000            .
1134        chr4    189631410  189631411       BATF         1000            .

     site2_chrom  site2_start  site2_end site2_name  site2_score site2_strand  \
0           chr4       699546     699547       JUNB          791            .
1           chr4       799177     799178       JUNB         1000            .
2           chr4       924307     924308       BATF         1000            .
3           chr4      1218986    1218987       JUNB         1000            .
4           chr4      1710546    1710547       JUNB          718            .
...          ...          ...        ...        ...          ...          ...
1130        chr4    185788469  185788470       JUNB         1000            .
1131        chr4    186134479  186134480       JUNB         1000            .
1132        chr4    186839701  186839702       BATF         1000            .
1133        chr4    189503109  189503110       BATF          744            .
1134        chr4    189631459  189631460       JUNB         1000            .

      site_distance site_orientation
0                 1               NA
1                14               NA
2                51               NA
3                18               NA
4                12               NA
...             ...              ...
1130             30               NA
1131             16               NA
1132             24               NA
1133              8               NA
1134             48               NA

[1135 rows x 14 columns]
```

Annotate regions

We can now use *annotate_regions* to annotate these locations to genes:

[8]:
```
annotated = tfcomb.annotation.annotate_regions(location_table, gtf="../data/chr4_genes.
↪gtf")
```

```
[W::hts_idx_load2] The index file is older than the data file: ../data/chr4_genes.gtf.gz.
↪tbi
```

[9]:
```
annotated
```

[9]:
```
     site1_chrom  site1_start  site1_end site1_name  site1_score site1_strand  \
0           chr4       699544     699545       BATF         1000            .
1           chr4       799162     799163       BATF         1000            .
2           chr4       924255     924256       JUNB         1000            .
3           chr4      1218967    1218968       BATF         1000            .
4           chr4      1710533    1710534       BATF         1000            .
...          ...          ...        ...        ...          ...          ...
1130        chr4    185788438  185788439       BATF         1000            .
1131        chr4    186134462  186134463       BATF         1000            .
1132        chr4    186839676  186839677       JUNB         1000            .
1133        chr4    189503100  189503101       JUNB         1000            .
1134        chr4    189631410  189631411       BATF         1000            .

     site2_chrom  site2_start  site2_end site2_name  site2_score site2_strand  \
0           chr4       699546     699547       JUNB          791            .
1           chr4       799177     799178       JUNB         1000            .
2           chr4       924307     924308       BATF         1000            .
3           chr4      1218986    1218987       JUNB         1000            .
4           chr4      1710546    1710547       JUNB          718            .
...          ...          ...        ...        ...          ...          ...
1130        chr4    185788469  185788470       JUNB         1000            .
1131        chr4    186134479  186134480       JUNB         1000            .
1132        chr4    186839701  186839702       BATF         1000            .
1133        chr4    189503109  189503110       BATF          744            .
1134        chr4    189631459  189631460       JUNB         1000            .

      site_distance site_orientation feature feat_strand  feat_start  \
0                 1               NA    gene           +    705747.0
1                14               NA     NaN         NaN         NaN
2                51               NA    gene           +    932386.0
3                18               NA     NaN         NaN         NaN
4                12               NA    gene           +   1712857.0
...             ...              ...     ...         ...         ...
1130             30               NA     NaN         NaN         NaN
1131             16               NA     NaN         NaN         NaN
1132             24               NA     NaN         NaN         NaN
1133              8               NA     NaN         NaN         NaN
1134             48               NA     NaN         NaN         NaN

       feat_end query_name  distance feat_anchor feat_ovl_peak peak_ovl_feat  \
0      770640.0    query_1    6203.0       start           0.0           0.0
1           NaN        NaN       NaN         NaN           NaN           NaN
2      958656.0    query_1    8131.0       start           0.0           0.0
3           NaN        NaN       NaN         NaN           NaN           NaN
4     1745171.0    query_1    2324.0       start           0.0           0.0
...         ...        ...       ...         ...           ...           ...
1130        NaN        NaN       NaN         NaN           NaN           NaN
1131        NaN        NaN       NaN         NaN           NaN           NaN
1132        NaN        NaN       NaN         NaN           NaN           NaN
1133        NaN        NaN       NaN         NaN           NaN           NaN
1134        NaN        NaN       NaN         NaN           NaN           NaN
```

(continues on next page)

(continued from previous page)

```
     relative_location          gene_id gene_version gene_name  \
0             Upstream  ENSG00000185619           18     PCGF3
1                  NaN              NaN          NaN       NaN
2             Upstream  ENSG00000127419           17   TMEM175
3                  NaN              NaN          NaN       NaN
4             Upstream  ENSG00000013810           21     TACC3
...                ...              ...          ...       ...
1130               NaN              NaN          NaN       NaN
1131               NaN              NaN          NaN       NaN
1132               NaN              NaN          NaN       NaN
1133               NaN              NaN          NaN       NaN
1134               NaN              NaN          NaN       NaN

         gene_source    gene_biotype
0     ensembl_havana  protein_coding
1                NaN             NaN
2     ensembl_havana  protein_coding
3                NaN             NaN
4     ensembl_havana  protein_coding
...              ...             ...
1130             NaN             NaN
1131             NaN             NaN
1132             NaN             NaN
1133             NaN             NaN
1134             NaN             NaN

[1135 rows x 29 columns]
```

By subsetting all sites, we can highlight the pairs annotated to promoters of any genes:

[10]:
```
annotated[~annotated["gene_id"].isna()]
```

[10]:
```
     site1_chrom  site1_start  site1_end site1_name  site1_score site1_strand  \
0           chr4       699544     699545       BATF         1000            .
2           chr4       924255     924256       JUNB         1000            .
4           chr4      1710533    1710534       BATF         1000            .
13          chr4      2761381    2761382       JUNB          855            .
15          chr4      2787533    2787534       BATF         1000            .
...          ...          ...        ...        ...          ...          ...
1106        chr4    184734256  184734257       BATF         1000            .
1117        chr4    185205229  185205230       BATF          607            .
1124        chr4    185399224  185399225       JUNB         1000            .
1125        chr4    185405783  185405784       JUNB         1000            .
1127        chr4    185479574  185479575       BATF         1000            .

     site2_chrom  site2_start  site2_end site2_name  site2_score site2_strand  \
0           chr4       699546     699547       JUNB          791            .
2           chr4       924307     924308       BATF         1000            .
4           chr4      1710546    1710547       JUNB          718            .
13          chr4      2761397    2761398       BATF         1000            .
15          chr4      2787566    2787567       JUNB          954            .
...          ...          ...        ...        ...          ...          ...
```

(continues on next page)

(continued from previous page)

```
1106        chr4    184734285  184734286       JUNB          580            .
1117        chr4    185205254  185205255       JUNB          878            .
1124        chr4    185399267  185399268       BATF         1000            .
1125        chr4    185405801  185405802       BATF         1000            .
1127        chr4    185479580  185479581       JUNB         1000            .

      site_distance site_orientation feature feat_strand   feat_start  \
0                 1               NA    gene           +     705747.0
2                51               NA    gene           +     932386.0
4                12               NA    gene           +    1712857.0
13               15               NA    gene           -    2741647.0
15               32               NA    gene           +    2793070.0
...             ...              ...     ...         ...          ...
1106             28               NA    gene           -  184694084.0
1117             24               NA    gene           +  185204236.0
1124             42               NA    gene           -  185363871.0
1125             17               NA    gene           -  185363871.0
1127              5               NA    gene           -  185445181.0

         feat_end query_name  distance feat_anchor feat_ovl_peak  \
0        770640.0    query_1    6203.0       start           0.0
2        958656.0    query_1    8131.0       start           0.0
4       1745171.0    query_1    2324.0       start           0.0
13      2756342.0    query_1    5039.0       start           0.0
15      2841098.0    query_1    5537.0       start           0.0
...           ...        ...       ...         ...           ...
1106  184734130.0    query_1     126.0       start           0.0
1117  185370185.0    query_1     993.0       start           1.0
1124  185395924.0    query_1    3300.0       start           0.0
1125  185395924.0    query_1    9859.0       start           0.0
1127  185471752.0    query_1    7822.0       start           0.0

     peak_ovl_feat  relative_location          gene_id gene_version gene_name  \
0              0.0           Upstream  ENSG00000185619           18     PCGF3
2              0.0           Upstream  ENSG00000127419           17   TMEM175
4              0.0           Upstream  ENSG00000013810           21     TACC3
13             0.0           Upstream  ENSG00000168884           15     TNIP2
15             0.0           Upstream  ENSG00000087266           17    SH3BP2
...            ...                ...              ...          ...       ...
1106           0.0           Upstream  ENSG00000151725           12     CENPU
1117      0.000006  PeakInsideFeature  ENSG00000109762           16     SNX25
1124           0.0           Upstream  ENSG00000109771           16    LRP2BP
1125           0.0           Upstream  ENSG00000109771           16    LRP2BP
1127           0.0           Upstream  ENSG00000168491           10   CCDC110

         gene_source    gene_biotype
0     ensembl_havana  protein_coding
2     ensembl_havana  protein_coding
4     ensembl_havana  protein_coding
13    ensembl_havana  protein_coding
15    ensembl_havana  protein_coding
...              ...             ...
```

(continues on next page)

(continued from previous page)

```
1106  ensembl_havana  protein_coding
1117  ensembl_havana  protein_coding
1124  ensembl_havana  protein_coding
1125  ensembl_havana  protein_coding
1127  ensembl_havana  protein_coding

[94 rows x 29 columns]
```

1.2.7 Differential analysis

This notebook shows how to create a differential analysis based on two CombObj's (A and B) from two different cell types.

[1]:
```
import tfcomb.objects
```

Prepare GM12878 and K562 CombObjs

The two objects contain ENCODE ChIP-seq peaks (1bp centered on the middle of the peak) from the celltypes GM12878 and K562 respectively.

[2]:
```
A = tfcomb.objects.CombObj(verbosity=0)
A.prefix = "GM12878"
A.TFBS_from_bed("../data/GM12878_hg38_chr4_TF_chipseq.bed")
A.market_basket()
A.set_verbosity(1) #reset verbosity to INFO
```

```
Internal counts for 'TF_counts' were not set. Please run .count_within() to obtain TF-TF␣
↪co-occurrence counts.
```

[3]:
```
B = tfcomb.objects.CombObj(verbosity=0)
B.prefix = "K562"
B.TFBS_from_bed("../data/K562_hg38_chr4_TF_chipseq.bed")
B.market_basket()
```

```
Internal counts for 'TF_counts' were not set. Please run .count_within() to obtain TF-TF␣
↪co-occurrence counts.
```

Compare two CombObj's

The two objects contain different amounts of TFs and rules:

[4]:
```
print(A)
print(B)
```

```
<CombObj: 112109 TFBS (151 unique names) | Market basket analysis: 21284 rules>
<CombObj: 216370 TFBS (447 unique names) | Market basket analysis: 166088 rules>
```

We will now use the *.compare*-function of CombObj 'A' to directly compare it with CombObj 'B'. What you will see is that many of the TFs are different between the object and are thus removed:

[5]:
```
compare_obj = A.compare(B)
```

```
WARNING: 365 TFs were not common between objects and were excluded from .rules. Set 'join
↪' to 'outer' in order to use all TFs across objects. The TFs excluded were: ['TEAD2',
↪'ZNF215', 'NR3C1', 'MEF2C', 'ZNF263', 'AGO1', 'ZEB1', 'ZEB2', 'ILF3', 'ZNF584', 'POLR3A
↪', 'XRCC3', 'KDM5B', 'POLR2G', 'PBX3', 'KLF5', 'ZNF695', 'NFRKB', 'SAP30', 'GTF2A2',
↪'CC2D1A', 'MAFG', 'ZNF197', 'RBM22', 'MCM3', 'IRF9', 'XRCC5', 'MCM7', 'THAP12', 'HNRNPK
↪', 'TRIP13', 'ZNF764', 'TFE3', 'U2AF1', 'JUN', 'ETV5', 'SNAPC5', 'KLF1', 'ZNF830',
↪'ZNF444', 'ZFP91', 'ZNF354B', 'GTF2F1', 'LEF1', 'ZBTB8A', 'MYB', 'ZC3H8', 'NCOA4',
↪'KLF6', 'POLR2H', 'STAT3', 'BCOR', 'ZNF165', 'POU2F2', 'ZNF644', 'DACH1', 'SMARCB1',
↪'IKZF2', 'HEY1', 'PTTG1', 'BATF', 'MEF2D', 'FIP1L1', 'FOXJ2', 'YBX3', 'BRD9', 'ZNF217',
↪ 'RBM34', 'DLX4', 'ZNF75A', 'ZSCAN32', 'PBX2', 'ZNF84', 'PHB', 'SP2', 'PHF20', 'POU5F1
↪', 'CCAR2', 'FOXA1', 'PATZ1', 'ZNF274', 'ZNF148', 'MNT', 'HLTF', 'HDAC3', 'ZNF175',
↪'ZNF436', 'ATF1', 'ZNF311', 'ELF4', 'HDAC1', 'ZNF507', 'ZFP1', 'RBM15', 'PRDM15',
↪'ZBTB7A', 'ARID2', 'TAL1', 'ZNF3', 'ERF', 'TFCP2', 'NR2F2', 'ZFX', 'ELF2', 'PAX8',
↪'STAG1', 'ZNF184', 'NCOR1', 'ZNF551', 'SIN3B', 'ZBTB2', 'TBX21', 'KHSRP', 'PTBP1',
↪'THAP7', 'NELFE', 'RBFOX2', 'ARID3B', 'PHF21A', 'EP400', 'ZNF700', 'WRNIP1', 'MYNN',
↪'FOSL1', 'HNRNPLL', 'SMARCE1', 'TEAD1', 'ZBTB11', 'ZNF655', 'RELA', 'HNRNPL', 'SMARCA4
↪', 'HMBOX1', 'PRMT5', 'ZNF83', 'PCBP2', 'E2F1', 'SUPT5H', 'HIVEP1', 'ZNF257', 'TOE1',
↪'ZKSCAN1', 'ZNF76', 'RBPJ', 'CGGBP1', 'PAX5', 'ZNF282', 'U2AF2', 'CBX2', 'ID3',
↪'NEUROD1', 'PCBP1', 'ZNF445', 'CBX1', 'FOXO4', 'MAFF', 'ARID1B', 'CHAMP1', 'PHF8',
↪'ETV1', 'SOX6', 'ZNF407', 'GTF2E2', 'ZNF589', 'ATF6', 'TRIM24', 'TBX18', 'ADNP', 'KLF10
↪', 'E2F6', 'BCL3', 'ZNF174', 'ZNF57', 'ZNF639', 'ZNF319', 'NR2F6', 'GABPB1', 'SNIP1',
↪'MGA', 'SRSF7', 'RUNX1', 'E2F3', 'SIRT6', 'CBFA2T2', 'TRIM28', 'HDAC8', 'ELK3', 'ASH2L
↪', 'KDM4B', 'ZNF622', 'CREB3L1', 'ZNF79', 'EHMT2', 'C11orf30', 'HSF1', 'ZBTB17', 'ZMYM3
↪', 'ZNF281', 'CLOCK', 'BDP1', 'ZNF395', 'ZNF250', 'TSHZ1', 'TFDP1', 'BCL11A', 'STAT1',
↪'CTBP1', 'HES1', 'TBPL1', 'IRF4', 'DIDO1', 'SFPQ', 'PYGO2', 'ZNF133', 'GATAD2A',
↪'ZNF280B', 'RBM39', 'EBF1', 'POLR2B', 'HNRNPUL1', 'ZNF397', 'CDC5L', 'ZC3H11A', 'IRF3',
↪ 'BCL6', 'ZBTB12', 'KLF13', 'MCM5', 'SMARCC2', 'ZNF408', 'NR0B1', 'ZNF518B', 'ZKSCAN8',
↪ 'NR1H2', 'ZC3H4', 'ZNF780A', 'ZNF146', 'CHD7', 'ZNF717', 'RUNX3', 'E2F7', 'ZKSCAN3',
↪'WHSC1', 'HOMEZ', 'TRIM25', 'CREB5', 'FOXJ3', 'ZNF280A', 'DEAF1', 'IRF1', 'HMG20B',
↪'MBD2', 'PHTF2', 'GMEB1', 'TCF7L2', 'MCM2', 'NFATC1', 'BRD4', 'ILK', 'TAF15', 'MTA1',
↪'SNRNP70', 'RNF2', 'RLF', 'ZBTB9', 'NFIX', 'ZBTB5', 'IRF2', 'ESRRB', 'COPS2', 'ARHGAP35
↪', 'ZNF687', 'ZNF23', 'DDX20', 'MEIS2', 'MEF2B', 'ASH1L', 'SRSF1', 'FUS', 'NFE2',
↪'HMGN3', 'EWSR1', 'KAT2B', 'SRSF3', 'ZNF239', 'TRIM22', 'RBM17', 'CSDE1', 'RREB1',
↪'MIER1', 'GATA2', 'NFE2L1', 'CREBBP', 'HNRNPH1', 'ZNF7', 'RFX1', 'STAT5B', 'TAF7',
↪'KAT8', 'KLF16', 'ZNF778', 'TAF9B', 'PTRF', 'MYBL2', 'ZNF12', 'ZNF316', 'ZNF207', 'PHB2
↪', 'ATF4', 'TEAD4', 'ZNF212', 'NUFIP1', 'RXRA', 'GATA1', 'ETS2', 'GTF2I', 'NCOA2',
↪'SAFB', 'NR4A1', 'PRPF4', 'MITF', 'THAP1', 'L3MBTL2', 'ZNF740', 'VEZF1', 'ZNF583',
↪'RBM25', 'TSC22D4', 'DNMT1', 'GTF3C2', 'THRA', 'IRF5', 'HINFP', 'ZMIZ1', 'ZNF785',
↪'GTF2B', 'CREB3', 'NONO', 'ZNF77', 'CBX8', 'ZNF318', 'RBM14', 'PRDM10', 'SETDB1',
↪'ZNF512', 'KAT2A', 'NCOA1', 'ZNF766', 'EED', 'SHOX2', 'ZHX1', 'CCNT2', 'RELB', 'CBFA2T3
↪', 'AFF1', 'E2F5', 'SAFB2', 'CEBPG', 'DDIT3', 'CTCFL', 'THRAP3', 'NCOA6', 'RFX7',
↪'ZNF324', 'ZNF347']
INFO: Calculating foldchange for contrast: GM12878 / K562
INFO: The calculated log2fc's are found in the rules table (<DiffCombObj>.rules)
```

The results of the differential analysis are now found in the .rules of the CombObj:

[6]:
```
compare_obj.rules
```

[6]:

	TF1	TF2	GM12878_cosine	\
SIX5-SUZ12	SIX5	SUZ12	0.005640	
SUZ12-SIX5	SUZ12	SIX5	0.005640	

(continues on next page)

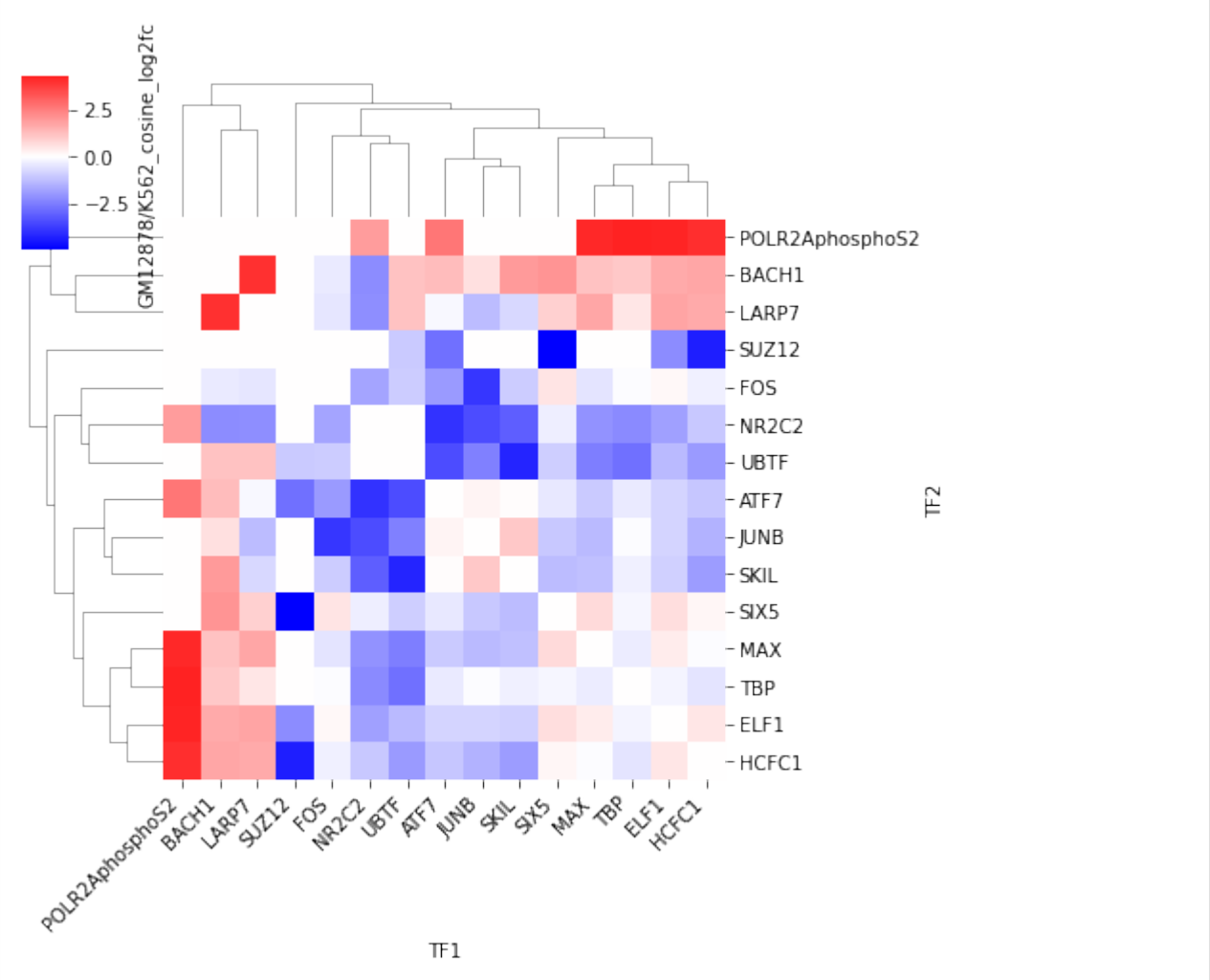
(continued from previous page)

```
SUZ12-HCFC1                    SUZ12            HCFC1        0.004308
HCFC1-SUZ12                    HCFC1            SUZ12        0.004308
UBTF-SKIL                       UBTF             SKIL        0.004266
...                              ...              ...             ...
POLR2AphosphoS2-MAX   POLR2AphosphoS2              MAX        0.206904
ELF1-POLR2AphosphoS2             ELF1  POLR2AphosphoS2        0.245057
POLR2AphosphoS2-ELF1  POLR2AphosphoS2             ELF1        0.245057
POLR2AphosphoS2-TBP   POLR2AphosphoS2              TBP        0.254481
TBP-POLR2AphosphoS2               TBP  POLR2AphosphoS2        0.254481

                      K562_cosine  GM12878/K562_cosine_log2fc
SIX5-SUZ12               0.478929                   -4.966291
SUZ12-SIX5               0.478929                   -4.966291
SUZ12-HCFC1              0.282011                   -4.351172
HCFC1-SUZ12              0.282011                   -4.351172
UBTF-SKIL                0.253694                   -4.208163
...                           ...                         ...
POLR2AphosphoS2-MAX      0.002029                    4.172457
ELF1-POLR2AphosphoS2     0.003654                    4.223385
POLR2AphosphoS2-ELF1     0.003654                    4.223385
POLR2AphosphoS2-TBP      0.003560                    4.285724
TBP-POLR2AphosphoS2      0.003560                    4.285724

[12114 rows x 5 columns]
```

Plot differential co-occurring TFs

We can now have a look at the changes between the two objects in terms of 'cosine' measure:

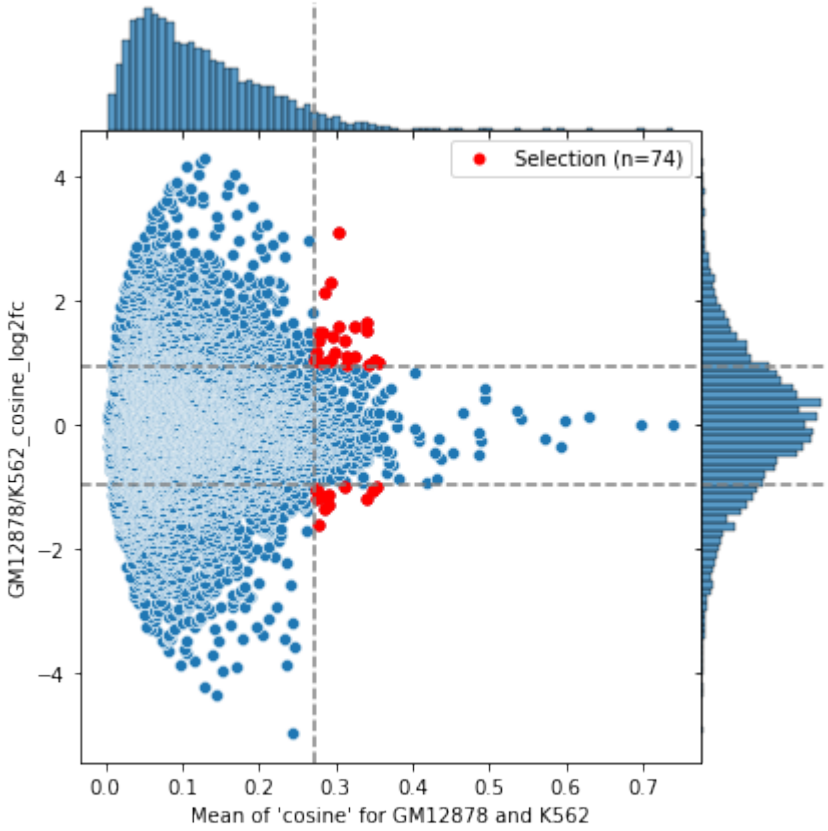
[7]:
```
compare_obj.plot_heatmap()
```



Like in the case for CombObjs, we can also select a subset of interesting differentially co-occurring TFs:

[8]:
```
selection = compare_obj.select_rules(measure_threshold_percent=0.2)
```

```
INFO: Selecting rules for contrast: ('GM12878', 'K562')
INFO: measure_threshold is None; trying to calculate optimal threshold
INFO: mean_threshold is None; trying to calculate optimal threshold
INFO: Creating subset of rules using thresholds
```



We can also plot the network to show the pairs which are either increasing or decreasing in 'cosine' measure between the two cell types:

[9]:
```
selection.plot_network()
```

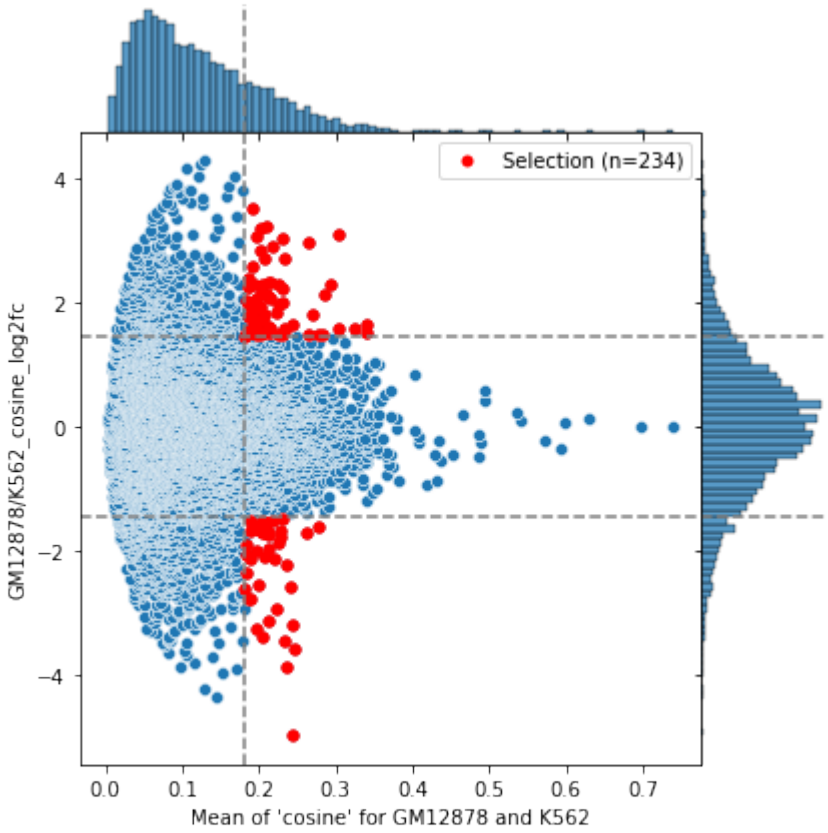
```
INFO: Finished! The network is found within <CombObj>.network.
```

[9]:

The strictness of the automatic threshold can be adjusted with `measure_threshold_percent` and `mean_threshold_percent`:

[10]:
```
selection2 = compare_obj.select_rules(measure_threshold_percent=0.1, mean_threshold_
↪percent=0.2)
```

```
INFO: Selecting rules for contrast: ('GM12878', 'K562')
INFO: measure_threshold is None; trying to calculate optimal threshold
INFO: mean_threshold is None; trying to calculate optimal threshold
INFO: Creating subset of rules using thresholds
```



[11]:
```
selection2.plot_network()
```

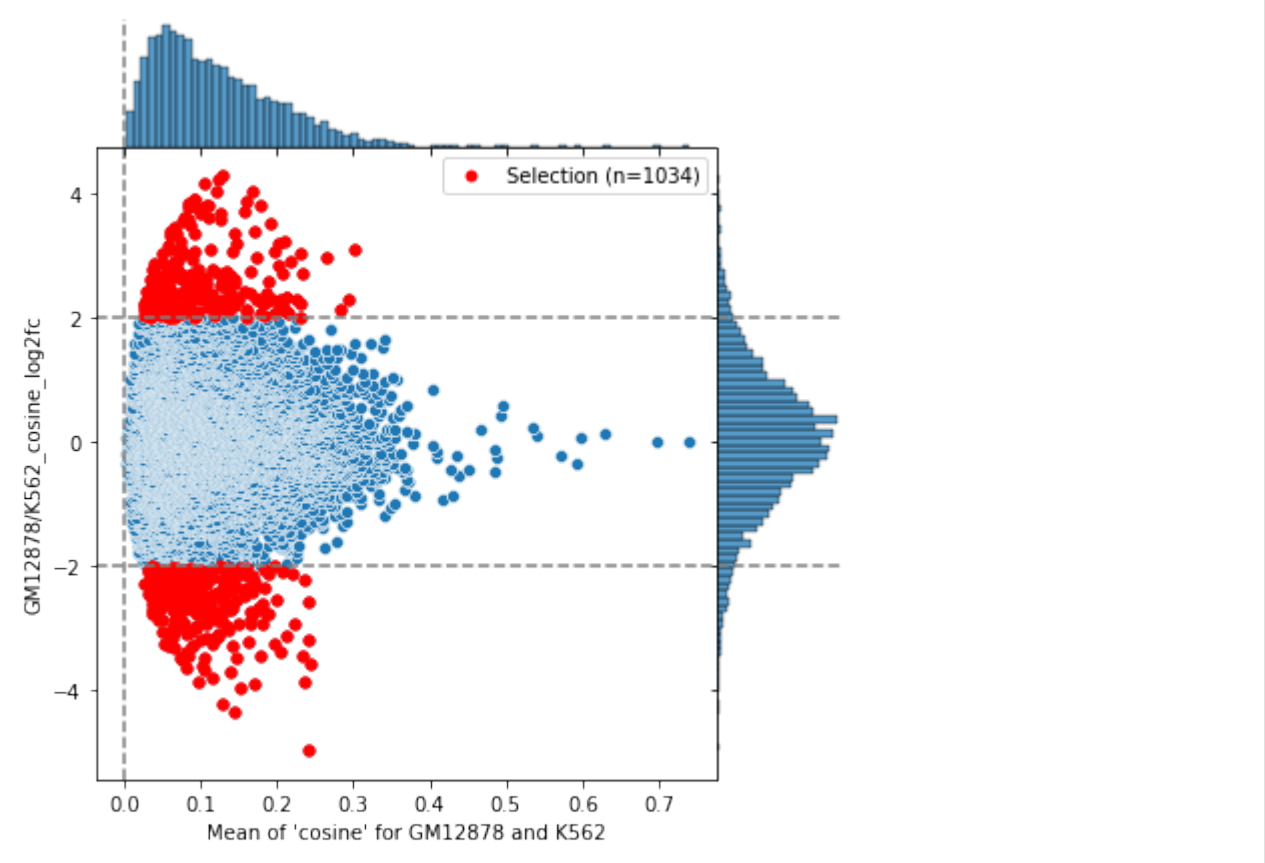
```
INFO: Finished! The network is found within <CombObj>.network.
```

[11]:

It is also possible to set specific thresholds with `measure_threshold` and `mean_threshold` (these will overwrite the automatic thresholding set by `measure_threshold_percent` and `mean_threshold_percent`):

[12]:
```
selection3 = compare_obj.select_rules(measure_threshold=(-2,2), mean_threshold=0)
```

```
INFO: Selecting rules for contrast: ('GM12878', 'K562')
INFO: Creating subset of rules using thresholds
```



[13]:
```
selection3.plot_network()
```

```
INFO: Finished! The network is found within <CombObj>.network.
```

[13]:

1.2.8 Binding distance analysis

In this example notebook, a binding distance analysis will be shown and explained step by step.
To start such an analysis, a market basket analysis is required first. The data used is the same as in the *TFBS from motif*.

For more details on how to perform the market basket step please have a look at the *TFBS from motif* or *ChIP-seq analysis* examples.

Content:

- Prepare object
- *Select rules*
- *Automated analysis*
- *Step-by-step:*
  1. Create object
  2. Count distances
  3. Smoothing counts
  4. Scale counts
  5. Correct background
  6. Analyse signal
  - z-score
  - Flat
- *Further downstream analysis*
  1. Analyzing hubs
  2. Signal classification

Prepare a CombObj

[1]:
```
import tfcomb.objects

C = tfcomb.objects.CombObj()
C.TFBS_from_motifs(regions="../data/GM12878_hg38_chr4_ATAC_peaks.bed",
                   motifs="../data/HOCOMOCOv11_HUMAN_motifs.txt",
                   genome="../data/hg38_chr4.fa.gz",
                   threads=4)
C.count_within(max_overlap=0.0, threads=4)
C.market_basket()
C.rules.head()
```

```
INFO: Scanning for TFBS with 4 thread(s)...
INFO: Progress: 11%
INFO: Progress: 20%
INFO: Progress: 30%
INFO: Progress: 40%
INFO: Progress: 50%
INFO: Progress: 60%
INFO: Progress: 71%
INFO: Progress: 82%
INFO: Progress: 91%
INFO: Finished!
INFO: Processing scanned TFBS
INFO: Identified 165810 TFBS (401 unique names) within given regions
```

(continues on next page)

(continued from previous page)

```
INFO: Setting up binding sites for counting
INFO: Counting co-occurrences within sites
INFO: Counting co-occurrence within background
INFO: Progress: 16%
INFO: Progress: 20%
INFO: Progress: 32%
INFO: Progress: 40%
INFO: Progress: 50%
INFO: Progress: 62%
INFO: Progress: 72%
INFO: Progress: 80%
INFO: Progress: 92%
INFO: Finished!
INFO: Done finding co-occurrences! Run .market_basket() to estimate significant pairs
INFO: Market basket analysis is done! Results are found in <CombObj>.rules
```

[1]:
```
                    TF1      TF2  TF1_TF2_count  TF1_count  TF2_count  \
POU3F2-SMARCA5   POU3F2  SMARCA5            239        302        241
SMARCA5-POU3F2  SMARCA5   POU3F2            239        241        302
POU2F1-SMARCA5   POU2F1  SMARCA5            263        426        241
SMARCA5-POU2F1  SMARCA5   POU2F1            263        241        426
SMARCA5-ZNF582  SMARCA5   ZNF582            172        241        195

                  cosine      zscore
POU3F2-SMARCA5  0.885902  129.586528
SMARCA5-POU3F2  0.885902  129.586528
POU2F1-SMARCA5  0.820810  135.355691
SMARCA5-POU2F1  0.820810  135.355691
SMARCA5-ZNF582  0.793419  117.370387
```

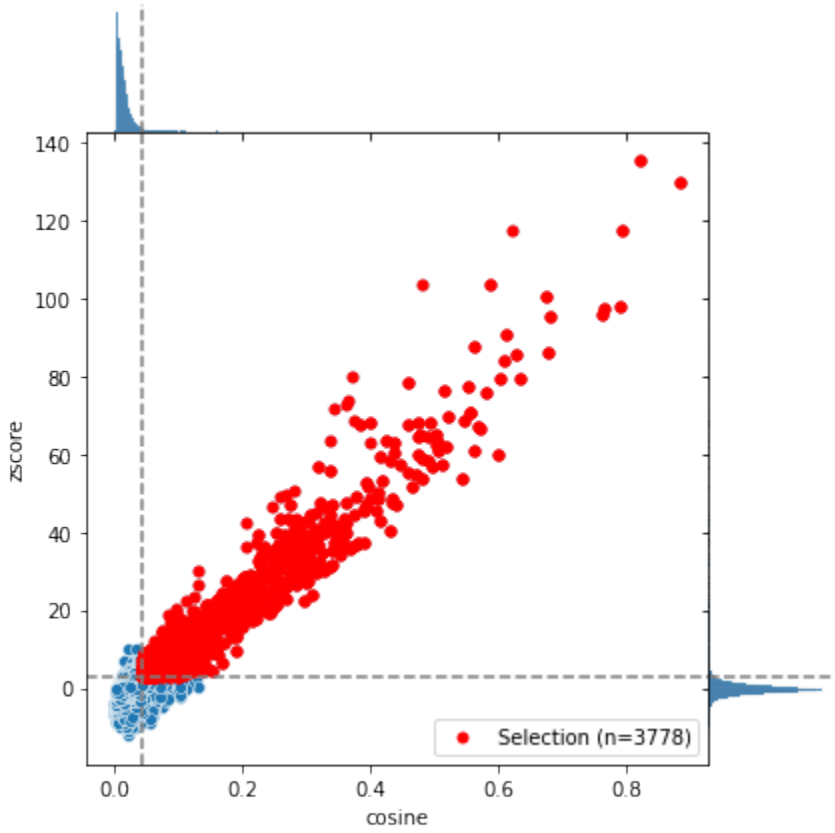
Selecting rules

Due to the way the market basket analysis is working, the C.rules result table contains entries for every transcription factor combination present in the data.

For a binding analysis not all of these rules are of particular interest. In this notebook the method *.select_significant_rules()* is used to filter rules of interest. For more details on rule selection please refer to the example notebook *Select rules*.

[2]:
```
selection = C.select_significant_rules()
```

```
INFO: x_threshold is None; trying to calculate optimal threshold
INFO: y_threshold is None; trying to calculate optimal threshold
INFO: Creating subset of TFBS and rules using thresholds
```



Method 1: Automated analysis

There are two different ways to run this analysis. The automated way will be shown in this chapter. Followed by an in depth guide showing the analysis step by step in the next chapter.

Here we will start with the **automated** analysis for **all** selected rules.

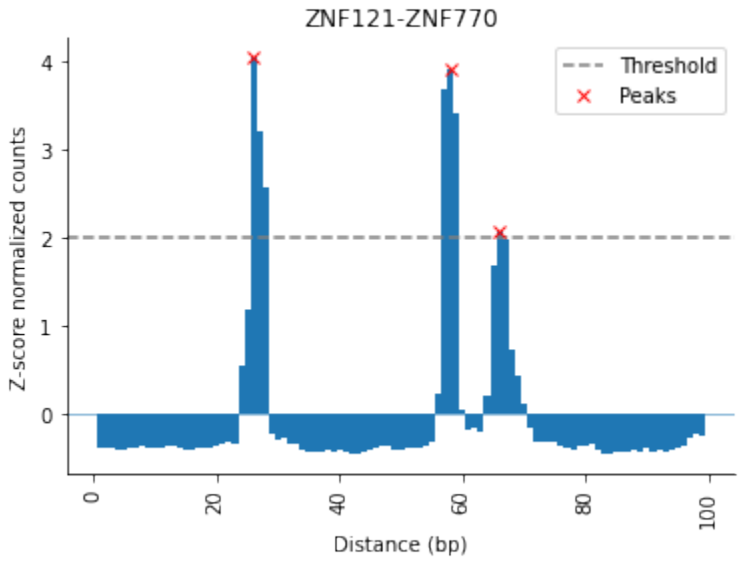
[3]:
```
selection.analyze_distances(threads=6) # adjust threads if needed
```

```
INFO: DistObject successfully created! It can be accessed via <CombObj>.distObj
INFO: Preparing to count distances.
INFO: Setting up binding sites for counting
INFO: Calculating distances
INFO: Done finding distances! Results are found in .distances
INFO: You can now run .smooth() and/or .correct_background() to preprocess sites before␣
↪finding peaks.
INFO: Or you can find peaks directly using .analyze_signal_all()
INFO: Smoothing signals with window size 3
INFO: Background correction finished! Results can be found in .corrected
INFO: Analyzing Signal with threads 6
INFO: Calculating zscores for signals
INFO: Finding preferred distances
INFO: Done analyzing signal. Results are found in .peaks
```

[4]:
```
selection.distObj.peaks.loc[(selection.distObj.peaks.TF1 == "ZNF121") & (selection.
↪distObj.peaks.TF2 == "ZNF770")]
```

[4]:
```
                   TF1     TF2  Distance  Peak Heights  Prominences  Threshold  \
ZNF121-ZNF770  ZNF121  ZNF770        26      4.043651     4.443567          2
ZNF121-ZNF770  ZNF121  ZNF770        58      3.906961     4.353073          2
ZNF121-ZNF770  ZNF121  ZNF770        66      2.068096     2.270661          2

               TF1_TF2_count  Distance_count  Distance_percent Distance_window
ZNF121-ZNF770           1319             381         28.885519         [25;27]
ZNF121-ZNF770           1319             376         28.506444         [57;59]
ZNF121-ZNF770           1319             220         16.679303         [65;67]
```

[5]:
```
_ = selection.distObj.plot(("ZNF121", "ZNF770"))
```



On the **x-axis** the distance in bp is shown. For example a distance of 100 means the anchors (depending on anchor mode, please refer to *Anchor mode* on this topic) are 100 bp away of each other. On the **y-axis** of the plot the calculated **zscore** per distance is shown.

For an explaination of the results please refer to the in depth guide below.

Method 2: Step-by-Step Analysis

Besides the automated way, the analysis can be done **step-by-step**. This chapter is an detailed guide, covering all 5 major steps.

1. Create a distance object

The binding distance analysis can be started from within any *combObj*. In this example the appropiate object is called **selection**, since we only want to use **significant** rules, not all. The analysis can also be done without pre selection.

[6]:
```
selection.create_distObj()
```

```
INFO: DistObject successfully created! It can be accessed via <CombObj>.distObj
```

As stated in the information message, the distObj should be created successfully and filled with all important information to start the distance analysis. This includes parameters set for the market masket analysis.

[7]:
```
selection.distObj
```

[7]:
```
<tfcomb.objects.DistObj at 0x7f87d1e6a950>
```

The 3792 rules (market basket results) selected earlier by *select.significant_rules()* are automatically passed to the distance object during creation:

[8]:
```
selection.distObj.rules
```

[8]:
```
                    TF1      TF2  TF1_TF2_count  TF1_count  TF2_count  \
POU3F2-SMARCA5   POU3F2  SMARCA5            239        302        241
SMARCA5-POU3F2  SMARCA5   POU3F2            239        241        302
POU2F1-SMARCA5   POU2F1  SMARCA5            263        426        241
SMARCA5-POU2F1  SMARCA5   POU2F1            263        241        426
SMARCA5-ZNF582  SMARCA5   ZNF582            172        241        195
...                ...      ...            ...        ...        ...
NFYB-EGR2          NFYB     EGR2             24        219       1304
MYOG-TAF1          MYOG     TAF1             23        386        681
TAF1-MYOG          TAF1     MYOG             23        681        386
ETV1-ZBTB17        ETV1   ZBTB17             20        117       1699
ZBTB17-ETV1      ZBTB17     ETV1             20       1699        117

                  cosine      zscore
POU3F2-SMARCA5  0.885902  129.586528
SMARCA5-POU3F2  0.885902  129.586528
POU2F1-SMARCA5  0.820810  135.355691
SMARCA5-POU2F1  0.820810  135.355691
SMARCA5-ZNF582  0.793419  117.370387
...                ...         ...
NFYB-EGR2       0.044911    3.401991
MYOG-TAF1       0.044860    3.375730
TAF1-MYOG       0.044860    3.375730
ETV1-ZBTB17     0.044858    3.064184
ZBTB17-ETV1     0.044858    3.064184

[3778 rows x 7 columns]
```

To unify the analysis steps between the market basket and the binding distance ones, the parameters for the:

1. *minimal distance*
2. *maximal distance*
3. *maximal allowed overlap*
4. *directionality*

5. *anchor*

will be set automatically when creating the distance object according to the values used for the *market basket analysis* step.

[9]:
```
selection.distObj.max_overlap
```

[9]:
```
0.0
```

2. Distance counting

After creating the distObj, the first step is to count the distances. This will be done with .count_distances(). In this step it is important to decide if the directionality should be taken into account.

If **directionality** is taken into account the position of the transcription factors do matter. This means there is a difference between *TFA -> TFB* and *TFB -> TFA* (compare *Orientation analysis* notebook). Otherwise *TFA -> TFB* and *TFB -> TFA* are the same.

Per default the **directionality** decision **is copied** from the market basket step.

[10]:
```
selection.distObj.directional
```

[10]:
```
False
```

[11]:
```
selection.distObj.count_distances()
```

```
INFO: Preparing to count distances.
INFO: Setting up binding sites for counting
INFO: Calculating distances
INFO: Done finding distances! Results are found in .distances
INFO: You can now run .smooth() and/or .correct_background() to preprocess sites before␣
↪finding peaks.
INFO: Or you can find peaks directly using .analyze_signal_all()
```

The resulting dataframe is constructed as followed: - **columns**: First the transcription factor for the pair (TF1, TF2) followed by the distances in bp - **rows**: each row representing one rule (pair) with the corresponding results

[12]:
```
selection.distObj.distances
```

[12]:
```
                    TF1      TF2  0  1   2   3   4   5   6  7  ...  91  92  \
POU3F2-SMARCA5   POU3F2  SMARCA5  2  0   0   7   0  15   0  1  ...   0   0
SMARCA5-POU3F2  SMARCA5   POU3F2  2  0   0   7   0  15   0  1  ...   0   0
POU2F1-SMARCA5   POU2F1  SMARCA5  0  2   0  39   0   0  12  1  ...   4   0
SMARCA5-POU2F1  SMARCA5   POU2F1  0  2   0  39   0   0  12  1  ...   4   0
SMARCA5-ZNF582  SMARCA5   ZNF582  0  0  38   0  36   1   1  2  ...   0   4
...                ...      ... .. ..  ..  ..  ..  ..  .. .. ...  ..  ..
NFYB-EGR2          NFYB     EGR2  2  0   1   0   0   0   1  0  ...   0   0
MYOG-TAF1          MYOG     TAF1  0  1   1   0   1   0   0  0  ...   0   0
TAF1-MYOG          TAF1     MYOG  0  1   1   0   1   0   0  0  ...   0   0
ETV1-ZBTB17        ETV1   ZBTB17  0  0   0   0   0   0   1  1  ...   0   0
ZBTB17-ETV1      ZBTB17     ETV1  0  0   0   0   0   0   1  1  ...   0   0

                93  94  95  96  97  98  99  100
POU3F2-SMARCA5   7   1   6   0   0   3   0    3
```

(continues on next page)

(continued from previous page)

```
SMARCA5-POU3F2   7   1   6   0   0   3   0    3
POU2F1-SMARCA5   5   0   0   4   0   1   0    1
SMARCA5-POU2F1   5   0   0   4   0   1   0    1
SMARCA5-ZNF582   0   2   0   0   2   0   1    0
...             ..  ..  ..  ..  ..  ..  ..  ...
NFYB-EGR2        0   0   0   0   1   0   0    0
MYOG-TAF1        0   0   0   0   0   0   1    0
TAF1-MYOG        0   0   0   0   0   0   1    0
ETV1-ZBTB17      0   1   1   0   1   0   0    0
ZBTB17-ETV1      0   1   1   0   1   0   0    0

[3778 rows x 103 columns]
```

Additional options for counting distances

Negative distances

Negative distances indicate overlapping. Caveat: this is basepair resolution and strongly dependend on motif length. Negative distance can occur if the distance anchor is set to *inner* mode (see *Anchor mode*) and overlapping is allowed.

Anchor mode

TFCOMB distance analysis supports three different anchor modes: *inner* , *outer* and *center*. The recommended mode is **inner**.

1. inner (default, recommented) is the distance *between* the transcription factors, it is measures as *start(transcription factor B) - end(transcription factor A)*. If for example transcription factor B is directly adjacent to Transcription factor A, the difference will be zero.
2. center is the distance measured from mid of transcription factor to mid of transcription factor
3. outer (uncommonly used) is the distance measured including both transcription factors. *end(transcription factor B) - start(transcription factor A)*

Directionality

Since we didn't count directional, the values for the pairs TF1-TF2 and TF2-TF1 should be equal. For example in the DataFrame above the results for *ZNF121-ZNF770* are the same as for *ZNF770-ZNF121*. This is not true if directionality is considered.

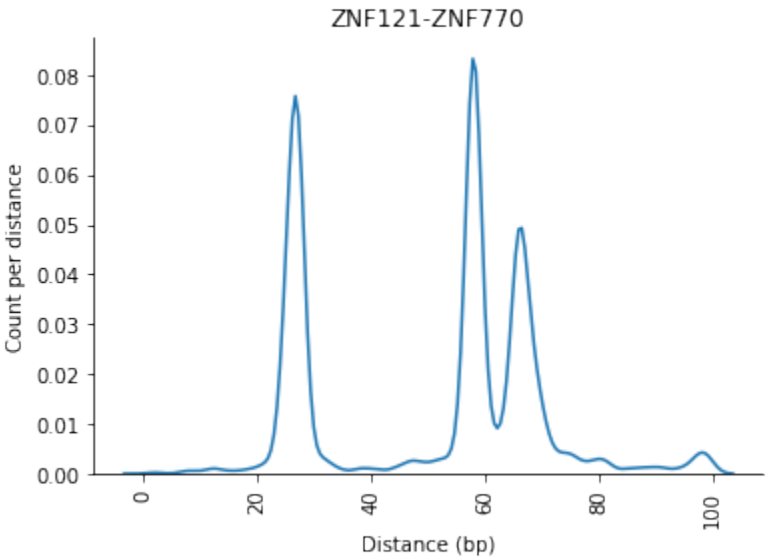
Note: If directionality is not considered, the duplicates can be filtered with .simplify_rules()

Plotting

There are different ways to plot the distance distribution, you can for example create a kernel density estimate (KDE) plot or a histogram of the distribution.

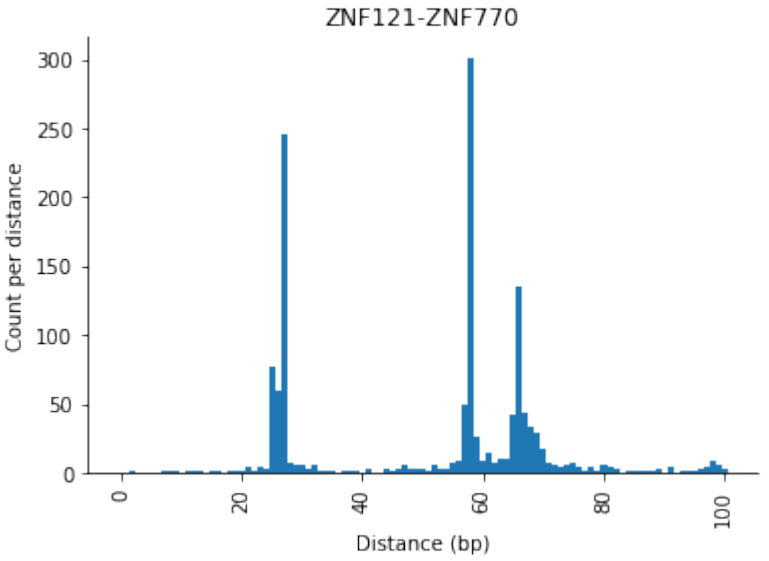
[13]:
```
selection.distObj.plot(("ZNF121", "ZNF770"), style="kde")
```

[13]:
```
<AxesSubplot:title={'center':'ZNF121-ZNF770'}, xlabel='Distance (bp)', ylabel='Count per␣
↪distance'>
```



[14]:
```
selection.distObj.plot(("ZNF121", "ZNF770"), style="hist")
```

[14]:
```
<AxesSubplot:title={'center':'ZNF121-ZNF770'}, xlabel='Distance (bp)', ylabel='Count per␣
↪distance'>
```



For **both** plots the **x-axis** shows the distance in bp.

For example a distance of 100 means the anchors (depending on anchor mode, please refere to *Anchor mode* on this topic) are 100 bp away of each other. *Here* is an explanation for the *neg* distance.

For the **y-axis** of the **kde plot** the density estimation is shown.

For the **y-axis** of the **histogram** the counts per distance is shown.

3. Smoothing counts

In order to collect distances from more than one basepair, e.g. in a window, it is possible to smooth the counted distances. This is done using the function .smooth() of the distObj:

[15]:
```
selection.distObj.smooth(window_size=3)
```

```
INFO: Smoothing signals with window size 3
```

The smoothed (min-max-scaled) distances are visible in the .smoothed variable:

[16]:
```
selection.distObj.smoothed.head()
```

[16]:

	TF1	TF2	1	2	3	4	\
POU3F2-SMARCA5	POU3F2	SMARCA5	0.666667	2.333333	2.333333	7.333333	
SMARCA5-POU3F2	SMARCA5	POU3F2	0.666667	2.333333	2.333333	7.333333	
POU2F1-SMARCA5	POU2F1	SMARCA5	0.666667	13.666667	13.000000	13.000000	
SMARCA5-POU2F1	SMARCA5	POU2F1	0.666667	13.666667	13.000000	13.000000	
SMARCA5-ZNF582	SMARCA5	ZNF582	12.666667	12.666667	24.666667	12.333333	

	5	6	7	8	...	90	\
POU3F2-SMARCA5	5.000000	5.333333	10.333333	10.333333	...	1.666667	
SMARCA5-POU3F2	5.000000	5.333333	10.333333	10.333333	...	1.666667	
POU2F1-SMARCA5	4.000000	4.333333	5.333333	1.333333	...	1.333333	
SMARCA5-POU2F1	4.000000	4.333333	5.333333	1.333333	...	1.333333	
SMARCA5-ZNF582	12.666667	1.333333	1.333333	3.333333	...	1.333333	

	91	92	93	94	95	96	\
POU3F2-SMARCA5	1.666667	2.333333	2.666667	4.666667	2.333333	2.000000	
SMARCA5-POU3F2	1.666667	2.333333	2.666667	4.666667	2.333333	2.000000	
POU2F1-SMARCA5	1.333333	3.000000	1.666667	1.666667	1.333333	1.333333	
SMARCA5-POU2F1	1.333333	3.000000	1.666667	1.666667	1.333333	1.333333	
SMARCA5-ZNF582	1.333333	1.333333	2.000000	0.666667	0.666667	0.666667	

	97	98	99
POU3F2-SMARCA5	1.000000	1.000000	2.000000
SMARCA5-POU3F2	1.000000	1.000000	2.000000
POU2F1-SMARCA5	1.666667	0.333333	0.666667
SMARCA5-POU2F1	1.666667	0.333333	0.666667
SMARCA5-ZNF582	0.666667	1.000000	0.333333

[5 rows x 101 columns]

4. Scale counts (optional)

Optionally, it is possible to scale the counts to be in the same ranges regardless of number of counts per pair.

[17]:
```
selection_copy = selection.copy() #create a copy to not alter the real selection object
selection_copy.distObj.scale()
```

[18]:
```
selection_copy.distObj.scaled.head()
```

[18]:
```
                    TF1      TF2         1         2         3      4  \
POU3F2-SMARCA5   POU3F2  SMARCA5  0.017857  0.107143  0.107143  0.375
SMARCA5-POU3F2  SMARCA5   POU3F2  0.017857  0.107143  0.107143  0.375
POU2F1-SMARCA5   POU2F1  SMARCA5  0.025000  1.000000  0.950000  0.950
SMARCA5-POU2F1  SMARCA5   POU2F1  0.025000  1.000000  0.950000  0.950
SMARCA5-ZNF582  SMARCA5   ZNF582  0.513514  0.513514  1.000000  0.500

                       5         6         7         8  ...        90  \
POU3F2-SMARCA5  0.250000  0.267857  0.535714  0.535714  ...  0.071429
SMARCA5-POU3F2  0.250000  0.267857  0.535714  0.535714  ...  0.071429
POU2F1-SMARCA5  0.275000  0.300000  0.375000  0.075000  ...  0.075000
SMARCA5-POU2F1  0.275000  0.300000  0.375000  0.075000  ...  0.075000
SMARCA5-ZNF582  0.513514  0.054054  0.054054  0.135135  ...  0.054054

                      91        92        93        94        95        96  \
POU3F2-SMARCA5  0.071429  0.107143  0.125000  0.232143  0.107143  0.089286
SMARCA5-POU3F2  0.071429  0.107143  0.125000  0.232143  0.107143  0.089286
POU2F1-SMARCA5  0.075000  0.200000  0.100000  0.100000  0.075000  0.075000
SMARCA5-POU2F1  0.075000  0.200000  0.100000  0.100000  0.075000  0.075000
SMARCA5-ZNF582  0.054054  0.054054  0.081081  0.027027  0.027027  0.027027

                      97        98        99
POU3F2-SMARCA5  0.035714  0.035714  0.089286
SMARCA5-POU3F2  0.035714  0.035714  0.089286
POU2F1-SMARCA5  0.100000  0.000000  0.025000
SMARCA5-POU2F1  0.100000  0.000000  0.025000
SMARCA5-ZNF582  0.027027  0.040541  0.013514

[5 rows x 101 columns]
```

5. Background correction

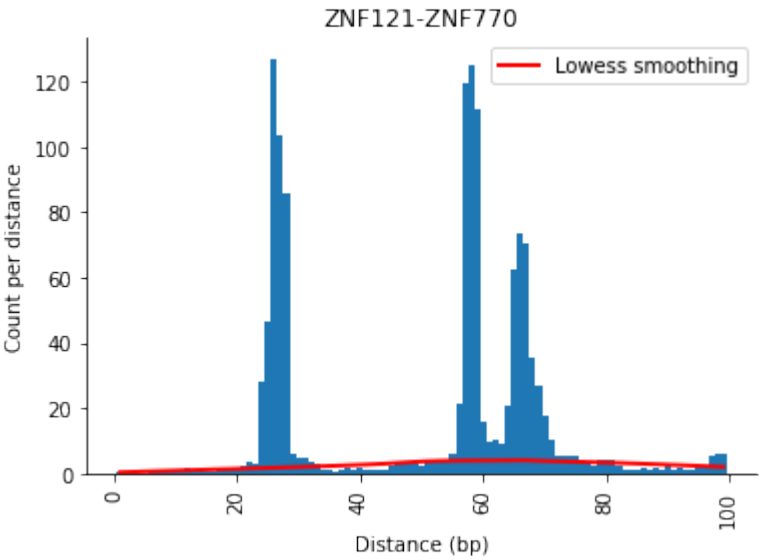
To separate the signal from the background noise a linear regression for every signal needs to be fitted next. The result of the fitted line for every pair can be found in the .linres attribute of the distance object.

[19]:
```
selection.distObj.correct_background(threads=6)
```

```
INFO: Background correction finished! Results can be found in .corrected
```

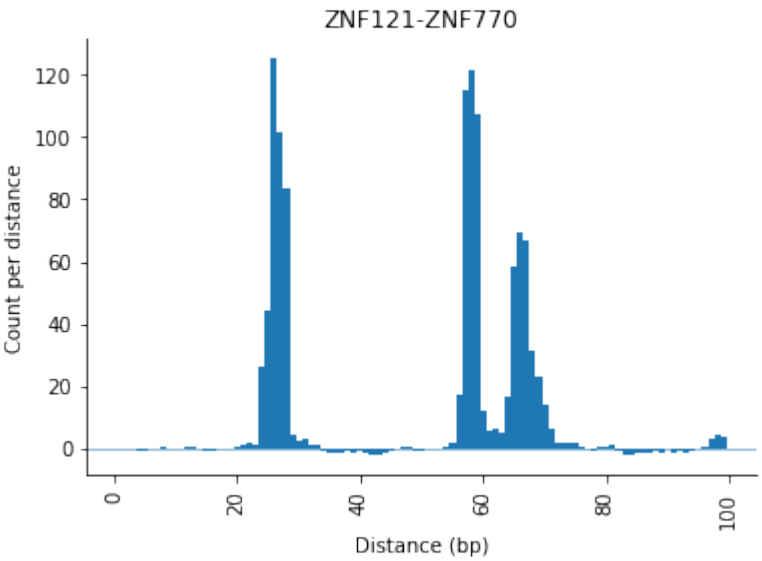
The fitted lowess function can be plotted with the .plot() command. The **red line** indicates the linear regression line which was fitted during this step.

[20]:
```
_ = selection.distObj.plot(("ZNF121", "ZNF770"), method="correction")
```



After correction, the plotting function will show the corrected counts:

[21]:
```
_ = selection.distObj.plot(("ZNF121", "ZNF770"))
```



6. Analyse signal

As a last step, the corrected signal can now be analyzed. Peaks will be called with *scipy.signal.find_peaks()*. Two different methods are available:

1. "zscore"
2. "flat" (number)

Zscore method

If prominence is set to **zscore**, the threshold is set in relation to the zscore (of the corrected signal) for each signal. For example if threshold is 2, the threshold prominence (for the score translated signal) will be a zscore of two.

[22]:
```
selection.distObj.analyze_signal_all(method="zscore", threshold=2, threads=6)
selection.distObj.peaks
```

```
INFO: Analyzing Signal with threads 6
INFO: Calculating zscores for signals
INFO: Finding preferred distances
INFO: Done analyzing signal. Results are found in .peaks
```

[22]:
```
                    TF1      TF2  Distance  Peak Heights  Prominences  \
ZFP82-SMARCA5     ZFP82  SMARCA5         8      6.547058     7.582435
SMARCA5-ZFP82   SMARCA5    ZFP82         8      6.547058     7.582435
ZNF394-SMARCA5   ZNF394  SMARCA5         5      6.940786     7.474373
SMARCA5-ZNF394  SMARCA5   ZNF394         5      6.940786     7.474373
POU3F2-SMARCA5   POU3F2  SMARCA5         9      6.584317     7.402690
...                 ...      ...       ...           ...          ...
ETV5-ZSCAN22       ETV5  ZSCAN22        98      2.010752     2.010752
ZBTB17-RFX1      ZBTB17     RFX1         1      2.010639     2.010639
RFX1-ZBTB17        RFX1   ZBTB17         1      2.010639     2.010639
WT1-ZIC3            WT1     ZIC3        48      2.226574     2.005542
ZIC3-WT1            ZIC3      WT1       48      2.226574     2.005542

                Threshold  TF1_TF2_count  Distance_count  Distance_percent  \
ZFP82-SMARCA5           2            198              46         23.232323
SMARCA5-ZFP82           2            198              46         23.232323
ZNF394-SMARCA5          2            187              74         39.572193
SMARCA5-ZNF394          2            187              74         39.572193
POU3F2-SMARCA5          2            234              57         24.358974
...                   ...            ...             ...               ...
ETV5-ZSCAN22            2             64               3          4.687500
ZBTB17-RFX1             2             30               1          3.333333
RFX1-ZBTB17             2             30               1          3.333333
WT1-ZIC3                2             51               4          7.843137
ZIC3-WT1                2             51               4          7.843137

               Distance_window
ZFP82-SMARCA5            [7;9]
SMARCA5-ZFP82            [7;9]
ZNF394-SMARCA5           [4;6]
SMARCA5-ZNF394           [4;6]
POU3F2-SMARCA5          [8;10]
...                        ...
ETV5-ZSCAN22           [97;99]
ZBTB17-RFX1              [1;2]
RFX1-ZBTB17              [1;2]
WT1-ZIC3                [47;49]
ZIC3-WT1                [47;49]

[8160 rows x 10 columns]
```

The resulting dataframe is constructed as followed: - **columns**: First the transcription factor for the pair (TF1, TF2)

followed by

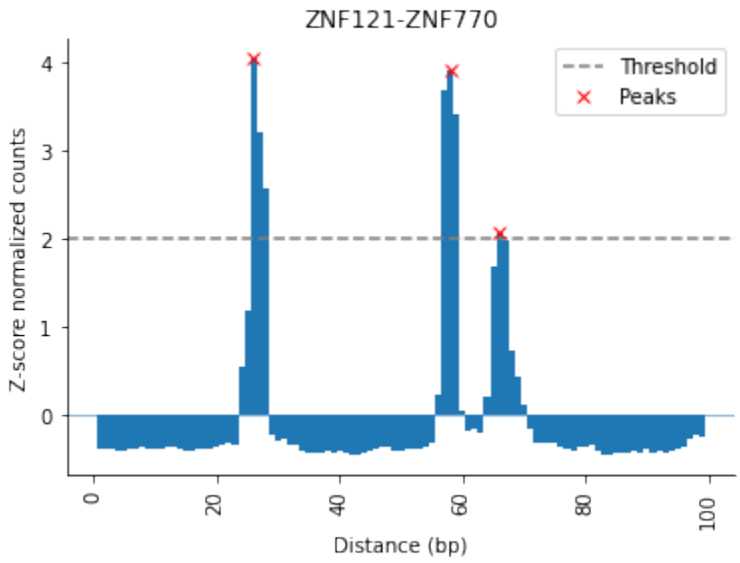
```
1. __Distance__ in bp \
2. __Peak Heights__ height of the peak (calculated by [_find_peaks()_](https://docs.
↪scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html)). \
3. __Prominences__ prominence of the peak (calculated by [_find_peaks()_](https://docs.
↪scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html)). \
4. __Threshold__ minimum prominence needed to be considered as peak (in this example the␣
↪[zscore](#5.B.-Zscore) of the signal). \
5. __TF1_TF2_count__ is the total number of co-occurring TF1-TF2 sites. \
6. __Distance_count__ The number of co-occurrences for the given distance. \
7. __Distance_percent__ The percent of Distance_count of the total sites (TF1_TF2_count).
↪ \
8. __Distance_window__ The distances collected for the given distance. Will be a window␣
↪if smoothing was applied. \
```

- **rows**: each row representing one rule (pair) at a distinct prefered binding distance with the corresponding results

The signal and peaks can now be plotted with the .plot() command.

[23]:
```
selection.distObj.plot(("ZNF121", "ZNF770"))
```

[23]:
```
<AxesSubplot:title={'center':'ZNF121-ZNF770'}, xlabel='Distance (bp)', ylabel='Z-score␣
↪normalized counts'>
```



This plot shows the **corrected** signal with the **called peaks**, indicated by a cross.

The **grey dottet line** shows the decision boundary (in this example a zscore of 2)

[24]:
```
selection.distObj.peaks.loc[(selection.distObj.peaks.TF1 == "ZNF121") & (selection.
↪distObj.peaks.TF2 == "ZNF770")]
```

[24]:

	TF1	TF2	Distance	Peak Heights	Prominences	Threshold	\
ZNF121-ZNF770	ZNF121	ZNF770	26	4.043651	4.443567	2	
ZNF121-ZNF770	ZNF121	ZNF770	58	3.906961	4.353073	2	
ZNF121-ZNF770	ZNF121	ZNF770	66	2.068096	2.270661	2	

(continues on next page)

(continued from previous page)

```
               TF1_TF2_count  Distance_count  Distance_percent Distance_window
ZNF121-ZNF770           1319             381         28.885519         [25;27]
ZNF121-ZNF770           1319             376         28.506444         [57;59]
ZNF121-ZNF770           1319             220         16.679303         [65;67]
```

The results in the .peaks table and the plot matches. For this *ZNF121-ZNF770* 3 peaks were found, which also can be seen in the plot at positions:

1. distance: 26
2. distance: 58
3. distance: 66

Meaning the zscore threshold with stringency of 2 identifies 3 different preferred distances for this pair. Please note, that the height of the distances differ!

Flat threshold

Using a flat threshold, it is possible to only find peaks with a certain number of counts as seen here:

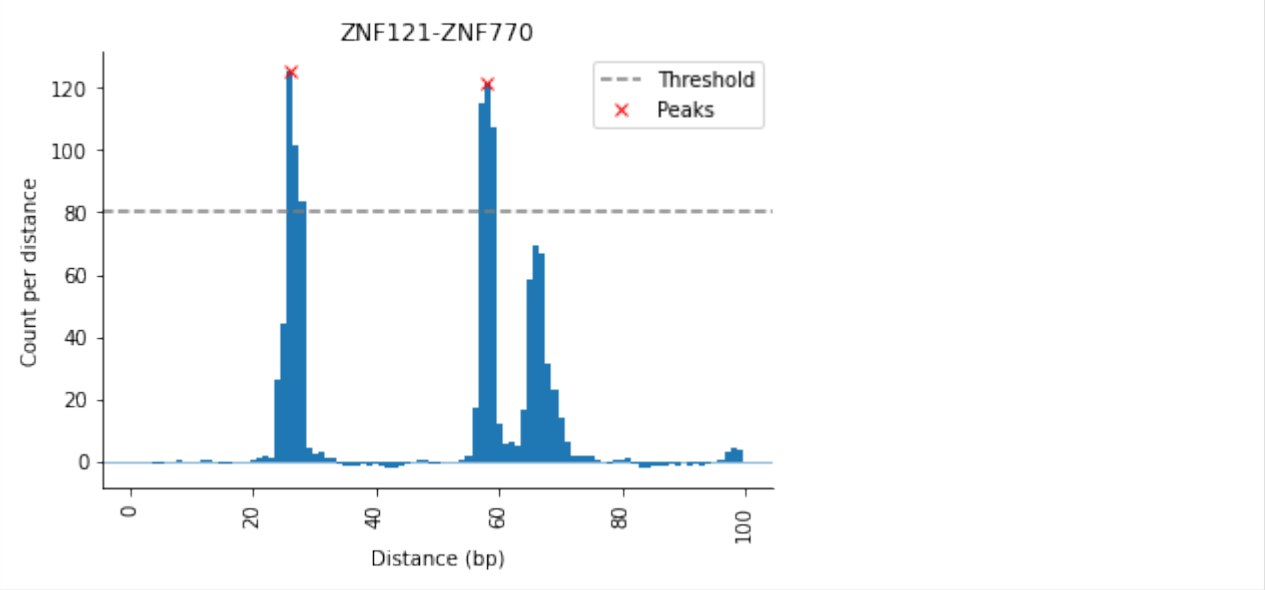
[25]:
```
selection.distObj.analyze_signal_all(method="flat", threshold=80, threads=6)
selection.distObj.peaks
```

```
INFO: Analyzing Signal with threads 6
INFO: Calculating zscores for signals
INFO: Finding preferred distances
INFO: Done analyzing signal. Results are found in .peaks
```

[25]:
```
                 TF1     TF2  Distance  Peak Heights  Prominences  Threshold  \
ZNF121-ZNF770  ZNF121  ZNF770        26    125.175641   125.839985         80
ZNF770-ZNF121  ZNF770  ZNF121        26    125.175641   125.839985         80
ZNF121-ZNF770  ZNF121  ZNF770        58    121.304647   123.277229         80
ZNF770-ZNF121  ZNF770  ZNF121        58    121.304647   123.277229         80
PAX5-ZNF770      PAX5  ZNF770        56    104.564327   106.526327         80
ZNF770-PAX5    ZNF770    PAX5        56    104.564327   106.526327         80

               TF1_TF2_count  Distance_count  Distance_percent Distance_window
ZNF121-ZNF770           1319             381         28.885519         [25;27]
ZNF770-ZNF121           1319             381         28.885519         [25;27]
ZNF121-ZNF770           1319             376         28.506444         [57;59]
ZNF770-ZNF121           1319             376         28.506444         [57;59]
PAX5-ZNF770              896             325         36.272321         [55;57]
ZNF770-PAX5              896             325         36.272321         [55;57]
```

[26]:
```
selection.distObj.plot(("ZNF121", "ZNF770"))
```

[26]:
```
<AxesSubplot:title={'center':'ZNF121-ZNF770'}, xlabel='Distance (bp)', ylabel='Count per␣
↪distance'>
```



Further downstream analysis

1. Analyzing hubs

This function allows to summarize the number of different partners (with at least one peak) each transcription factor has.

[27]:
```
selection.distObj.analyze_hubs().sort_values()
```

[27]:
```
PAX5      1
ZNF121    1
ZNF770    2
dtype: int64
```

2. Signal classification

This function allows to classify the pairs (in .distance DataFrame) wheather the signal is peaking or not.

[28]:
```
selection.distObj.classify_rules()
selection.distObj.classified.sort_values(by="isPeaking")
```

```
INFO: classifying rules
INFO: classifcation done. Results can be found in .classified
```

[28]:
```
                    TF1      TF2  isPeaking
POU3F2-SMARCA5  POU3F2  SMARCA5      False
ELK4-WT1          ELK4      WT1      False
WT1-ELK4           WT1     ELK4      False
ELF2-SP1          ELF2      SP1      False
SP1-ELF2           SP1     ELF2      False
...                ...      ...        ...
```

(continues on next page)

(continued from previous page)

```
ZBTB17-ETV1     ZBTB17     ETV1      False
ZNF121-ZNF770   ZNF121   ZNF770       True
PAX5-ZNF770       PAX5   ZNF770       True
ZNF770-PAX5     ZNF770     PAX5       True
ZNF770-ZNF121   ZNF770   ZNF121       True

[3778 rows x 3 columns]
```

True means the signal has at least one peak found, False otherwise.

1.2.9 GO-term analysis

TF-COMB contains functionality for performing GO-term analysis on gene lists as obtained e.g. from TFBS annotation.

Obtain a list of genes from previous analysis

We will use a gene list containing genes upregulated in response to hypoxia (source msigdb: https://www.gsea-msigdb.org/gsea/msigdb/cards/HALLMARK_HYPOXIA.html)

[1]:
```
with open("../data/gene_list.txt") as f:
    genes = f.read().split("\n")
```

[2]:
```
genes[:10]
```

[2]:
```
['ACKR3',
 'ADM',
 'ADORA2B',
 'AK4',
 'AKAP12',
 'ALDOA',
 'ALDOB',
 'ALDOC',
 'AMPD3',
 'ANGPTL4']
```

Given a list of genes, you can use the GOAnalysis class to perform a GO-term analysis of the genes:

[3]:
```
from tfcomb.annotation import GOAnalysis
go_table = GOAnalysis().enrichment(genes)
```

```
INFO: Running GO-term enrichment for organism 'hsapiens' (taxid: 9606)
go-basic.obo: fmt(1.2) rel(2022-05-16) 47,071 GO Terms
HMS:0:00:05.707700 349,061 annotations, 20,717 genes, 18,963 GOs, 1 taxids READ: gene2go
INFO: Setting up gene ids
INFO: Setting up GO enrichment

Load BP Gene Ontology Analysis ...
Propagating term counts up: is_a

Load CC Gene Ontology Analysis ...
Propagating term counts up: is_a
```

(continues on next page)

(continued from previous page)

```
Load MF Gene Ontology Analysis ...
Propagating term counts up: is_a
```

The results are seen in the returned table:

[4]:
```
go_table.head()
```

[4]:
```
            GO                                          name  NS  depth  \
36  GO:0005996               monosaccharide metabolic process  BP      4
31  GO:0006091  generation of precursor metabolites and energy  BP      3
37  GO:0006006                     glucose metabolic process  BP      6
11  GO:0016052                carbohydrate catabolic process  BP      4
21  GO:0009141       nucleoside triphosphate metabolic process  BP      7

   enrichment ratio_in_study ratio_in_pop  p_uncorrected  p_fdr_bh  \
36  increased         35/200    174/19351   2.699871e-07  0.000111
31  increased         32/200    365/19351   2.661620e-07  0.000111
37  increased         29/200    113/19351   2.737493e-07  0.000111
11  increased         26/200    105/19351   1.366245e-07  0.000111
21  increased         23/200    203/19351   2.011868e-07  0.000111

    study_count                                        study_items
36           35  ALDOA, ALDOB, ALDOC, ATF3, BRS3, ENO1, ENO2, E...
31           32  ALDOA, ALDOB, ALDOC, ENO1, ENO2, ENO3, GAA, GA...
37           29  ALDOC, ATF3, BRS3, ENO1, ENO2, ENO3, FBP1, GAA...
11           26  ALDOA, ALDOB, ALDOC, ENO1, ENO2, ENO3, GAA, GA...
21           23  AK4, ALDOA, ALDOB, ALDOC, ENO1, ENO2, ENO3, GA...
```

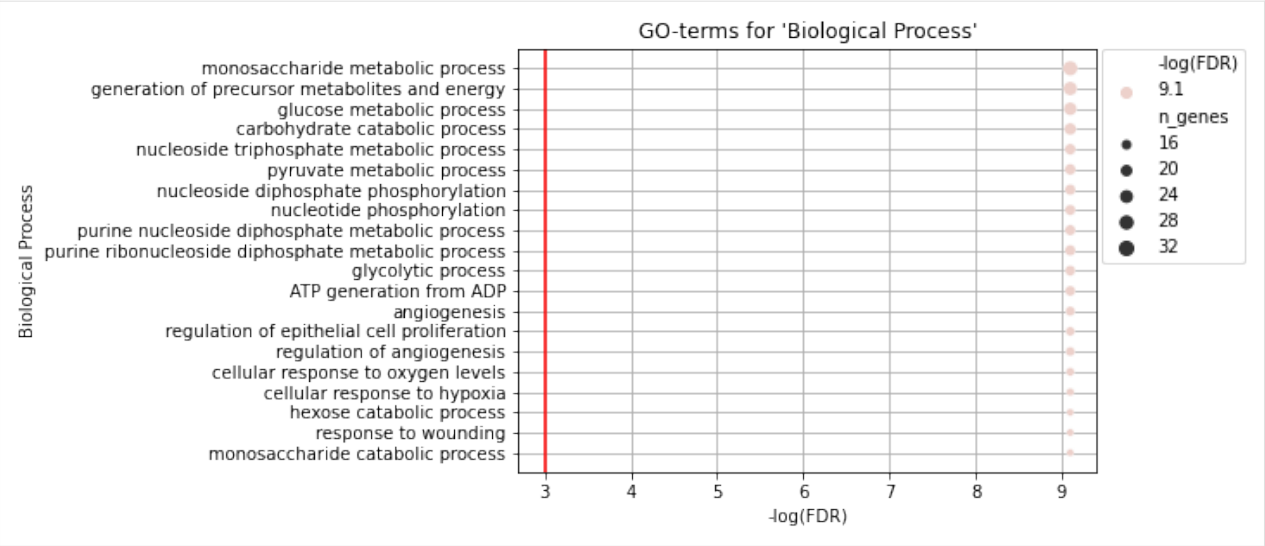
Plotting the enriched terms

The returned `go_table` is a subclass of pandas.DataFrame, which contains options for plotting the GO-enrichment results as seen here:

[5]:
```
type(go_table)
```

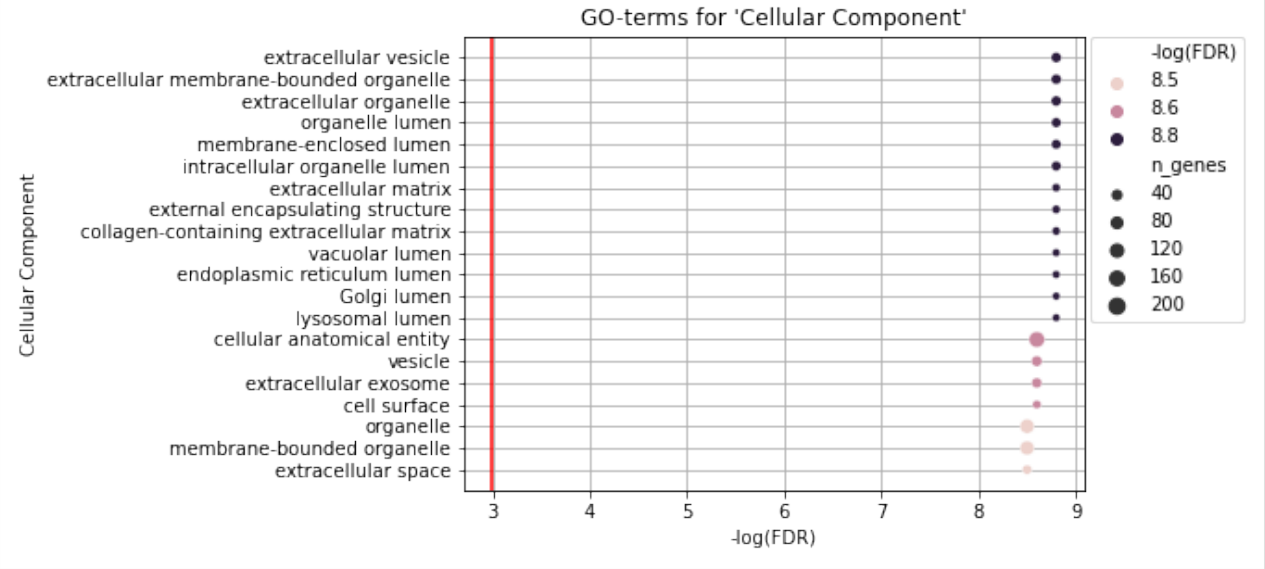
[5]:
```
tfcomb.annotation.GOAnalysis
```

[6]:
```
_ = go_table.plot_bubble()
```

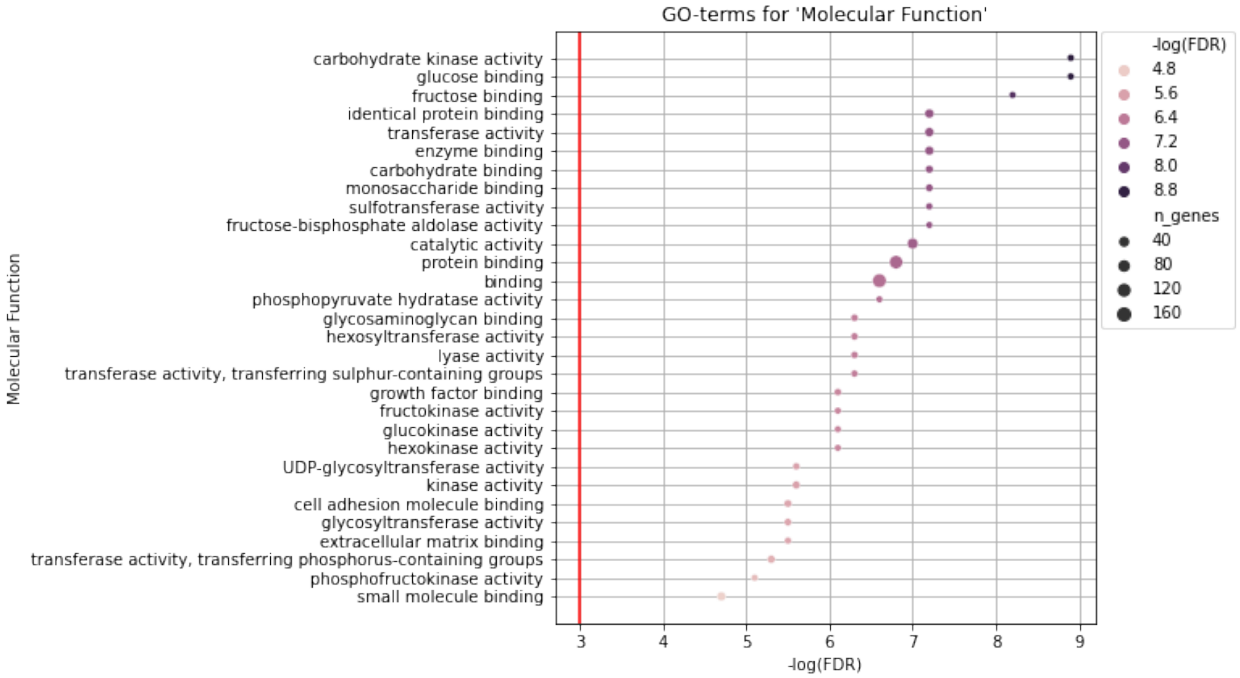


The default aspect shown is “BP” (Biological Process), but by setting `aspect`, either of “CC” (Cellular Component) and “MF” (Molecular Function can be shown:

[7]:
```
_ = go_table.plot_bubble(aspect="CC")
```



[8]:
```
_ = go_table.plot_bubble(aspect="MF", n_terms=30)
```



[ ]:

1.2.10 Genomic locations of TF-TF pairs

In this notebook, we will go over how to get the locations of two TFs co-occurring. We will start by creating a TF-COMB analysis from motif positions:

[1]:
```
import tfcomb

C = tfcomb.CombObj()
C.TFBS_from_motifs(regions="../data/GM12878_hg38_chr4_ATAC_peaks.bed",
                   motifs="../data/HOCOMOCOv11_HUMAN_motifs.txt",
                   genome="../data/hg38_chr4.fa.gz",
                   threads=4)
C.market_basket()
```

```
INFO: Scanning for TFBS with 4 thread(s)...
INFO: Progress: 12%
INFO: Progress: 20%
INFO: Progress: 30%
INFO: Progress: 40%
INFO: Progress: 50%
INFO: Progress: 60%
INFO: Progress: 70%
INFO: Progress: 80%
INFO: Progress: 91%
INFO: Finished!
INFO: Processing scanned TFBS
INFO: Identified 165810 TFBS (401 unique names) within given regions
```

(continues on next page)

(continued from previous page)

```
Internal counts for 'TF_counts' were not set. Please run .count_within() to obtain TF-TF␣
↪co-occurrence counts.
WARNING: No counts found in <CombObj>. Running <CombObj>.count_within() with standard␣
↪parameters.
INFO: Setting up binding sites for counting
INFO: Counting co-occurrences within sites
INFO: Counting co-occurrence within background
INFO: Running with multiprocessing threads == 1. To change this, give 'threads' in the␣
↪parameter of the function.
INFO: Progress: 10%
INFO: Progress: 20%
INFO: Progress: 30%
INFO: Progress: 40%
INFO: Progress: 50%
INFO: Progress: 60%
INFO: Progress: 70%
INFO: Progress: 80%
INFO: Progress: 90%
INFO: Done finding co-occurrences! Run .market_basket() to estimate significant pairs
INFO: Market basket analysis is done! Results are found in <CombObj>.rules
```

[2]:
```
C.rules.head()
```

[2]:

	TF1	TF2	TF1_TF2_count	TF1_count	TF2_count	cosine	zscore
POU3F2-SMARCA5	POU3F2	SMARCA5	239	302	241	0.885902	129.586528
SMARCA5-POU3F2	SMARCA5	POU3F2	239	241	302	0.885902	129.586528
POU2F1-SMARCA5	POU2F1	SMARCA5	263	426	241	0.820810	135.355691
SMARCA5-POU2F1	SMARCA5	POU2F1	263	241	426	0.820810	135.355691
SMARCA5-ZNF582	SMARCA5	ZNF582	172	241	195	0.793419	117.370387

Getting locations for a selected TF-TF pair

We choose the highest ranking TF pair from the .rules:

[3]:
```
TF1, TF2 = C.rules.iloc[0, [0,1]]
TF1, TF2
```

[3]:
```
('POU3F2', 'SMARCA5')
```

We can now apply *get_pair_locations()* to get the locations of the TF-TF pairs

[4]:
```
pairs = C.get_pair_locations((TF1, TF2))
```

[5]:
```
pairs[:10]
```

[5]:
```
TFBSPairList([<TFBSPair | TFBS1: (chr4,49092715,49092730,SMARCA5,13.72436,-) | TFBS2:␣
↪(chr4,49092743,49092775,POU3F2,11.25323,+) | distance: 13 | orientation: divergent >,
↪<TFBSPair | TFBS1: (chr4,49092715,49092730,SMARCA5,13.72436,-) | TFBS2: (chr4,49092788,
↪49092865,POU3F2,11.46462,+) | distance: 58 | orientation: divergent >, <TFBSPair |␣
↪TFBS1: (chr4,49092743,49092775,POU3F2,11.25323,+) | TFBS2: (chr4,49092785,49092880,
↪SMARCA5,15.00425,-) | distance: 10 | orientation: convergent >, <TFBSPair | TFBS1:␣
↪(chr4,49092745,49092780,SMARCA5,11.81125,-) | TFBS2: (chr4,49092788,49092865,POU3F2,11.
↪46462,+) | distance: 8 | orientation: divergent >, <TFBSPair | TFBS1: (chr4,49092785,
↪49092880,SMARCA5,15.00425,-) | TFBS2: (chr4,49092893,49092930,POU3F2,11.46462,+) |␣
↪distance: 13 | orientation: divergent >, <TFBSPair | TFBS1: (chr4,49092788,49092865,
↪POU3F2,11.46462,+) | TFBS2: (chr4,49092885,49092930,SMARCA5,12.69622,-) | distance: 20␣
↪| orientation: convergent >, <TFBSPair | TFBS1: (chr4,49096721,49096746,SMARCA5,12.
↪97124,-) | TFBS2: (chr4,49096779,49096836,POU3F2,11.46462,+) | distance: 33 |␣
↪orientation: divergent >, <TFBSPair | TFBS1: (chr4,49096721,49096746,SMARCA5,12.97124,-
↪) | TFBS2: (chr4,49096839,49096896,POU3F2,11.63189,+) | distance: 93 | orientation:␣
↪divergent >, <TFBSPair | TFBS1: (chr4,49096744,49096761,POU3F2,10.58175,+) | TFBS2:␣
↪(chr4,49096771,49096836,SMARCA5,13.02056,-) | distance: 10 | orientation: convergent >,
↪ <TFBSPair | TFBS1: (chr4,49096744,49096761,POU3F2,10.58175,+) | TFBS2: (chr4,49096841,
↪49096916,SMARCA5,13.70506,-) | distance: 80 | orientation: convergent >])
```

We can write these locations to a file:

[6]:
```
pairs.write_bed("TFBS_pair_positions.bed", fmt="bed")

#show the content of file
import pandas as pd
pd.read_csv("TFBS_pair_positions.bed", sep="\t", header=None, nrows=5)
```

[6]:

	0	1	2	3	4	5
0	chr4	49092715	49092730	SMARCA5	.	-
1	chr4	49092743	49092775	POU3F2	.	+
2	chr4	49092715	49092730	SMARCA5	.	-
3	chr4	49092788	49092865	POU3F2	.	+
4	chr4	49092743	49092775	POU3F2	.	+

The pairs can also be written out as 'bedpe' format, which contains the positions of both sites:

[7]:
```
pairs.write_bed("TFBS_pair_positions.bedpe", fmt="bedpe")

pd.read_csv("TFBS_pair_positions.bedpe", sep="\t", header=None, nrows=5)
```

[7]:

	0	1	2	3	4	5	6	7	8	9
0	chr4	49092715	49092730	chr4	49092743	49092775	SMARCA5-POU3F2	13	-	+
1	chr4	49092715	49092730	chr4	49092788	49092865	SMARCA5-POU3F2	58	-	+
2	chr4	49092743	49092775	chr4	49092785	49092880	POU3F2-SMARCA5	10	+	-
3	chr4	49092745	49092780	chr4	49092788	49092865	SMARCA5-POU3F2	8	-	+
4	chr4	49092785	49092880	chr4	49092893	49092930	SMARCA5-POU3F2	13	-	+

1.2.11 Plot TFBS in genome view

We will first plot the TFBS sites from ChIP-seq of GM12878 as loaded previously:

[1]:
```
import tfcomb
C = tfcomb.CombObj().from_pickle("../data/GM12878.pkl")
```

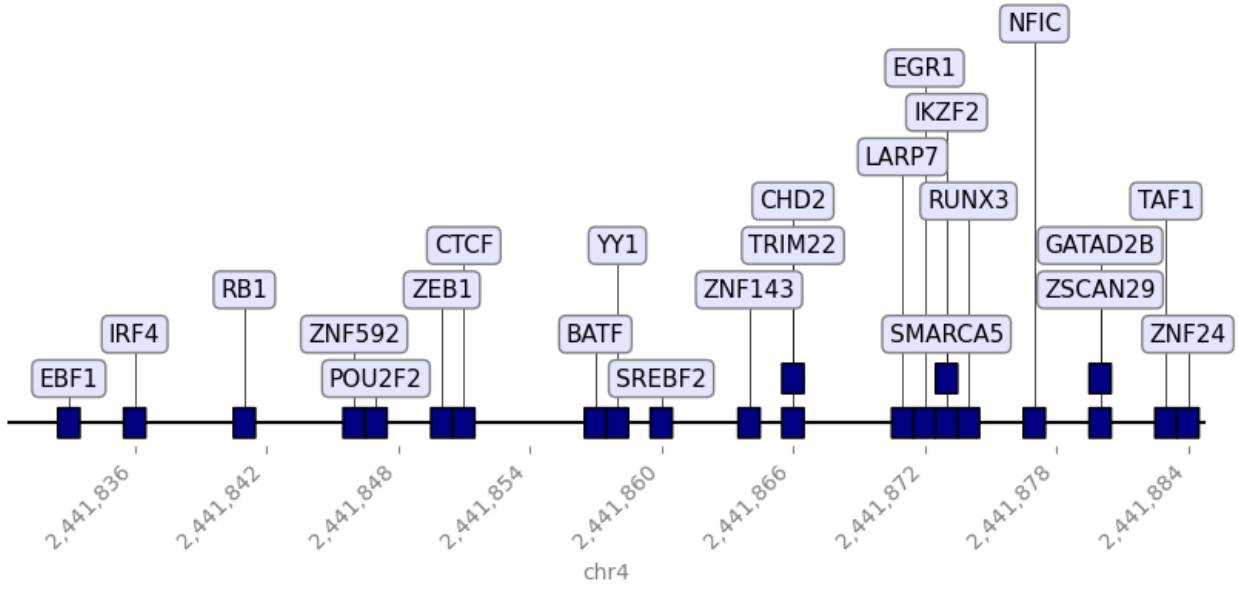
This object already contains TFBS to show:

[2]:
```
C.TFBS[:10]
```

[2]:
```
[chr4   11875   11876   ZBTB33  1000    .,
 chr4   116639  116640  JUNB    974     .,
 chr4   116678  116679  RUNX3   1000    .,
 chr4   121620  121621  RUNX3   1000    .,
 chr4   124050  124051  ZNF217  948     .,
 chr4   124052  124053  SMARCA5 802     .,
 chr4   124169  124170  NR2F1   634     .,
 chr4   124289  124290  CBX5    906     .,
 chr4   124363  124364  E4F1    1000    .,
 chr4   124365  124366  PKNOX1  1000    .]
```

Now, using *.plot_TFBS*, we can show the TFBS in a certain window:

[3]:
```
C.plot_TFBS(window_chrom = "chr4",
            window_start = 2441831,
            window_end = 2441885)
```



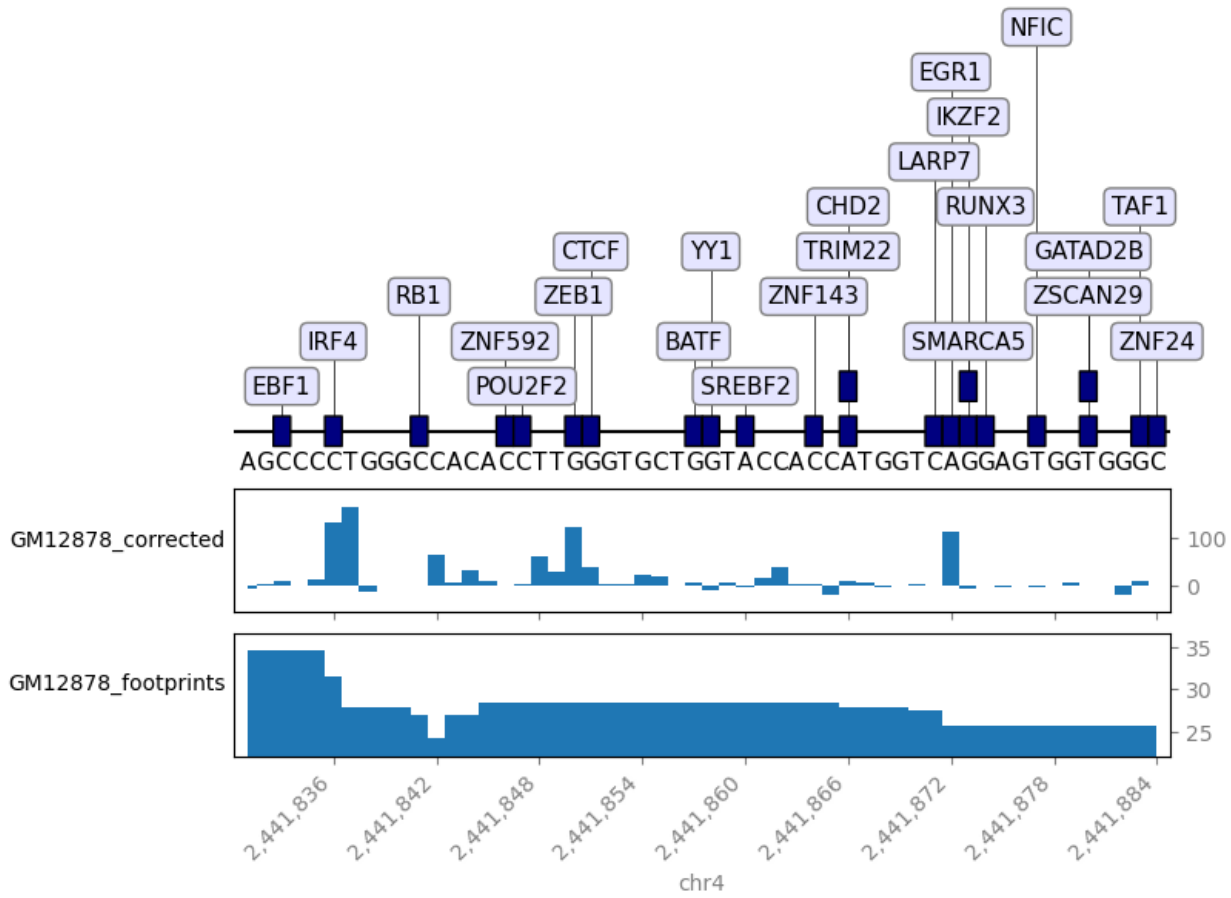
The function can also take *fasta* and *bigwigs* to show additional tracks below the TFBS:

[4]:
```
C.plot_TFBS(window_chrom = "chr4",
            window_start = 2441831,
            window_end = 2441885,
            fasta="../data/hg38_chr4.fa.gz",
```

(continues on next page)

(continued from previous page)

```
            bigwigs = ["../data/GM12878_corrected.bw",
                       "../data/GM12878_footprints.bw"])
```

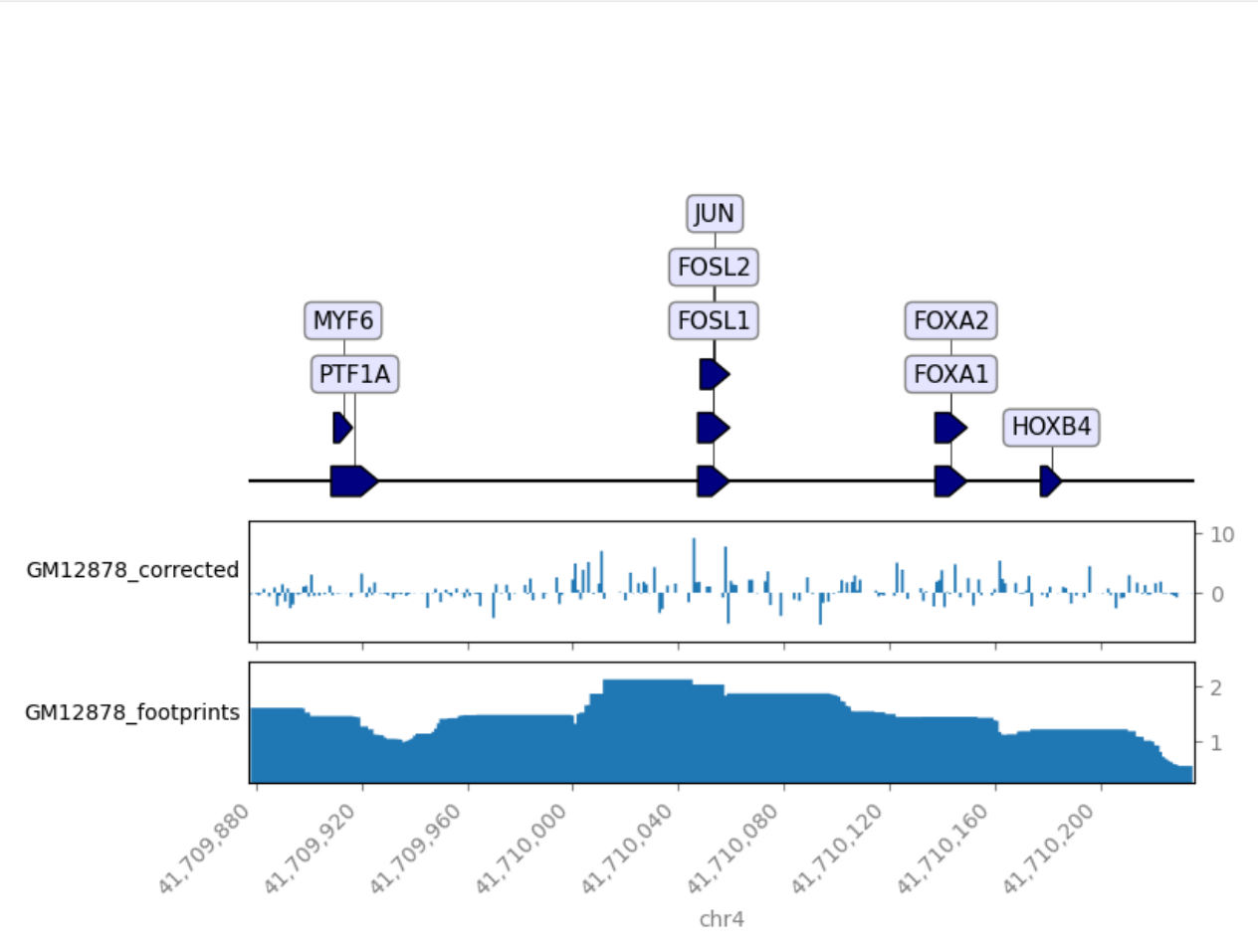


Plot TFBS from motifs

For TFBS with strand information, e.g. from motifs, *.plot_TFBS* will show the direction of the TFBS as arrows:

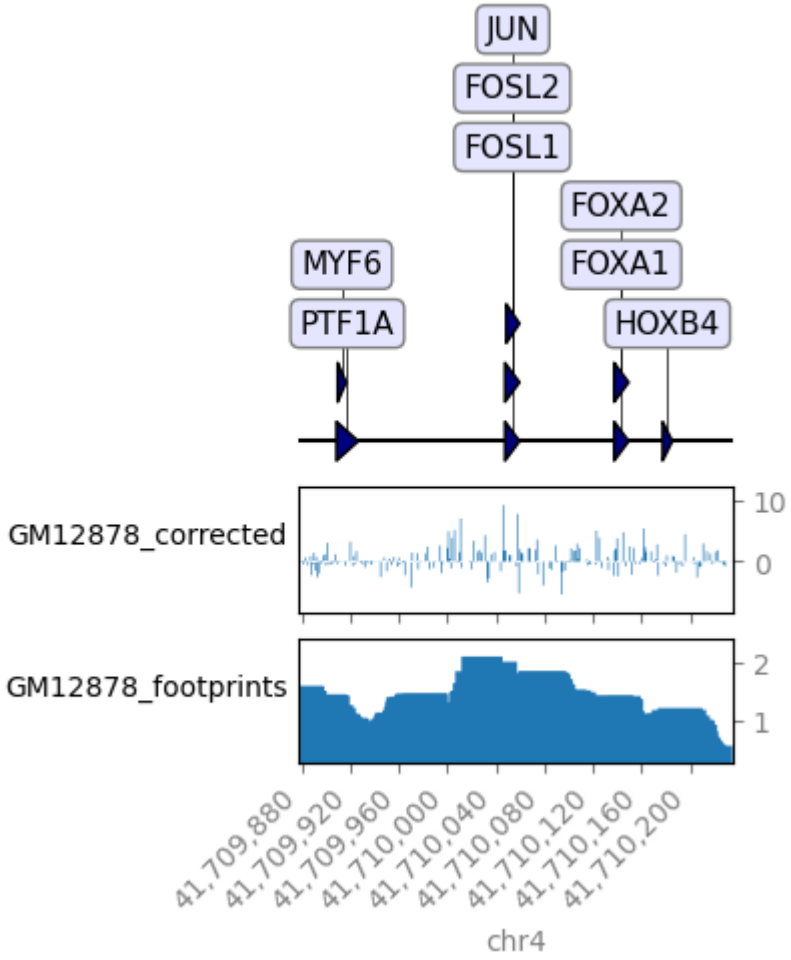
[5]:
```
C = tfcomb.CombObj(verbosity=0)
C.TFBS_from_motifs(regions="../data/GM12878_hg38_chr4_ATAC_peaks.bed",
                   motifs="../data/HOCOMOCOv11_HUMAN_motifs.txt",
                   genome="../data/hg38_chr4_masked.fa.gz",
                   threads=8)
```

[6]:
```
C.plot_TFBS(window_chrom = "chr4",
            window_start = 41709878,
            window_end = 41710236,
            bigwigs = ["../data/GM12878_corrected.bw",
                       "../data/GM12878_footprints.bw"])
```



Set figsize to control size of plot (the default is to estimate it from the number of tracks)

[7]:
```
C.plot_TFBS(window_chrom = "chr4",
            window_start = 41709878,
            window_end = 41710236,
            bigwigs = ["../data/GM12878_corrected.bw",
                       "../data/GM12878_footprints.bw"],
            figsize=(4,5))
```



1.2.12 Plot footprints for TF pair locations

Calculate TF positions from motifs

We calculate the location of TFBS by scanning the genome for motif positions, and run market basket analysis to find co-occurring TFs:

[1]:
```
import tfcomb
C = tfcomb.CombObj()
C.TFBS_from_motifs(regions="../data/GM12878_hg38_chr4_ATAC_peaks.bed",
                   motifs="../data/HOCOMOCOv11_HUMAN_motifs.txt",
                   genome="../data/hg38_chr4_masked.fa.gz",
                   resolve_overlapping="highest_score",
                   motif_pvalue=5e-05,
                   threads=4)
```

```
/workspace/.conda/tfcomb_env/lib/python3.7/site-packages/tfcomb/network.py:12:␣
↪MatplotlibDeprecationWarning:
The mpl_toolkits.axes_grid module was deprecated in Matplotlib 2.1 and will be removed␣
```

(continues on next page)

(continued from previous page)

```
↪two minor releases later. Use mpl_toolkits.axes_grid1 and mpl_toolkits.axisartist,␣
↪which provide the same functionality instead.
  from mpl_toolkits.axes_grid.inset_locator import inset_axes
INFO: Scanning for TFBS with 4 thread(s)...
INFO: Progress: 11%
INFO: Progress: 20%
INFO: Progress: 30%
INFO: Progress: 40%
INFO: Progress: 50%
INFO: Progress: 60%
INFO: Progress: 70%
INFO: Progress: 80%
INFO: Progress: 90%
INFO: Finished!
INFO: Processing scanned TFBS
INFO: Identified 336722 TFBS (401 unique names) within given regions
```

[2]:
```
C.count_within(threads=4)
C.market_basket()
C.simplify_rules()
```

```
INFO: Setting up binding sites for counting
INFO: Counting co-occurrences within sites
INFO: Counting co-occurrence within background
INFO: Progress: 10%
INFO: Progress: 20%
INFO: Progress: 30%
INFO: Progress: 40%
INFO: Progress: 52%
INFO: Progress: 62%
INFO: Progress: 72%
INFO: Progress: 80%
INFO: Progress: 92%
INFO: Finished!
INFO: Done finding co-occurrences! Run .market_basket() to estimate significant pairs
INFO: Market basket analysis is done! Results are found in <CombObj>.rules
```

[3]:
```
C.rules.head(10)
```

[3]:

	TF1	TF2	TF1_TF2_count	TF1_count	TF2_count	cosine	\
SP1-SP2	SP1	SP2	3323	2877	3415	1.060144	
SP2-SP3	SP2	SP3	3191	3415	2891	1.015564	
SP1-SP3	SP1	SP3	2640	2877	2891	0.915398	
PATZ1-SP2	PATZ1	SP2	2776	3237	3415	0.834935	
KLF3-SP2	KLF3	SP2	2254	2155	3415	0.830874	
SP2-WT1	SP2	WT1	2666	3415	3163	0.811176	
KLF6-SP2	KLF6	SP2	2375	2557	3415	0.803717	
SP2-THAP1	SP2	THAP1	1779	3415	1444	0.801119	
SP2-SP4	SP2	SP4	2247	3415	2431	0.779857	
KLF3-SP3	KLF3	SP3	1910	2155	2891	0.765219	

	zscore

(continues on next page)

(continued from previous page)

```
SP1-SP2    91.794181
SP2-SP3    93.100144
SP1-SP3    99.371057
PATZ1-SP2  76.407190
KLF3-SP2   95.606695
SP2-WT1    65.990438
KLF6-SP2   84.420823
SP2-THAP1  78.441254
SP2-SP4    88.228572
KLF3-SP3   75.631093
```

Obtain locations of pairs

[4]:
```
pairs = C.get_pair_locations(("CTCF","SP2"))
```

[5]:
```
len(pairs)
```

[5]:
```
547
```

Plot pair footprints

[6]:
```
# mandatory setup for plotting (except pairLines)
pairs.bigwig_path = "../data/GM12878_corrected.bw"

# optional
# globally set signal windows size counted from alignment point
# pairs.comp_plotting_tables(flank=(10, 150), # default = 100
#                            align="left") # one of ['center', 'left', 'right']
```

PairMap

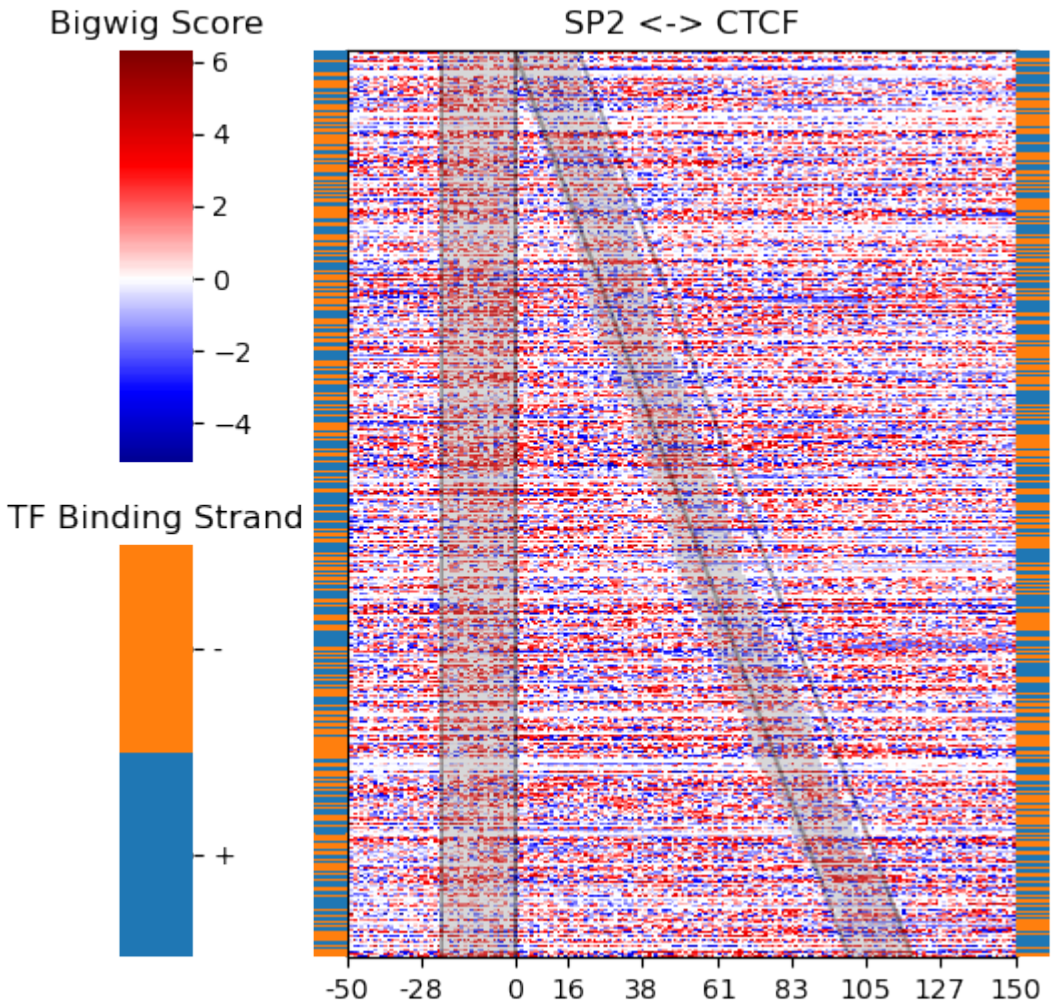
A heatmap of all binding pairs sorted by distance between the transcription factors.

[7]:
```
import numpy as np

pairs.pairMap(logNorm_cbar=None, # One of [None, "centerLogNorm", "SymLogNorm"]. Select␣
↪type of colorbar normalization.
              show_binding=True, # Show the TF binding positions.
              flank_plot="strand", # One of ["strand", "orientation"]. Select what is␣
↪shown in the flanking plots.
              figsize=(6, 6), # Figure size
              output=None, # Path to output file.
              flank=(50, 150), # Number of bases extended from center. Default = 100 or␣
↪last used size
              align="left", # Pair alignment one of ["center", "left", "right"]. Default␣
↪= "center" or last used.
              alpha=0.3, # Alpha for center line, TF binding positions and diagonal lines
              cmap="seismic", # Color palette to use
```

(continues on next page)

(continued from previous page)

```
            show_diagonal=True, # Show diagonal binding lines
            legend_name_score="Bigwig Score", # Set name of scoring legend (upper left)
            xtick_num=10, # Number of shown x-axis ticks
            log=np.log1p, # Log function to apply on the scores (any provided by␣
↪numpy). Default np.log1p
            dpi=100) # Dots per inch
```



[7]:
```
GridSpec(1, 2, width_ratios=[1, 10])
```

PairTrack

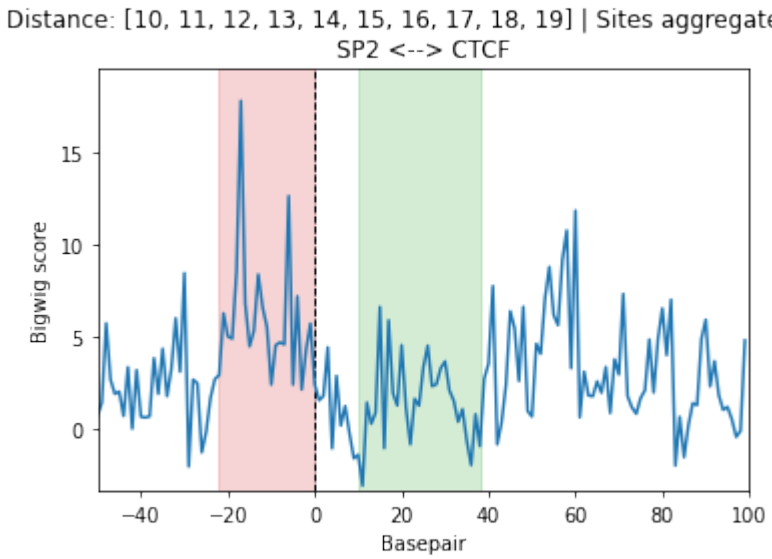
Aggregated binding of selected positions. Either select by one or more distances (between TFs) to aggregate signal or select a range of binding positions sorted by increasing distance.

[8]:
```
pairs.pairTrack(# either
                dist=list(range(10,20)), # Select sites to aggregate by one or more␣
↪distances between TF-pair.
                # or
```

(continues on next page)

(continued from previous page)

```
                # (range will be used if both are set)
                start=None, # Start of site range to be aggregated.
                end=None, # End of site range to be aggregated. Sites are sorted by␣
↪increasing distance.

                ymin=None, # y-axis minimum. Default is min value of selected data.
                ymax=None, # y-axis maximum. Default is max value of selected data.
                ylabel="Bigwig score", # y-axis label
                output=None, # Path to output file.
                flank=(50, 100), # Number of bases extended from center. Default = 100␣
↪or last used size
                align="left", # Pair alignment one of ["center", "left", "right"].␣
↪Default = "center" or last used.
                dpi=70, # Dots per inch
                figsize=(6, 4)) # Figure size
```



[8]:
```
<AxesSubplot:title={'center':'Distance: [10, 11, 12, 13, 14, 15, 16, 17, 18, 19] | Sites␣
↪aggregated: 65\nSP2 <--> CTCF'}, xlabel='Basepair', ylabel='Bigwig score'>
```

PairTrackAnimation

Multiple pairTracks combined as to a gif.

Note:
The memory limit can be increased with the following if necessary. Default is 20 MB.
`matplotlib.rcParams['animation.embed_limit'] = 100 # in MB`

[9]:
```
import matplotlib
matplotlib.rcParams['animation.embed_limit'] = 100
```

[10]:
```
pairs.pairTrackAnimation(site_num=None, # Number of sites aggregated in each plot. If␣
↪None will aggregate by distance.
                         step=10, # Step size between plots. Ignored if site_num=None
                         ymin=None, # y-axis minimum. Default is min value of selected␣
↪data.
                         ymax=None, # y-axis maximum. Default is max value of selected␣
↪data.
                         ylabel="Bigwig score", # y-axis label
                         interval=50, # Time between plots in milliseconds.
                         repeat_delay=0, # Delay until the gif is repeated in␣
↪milliseconds. Not functional
                         repeat=True, # Whether the gif should be repeated. Only affects␣
↪jupyter.
                         output=None, # Path to output file.
                         flank=(50, 150), # Number of bases extended from center.␣
↪Default = 100 or last used size
                         align="left", # Pair alignment one of ["center", "left", "right
↪"]. Default = "center" or last used.
                         dpi=70, # Dots per inch
                         figsize=(6, 4)) # Figure size
```

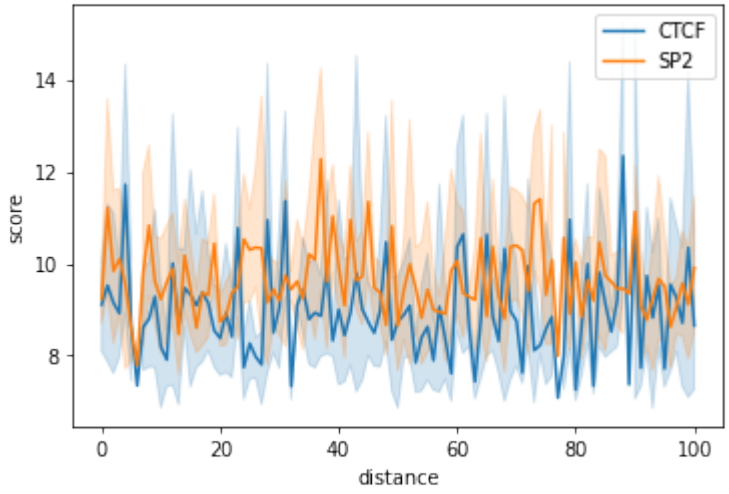
```
Create frames: 100%|| 102/102 [00:02<00:00, 39.47it/s]
Render animation: 100%|| 102/102 [00:06<00:00, 15.18it/s]
```

[10]:
```
<IPython.core.display.HTML object>
```

PairLines

Compare TF pair in lineplot. Choose from various options for x- and y-axis. To view x & y options set parameter to None

[11]:
```
pairs.pairLines(x="distance", # X-axis values
                y="score", # Y-axis values
                figsize=(6, 4), # Figure size
                dpi=70, # Dots per inch
                output=None) # Path to output file.
```



[11]: <AxesSubplot:xlabel='distance', ylabel='score'>

1.3 Methods

TF-COMB consist of several modules, for which the methods are described in this section.

Methods:

1.3.1 Adopted Market Basket Analysis (adopted MBA)

To explore relationships between items bought by customers a frequent pattern mining approach called market basket analysis (MBA) can be applied. The basic idea behind TF-COMB is utilizing such a MBA in a way applicable to transcription factor (TF) binding data to uncover co-occurrences between TFs. Here we present the classical MBA approach and how we altered it to cope with the special biological challenges.

Market Basket Analysis (MBA)

The MBA is part of the broader field of data mining and aims to reveal patterns of items frequently bought (observed) together.
For example to answer the question: "If customers are buying cereal (A), is it likely they will also buy milk (B)?".

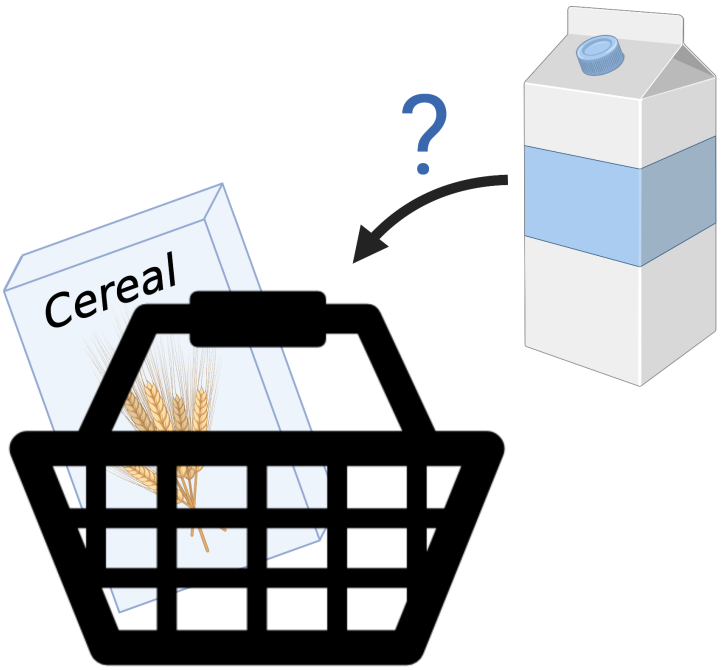


Fig. 1: Created with BioRender.com.

What kind of data is needed to calculate relationships?

For a classical MBA the transactions / baskets for each customer are examined and encoded binarized in a large matrix.

Based on the matrix a variety of **metrics** can be calculated like the widely used *support* and *confidence*:

support

$$(A \rightarrow B) = P(A, B) = \frac{frequency(A, B)}{N}$$

confidence

$$(A \rightarrow B) = P(B|A) = \frac{frequency(A, B)}{frequency(A)}$$

Frequent itemsets, also called *rules*, are composed by algorithms like *apriori* or *Frequent Pattern (FP) Growth* taking these measures into account.

But TF binding data is no (super-)market?

As mentioned above we took this approach and asked if we can transfer the concept to be applied to biological data in a way that we can answer the question: "If we observe *TFA*, is it likely we will also observe *TFB*?" Assuming TF factors to be items posed some challenges due to the nature of the data, explained below.

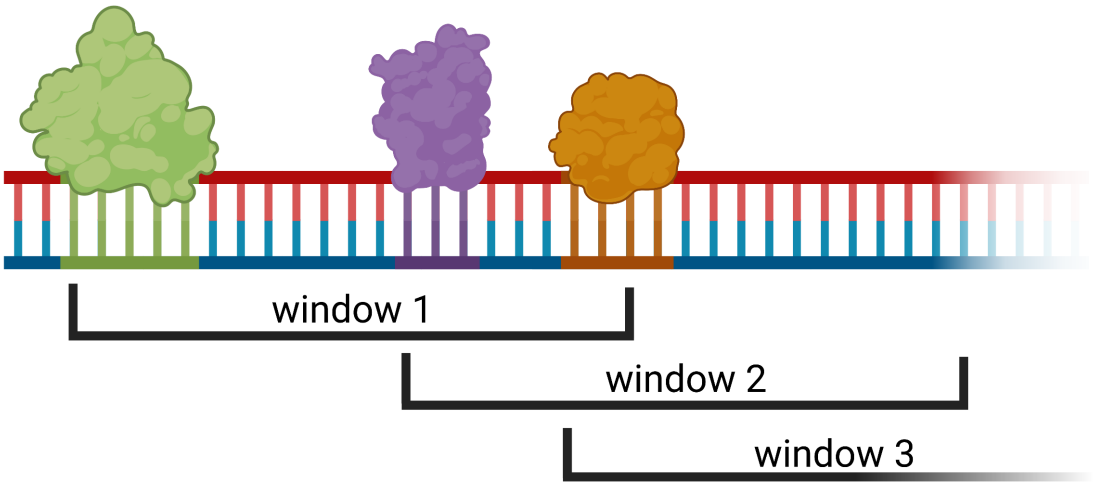
What are transactions/windows in TF binding data?

The first question translating the MBA concept to TF binding data was how to define transactions/baskets. Since we assumed TF factors to be the items we want to examine, we need a counterpart for transactions. Therefore, we constructed arbitrary windows of a given size w. However, the human genome for example is huge and utilizing a *sliding window approach* to slice the genome is computational expensive as well as resulting in many windows without valuable information content. Therefore, we assumed every genomic TF start position to denote the start of a window, resulting in a number of windows (transactions) equals observed TFs.

Created with BioRender.com.

How we resolved overlapping

TFs and their respective binding sites do not only occur side-by-side but sometimes also overlap each other. To enable full control of these special cases, TF-COMB offers a parameter to control the amount of allowed overlap ranging from 0 (no overlap allowed) to 1 (full overlap allowed). Overlap is calculated in percentage of the shorter binding site overlapping the larger binding site.



Metrics

To assess the importance of the found rules a variety of metrics exist[1] . Besides the above mentioned support, which is calculated automatically, TF-COMB supports *cosine*, *confidence*, *lift* or *jaccard*.

lift

$$\frac{confidence}{support(B)}$$

cosine

$$\frac{support(A,B)}{\sqrt{support(A) * support(B)}}$$

confidence

$$\frac{support(A,B)}{support(A)}$$

[1] Pang-Ning Tan, Vipin Kumar, Jaideep Srivastava, *Selecting the right objective measure for association analysis*, Information Systems Volume 29, Issue 4, 293-313 (2004)

jaccard

$$\frac{support(A, B)}{support(A) + support(B) - support(A, B)}$$

1.3.2 Anchor Modes

To calculate whether a pair is within a specific window of size w and to interpret predicted preferred distances, it is important to define an anchor point from which the distance is measured. TF-COMB supports three different anchor modes: inner, outer and center. The default mode is **inner**.

1. **inner** (default) is the distance between the transcription factors, it is measures as $start(TFB) - end(TFA)$. If for example transcription factor B is directly adjacent to Transcription factor A, the difference will be zero.
2. **center** is the distance measured from mid of transcription factor to mid of transcription factor $center(TFB) - center(TFA)$
3. **outer** (uncommonly used) is the distance measured including both transcription factors. $end(TFB) - start(TFA)$

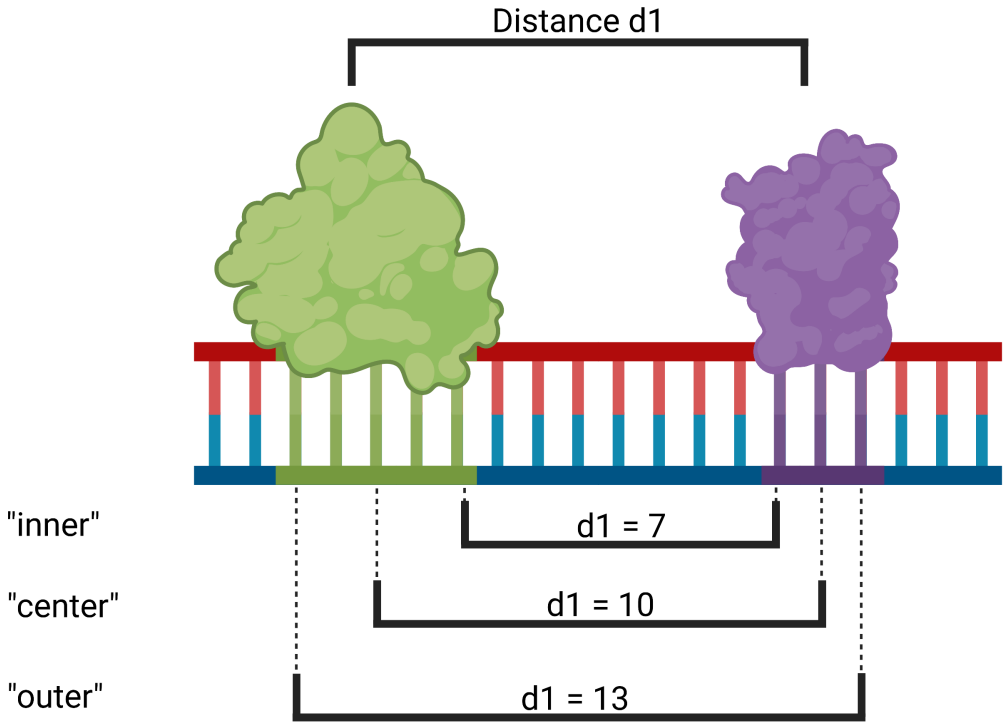


Fig. 2: Created with BioRender.com.

1.4 API reference

1.4.1 tfcomb.objects module

class tfcomb.objects.**CombObj**(*verbosity=1*)

Bases: object

The main class for collecting and working with co-occurring TFs.

Examples

```
>>> C = tfcomb.objects.CombObj()
```

Verbosity of the output log can be set using the 'verbosity' parameter:

```
>>> C = tfcomb.objects.CombObj(verbosity=2)
```

copy()

Returns a deep copy of the CombObj

set_verbosity(*level*)

Set the verbosity level for logging after creating the CombObj.

Parameters

level (*int*) – A value between 0-3 where 0 (only errors), 1 (info), 2 (debug), 3 (spam debug). Default: 1.

set_prefix(*prefix*)

Sets the .prefix variable of the object. Useful when comparing two objects in a DiffCombObj.

Parameters

prefix (*str*) – A string to add as .prefix for this object, e.g. 'control', 'treatment' or 'analysis1'.

check_pair(*pair*)

Checks if a pair is valid and present.

Parameters

pair (*tuple(str,str)*) – TF names for which the test should be performed. e.g. ("NFYA","NFYB")

to_pickle(*path*)

Save the CombObj to a pickle file.

Parameters

path (*str*) – Path to the output pickle file e.g. 'my_combobj.pkl'.

See also:

from_pickle

from_pickle(*path*)

Import a CombObj from a pickle file.

Parameters

path (*str*) – Path to an existing pickle file to read.

Raises
InputError – If read object is not an instance of CombObj.

See also:

to_pickle

TFBS_from_motifs(*regions*, *motifs*, *genome*, *motif_pvalue=1e-05*, *motif_naming='name'*, *gc=0.5*, *resolve_overlapping='merge'*, *extend_bp=0*, *threads=1*, *overwrite=False*, *_suffix=''*)

Function to calculate TFBS from motifs and genome fasta within the given genomic regions.

Parameters

- **regions** (*str or tobias.utils.regions.RegionList*) – Path to a .bed-file containing regions or a tobias-format RegionList object.
- **motifs** (*str or tobias.utils.motifs.MotifList*) – Path to a file containing JASPAR/MEME-style motifs or a tobias-format MotifList object.
- **genome** (*str*) – Path to the genome fasta-file to use for scan.
- **motif_pvalue** (*float, optional*) – The pvalue threshold for the motif search. Default: 1e-05.
- **motif_naming** (*str, optional*) – How to name TFs based on input motifs. Must be one of: 'name', 'id', 'name_id' or 'id_name'. Default: "name".
- **gc** (*float between 0-1, optional*) – Set GC-content for the motif background model. Default: 0.5.
- **resolve_overlapping** (*str, optional*) – Control how to treat overlapping occurrences of the same TF. Must be one of "merge", "highest_score" or "off". If "highest_score", the highest scoring overlapping site is kept. If "merge", the sites are merged, keeping the information of the first site. If "off", overlapping TFBS are kept. Default: "merge".
- **extend_bp** (*int, optional*) – Extend input regions with 'extend_bp' before scanning. Default: 0.
- **threads** (*int, optional*) – How many threads to use for multiprocessing. Default: 1.
- **overwrite** (*boolean, optional*) – Whether to overwrite existing sites within .TFBS. Default: False (sites are appended to .TFBS).

Returns
.TFBS_from_motifs fills the objects' .TFBS variable

Return type
None

TFBS_from_bed(*bed_file*, *overwrite=False*)

Fills the .TFBS attribute using a precalculated set of binding sites e.g. from ChIP-seq.

Parameters

- **bed_file** (*str*) – A path to a .bed-file with precalculated binding sites. The 4th column of the file should contain the name of the TF in question.
- **overwrite** (*boolean*) – Whether to overwrite existing sites within .TFBS. Default: False (sites are appended to .TFBS).

Returns
The .TFBS variable is filled in place

Return type
None

TFBS_from_TOBIAS(*bindetect_path*, *condition*, *overwrite=False*)

Fills the .TFBS variable with pre-calculated bound binding sites from TOBIAS BINDetect.

Parameters

- **bindetect_path** (*str*) – Path to the BINDetect-output folder containing <TF1>, <TF2>, <TF3> (...) folders.
- **condition** (*str*) – Name of condition to use for fetching bound sites.
- **overwrite** (*boolean*) – Whether to overwrite existing sites within .TFBS. Default: False (sites are appended to .TFBS).

Returns
The .TFBS variable is filled in place

Return type
None

Raises
InputError – If no files are found in path or if condition is not one of the avaiable conditions.

cluster_TFBS(*threshold=0.5*, *merge_overlapping=True*)

Cluster TFBS based on overlap of individual binding sites. This can be used to pre-process motif-derived TFBS into TF "families" of TFs with similar motifs. This changes the .name attribute of each site within .TFBS to represent the cluster (or the original TF name if no cluster was found).

Parameters

- **threshold** (*float from 0-1, optional*) – The threshold to set when clustering binding sites. Default: 0.5.
- **merge_overlapping** (*bool, optional*) – Whether to merge overlapping sites following clustering. If True, overlapping sites from the same cluster will be merged to one site (spanning site1-start -> site2-end). If False, the original sites (but with updated names) will be kept in .TFBS. Default: True.

Returns
The .TFBS names are updated in place.

Return type
None

subset_TFBS(*names=None*, *regions=None*)

Subset .TFBS in object to specific regions or TF names. Can be used to select only a subset of TFBS (e.g. only in promoters) to run analysis on. Note: Either 'names' or 'regions' must be given - not both.

Parameters

- **names** (*list of strings, optional*) – A list of names to keep. Default: None.
- **regions** (*str or RegionList, optional*) – Path to a .bed-file containing regions or a tobias-format RegionList object. Default: None.

Returns
The .TFBS attribute is updated in place.

Return type
None

TFBS_to_bed(*path*)

Writes out the .TFBS regions to a .bed-file. This is a wrapper for the tobias.utils.regions.RegionList().write_bed() utility.

Parameters

path (*str*) – File path to write .bed-file to.

count_within(*min_dist=0*, *max_dist=100*, *min_overlap=0*, *max_overlap=0*, *stranded=False*, *directional=False*, *binarize=False*, *anchor='inner'*, *n_background=50*, *threads=1*)

Count co-occurrences between TFBS. This function requires .TFBS to be filled by either *TFBS_from_motifs*, *TFBS_from_bed* or *TFBS_from_tobias*. This function can be followed by .market_basket to calculate association rules.

Parameters

- **min_dist** (*int*) – Minimum distance between two TFBS to be counted as co-occurring. Distances are calculated depending on the 'anchor' given. Default: 0.
- **max_dist** (*int*) – Maximum distance between two TFBS to be counted as co-occurring. Distances are calculated depending on the 'anchor' given. Default: 100.
- **min_overlap** (*float between 0-1, optional*) – Minimum overlap fraction needed between sites, e.g. 0 = no overlap needed, 1 = full overlap needed. Default: 0.
- **max_overlap** (*float between 0-1, optional*) – Controls how much overlap is allowed for individual sites. A value of 0 indicates that overlapping TFBS will not be saved as co-occurring. Float values between 0-1 indicate the fraction of overlap allowed (the overlap is always calculated as a fraction of the smallest TFBS). A value of 1 allows all overlaps. Default: 0 (no overlap allowed).
- **stranded** (*bool*) – Whether to take strand of TFBSs into account. Default: False.
- **directional** (*bool*) – Decide if direction of found pairs should be taken into account, e.g. whether "<—TF1—> <—TF2—>" is only counted as TF1-TF2 (directional=True) or also as TF2-TF1 (directional=False). Default: False.
- **binarize** (*bool, optional*) – Whether to count a TF1-TF2 more than once per window (e.g. in the case of "<TF1> <TF2> <TF2> (...)"). Default: False.
- **anchor** (*str, optional*) – The anchor to use for calculating distance. Must be one of ["inner", "outer", "center"]
- **n_background** (*int, optional*) – Number of random co-occurrence backgrounds to obtain. This number effects the runtime of .count_within, but 'threads' can be used to speed up background calculation. Default: 50.
- **threads** (*int, optional*) – Number of threads to use. Default: 1.

Returns

Fills the object variables .TF_counts and .pair_counts.

Return type

None

Raises

ValueError – If .TFBS has not been filled.

get_pair_locations(*pair*, *TF1_strand=None*, *TF2_strand=None*, ***kwargs*)

Get genomic locations of a particular TF pair. Requires .TFBS to be filled. If 'count_within' was run, the parameters used within the latest 'count_within' run are used. Else, the default values of tfcomb.utils.get_pair_locations() are used. Both options can be overwritten by setting kwargs.

Parameters

- **pair** (*tuple*) – Name of TF1, TF2 in pair.
- **TF1_strand** (*str, optional*) – Strand of TF1 in pair. Default: None (strand is not taken into account).
- **TF2_strand** (*str, optional*) – Strand of TF2 in pair. Default: None (strand is not taken into account).
- **kwargs** (*arguments*) – Any additional arguments are passed to tfcomb.utils.get_pair_locations.

Return type
tfcomb.utils.TFBSPairList

market_basket(*measure='cosine'*, *threads=1*, *keep_zero=False*, *n_baskets=1000000.0*, *_show_columns=['TF1_TF2_count', 'TF1_count', 'TF2_count']*)

Runs market basket analysis on the TF1-TF2 counts. Requires prior run of .count_within().

Parameters

- **measure** (*str or list of strings, optional*) – The measure(s) to use for market basket analysis. Can be any of: ["cosine", "confidence", "lift", "jaccard"]. Default: 'cosine'.
- **threads** (*int, optional*) – Threads to use for multiprocessing. This is passed to .count_within() in case the <CombObj> does not contain any counts yet. Default: 1.
- **keep_zero** (*bool, optional*) – Whether to keep rules with 0 occurrences in .rules table. Default: False (remove 0-rules).
- **n_baskets** (*int, optional*) – The number of baskets used for calculating market basket measures. Default: 1e6.

Raises
InputError – If the measure given is not within available measures.

reduce_TFBS()

Reduce TFBS to the TFs present in .rules.

Return type
None - changes .TFBS in place

simplify_rules()

Simplify rules so that TF1-TF2 and TF2-TF1 pairs only occur once within .rules. This is useful for association metrics such as 'cosine', where the association of TF1->TF2 equals TF2->TF1. This function keeps the first unique pair occurring within the rules table.

select_TF_rules(*TF_list*, *TF1=True*, *TF2=True*, *reduce_TFBS=True*, *inplace=False*, *how='inner'*)

Select rules based on a list of TF names. The parameters TF1/TF2 can be used to select for which TF to create the selection on (by default: both TF1 and TF2).

Parameters

- **TF_list** (*list*) – List of TF names fitting to TF1/TF2 within .rules.
- **TF1** (*bool, optional*) – Whether to subset the rules containing 'TF_list' TFs within "TF1". Default: True.
- **TF2** (*bool, optional*) – Whether to subset the rules containing 'TF_list' TFs within "TF2". Default: True.

- **reduce_TFBS** (*bool, optional*) – Whether to reduce the .TFBS of the new object to the TFs remaining in *.rules* after selection. Setting this to 'False' will improve speed, but also increase memory consumption. Default: True.
- **inplace** (*bool, optional*) – Whether to make selection on current CombObj. If False,
- **how** (*string, optional*) – How to join TF1 and TF2 subset. Default: inner

Raises
InputError – If both TF1 and TF2 are False or if no rules were selected based on input.

Returns

- *If inplace == False; tfcomb.objects.CombObj()* – An object containing a subset of <Combobj>.rules.
- *if inplace == True;* – Returns None

select_custom_rules(*custom_list*, *reduce_TFBS=True*)

Select rules based on a custom list of TF pairs.

Parameters

- **custom_list** (*list of strings*) – List of TF pairs (e.g. a string "TF1-TF2") fitting to TF1/TF2 combination within .rules.
- **reduce_TFBS** (*bool, optional*) – Whether to reduce the .TFBS of the new object to the TFs remaining in *.rules* after selection. Setting this to 'False' will improve speed, but also increase memory consumption. Default: True.

Returns
An object containing a subset of <Combobj>.rules

Return type
tfcomb.objects.CombObj()

select_top_rules(*n*, *reduce_TFBS=True*)

Select the top 'n' rules within .rules. By default, the .rules are sorted for the measure value, so n=100 will select the top 100 highest values for the measure (e.g. cosine).

Parameters

- **n** (*int*) – The number of rules to select.
- **reduce_TFBS** (*bool, optional*) – Whether to reduce the .TFBS of the new object to the TFs remaining in *.rules* after selection. Setting this to 'False' will improve speed, but also increase memory consumption. Default: True.

Returns
An object containing a subset of <Combobj>.rules

Return type
tfcomb.objects.CombObj()

select_significant_rules(*x='cosine'*, *y='zscore'*, *x_threshold=None*, *x_threshold_percent=0.05*, *y_threshold=None*, *y_threshold_percent=0.05*, *reduce_TFBS=True*, *plot=True*, ***kwargs*)

Make selection of rules based on distribution of x/y-measures

Parameters

- **x** (*str, optional*) – The name of the column within .rules containing the measure to be selected on. Default: 'cosine'.

- **y** (*str, optional*) – The name of the column within .rules containing the pvalue to be selected on. Default: 'zscore'
- **x_threshold** (*float, optional*) – A minimum threshold for the x-axis measure to be selected. If None, the threshold will be estimated from the data. Default: None.
- **x_threshold_percent** (*float between 0-1, optional*) – If x_threshold is not given, x_threshold_percent controls the strictness of the automatic threshold selection. Default: 0.05.
- **y_threshold** (*float, optional*) – A minimum threshold for the y-axis measure to be selected. If None, the threshold will be estimated from the data. Default: None.
- **y_threshold_percent** (*float between 0-1, optional*) – If y_threshold is not given, y_threshold_percent controls the strictness of the automatic threshold selection. Default: 0.05.
- **reduce_TFBS** (*bool, optional*) – Whether to reduce the .TFBS of the new object to the TFs remaining in *.rules* after selection. Setting this to 'False' will improve speed, but also increase memory consumption. Default: True.
- **plot** (*bool, optional*) – Whether to show the 'measure vs. pvalue'-plot or not. Default: True.
- **kwargs** (*arguments*) – Additional arguments are forwarded to tfcomb.plotting.scatter

Returns
An object containing a subset of <obj>.rules

Return type
tfcomb.objects.CombObj()

See also:

tfcomb.plotting.scatter

integrate_data(*table*, *merge='pair'*, *TF1_col='TF1'*, *TF2_col='TF2'*, *prefix=None*)

Function to add external data to object rules.

Parameters

- **table** (*str or pandas.DataFrame*) – A table containing data to add to .rules. If table is a string, 'table' is assumed to be the path to a tab-separated table containing a header line and rows of data.
- **merge** (*str*) – Which information to merge - must be one of "pair", "TF1" or "TF2". The option "pair" is used to merge infromation about TF-TF pairs such as protein-protein-interactions. The 'TF1' and 'TF2' can be used to include TF-specific information such as expression levels.
- **TF1_col** (*str, optional*) – The column in table corresponding to "TF1" name. If merge == "TF2", 'TF1' is ignored. Default: "TF1".
- **TF2_col** (*str, optional*) – The column in table corresponding to "TF2" name. If merge == "TF1", 'TF2' is ignored. Default: "TF2".
- **prefix** (*str, optional*) – A prefix to add to the columns. Can be useful for adding the same information to both TF1 and TF2 (e.g. by using "TF1" and "TF2" prefixes), or adding same-name columns from different tables. Default: None (no prefix).

plot_TFBS(***kwargs*)

This is a wrapper for the plotting function *tfcomb.plotting.genome_view*

Parameters

kwargs (*arguments*) – All arguments are passed to *tfcomb.plotting.genome_view*. Please see the documentation for input parameters.

plot_heatmap(*n_rules=20*, *color_by='cosine'*, *sort_by=None*, ***kwargs*)

Plot a heatmap of rules and their attribute values. This is a wrapper for the plotting function *tfcomb.plotting.heatmap*.

Parameters

- **n_rules** (*int, optional*) – The number of rules to show. The first *n_rules* rules of .rules are taken. Default: 20.
- **color_by** (*str, optional*) – A column within .rules to color the heatmap by. Note: Can be different than sort_by. Default: "cosine".
- **sort_by** (*str, optional*) – A column within .rules to sort by before choosing n_rules. Default: None (rules are not sorted before selection).
- **kwargs** (*arguments*) – Any additional arguments are passed to tfcomb.plotting.heatmap.

See also:

tfcomb.plotting.heatmap

plot_bubble(*n_rules=20*, *yaxis='cosine'*, *color_by='TF1_TF2_count'*, *size_by=None*, *sort_by=None*, ***kwargs*)

Plot a bubble-style scatterplot of the object rules. This is a wrapper for the plotting function *tfcomb.plotting.bubble*.

Parameters

- **n_rules** (*int, optional*) – The number of rules to show. The first *n_rules* rules of .rules are taken. Default: 20.
- **yaxis** (*str, optional*) – A column within .rules to depict on the y-axis of the plot. Default: "cosine".
- **color_by** (*str, optional*) – A column within .rules to color points in the plot by. Default: "TF1_TF2_count".
- **size_by** (*str, optional*) – A column within .rules to size points in the plot by. Default: None.
- **sort_by** (*str, optional*) – A column within .rules to sort by before choosing n_rules. Default: None (rules are not sorted before selection).
- **unique** (*bool, optional*) – Only show unique pairs in plot, e.g. only the first occurrence of TF1-TF2 / TF2-TF1. Default: True.
- **kwargs** (*arguments*) – Any additional arguments are passed to tfcomb.plotting.bubble.

See also:

tfcomb.plotting.bubble

plot_scatter(*x*, *y*, *hue=None*, ***kwargs*)

Plot a scatterplot of information from .rules.

Parameters

- **x** (*str*) – The name of the column in .rules containing values to plot on x-axis.
- **y** (*str*) – The name of the column in .rules containing values to plot on y-axis.

create_distObj()

Creates a distObject, useful for manual analysis. Fills self.distObj.

analyze_distances(*parent_directory=None*, *threads=4*, *correction=True*, *scale=True*, ***kwargs*)

Standard distance analysis workflow. Use create_distObj for own workflow steps and more options!

analyze_orientation()

Analyze preferred orientation of sites in .TFBS. This is a wrapper for tfcomb.analysis.orientation().

Return type
pd.DataFrame

See also:

tfcomb.analysis.orientation

build_network(***kwargs*)

Builds a TF-TF co-occurrence network for the rules within object. This is a wrapper for the tfcomb.network.build_nx_network() function, which uses the python networkx package.

Parameters
kwargs (*arguments*) – Any additional arguments are passed to tfcomb.network.build_nx_network().

Return type
None - fills the .network attribute of the *CombObj* with a networkx.Graph object

cluster_network(*method='louvain'*, *weight=None*)

Creates a clustering of nodes within network and add a new node attribute "cluster" to the network.

Parameters

- **method** (*str, one of ["louvain", "blockmodel"]*) – The method Default: "louvain".
- **weight** (*str, optional*) – The name of the edge attribute to use as weight. Default: None (not weighted).

plot_network(*color_node_by='TF1_count'*, *color_edge_by='cosine'*, *size_edge_by='TF1_TF2_count'*, ***kwargs*)

Plot the rules in .rules as a network using Graphviz for python. This function is a wrapper for building the network (using tfcomb.network.build_network) and subsequently plotting the network (using tfcomb.plotting.network).

Parameters

- **color_node_by** (*str, optional*) – A column in .rules or .TF_table to color nodes by. Default: 'TF1_count'.
- **color_edge_by** (*str, optional*) – A column in .rules to color edges by. Default: 'cosine'.
- **size_edge_by** (*str, optional*) – A column in rules to size edge width by. Default: 'TF1_TF2_count'.
- **kwargs** (*arguments*) – All other arguments are passed to tfcomb.plotting.network.

See also:

tfcomb.network.build_network

compare(*obj_to_compare*, *measure='cosine'*, *join='inner'*, *normalize=True*)

Utility function to create a DiffCombObj directly from a comparison between this CombObj and another CombObj. Requires .market_basket() run on both objects. Runs DiffCombObj.normalize (if chosen) and DiffCombObj.calculate_foldchanges() under the hood.

Note: Set .prefix for each object to get proper naming of output log2fc columns.

Parameters

- **obj_to_compare** (tfcomb.objects.CombObj) – Another CombObj to compare to the current CombObj.
- **measure** (*str, optional*) – The measure to compare between objects. Default: 'cosine'.
- **join** (*string*) – How to join the TF names of the two objects. Must be one of "inner" or "outer". If "inner", only TFs present in both objects are retained. If "outer", TFs from both objects are used, and any missing counts are set to 0. Default: "inner".
- **normalize** (*bool, optional*) – Whether to normalize values between objects. Default: True.

Return type
DiffCombObj

class tfcomb.objects.**DiffCombObj**(*objects=[]*, *measure='cosine'*, *join='inner'*, *fillna=True*, *verbosity=1*)

Bases: object

add_object(*obj*, *join='inner'*, *fillna=True*)

Add one CombObj to the DiffCombObj.

Parameters

- **obj** (CombObj) – An instance of CombObj
- **join** (*string*) – How to join the TF names of the two objects. Must be one of "inner" or "outer". If "inner", only TFs present in both objects are retained. If "outer", TFs from both objects are used, and any missing counts are set to 0. Default: "inner".
- **fillna** (*True*) – If "join" == "outer", there can be missing counts for individual rules. If fillna == True, these counts are set to 0. Else, the counts are NA. Default: True.

Returns
Object is added in place

Return type
None

normalize()

Normalize the values for the DiffCombObj given measure (.measure) using quantile normalization. Overwrites the <prefix>_<measure> columns in .rules with the normalized values.

calculate_foldchanges(*pseudo=0.01*)

Calculate measure foldchanges between objects in DiffCombObj. The measure is chosen at the creation of the DiffCombObj and defaults to 'cosine'.

Parameters
pseudo (*float, optional*) – Set the pseudocount to add to all values before log2-foldchange transformation. Default: 0.01.

See also:

`tfcomb.DiffCombObj.normalize`

select_rules(*contrast=None*, *measure='cosine'*, *measure_threshold=None*, *measure_threshold_percent=0.05*, *mean_threshold=None*, *mean_threshold_percent=0.05*, *plot=True*, ***kwargs*)

Select differentially regulated rules using a MA-plot on the basis of measure and mean of measures per contrast.

Parameters

- **contrast** (*tuple*) – Name of the contrast to use in tuple format e.g. (<prefix1>,<prefix2>). Default: None (the first contrast is shown).
- **measure** (*str, optional*) – The measure to use for selecting rules. Default: "cosine" (internally converted to <prefix1>/<prefix2>_<measure>_log2fc).
- **measure_threshold** (*tuple, optional*) – Threshold for 'measure' for selecting rules. Default: None (the threshold is estimated automatically)
- **measure_threshold_percent** (*float between 0-1*) – If measure_threshold is not set, measure_threshold_percent controls the strictness of the automatic threshold. If you increase this value, more differential rules will be found and vice versa. Default: 0.05.
- **mean_threshold** (*float, optional*) – Threshold for 'mean' for selecting rules. Default: None (the threshold is estimated automatically)
- **mean_threshold_percent** (*float between 0-1*) – if mean_threshold is not set, mean_threshold_percent controls the strictness of the automatic threshold. If you increase this value, more differential rules will be found and vice versa. Default: 0.05.
- **plot** (*boolean, optional*) – Whether to plot the volcano plot. Default: True.
- **kwargs** (*arguments, optional*) – Additional arguments are passed to tfcomb.plotting.scatter.

Returns
An object containing a subset of <DiffCombobj>.rules

Return type
tfcomb.objects.DiffCombObj()

See also:

`tfcomb.plotting.volcano`

plot_correlation(*method='pearson'*, *save=None*, ***kwargs*)

Plot correlation of 'measure' between rules across objects.

Parameters

- **method** (*str, optional*) – Either 'pearson' or 'spearman'. Default: 'pearson'.
- **save** (*str, optional*) – Save the plot to the file given in 'save'. Default: None.
- **kwargs** (*arguments, optional*) – Additional arguments are passed to sns.clustermap.

plot_rules_heatmap(***kwargs*)

Plot a heatmap of size n_rules x n_objects

plot_heatmap(*contrast=None*, *n_rules=10*, *color_by='cosine_log2fc'*, *sort_by=None*, ***kwargs*)

Functionality to plot a heatmap of differentially co-occurring TF pairs for a certain contrast.

Parameters

- **contrast** (*tuple, optional*) – Name of the contrast to use in tuple format e.g. (<prefix1>,<prefix2>). Default: None (the first contrast is shown).
- **n_rules** (*int, optional*) – Number of rules to show from each contrast (default: 10). Note: This is the number of rules either up/down, meaning that the rules shown are n_rules * 2.
- **color_by** (*str, optional*) – Default: "cosine" (converted to "<prefix1>/<prefix2>_<color_by>")
- **sort_by** (*str, optional*) – Column in .rules to sort rules by. Default: None (keep sort)
- **kwargs** (*arguments, optional*) – Additional arguments are passed to tfcomb.plotting.heatmap.

See also:

tfcomb.plotting.heatmap

plot_bubble(*contrast=None*, *n_rules=20*, *yaxis='cosine_log2fc'*, *color_by=None*, *size_by=None*, ***kwargs*)

Plot bubble scatterplot of information within .rules.

Parameters

- **contrast** (*tuple, optional*) – Name of the contrast to use in tuple format e.g. (<prefix1>,<prefix2>). Default: None (the first contrast is shown).
- **n_rules** (*int, optional*) – Number of rules to show (in each direction). Default: 20.
- **yaxis** (*str, optional*) – Measure to show on the y-axis. Default: "cosine_log2fc".
- **color_by** (*str, optional*) – If column is not in rules, the string is supposed to be in the form "prefix1/prefix2_<color_by>". Default: None.
- **size_by** (*str, optional*) – Column to size bubbles by. Default: None.
- **kwargs** (*arguments*) – Any additional arguments are passed to tfcomb.plotting.bubble.

See also:

tfcomb.plotting.bubble

plot_network(*contrast=None*, *color_node_by=None*, *size_node_by=None*, *color_edge_by='cosine_log2fc'*, *size_edge_by=None*, ***kwargs*)

Plot the network of differential co-occurring TFs.

Parameters

- **contrast** (*tuple*) – Name of the contrast to use in tuple format e.g. (<prefix1>,<prefix2>). Default: None (the first contrast is shown).
- **color_node_by** (*str, optional*) – Name of measure to color node by. If column is not in .rules, the name will be internally converted to "prefix1/prefix2_<color_edge_by>". Default: None.
- **size_node_by** (*str, optional*) – Column in .rules to size_node_by. If column is not in .rules, the name will be internally converted to "prefix1/prefix2_<size_node_by>" Default: None.
- **color_edge_by** (*str, optional*) – The name of measure or column to color edge by (will be internally converted to "prefix1/prefix2_<color_edge_by>"). Default: "cosine_log2fc".

- **size_edge_by** (*str, optional*) – The name of measure or column to size edge by.
- **kwargs** (*arguments*) – Any additional arguments are passed to tfcomb.plotting.network.

Return type
dot network object

See also:

tfcomb.plotting.network

to_pickle(*path: str*)

Save the DiffCombObj to a pickle file.

Parameters
path (*str*) – Path to the output pickle file e.g. 'my_diff_comb_obj.pkl'.

See also:

from_pickle

from_pickle(*path: str*)

Import a DiffCombObj from a pickle file.

Parameters
path (*str*) – Path to an existing pickle file to read.

Raises
InputError – If read object is not an instance of DiffCombObj.

See also:

to_pickle

class tfcomb.objects.**DistObj**(*verbosity=1*)

Bases: object

The main class for analyzing preferred binding distances for co-occurring TFs.

Examples

```
>>> D = tfcomb.distances.DistObj()
```

Verbosity of the output log can be set using the 'verbosity' parameter: >>> D = tfcomb.distances.DistObj(verbosity=2)

set_verbosity(*level*)

Set the verbosity level for logging after creating the CombObj.

Parameters
level (*int*) – A value between 0-3 where 0 (only errors), 1 (info), 2 (debug), 3 (spam debug).

Returns
Sets the verbosity level for the Logger inplace

Return type
None

fill_rules(*comb_obj*)

Fill DistanceObject according to reference object with all needed Values and parameters to perform standard prefered distance analysis

Parameters
comb_obj (*tfcomb.objects (or any other object contain all necessary rules)*) – Object from which the rules and parameters should be copied from

Returns
Copies values and parameters from a combObj or diffCombObj.

Return type
None

reset_signal()

Resets the signals to their original state.

Returns
Resets the object datasource variable to the original raw distances

Return type
None

check_datasource(*att*)

Utility function to check if distances in .<att> were set. If not, InputError is raised.

Parameters
att (*str*) – Attribute name for a dataframe in self.

check_peaks()

Utility function to check if peaks were called. If not, InputError is raised.

check_min_max_dist()

Utility function to check if min and max distance are valid.

static chunk_table(*table*, *n*)

Split a pandas dataframe row-wise into n chunks.

Parameters
n (*int*) – A positive number of chunks to split table into.

Return type
list of pd.DataFrames

count_distances(*directional=None*, *stranded=None*, *percentage=False*, *percentage_bins=100*)

Count distances for co_occurring TFs, can be followed by analyze_distances
to determine preferred binding distances

Parameters

- **directional** (*bool or None, optional*) – Decide if direction of found pairs should be taken into account, e.g. whether "<—TF1—> <—TF2—>" is only counted as TF1-TF2 (directional=True) or also as TF2-TF1 (directional=False). If directional is None, self.directional will be used. Default: None.
- **stranded** (*bool or None, optional*) – Whether to take strand of TFBS into account when counting distances. If stranded is None, self.stranded will be used. Default: None
- **percentage** (*bool, optional*) – Whether to count distances as bp or percentage of longest TF1/TF2 region. If True, output will be collected in 1-percent increments from 0-1. If False, output depends on the min/max distance values given in the DistObj. Default: False.

Returns
Fills the object variable .distances.

Return type
None

`scale`(*how='min-max'*)

Scale the counted distances per pair. Saves the scaled counts into .scaled and updates .datasource.

Parameters
how (*str, optional*) – How to scale the counts. Must be one of: ["min-max", "fraction"]. If "min-max", all counts are scaled between 0 and 1. If "fraction", the sum of all counts are scled between 0 and 1. Default: "min-max".

`smooth`(*window_size=3*, *reduce=True*)

Helper function for smoothing all rules with a given window size.

Parameters

- **window_size** (*int, optional*) – Window size for the rolling smoothing window. A bigger window produces larger flanking ranks at the sides. Default: 3.
- **reduce** (*bool, optional*) – Reduce the distances to the positions with a full window, i.e. if the window size is 3, the first and last distances are removed. This prevents flawed enrichment of peaks at the borders of the distances. Default: True.

Returns
Fills the object variable .smoothed and updates .datasource

Return type
None

`is_smoothed`()

Return True if data was smoothed during analysis, False otherwise

Returns
True if smoothed, False otherwiese

Return type
bool

`correct_background`(*frac=0.66*, *threads=1*)

Corrects the background of distances.

Parameters

- **frac** (*float, optional*) – Fraction of data used to calculate smooth. Setting this fraction lower will cause stronger smoothing effect. Default: 0.66
- **threads** (*int, optional*) – Number of threads to use in functions. Default: 1.

Returns
Fills the object variable .corrected

Return type
None

`analyze_signal_all`(*threads=1*, *method='zscore'*, *threshold=2*, *min_count=1*, *save=None*)

After background correction is done, the signal is analyzed for peaks, indicating preferred binding distances. There can be more than one peak (more than one preferred binding distance) per Signal. Peaks are called with scipy.signal.find_peaks().

Parameters

- **threads** (*int*) – Number of threads used. Default: 1.

- **method** (*str*) – Method for transforming counts. Can be one of: "zscore" or "flat". If "zscore", the zscore for the pairs is used. If "flat", no transformation is performed. Default: "zscore".
- **threshold** (*float*) – The lower threshold for selecting peaks. Default: 2.
- **min_count** (*int*) – Minimum count of TF1-TF2 occurrences for a preferred distance to be called. Default: 1 (all occurrences are considered).
- **save** (*str*) – Path to save the peaks table to. Default: None (table is not written).

Returns
Fills the object variable self.peaks, self.peaking_count

Return type
None

evaluate_noise(*threads=1*, *method='median'*, *height_multiplier=0.75*)

Evaluates the noisiness of the signal. Therefore the peaks are cut out and the remaining signal is analyzed.

Parameters

- **threads** (*int*) – Number of threads used for evaluation. Default: 1
- **method** (*str*) – Measurement to calculate the noisiness of a signal. One of ["median", "min_max"]. Default: "median"
- **height_multiplier** (*float*) – Height multiplier (percentage) to calculate cut points. Must be between 0 and 1. Default: 0.75

See also:

tfcomb.utils.evaluate_noise_chunks

rank_rules(*by=['Distance_percent', 'Peak Heights', 'Noisiness']*, *calc_mean=True*)

ranks rules within each column specified.

Parameters

- **by** (*list of strings*) – Columns for wich the rules should be ranked Default: ["Distance_percent", "Peak Heights", "Noisiness"]
- **calc_mean** (*bool*) – True if an extra column should be calculated containing the mean rank, false otherwise Default: True

Raises
InputError – If columns selection (parameter: by) is not valid.

Returns
adds a rank column for each criteria given plus one for the mean if set to True

Return type
None

mean_distance(*source='datasource'*)

Get the mean distance for each rule in .rules.

Return type
pandas.DataFrame containing "mean_distance" per rule.

max_distance(*source='datasource'*)

Get the distance with the maximum signal for each rule in .rules.

Parameters
source (*str*) – The name of the datasource to use for calculation. Default: "datasource" (the current state of data).

Return type
pandas.DataFrame containing "max_distance" per rule.

`analyze_hubs()`

Counts the number of different partners each transcription factor forms a peak with, **with at least one peak**.

Returns
A panda series with the tf as index and the count as integer

Return type
pd.Series

`count_peaks()`

Counts the number of identified distance peaks per rules.

Returns
A dataframe containing 'n_peaks' (column) for each TF1-TF2 rule (index)

Return type
pd.DataFrame

`classify_rules()`

Classify all rules True if at least one peak was found, False otherwise.

Returns
fills .classified

Return type
None

`get_periodicity()`

Calculate periodicity for all rules via autocorrelation.

Returns
Fills the object variable .autocorrelation and .periodicity

Return type
None

`plot_autocorrelation(pair)`

Plot the autocorrelation for a pair, which shows the lag of periodicity in the counted distances.

Parameters
pair (*tuple(str, str)*) – TF names to plot. e.g. ("NFYA","NFYB")

`build_network()`

Builds a TF-TF co-occurrence network for the rules within object.

Returns
fills the .network attribute of the *CombObj* with a networkx.Graph object

Return type
None

See also:

`tfcomb.network.build_nx_network`

plot_bg_estimation(*pair*)

Plot the background estimation for pair for debugging background estimation

Parameters

pair (*tuple(str, str)*) – TF names to plot. e.g. ("NFYA","NFYB")

plot(*pair*, *method='peaks'*, *style='hist'*, *show_peaks=True*, *save=None*, *config=None*, *collapse=None*, *ax=None*, *color='tab:blue'*, *max_dist=None*, ***kwargs*)

Produces different plots.

Parameters

- **pair** (*tuple(str, str)*) – TF names to plot. e.g. ("NFYA","NFYB")
- **method** (*str, optional*) – Plotting method. One of: - 'peaks': Shows the z-score signal and any peaks found by analyze_signal_all. - 'correction': Shows the fit of the lowess curve to the data. - 'datasource', 'distances', 'scaled', 'corrected', 'smoothed': Shows the signal of the counts given in the .<method> table. Default: 'peaks'.
- **style** (*str, optional*) – What style to plot the datasource in. Can be one of: ["hist", "kde", "line"]. Default: "hist".
- **show_peaks** (*bool, optional*) – Whether to show the identified peak(s) (if any were found) in the plot. Default: True.
- **save** (*str, optional*) – Path to save the plots to. If save is None plots won't be plotted. Default: None
- **config** (*dict, optional*) – Config for some plotting methods.

  e.g. {"nbins":100} for histogram like plots or {"bwadjust":0.1} for kde (densitiy) plot.

  If set to *None*, below mentioned default parameters are used.

  possible parameters:

  [hist]: n_bins, Default: self.max_dist - self.min_dist + 1

  [kde]: bwadjust, Default: 0.1 (see seaborn.kdeplot())

  Default: None
- **collapse** (*str, optional*) – None if negative data should not be collapsed. ["min","max","mean","sum"] allowed as methods. See ._collapse_negative() for more information.
- **ax** (*plt.axis*) – Plot to an existing axis object. Default: None (a new axis will be created).
- **color** (*str, optional*) – Color of the plot hist/line/kde. Default: "tab:blue".
- **max_dist** (*int, optional*) – Option to set the max_dist independent of the max_dist used for counting distances. Default: None (max_dist is not changed).
- **kwargs** (*arguments*) – Additional arguments are passed to plt.hist().

plot_network(*color_node_by='TF1_count'*, *color_edge_by='Distance'*, *size_edge_by='Distance_percent'*, ***kwargs*)

Plot the rules in .rules as a network using Graphviz for python. This function is a wrapper for building the network (using tfcomb.network.build_network) and subsequently plotting the network (using tfcomb.plotting.network).

Parameters

- **color_node_by** (*str, optional*) – A column in .rules or .TF_table to color nodes by. Default: 'TF1_count'.

- **color_edge_by** (*str, optional*) – A column in .rules to color edges by. Default: 'Distance'.
- **size_edge_by** (*str, optional*) – A column in rules to size edge width by. Default: 'TF1_TF2_count'.
- ****kwargs** (*arguments*) – All other arguments are passed to tfcomb.plotting.network.

See also:

tfcomb.network.build_network

1.4.2 tfcomb.network module

tfcomb.network.**build_network**(*edge_table*, *node1='TF1'*, *node2='TF2'*, *node_table=None*, *directed=False*, *multi=False*, *tool='networkx'*, *verbosity=1*)

Build a network object from a table using either 'networkx' or 'graph-tool'.

Parameters

- **edge_table** (*pd.DataFrame*) – Table containing rows of edges and edge information between node1/node2.
- **node1** (*str, optional*) – The column to use as node1 ID. Default: "TF1".
- **node2** (*str, optional*) – The column to use as node2 ID. Default: "TF2".
- **node_table** (*pandas.DataFrame*) – A table of attributes to use for nodes. Default: node attributes are estimated from the columns in edge_table.
- **directed** (*bool, optional*) – Whether edges are directed or not. Default: False.
- **multi** (*bool, optional*) – Allow multiple edges between two vertices. NOTE: Only valid for tool == 'networkx'. If False, the first occurrence of TF1-TF2/TF2-TF1 in the table is used. Default: False.
- **tool** (*str, optional*) – Which module to use for generating network. Must be one of 'networkx' or 'graph-tool'. Default: 'networkx'.
- **verbosity** (*int, optional*) – Verbosity of logging (0/1/2/3). Default: 1.

Returns

- **if tool is 'networkx'** (*networkx.Graph / networkx.DiGraph / networkx.MultiGraph / networkx.MultiDiGraph - depending on parameters given.*)
- **if tool is 'graph-tool'** (*graph_tool.Graph*)

tfcomb.network.**get_degree**(*G*, *weight=None*, *direction='both'*)

Get degree per node in graph. If weight is given, the degree is the sum of weighted edges.

Parameters

- **G** (*networkx.Graph*) – An instance of networkx.Graph
- **weight** (*str, optional*) – Name of an edge attribute within network. Default: None.
- **direction** (*str, optional*) – Which edge direction to use for calculating degrees. Can be one of: ["both", "in", "out"]. Default: 'both'.

Returns

A table of format (...)

Return type
DataFrame

tfcomb.network.**get_betweenness_centrality**(*G*, *weight=None*)

Parameters

- **G** (*networkx.Graph*) – An instance of networkx.Graph
- **weight** – Edge attribute. Default: None.

tfcomb.network.**subset_graph**(*G*, *nodes*, *depth=0*)

Subset a graph to a subset of nodes and their neighborhoods at the depth given by 'depth'.

Parameters

- **G** (*networkx.Graph*) – An instance of networkx.Graph.
- **nodes** (*str or list of str*) – Nodes to keep.
- **depth** (*int, optional*) – Default: 0 (only edges between given nodes)

tfcomb.network.**cluster_louvain**(*G*, *weight=None*, *attribute_name='cluster'*, *logger=None*)

Cluster a network using community louvain clustering. By default, sets the attribute "cluster" to each node.

Parameters

- **G** (*networkx.Graph*) – An instance of a network graph to cluster.
- **weight** (*str*) – Attribute in graph to use as weight. The higher the weight, the stronger the link. Default: None.
- **attribute_name** (*str*) – The attribute name to use for saving clustering. Default: "cluster".
- **logger** (*a logger object*) – An instance of a logger. Default: No logging.

Return type
None - clustering is added to 'G' in place.

tfcomb.network.**cluster_blockmodel**(*g*, *attribute_name='cluster'*)

Clustering of a graph-tool graph using stochastic block model minimization.

Parameters

- **g** (*a graph.tool graph*) – An instance of a graph.tool graph.
- **attribute_name** (*str*) – The attribute name to use for saving clustering. Default: "cluster".

tfcomb.network.**get_node_table**(*G*)

Get a table containing node names and node attributes for G.

Parameters
G (*a networkx Graph object or graph_tool Graph object*) –

Return type
pandas.DataFrame

tfcomb.network.**get_edge_table**(*G*)

Get a table containing edge names and edge attributes for G.

Parameters
G (*a networkx Graph object.*) –

Return type
pandas.DataFrame

tfcomb.network.**create_random_network**(*nodes*, *edges*)

Create a random network with the given list of nodes and total number of edges.

Parameters

- **nodes** (*list*) – List of nodes to use in network.
- **edges** (*int*) – Number of edges between nodes.

Return type
networkx.Graph containing random edges between nodes.

tfcomb.network.**plot_powerlaw**(*G*, *title='Node degree powerlaw fit'*, *color='blue'*, *save=None*)

Fit and plot a powerlaw distribution to the node degrees in the network.

Parameters

- **G** (*a networkx Graph object*) – Networkx containing nodes and edges to analyze.
- **title** (*str, optional*) – The title of the resulting plot. Default: "Node degree powerlaw fit".
- **color** (*str, optional*) – The color of the data plotted. Default: "blue".
- **save** (*str, optional*) – If not None, save the plot to the given path. Default: None.

Returns
ax – Axes object containing the plot.

Return type
matplotlib.axes

1.4.3 tfcomb.analysis module

tfcomb.analysis.**orientation**(*rules*, *verbosity=1*)

Perform orientation analysis on the TF pairs in a directional / strand-specific table. The analysis counts different scenarios depending on the input.

If the input matrix is symmetric, the analysis contains two scenarios:

```
1. Same:      (TF1+)  (TF2+)  =  (TF2+)  (TF1+)  =  (TF1-)  (TF2-)  =  (TF2-)  (TF1-
↪)
2. Opposite:  (TF1+)  (TF2-)  =  (TF2+)  (TF1-)  =  (TF1-)  (TF2+)  =  (TF2-) ␣
↪(TF1+)
```

If the input is directional, the analysis contains four different scenarios:

```
1. TF1-TF2:     (TF1+)  (TF2+)   =   (TF2-)  (TF1-)
2. TF2-TF1:     (TF2+)  (TF1+)   =   (TF1-)  (TF2-)
3. convergent:  (TF1+)  (TF2-)   =   (TF2+)  (TF1-)
4. divergent:   (TF1-)  (TF2+)   =   (TF2-)  (TF1+)
```

Parameters

- **rules** (*pd.DataFrame*) – The .rules output of a CombObj analysis.

- **verbosity** (*int*) – A value between 0-3 where 0 (only errors), 1 (info), 2 (debug), 3 (spam debug). Default: 1.

Returns

- *An OrientationAnalysis object (subclass of pd.DataFrame). The table contains frequencies of pairs related to each scenario.*
- *The dataframe has the following columns –*
  - TF1: name of the first TF in pair
  - TF2: name of the second TF in pair
  - TF1_TF2_count: The total count of TF1-TF2 co-occurring pairs
  - **If symmetric:**
    - Same
    - Opposite
  - **If directional:**
    - TF1_TF2
    - TF2_TF1
    - convergent
    - divergent
  - std: Standard deviation of scenario frequencies
  - pvalue: A chi-square test to test the hypothesis that the scenarios are equally distributed

class tfcomb.analysis.**OrientationAnalysis**(**args: Any*, ***kwargs: Any*)

Bases: DataFrame

Analysis of the orientation of TF co-ocurring pairs

plot_heatmap(*yticklabels=False*, *figsize=(6, 6)*, *save=None*, ***kwargs*)

Plot a heatmap of orientation scenarios for the output of the orientation analysis.

Parameters

- **yticklabels** (*bool, optional*) – Show yticklabels (TF-pairs) in plot. Default: False.
- **figsize** (*tuple*) – The size of the output heatmap. Default: (6,6)
- **save** (*str, optional*) – Save the plot to the file given in 'save'. Default: None.
- **kwargs** (*arguments*) – Any additional arguments are passed to sns.clustermap.

Return type

seaborn.matrix.ClusterGrid

1.4.4 tfcomb.annotation module

tfcomb.annotation.**annotate_regions**(*regions*, *gtf*, *config=None*, *best=True*, *threads=1*, *verbosity=1*)

Annotate regions with genes from .gtf using UROPA[1].

Parameters

- **regions** (*tobias.utils.regions.RegionList() or pandas.DataFrame*) – A RegionList object with positions of genomic elements e.g. TFBS or a DataFrame containing chr/start/stop-coordinates. If DataFrame, the function assumes that the order of columns is: 'chromosome', 'start', 'end', 'id', 'score', 'strand'.
- **gtf** (*str*) – Path to .gtf file containing genomic elements for annotation.
- **config** (*dict, optional*) – A dictionary indicating how regions should be annotated. Default is to annotate feature 'gene' within -10000;1000bp of the gene start. See 'Examples' of how to set up a custom configuration dictionary.
- **best** (*boolean*) – Whether to return the best annotation or all valid annotations. Default: True (only best are kept).
- **threads** (*int, optional*) – Number of threads to use for multiprocessing. Default: 1.
- **verbosity** (*int, optional*) – Level of verbosity of logger. One of 0,1, 2. Default: 1.

Returns

Dataframe including regions and annotation information (if applicable, otherwise a warning will be displayed and None is returned).

Return type

pd.DataFrame or None

References

Examples

```
>>> custom_config = {"queries": [{"distance": [10000, 1000],
...                               "feature_anchor": "start",
...                               "feature": "gene"}],
...                               "priority": True,
...                               "show_attributes": "all"}
```

#Annotate regions (data/ refers to the data directory of the tfcomb github repository)

```
>>> regions = pd.read_csv("data/GM12878_hg38_chr4_ATAC_peaks.bed")
>>> annotate_regions(regions, gtf="data/chr4_genes.gtf",
                                                  config=custom_config)
```

tfcomb.annotation.**get_annotated_genes**(*regions*, *attribute='gene_name'*)

Get list of genes from the list of annotated regions from annotate_regions().

Parameters

- **regions** (*RegionList() or list of OneTFBS objects*) –

[1] Kondili M, Fust A, Preussner J, Kuenne C, Braun T, and Looso M. UROPA: a tool for Universal RObust Peak Annotation. Scientific Reports 7 (2017), doi: 10.1038/s41598-017-02464-y

- **attribute** (*str*) – The name of the attribute in the 9th column of the .gtf file. Default: 'gene_name'.

class tfcomb.annotation.**GOAnalysis**(**args: Any*, ***kwargs: Any*)

Bases: DataFrame

aspect_translation = {'BP': 'Biological Process', 'CC': 'Cellular Component', 'MF': 'Molecular Function'}

enrichment(*genes*, *organism='hsapiens'*, *background=None*, *propagate_counts=True*, *min_depth=1*, *verbosity=1*)

Perform a GO-term enrichment based on a list of genes. This is a TF-COMB wrapper for goatools.

Parameters

- **gene_ids** (*list*) – A list of gene ids.
- **organism** (str, optional) – The organism of which the gene_ids originate. Defaults to 'hsapiens'.
- **background** (*list, optional*) – A specific list of background gene ids to use. Default: The list of protein coding genes of the 'organism' given.
- **propagate_counts** (*bool*) – Whether to propagate counts up the tree to parent GO's. Default: True.
- **min_depth** (*int*) – Minimum depth of GO-terms to show in output table. Default: 1.
- **verbosity** (*int, optional*) – Default: 1.

See also:

tfcomb.annotation.GOAnalysis.plot_bubble

Returns

- *GOAnalysis object containing enrichment results*
- *Reference*
- ———-
- **https** (*//www.nature.com/articles/s41598-018-28948-z*)

plot_bubble(*aspect='BP'*, *n_terms=20*, *threshold=0.05*, *title=None*, *save=None*)

Plot a bubble-style plot of GO-enrichment results.

Parameters

- **aspect** (*str*) – The aspect for which GO-terms should be shown. Must be one of ["BP", "MF", "CC"]. Default: "BP".
- **n_terms** (*int*) – Maximum number of terms to show in graph. Default: 20
- **threshold** (*float between 0-1*) – FDR threshold for significant GO-terms. Default: 0.05.
- **title** (*str*) – Custom title for the plot. Default: 'GO-terms for <aspect>'.
- **save** (*str, optional*) – Save the plot to the file given in 'save'. Default: None.

Return type

ax

compare(*compare_table*)

IN PROGRESS: Plot a comparison of two GO-term analysis

Parameters

compare_table (*GOAnalysis object*) –

1.4.5 tfcomb.plotting module

tfcomb.plotting.**bubble**(*rules_table*, *yaxis='confidence'*, *size_by='TF1_TF2_support'*, *color_by='lift'*, *figsize=(7, 4)*, *save=None*)

Plot bubble plot with TF1-TF2 pairs on the x-axis and a choice of measure on the y-axis, as well as color and size of bubbles.

Parameters

- **rules_table** (*pandas.DataFrame*) – Dataframe containing data to plot.
- **yaxis** (*str, optional*) – Column containing yaxis information. Default: "confidence".
- **size_by** (*str*) – Default: "TF1_TF2_support".
- **color_by** (*str*) – Default: None
- **figsize** (*tuple*) – Default: (7,7).
- **save** (*str, optional*) – Save the plot to the file given in 'save'. Default: None.

Return type

ax

tfcomb.plotting.**heatmap**(*rules_table*, *columns='TF1'*, *rows='TF2'*, *color_by='cosine'*, *figsize=(7, 7)*, *save=None*)

Plot heatmap with TF1 and TF2 on rows and columns respectively. Heatmap colormap is chosen by .color_by.

Parameters

- **rules_table** (*pandas.DataFrame*) – The <CombObj>.rules table calculated by market basket analysis
- **columns** (*str, optional*) – The name of the column in rules_table to use as heatmap column names. Default: TF1.
- **rows** (*str, optional*) – The name of the column in rules_table to use as heatmap row names. Default: TF2.
- **color_by** (*str, optional*) – The name of the column in rules_table to use as heatmap colors. Default: "cosine".
- **figsize** (*tuple*) – The size of the output heatmap. Default: (7,7)
- **save** (*str*) – Save the plot to the file given in 'save'. Default: None.

tfcomb.plotting.**scatter**(*table*, *x*, *y*, *x_threshold=None*, *y_threshold=None*, *label=None*, *label_fontsize=9*, *label_color='red'*, *title=None*, *save=None*, ***kwargs*)

Plot scatter-plot of x/y values within table. Can also set thresholds and label values within plot.

Parameters

- **table** (*pd.DataFrame*) – A table containing columns of 'measure' and 'pvalue'.
- **x** (*str*) – Name of column in table containing values to map on the x-axis.

- **y** (*str*) – Name of column in table containing values to map on the y-axis.
- **x_threshold** (*float, tuple of floats or None, optional*) – Gives the option to visualize an x-axis threshold within plot. If None, no measure threshold is set. Default: None.
- **y_threshold** (*float, tuple of floats or None, optional*) – Gives the option to visualize an y-axis threshold within plot. If None, no measure threshold is set. Default: None.
- **label** (*str or list, optional*) – If None, no point labels are plotted. If "selection", the . Default: None.
- **label_fontsize** (*float, optional*) – Size of labels. Default: 9.
- **label_color** (*str, optional*) – Color of labels. Default: 'red'.
- **title** (*str, optional*) – Title of plot. Default: None.
- **kwargs** (*arguments*) – Any additional arguments are passed to sns.jointplot.

tfcomb.plotting.**go_bubble**(*table*, *aspect='MF'*, *n_terms=20*, *threshold=0.05*, *save=None*)

Plot a bubble-style plot of GO-enrichment results.

Parameters

- **table** (*pandas.DataFrame*) – The output of tfcomb.analysis.go_enrichment.
- **aspect** (*str*) – One of ["MF", "BP", "CC"]
- **n_terms** (*int*) – Maximum number of terms to show in graph. Default: 20
- **threshold** (*float between 0-1*) – The p-value-threshold to show in plot.
- **save** (*str, optional*) – Save the plot to the file given in 'save'. Default: None.

Return type

ax

tfcomb.plotting.**network**(*network*, *color_node_by=None*, *color_edge_by=None*, *size_node_by=None*, *size_edge_by=None*, *engine='sfdp'*, *size='8,8'*, *min_edge_size=2*, *max_edge_size=8*, *min_node_size=14*, *max_node_size=20*, *legend_size='auto'*, *node_border=False*, *node_cmap=None*, *edge_cmap=None*, *node_attributes={}*, *save=None*, *verbosity=1*)

Plot network of a networkx object using Graphviz for python.

Parameters

- **network** (*networkx.Graph*) – A networkx Graph/DiGraph object containing the network to plot.
- **color_node_by** (*str, optional*) – The name of a node attribute
- **color_edge_by** (*str, optional*) – The name of an edge attribute
- **size_node_by** (*str, optional*) – The name of a node attribute
- **size_edge_by** (*str, optional*) – The name of an edge attribute
- **engine** (*str, optional*) – The graphviz engine to use for finding network layout. Default: "sfdp".
- **size** (*str, optional*) – Size of the output figure. Default: "8,8".
- **min_edge_size** (*float, optional*) – Default: 2.
- **max_edge_size** (*float, optional*) – Default: 8.

- **min_node_size** (*float, optional*) – Default: 14.
- **max_node_size** (*float, optional*) – Default: 20.
- **legend_size** (*int, optional*) – Fontsize for legend explaining color_node_by/color_edge_by/size_node_by/size_edge_by. Set to 0 to hide legend. Default: 'auto'.
- **node_border** (*bool, optional*) – Whether to plot border on nodes. Can be useful if the node colors are very light. Default: False.
- **node_cmap** (*str, optional*) – Name of colormap for node coloring. Default: None (colors are automatically chosen).
- **edge_cmap** (*str, optional*) – Name of colormap for edge coloring. Default: None (colors are automatically chosen).
- **node_attributes** (*dict, optional*) – Additional node attributes to apply to graph. Default: No additional attributes.
- **save** (*str, optional*) – Path to save network figure to. Format is inferred from the filename - if not valid, the default format is '.pdf'.
- **verbosity** (*int, optional*) – verbosity of the logging. Default: 1.

Raises

- **TypeError** – If network is not a networkx.Graph object
- ***InputError*** – If any of 'color_node_by', 'color_edge_by' or 'size_edge_by' is not in node/edge attributes, or if 'engine' is not a valid graphviz engine.

tfcomb.plotting.**genome_view**(*TFBS*, *window_chrom=None*, *window_start=None*, *window_end=None*, *window=None*, *fasta=None*, *bigwigs=None*, *bigwigs_sharey=False*, *TFBS_track_height=4*, *title=None*, *highlight=None*, *save=None*, *figsize=None*, *verbosity=1*)

Plot TFBS in genome view via the 'DnaFeaturesViewer' package.

Parameters

- **TFBS** (*list*) – A list of OneTFBS objects or any other object containing .chrom, .start, .end and .name variables.
- **window_chrom** (*str, optional if 'window' is given*) – The chromosome of the window to show.
- **window_start** (*int, optional if 'window' is given*) – The genomic coordinates for the start of the window.
- **window_end** (*int, optional if 'window' is given*) – The genomic coordinates for the end of the window.
- **window** (*Object with .chr, .start, .end*) – If window_chrom/window_start/window_end are not given, window can be given as an object containing .chrom, .start, .end variables
- **fasta** (*str, optional*) – The path to a fasta file containing sequence information to show. Default: None.
- **bigwigs** (*str, list or dict of strings, optional*) – Give the paths to bigwig signals to show within graph. Default: None.
- **bigwigs_sharey** (*bool or list, optional*) – Whether bigwig signals should share y-axis range. If True, all signals will be shared. It is also possible to give a list of bigwig

indices (starting at 0), which should share y-axis values, e.g. [0,1,3] for the 1st, 2nd and 4th bigwig to share signal. If list of lists, each lists correspond to a grouping, e.g. [[0,2], [1,3]]. Default: False.

- **TFBS_track_height** (*float, optional*) – Relative track height of TFBS. Default: 4.
- **title** (*str, optional*) – Title of plot. Default: None.
- **highlight** (*list, optional*) – A list of OneTFBS objects or any other object containing .chrom, .start, .end and .name variables.
- **figsize** (*tuple, optional*) – The size of the figure. Default: None (8, TFBS_track_height + number of bigwig tracks).
- **save** (*str, optional*) – Save the plot to the file given in 'save'. Default: None.

1.4.6 tfcomb.utils module

class tfcomb.utils.**OneTFBS**(*lst=[]*)

Bases: list

Collects location information about one single TFBS

class tfcomb.utils.**TFBSPair**(*TFBS1*, *TFBS2*, *anchor='inner'*, *simplify=False*)

Bases: object

Collects information about a co-occurring pair of TFBS

class tfcomb.utils.**TFBSPairList**(*iterable=()*, */*)

Bases: list

Class for collecting and analyzing a list of TFBSPair objects

as_table()

Table representation of the pairs in the list

write_bed(*outfile*, *fmt='bed'*, *merge=False*)

Write the locations of (TF1, TF2) pairs to a bed-file.

Parameters

- **locations** (*list*) – The output of get_pair_locations().
- **outfile** (*str*) – The path which the pair locations should be written to.
- **fmt** (*str, optional*) – The format of the output file. Must be one of "bed" or "bedpe". If "bed", the TF1/TF2 sites are written individually (see merge option to merge sites). If "bedpe", the sites are written in BEDPE format. Default: "bed".
- **merge** (*bool, optional*) – If fmt="bed", 'merge' controls how the locations are written out. If True, will be written as one region spanning TF1.start-TF2.end. If False, TF1/TF2 sites are written individually. Default: False.

append(*element*)

Append object to the end of the list.

extend(*l*)

Extend list by appending elements from the iterable.

insert(*index*, *object*)

Insert object before index.

remove(*value*)

Remove first occurrence of value.

Raises ValueError if the value is not present.

pop(*index=-1*)

Remove and return item at index (default last).

Raises IndexError if list is empty or index is out of range.

clear() → None

Remove all items from list.

property bigwig_path

Get path to bigwig file.

property plotting_tables

Getter for plotting_tables. Will compute if necessary.

comp_plotting_tables(*flank=100*, *align='center'*)

Prepare pair and score tables for plotting.

Parameters

- **flank** (*int or tuple, default 100*) – Window size of TFBSpair. Adds given amount of bases in both directions counted from alignment anchor (see align) between binding sites. Use a tuple of ints to set left and right flank independently.
- **align** (*str, default 'center'*) –

  Position from which the flanking regions are added. Must be one of 'center', 'left', 'right'.
  'center': Midpoint between binding positions (rounded down if uneven). 'left': End of first binding position in pair. 'right': Start of second binding position in pair.

pairMap(*logNorm_cbar=None*, *show_binding=True*, *flank_plot='strand'*, *figsize=(7, 14)*, *output=None*, *flank=None*, *align=None*, *alpha=0.7*, *cmap='seismic'*, *show_diagonal=True*, *legend_name_score='Bigwig Score'*, *xtick_num=10*, *log=<ufunc 'log1p'>*, *dpi=300*)

Create a heatmap of TF binding pairs sorted for distance.

Parameters

- **logNorm_cbar** (*str, default None*) – [None, "centerLogNorm", "SymLogNorm"] Choose a type of normalization for the colorbar.

  SymLogNorm:
  Use matplotlib.colors.SymLogNorm. This does not center to 0

  centerLogNorm:
  Use custom matplotlib.colors.SymLogNorm from stackoverflow. Note this creates a weird colorbar.
- **show_binding** (*bool, default True*) – Shows the TF binding positions as a grey background.
- **flank_plot** (*str, default 'strand'*) – ["strand", "orientation", None] Decide if the plots flanking the heatmap should be colored for strand, strand-orientation or disabled.
- **figsize** (*int tuple, default (7, 14)*) – Figure dimensions.
- **output** (*str, default None*) – Save plot to given file.

- **flank** (*int or int tuple, default None*) – Bases added to both sides counted from center. Forwarded to comp_plotting_tables().
- **align** (*str, default None*) – Alignment of pairs. One of ['left', 'right', 'center']. Forwarded to comp_plotting_tables().
- **alpha** (*float, default 0.7*) – Alpha value for diagonal lines, TF binding positions and center line.
- **cmap** (*matplotlib colormap name or object, or list of colors, default 'seismic'*) – Color palette used in the main heatmap. Forwarded to seaborn.heatmap(cmap)
- **show_diagonal** (*boolean, default True*) – Shows diagonal lines for identifying preference in binding distance.
- **legend_name_score** (*str, default 'Bigwig Score'*) – Name of the score legend (upper legend).
- **xtick_num** (*int, default 10*) – Number of ticks shown on the x-axis. Disable ticks with None or values < 0.
- **log** (*function, default numpy.log1p*) – Function applied to each row of scores. Excludes 0 and will use absolute value for negative numbers adding the sign afterwards. Use any of the numpy.log functions. For example numpy.log, numpy.log10 or numpy.log1p. None to disable.
- **dpi** (*float, default 300*) – The resolution of the figure in dots-per-inch.

Returns
Object containing the finished pairMap.

Return type
matplotlib.gridspec.GridSpec

pairTrack(*dist=None*, *start=None*, *end=None*, *ymin=None*, *ymax=None*, *ylabel='Bigwig signal'*, *output=None*, *flank=None*, *align=None*, *figsize=(6, 4)*, *dpi=70*, *_ret_param=False*)

Create an aggregated footprint track on the paired binding sites. Either aggregate all sites for a specific distance or give a range of sites that should be aggregated. If the second approach spans multiple distances the binding locations are shown as a range as well.

Parameters

- **dist** (*int or int list, default None*) – Show track for one or more distances between binding sites.
- **start** (*int, default None*) – Define start of range of sites that should be aggregated. If set will ignore 'dist'.
- **end** (*int, default None*) – Define end of range of sites that should be aggregated. If set will ignore 'dist'.
- **ymin** (*int, default None*) – Y-axis minimum limit.
- **ymax** (*int, default None*) – Y-axis maximum limit.
- **ylabel** (*str, default 'Bigwig signal'*) – Label for the y-axis.
- **output** (*str, default None*) – Save plot to given file.
- **flank** (*int or int tuple, default None*) – Bases added to both sides counted from center. Forwarded to comp_plotting_tables().

- **align** (*str, default None*) – Alignment of pairs. One of ['left', 'right', 'center']. Forwarded to comp_plotting_tables().
- **figsize** (*int tuple, default (3, 3)*) – Figure dimensions.
- **dpi** (*float, default 70*) – The resolution of the figure in dots-per-inch.
- **_ret_param** (*bool, default False*) – Intended for internal animation use! If True will cause the function to return a dict of function call parameters used to create plot.

Returns
Return axes object of the plot.

Return type
matplotlib.axes._subplots.AxesSubplot or dict

pairTrackAnimation(*site_num=None*, *step=10*, *ymin=None*, *ymax=None*, *ylabel='Bigwig signal'*, *interval=50*, *repeat_delay=0*, *repeat=False*, *output=None*, *flank=None*, *align=None*, *figsize=(6, 4)*, *dpi=70*)

Combine a set of pairTrack plots to a .gif.

Note: The memory limit can be increased with the following if necessary. Default is 20 MB. matplotlib.rcParams['animation.embed_limit'] = 100 # in MB

Parameters

- **site_num** (*int, default None*) – Number of sites to aggregate for every step. If None will aggregate by distance between binding pair.
- **step** (*int, default None*) – Step size between aggregations. Will be ignored if site_num=None.
- **ymin** (*int, default None*) – Y-axis minimum limit
- **ymax** (*int, default None*) – Y-axis maximum limit
- **ylabel** (*str, default 'Bigwig signal'*) – Label for the y-axis.
- **interval** (*int, default 50*) – Delay between frames in milliseconds
- **repeat_delay** (*int, default 0*) – The delay in milliseconds between consecutive animation runs, if repeat is True.
- **repeat** (*boolean, default False*) – Whether the animation repeats when the sequence of frames is completed.
- **output** (*str, default None*) – Save plot to given file.
- **flank** (*int or int tuple, default None*) – Bases added to both sides counted from center. Forwarded to comp_plotting_tables().
- **align** (*str, default None*) – Alignment of pairs. One of ['left', 'right', 'center']. Forwarded to comp_plotting_tables().
- **figsize** (*int tuple, default (6, 4)*) – Figure dimensions.
- **dpi** (*float, default 70*) – The resolution of the figure in dots-per-inch.

Returns
Gif object ready to display in a jupyter notebook.

Return type
IPython.core.display.HTML

pairLines(*x*, *y*, *figsize=(6, 4)*, *dpi=70*, *output=None*)

Compare miscellaneous values between TF-pair.

Parameters

- **x** (*string*) – Data to show on the x-axis. Set None to get a list of options.
- **y** (*string*) – Data to show on the y-axis. Set None to get a list of options.
- **figsize** (*int tuple, default (6, 4)*) – Figure dimensions.
- **dpi** (*float, default 70*) – The resolution of the figure in dots-per-inch.
- **output** (*str, default None*) – Save plot to given file.

Returns
Return axes object of the plot.

Return type
matplotlib.axes._subplots.AxesSubplot

set_orientation(*simplify=False*)

Fill orientation of each TF pair

plot_distances(*groupby='orientation'*, *figsize=None*, *group_order=None*)

Plot the distribution of distances between TFBS-pairs.

Parameters

- **groupby** (*str*) – An attribute of each pair to group distances by. If None, all distances are shown without grouping. Default: "orientation".
- **figsize** (*tuple of ints*) – Set the figure size, e.g. (8,10). Default: None (default matplotlib figuresize).

exception tfcomb.utils.**InputError**

Bases: Exception

Raises an InputError exception without writing traceback

exception tfcomb.utils.**StopExecution**

Bases: Exception

Stop execution of a notebook cell with error message

tfcomb.utils.**check_graphtool**()

Utility to check if 'graph-tool' is installed on path. Raises an exception (if notebook) or exits (if script) if the module is not installed.

tfcomb.utils.**check_module**(*module*)

Check if <module> can be imported without error

tfcomb.utils.**check_columns**(*df*, *columns*)

Utility to check whether columns are found within a pandas dataframe.

Parameters

- **df** (*pandas.DataFrame*) – A pandas dataframe to check.
- **columns** (*list*) – A list of column names to check for within 'df'.

Raises
InputError – If any of the columns are not in 'df'.

tfcomb.utils.**check_dir**(*dir_path*, *create=True*)

Check if a dir is writeable.

Parameters
dir_path (*str*) – A path to a directory.

Raises
InputError – If dir_path is not writeable.

tfcomb.utils.**check_writeability**(*file_path*)

Check if a file is writeable.

Parameters
file_path (*str*) – A path to a file.

Raises
InputError – If file_path is not writeable.

tfcomb.utils.**check_type**(*obj*, *allowed*, *name=None*)

Check whether given object is within a list of allowed types.

Parameters

- **obj** (*object*) – Object to check type on
- **allowed** (*type or list of types*) – A type or a list of object types to be allowed
- **name** (*str, optional*) – Name of object to be written in error. Default: None (the input is referred to as 'object')

Raises
InputError – If object type is not within types.

tfcomb.utils.**check_string**(*astring*, *allowed*, *name=None*)

Check whether given string is within a list of allowed strings.

Parameters

- **astring** (*str*) – A string to check.
- **allowed** (*str or list of strings*) – A string or list of allowed strings to check against 'astring'.
- **name** (*str, optional*) – The name of the string to be written in error. Default: None (the value is referred to as 'string').

Raises
InputError – If 'astring' is not in 'allowed'.

tfcomb.utils.**check_value**(*value*, *vmin=-inf*, *vmax=inf*, *integer=False*, *name=None*)

Check whether given 'value' is a valid value (or integer) and if it is within the bounds of vmin/vmax.

Parameters

- **value** (*int or float*) – The value to check.
- **vmin** (*int or float, optional*) – Minimum the value is allowed to be. Default: -infinity (no bound)
- **vmax** (*int or float*) – Maxmum the value is allowed to be. Default: +infinity (no bound)
- **integer** (*bool, optional*) – Whether value must be an integer. Default: False (value can be float)

- **name** (*str, optional*) – The name of the value to be written in error. Default: None (the value is referred to as 'value').

Raises
InputError – If 'value' is not a valid value as given by parameters.

tfcomb.utils.**random_string**(*l=8*)

Get a random string of length l

class tfcomb.utils.**Progress**(*n_total*, *n_print*, *logger*)

Bases: object

write_progress(*n_done*)

Log the progress of the current tasks.

Parameters
n_done (*int*) – Number of tasks done (of n_total tasks)

tfcomb.utils.**log_progress**(*jobs*, *logger*, *n=10*)

Log progress of jobs within job list.

Parameters

- **jobs** (*list*) – List of multiprocessing jobs to write progress for.
- **logger** (*logger instance*) – A logger to use for writing out progress.
- **n** (*int, optional*) – Maximum number of progress statements to show. Default: 10.

tfcomb.utils.**prepare_motifs**(*motifs_file*, *motif_pvalue=0.0001*, *motif_naming='name'*)

Read motifs from motifs_file and set threshold/name.

tfcomb.utils.**open_genome**(*genome_f*)

Opens an internal genome object for fetching sequences.

Parameters
genome_f (*str*) – The path to a fasta file.

Return type
pysam.FastaFile

tfcomb.utils.**open_bigwig**(*bigwig_f*)

Parameters
bigwig_f (*str*) – The path to a bigwig file.

tfcomb.utils.**check_boundaries**(*regions*, *genome*)

Utility to check whether regions are within the boundaries of genome.

Parameters

- **regions** (*tobias.utils.regions.RegionList*) – A RegionList() object containing regions to check.
- **genome** (*pysam.FastaFile*) – An object (e.g. from open_genome()) to use as reference.

Raises
InputError – If a region is not available within genome

tfcomb.utils.**unique_region_names**(*regions*)

Get a list of unique region names within regions.

Parameters
regions (*tobias.utils.regions.RegionList*) – A RegionList() object containing regions with .name attributes.

Returns
The list of sorted names from regions.

Return type
list

tfcomb.utils.**calculate_TFBS**(*regions*, *motifs*, *genome*, *resolve='merge'*)

Multiprocessing-safe function to scan for motif occurrences

Parameters

- **genome** (*str or*) – If string , genome will be opened
- **regions** (*RegionList()*) – A RegionList() object of regions
- **resolve** (*str*) – How to resolve overlapping sites from the same TF. Must be one of "off", "highest_score" or "merge". If "highest_score", the highest scoring overlapping site is kept. If "merge", the sites are merged, keeping the information of the first site. If "off", overlapping TFBS are kept. Default: "merge".

Return type
List of TFBS within regions

tfcomb.utils.**resolve_overlaps**(*sites*, *how='merge'*, *per_name=True*)

Resolve overlapping sites within a list of genomic regions.

Parameters

- **sites** (*RegionList*) – A list of TFBS/regions with .chrom, .start, .end and .name information.
- **how** (*str*) – How to resolve the overlapping site. Must be one of "highest_score", "merge". If "highest_score", the highest scoring overlapping site is kept. If "merge", the sites are merged, keeping the information of the first site. Default: "merge".
- **per_name** (*bool*) – Whether to resolve overlapping only per name or across all sites. If 'True' overlaps are only resolved if the name of the sites are equal. If 'False', overlaps are resolved across all sites. Default: True.

tfcomb.utils.**add_region_overlap**(*a*, *b*, *att='overlap'*)

Overlap regions in regionlist 'a' with regions from regionlist 'b' and add a boolean attribute to the regions in 'a' containing overlap status with 'b'.

Parameters

- **a** (*list of OneTFBS objects*) – A list of objects containing genomic locations.
- **b** (*list of OneTFBS objects*) – A list of objects containing genomic locations to overlap with 'a' regions.
- **att** (*str, optional*) – The name of the attribute to add to 'a' objects. Default: "overlap".

tfcomb.utils.**shuffle_array**(*arr*, *seed=1*)

tfcomb.utils.**shuffle_sites**(*sites*, *seed=1*)

Shuffle TFBS names to existing positions and updates lengths of the new positions.

Parameters
sites (*np.array*) – An array of sites in shape (n_sites,4), where each row is a site and columns correspond to chromosome, start, end, name.

Return type
An array containing shuffled names with site lengths corresponding to original length of sites.

tfcomb.utils.**calculate_background**(*sites*, *seed=1*, *directional=False*, ***kwargs*)

Wrapper to shuffle sites and count co-occurrence of the shuffled sites.

Parameters

- **sites** (*np.array*) – An array of sites in shape (n_sites,4), where each row is a site and columns correspond to chromosome, start, end, name.
- **seed** (*int, optional*) – Seed for shuffling sites. Default: 1.
- **directional** (*bool*) – Decide if direction of found pairs should be taken into account. Default: False.
- **kwargs** (*arguments*) – Additional arguments for count_co_occurrence

tfcomb.utils.**get_threshold**(*data*, *which='upper'*, *percent=0.05*, *_n_max=10000*, *verbosity=0*, *plot=False*)

Function to get upper/lower threshold(s) based on the distribution of data. The threshold is calculated as the probability of "percent" (upper=1-percent).

Parameters

- **data** (*list or array*) – An array of data to find threshold on.
- **which** (*str*) – Which threshold to calculate. Can be one of "upper", "lower", "both". Default: "upper".
- **percent** (*float between 0-1*) – Controls how strict the threshold should be set in comparison to the distribution. Default: 0.05.

Return type
If which is one of "upper"/"lower", get_threshold returns a float. If "both", get_threshold returns a list of two float thresholds.

tfcomb.utils.**is_symmetric**(*matrix*)

Check if a matrix is symmetric around the diagonal

tfcomb.utils.**make_symmetric**(*matrix*)

Make a numpy matrix matrix symmetric by merging x-y and y-x

tfcomb.utils.**set_contrast**(*contrast*, *available_contrasts*)

Utility function for the plotting functions of tfcomb.objects.DiffCombObj

tfcomb.utils.**analyze_signal_chunks**(*datasource*, *threshold*)

Evaluating signal for chunks.

Parameters

- **datasource** (*pd.DataFrame*) – A (sub-)Dataframe with the (corrected) distance counts for the pairs
- **threshold** (*float*) – Threshold for prominence and height in peak calling (see scipy.signal.find_peaks() for detailed information)

Returns
list of found peaks in form [TF1, TF2, Distance, Peak Heights, Prominences, Prominence Threshold]

Return type
list

See also:

`tfcomb.object.analyze_signal_all`

`tfcomb.utils.evaluate_noise_chunks(signals, peaks, method='median', height_multiplier=0.75)`

Evaluate the noisiness of a signal for chunks (a chunk can also be the whole dataset).

Parameters

- **pairs** (*list(tuples(str,str))*) – list of pairs to perform analysis on
- **signals** (*pd.Dataframe*) – A (sub-)Dataframe containing signal data for pairs
- **method** (*str, otional*) – Method used to get noise measurement, either "median" or "min_max" allowed. Default: "median"
- **height_multiplier** (*float, optional*) – Height multiplier (percentage) to calculate cut points. Must be between 0 and 1. Default: 0.75

Raises
ValueError – If no signal data is given for a pair

Note: Constraint: DataFrame with signals need to contain a signal for each pair given within pairs.

`tfcomb.utils.getAllAttr(object, private=False, functions=False)`

Collect all attributes of an object and return as dict.

Parameters

- **private** (*boolean, default False*) – If private attributes should be included. Everything with '_' prefix.
- **functions** (*boolean, default False*) – If callable attributes ie functions shoudl be included.

Returns
Dict of all the objects attributes.

Return type
dictionary

1.5 Changelog

1.5.1 1.1 (11-08-2023)

- Fix for floating-point error handling (#55)
- Fix error reading MEME files without "name" per motif (#64)
- Fix get_pair_locations for motif lists with length > 1000 (#62)
- Added figsize to plotting.genome_view (#69)

1.5.2 1.0.3 (23-01-2023)

- Added pyproject.toml to fix error introduced in 1.0.2 when installing with pip

1.5.3 1.0.2 (20-01-2023)

- Fixed missing readthedocs documentation
- Added a warning if not at least one annotation was found with the given config (annotate_regions())

1.5.4 1.0.1 (24-11-2022)

Fixed error when importing mpl_toolkits.axes_grid for matplotlib >= 3.6 (#47); the import now depends on the matplotlib version

1.5.5 1.0.0 (25-10-2022)

- Initial release to PyPI
- Start of versioning

PYTHON MODULE INDEX

t

tfcomb.analysis, 96
tfcomb.annotation, 98
tfcomb.network, 94
tfcomb.objects, 76
tfcomb.plotting, 100
tfcomb.utils, 103

INDEX

A

add_object() (*tfcomb.objects.DiffCombObj method*), 85
add_region_overlap() (*in module tfcomb.utils*), 110
analyze_distances() (*tfcomb.objects.CombObj method*), 84
analyze_hubs() (*tfcomb.objects.DistObj method*), 92
analyze_orientation() (*tfcomb.objects.CombObj method*), 84
analyze_signal_all() (*tfcomb.objects.DistObj method*), 90
analyze_signal_chunks() (*in module tfcomb.utils*), 111
annotate_regions() (*in module tfcomb.annotation*), 98
append() (*tfcomb.utils.TFBSPairList method*), 103
as_table() (*tfcomb.utils.TFBSPairList method*), 103
aspect_translation (*tfcomb.annotation.GOAnalysis attribute*), 99

B

bigwig_path (*tfcomb.utils.TFBSPairList property*), 104
bubble() (*in module tfcomb.plotting*), 100
build_network() (*in module tfcomb.network*), 94
build_network() (*tfcomb.objects.CombObj method*), 84
build_network() (*tfcomb.objects.DistObj method*), 92

C

calculate_background() (*in module tfcomb.utils*), 111
calculate_foldchanges() (*tfcomb.objects.DiffCombObj method*), 85
calculate_TFBS() (*in module tfcomb.utils*), 110
check_boundaries() (*in module tfcomb.utils*), 109
check_columns() (*in module tfcomb.utils*), 107
check_datasource() (*tfcomb.objects.DistObj method*), 89
check_dir() (*in module tfcomb.utils*), 107
check_graphtool() (*in module tfcomb.utils*), 107
check_min_max_dist() (*tfcomb.objects.DistObj method*), 89
check_module() (*in module tfcomb.utils*), 107
check_pair() (*tfcomb.objects.CombObj method*), 76
check_peaks() (*tfcomb.objects.DistObj method*), 89
check_string() (*in module tfcomb.utils*), 108
check_type() (*in module tfcomb.utils*), 108
check_value() (*in module tfcomb.utils*), 108
check_writeability() (*in module tfcomb.utils*), 108
chunk_table() (*tfcomb.objects.DistObj static method*), 89
classify_rules() (*tfcomb.objects.DistObj method*), 92
clear() (*tfcomb.utils.TFBSPairList method*), 104
cluster_blockmodel() (*in module tfcomb.network*), 95
cluster_louvain() (*in module tfcomb.network*), 95
cluster_network() (*tfcomb.objects.CombObj method*), 84
cluster_TFBS() (*tfcomb.objects.CombObj method*), 78
CombObj (*class in tfcomb.objects*), 76
comp_plotting_tables() (*tfcomb.utils.TFBSPairList method*), 104
compare() (*tfcomb.annotation.GOAnalysis method*), 99
compare() (*tfcomb.objects.CombObj method*), 84
copy() (*tfcomb.objects.CombObj method*), 76
correct_background() (*tfcomb.objects.DistObj method*), 90
count_distances() (*tfcomb.objects.DistObj method*), 89
count_peaks() (*tfcomb.objects.DistObj method*), 92
count_within() (*tfcomb.objects.CombObj method*), 79
create_distObj() (*tfcomb.objects.CombObj method*), 83
create_random_network() (*in module tfcomb.network*), 96

D

DiffCombObj (*class in tfcomb.objects*), 85
DistObj (*class in tfcomb.objects*), 88

E

enrichment() (*tfcomb.annotation.GOAnalysis method*), 99

evaluate_noise() (*tfcomb.objects.DistObj method*), 91
evaluate_noise_chunks() (*in module tfcomb.utils*), 112
extend() (*tfcomb.utils.TFBSPairList method*), 103

F

fill_rules() (*tfcomb.objects.DistObj method*), 88
from_pickle() (*tfcomb.objects.CombObj method*), 76
from_pickle() (*tfcomb.objects.DiffCombObj method*), 88

G

genome_view() (*in module tfcomb.plotting*), 102
get_annotated_genes() (*in module tfcomb.annotation*), 98
get_betweenness_centrality() (*in module tfcomb.network*), 95
get_degree() (*in module tfcomb.network*), 94
get_edge_table() (*in module tfcomb.network*), 95
get_node_table() (*in module tfcomb.network*), 95
get_pair_locations() (*tfcomb.objects.CombObj method*), 79
get_periodicity() (*tfcomb.objects.DistObj method*), 92
get_threshold() (*in module tfcomb.utils*), 111
getAllAttr() (*in module tfcomb.utils*), 112
go_bubble() (*in module tfcomb.plotting*), 101
GOAnalysis (*class in tfcomb.annotation*), 99

H

heatmap() (*in module tfcomb.plotting*), 100

I

InputError, 107
insert() (*tfcomb.utils.TFBSPairList method*), 103
integrate_data() (*tfcomb.objects.CombObj method*), 82
is_smoothed() (*tfcomb.objects.DistObj method*), 90
is_symmetric() (*in module tfcomb.utils*), 111

L

log_progress() (*in module tfcomb.utils*), 109

M

make_symmetric() (*in module tfcomb.utils*), 111
market_basket() (*tfcomb.objects.CombObj method*), 80
max_distance() (*tfcomb.objects.DistObj method*), 91
mean_distance() (*tfcomb.objects.DistObj method*), 91
module
    tfcomb.analysis, 96
    tfcomb.annotation, 98
    tfcomb.network, 94
    tfcomb.objects, 76
    tfcomb.plotting, 100
    tfcomb.utils, 103

N

network() (*in module tfcomb.plotting*), 101
normalize() (*tfcomb.objects.DiffCombObj method*), 85

O

OneTFBS (*class in tfcomb.utils*), 103
open_bigwig() (*in module tfcomb.utils*), 109
open_genome() (*in module tfcomb.utils*), 109
orientation() (*in module tfcomb.analysis*), 96
OrientationAnalysis (*class in tfcomb.analysis*), 97

P

pairLines() (*tfcomb.utils.TFBSPairList method*), 107
pairMap() (*tfcomb.utils.TFBSPairList method*), 104
pairTrack() (*tfcomb.utils.TFBSPairList method*), 105
pairTrackAnimation() (*tfcomb.utils.TFBSPairList method*), 106
plot() (*tfcomb.objects.DistObj method*), 93
plot_autocorrelation() (*tfcomb.objects.DistObj method*), 92
plot_bg_estimation() (*tfcomb.objects.DistObj method*), 92
plot_bubble() (*tfcomb.annotation.GOAnalysis method*), 99
plot_bubble() (*tfcomb.objects.CombObj method*), 83
plot_bubble() (*tfcomb.objects.DiffCombObj method*), 87
plot_correlation() (*tfcomb.objects.DiffCombObj method*), 86
plot_distances() (*tfcomb.utils.TFBSPairList method*), 107
plot_heatmap() (*tfcomb.analysis.OrientationAnalysis method*), 97
plot_heatmap() (*tfcomb.objects.CombObj method*), 83
plot_heatmap() (*tfcomb.objects.DiffCombObj method*), 86
plot_network() (*tfcomb.objects.CombObj method*), 84
plot_network() (*tfcomb.objects.DiffCombObj method*), 87
plot_network() (*tfcomb.objects.DistObj method*), 93
plot_powerlaw() (*in module tfcomb.network*), 96
plot_rules_heatmap() (*tfcomb.objects.DiffCombObj method*), 86
plot_scatter() (*tfcomb.objects.CombObj method*), 83
plot_TFBS() (*tfcomb.objects.CombObj method*), 82
plotting_tables (*tfcomb.utils.TFBSPairList property*), 104
pop() (*tfcomb.utils.TFBSPairList method*), 104
prepare_motifs() (*in module tfcomb.utils*), 109

Progress (*class in tfcomb.utils*), 109

R

random_string() (*in module tfcomb.utils*), 109
rank_rules() (*tfcomb.objects.DistObj method*), 91
reduce_TFBS() (*tfcomb.objects.CombObj method*), 80
remove() (*tfcomb.utils.TFBSPairList method*), 103
reset_signal() (*tfcomb.objects.DistObj method*), 89
resolve_overlaps() (*in module tfcomb.utils*), 110

S

scale() (*tfcomb.objects.DistObj method*), 90
scatter() (*in module tfcomb.plotting*), 100
select_custom_rules() (*tfcomb.objects.CombObj method*), 81
select_rules() (*tfcomb.objects.DiffCombObj method*), 86
select_significant_rules() (*tfcomb.objects.CombObj method*), 81
select_TF_rules() (*tfcomb.objects.CombObj method*), 80
select_top_rules() (*tfcomb.objects.CombObj method*), 81
set_contrast() (*in module tfcomb.utils*), 111
set_orientation() (*tfcomb.utils.TFBSPairList method*), 107
set_prefix() (*tfcomb.objects.CombObj method*), 76
set_verbosity() (*tfcomb.objects.CombObj method*), 76
set_verbosity() (*tfcomb.objects.DistObj method*), 88
shuffle_array() (*in module tfcomb.utils*), 110
shuffle_sites() (*in module tfcomb.utils*), 110
simplify_rules() (*tfcomb.objects.CombObj method*), 80
smooth() (*tfcomb.objects.DistObj method*), 90
StopExecution, 107
subset_graph() (*in module tfcomb.network*), 95
subset_TFBS() (*tfcomb.objects.CombObj method*), 78

T

TFBS_from_bed() (*tfcomb.objects.CombObj method*), 77
TFBS_from_motifs() (*tfcomb.objects.CombObj method*), 77
TFBS_from_TOBIAS() (*tfcomb.objects.CombObj method*), 78
TFBS_to_bed() (*tfcomb.objects.CombObj method*), 78
TFBSPair (*class in tfcomb.utils*), 103
TFBSPairList (*class in tfcomb.utils*), 103
tfcomb.analysis
    module, 96
tfcomb.annotation
    module, 98
tfcomb.network
    module, 94
tfcomb.objects
    module, 76
tfcomb.plotting
    module, 100
tfcomb.utils
    module, 103
to_pickle() (*tfcomb.objects.CombObj method*), 76
to_pickle() (*tfcomb.objects.DiffCombObj method*), 88

U

unique_region_names() (*in module tfcomb.utils*), 109

W

write_bed() (*tfcomb.utils.TFBSPairList method*), 103
write_progress() (*tfcomb.utils.Progress method*), 109