What is GimmeMotifs?

GimmeMotifs is an analysis framework for transcription factor motif analysis written in Python. It contains command-line scripts to predict de novo motifs, scan for known motifs, identify differential motifs, calculate motif enrichment statistics, plot sequence logos and more. In addition, all this functionality is available from a Python API.

GimmeMotifs is free and open source research software. If you find it useful please cite our paper:

CHAPTER 2

Getting started

• The easiest way to install GimmeMotifs is using bioconda on Linux or Mac. From version 0.13.0 only Python 3 (>= 3.4) is supported.
• Have a look at these simple examples to get a taste of what is possible.
• Check out the more detailed tutorials.
• Full command-line reference can be found here.
• There’s also an API documentation.
Get help

• First, check the FAQ for common issues.
• The preferred way to get support is through the Github issues page.
• Finally, you can reach me by mail or via twitter.
4.1 Installation

GimmeMotifs runs on Linux. On Windows 10 it will run fine using the Windows Subsystem for Linux. Mac OSX should work and is included in the build test. However, as I don’t use it myself, unexpected issues might pop up. Let me know, so I can try to fix it.

4.1.1 The easiest way to install

The preferred way to install GimmeMotifs is by using conda. Activate the bioconda channel if you haven’t used bioconda before. You only have to do this once.

```
$ conda config --add channels defaults
$ conda config --add channels bioconda
$ conda config --add channels conda-forge
```

You can install GimmeMotifs with one command. In the current environment:

```
$ conda install gimmemotifs
```

Or create a specific environment:

```
$ conda create -n gimme python=3 gimmemotifs
# Activate the environment before you use GimmeMotifs
$ conda activate gimme
```

GimmeMotifs only supports Python 3. Don’t forget to activate the environment with `source activate gimme` whenever you want to use GimmeMotifs.

Installation successful? Good. Have a look at the configuration section.
Important note on upgrading from 0.11.1

The way genomes are installed and used has been changed from 0.11.1 to 0.12.0. Basically, we have switched to the faidx index used and supported by many other tools. This means that the old (<=0.11.1) GimmeMotifs index cannot be used by GimmeMotifs 0.12.0 and higher. You can re-install genomes using genomepy, which is now the preferred tool for genome management for GimmeMotifs. However, because of this change you can now also directly supply a genome FASTA instead of a genome name. Pre-indexing is not required anymore.

4.1.2 Alternative installation

These are the prerequisites for a full GimmeMotifs installation.

- bedtools http://bedtools.readthedocs.io
- UCSC genePredToBed http://hgdownload.cse.ucsc.edu/admin/exe/linux.x86_64/genePredToBed
- UCSC bigBedToBed http://hgdownload.cse.ucsc.edu/admin/exe/linux.x86_64/bigBedToBed
- R + RobustRankAggreg https://cran.r-project.org/web/packages/RobustRankAggreg/index.html
- Perl + Algorithm::Cluster

In addition many of the motif tools (such as MEME) will need to be installed separately. Instructions for doing so are not included here.

Installation from PyPI with pip is a relatively straightforward option. Install with pip as follows:

```bash
dsudo pip install gimmemotifs
```

Or the (unstable) develop branch with the newest bells, whistles and bugs:

```bash
dsudo pip install git+https://github.com/vanheeringen-lab/gimmemotifs.git@develop
```

If you don’t have root access, see the option below.

Ubuntu prerequisites

To install GimmeMotifs in a virtualenv, several Python packages need to be built from source.

Install the necessary packages to build numpy, scipy, matplotlib and GimmeMotifs:

```bash
sudo apt-get install python-pip python-dev build-essential libatlas-base-dev \
gfortran liblapack-dev libatlas-base-dev cython libpng12-dev libfreetype6-dev \
libgsl0-dev
```

Install via pip

Create a virtualenv and activate it according to the documentation.

Install numpy:

```bash
$ pip install numpy
```

Now you can install GimmeMotifs using pip. Latest stable release:

```bash
$ pip install gimmemotifs
```
Installation from source

Did I mention conda?
You know bioconda is amazing, right?
So...
These instructions are not up-to-date! Basically, you’re on your own!
Make sure to install all required dependencies.
You can download the latest stable version of GimmeMotifs at:

https://github.com/simonvh/gimmemotifs/releases

Start by unpacking the source archive

```
tar xvzf gimmemotifs-0.11.0.tar.gz
cd gimmemotifs-0.11.0
```

You can build GimmeMotifs with the following command:

```
python setup.py build
```

Run the tests to check if the basics work correctly:

```
python run_tests.py
```

If you encounter no errors, go ahead with installing GimmeMotifs (root privileges required):

```
sudo python setup.py install
```

On first run GimmeMotifs will try to locate the tools you have installed. If you have recently installed them, running an `updatedb` will be necessary. Using this option GimmeMotifs will create a configuration file, the default is:

```
~/.config/gimmemotifs/gimmemotifs.cfg
```

This is a personal configuration file.

It is also possible to run the `setup.py install` command with the `--prefix`, `--home`, or `--install-data` options, to install in GimmeMotifs in a different location (for instance, in your own home directory). This should be fine, however, these alternative methods of installing GimmeMotifs have not been extensively tested.

4.1.3 Configuration

You will need genome FASTA files for a lot of the tools that are included with GimmeMotifs.

Download genomes automatically

The most straightforward way to download and index a genome is to use the `genomepy` tool, which is installed with GimmeMotifs.

```
$ genomepy install hg19 UCSC --annotation
```
Here, the hg19 genome and accompanying gene annotation will be downloaded from UCSC to the directory ~/.local/share/genomes/hg19. You can change this default location by creating/editing the file ~/.config/genomepy/genomepy.yaml and change the following line:

```
genome_dir: /data/genomes
```

Please note: in contrast to earlier versions of GimmeMotifs it is no longer necessary to index a genome.

**Adding gene files**

Note: If you used the genomepy command, annotation will be included automatically.

For some applications a gene file is used. This is a file containing gene annotation in BED12 format. It should be located in the gene_dir, which is defined in the configuration file (see below). The file needs to be named <index_name>.bed, so for instance hg19.bed.

**Other configuration options**

All of GimmeMotifs configuration is stored in ~/.config/gimmemotifs/gimmemotifs.cfg. The configuration file is created at first run with all defaults set, but you can always edit it afterwards. It contains two sections main and params that take care of paths, file locations, parameter settings etc. Additionally, every motif tool has its own section. Let’s have a look at the options.

```
[main]
template_dir = /usr/share/gimmemotifs/templates
seqlogo = /usr/local/bin/seqlogo
score_dir = /usr/share/gimmemotifs/score_dists
motif_databases = /usr/share/gimmemotifs/motif_databases
gene_dir = /usr/share/gimmemotifs/genes
tools_dir = /usr/share/gimmemotifs/tools
```

- **template_dir** The location of the jinja2 html templates, used to generate the reports.
- **seqlogo** The seqlogo executable.
- **score_dir** To generate p-values, a pre-calculated file with mean and sd of score distributions is needed. These are located here.
- **motif_databases** For now contains only the JASPAR motifs.
- **gene_dir** Directory with bed-files containing gene locations. This is needed to create promoter background sequences.
- **tools_dir** Here all tools included with GimmeMotifs are stored.

```
[params]
fraction = 0.2
use_strand = False
abs_max = 1000
analysis = medium
enrichment = 1.5
width = 200
lwidth = 500
genome = hg19
background = gc,random
cluster_threshold = 0.95
available_tools = MDmodule,MEME,Weeder,GADEM,MotifSampler,trawler,Improbizer,BioProspector,Psemo,CHIPunk,JASPAR,AME,HMS,Homer
```

(continues on next page)
This section specifies all the default GimmeMotifs parameters. Most of these can also be specified at the command-line when running GimmeMotifs, in which case they will override the parameters specified.

**Configuration of MotifSampler**

If you want to use MotifSampler there is one more step that you'll have to take after installation of GimmeMotifs. For every organism, you will need a MotifSampler background. Note that human (hg19, hg38) and mouse (mm9, mm10) background models are included, so for these organisms MotifSampler will work out of the box. For other organisms the necessary background files can be created with CreateBackgroundModel (which is included with GimmeMotifs or can be downloaded from the same site as MotifSampler). The background model file needs to be saved in the directory `/usr/share/gimmemotifs/MotifSampler` and it should be named `<organism_index_name>.bg`. So, for instance, if I downloaded the human epd background (`epd_homo_sapiens_499_chromgenes_non_split_3.bg`), this file should be saved as `/usr/share/gimmemotifs/MotifSampler/hg19.bg` here.

### 4.2 Overview

#### 4.2.1 Motif databases

By default GimmeMotifs uses a non-redundant, clustered database of known vertebrate motifs: `gimme.vertebrate.v5.0`. These motifs come from CIS-BP (http://cisbp.ccbr.utoronto.ca/) and other sources. Large-scale benchmarks using ChIP-seq peaks show that this database shows good performance and should be a good default choice.

In addition, many other motif databases come included with GimmeMotifs:

- **CIS-BP** - All motifs from the CIS-BP database (version 1.02).
- **ENCODE** - ENCODE motifs from Kheradpour & Kellis (2013).
- **factorbook** - All non-redundant motifs from Factorbook.
- **HOCOMOCOv11_HUMAN** - All human motifs from HOCOMOCO version 11.
- **HOCOMOCOv11_MOUSE** - All mouse motifs from HOCOMOCO version 11.
- **HOMER** - All motifs from HOMER (downloaded Oct. 2018).
- **IMAGE** - The motif database from Madsen et al. (2018).
- **JASPAR2018** - All CORE motifs from JASPAR 2018.
- **JASPAR2018_vertebrates** - CORE vertebrates motifs from JASPAR 2018.
- **JASPAR2018_plants** - CORE plants motifs from JASPAR 2018.
• JASPAR2018_insects - CORE insects motifs from JASPAR 2018.
• JASPAR2018_fungi - CORE fungi motifs from JASPAR 2018.
• JASPAR2018_nematodes - CORE nematodes motifs from JASPAR 2018.
• JASPAR2018_urochordata - CORE urochordata motifs from JASPAR 2018.
• SwissRegulon - The SwissRegulon motifs.

You can specify any of these motif databases by name in any GimmeMotifs tool. For instance:

```bash
$ gimme scan TAp73alpha.fa -p JASPAR2018_vertebrates
```

or

```bash
$ gimme roc TAp73alpha.fa bg.fa -p HOMER -r roc.report
```

4.3 Simple examples

4.3.1 Install a genome

Any genome on UCSC, Ensembl or NCBI can be installed automatically using genomepy. The genomepy command tool comes installed with gimmemotifs. For instance, to download the hg38 genome from UCSC:

```bash
$ genomepy install hg38 UCSC --annotation
```

4.3.2 Predict de novo motifs

For a quick analysis, to see if it works:

```bash
$ gimme motifs TAp73alpha.bed -g hg19 -n gimme.p73 -a small -t Homer,MDmodule, ...
```

For a full analysis:

```bash
$ gimme motifs TAp73alpha.bed -g hg19 -n gimme.p73
```

Open the file gimme.p73/motif_report.html in your web browser to see the results.

4.3.3 Compare motifs between data sets

```bash
$ gimme maelstrom hg19.blood.most_variable.1k.txt hg19 maelstrom.out/
```

The output scores of gimme maelstrom represents the combined result of multiple methods. The individual results from different methods are ranked from high-scoring motif to low-scoring motif and then aggregated using rank aggregation. The score that is shown is the -log10(p-value), where the p-value (from the rank aggregation) is corrected for multiple testing. This procedure is then repeated with the ranking reversed. These are shown as negative values.
4.3.4 Create sequence logos

$ gimme logo -i p53_Average_8

This will generate p53_Average_8.png.

Use the `-p` argument for a different pwm file. The following command will generate a logo for every motif in custom.pwm.

$ gimme logo -p custom.pwm

4.4 Tutorials

While GimmeMotifs was originally developed to predict de novo motifs in ChIP-seq peaks, it is now a full-fledged suite of TF motif analysis tools. You can still identify new motifs, but also scan for known motifs, find differential motifs in multiple sets of sequences, create sequence logos, calculate all kinds of enrichment statistics, and more!

For this tutorial I’ll assume you use bioconda. If you haven’t already done so, install GimmeMotifs.

$ conda create -n gimme python=3 gimmemotifs

And activate it!

$ source activate gimme

To locate the example files mentioned in the tutorial, locate the examples/ directory of your GimmeMotifs installation. When using conda:

$ echo `conda info | grep default | awk '{print $4}'`/share/gimmemotifs/examples
/home/simon/anaconda3/share/gimmemotifs/examples

Alternatively, the example data is also available from figshare and you can download it from there.

$ curl -L -o gimme.example_data.tgz https://ndownloader.figshare.com/files/8834965
$ tar xzvf gimme.example_data.tgz
### 4.4.1 Find de novo motifs

As a simple example, let’s predict the CTCF motif based on ChIP-seq data from ENCODE. Download the peaks:

```bash

→ wgEncodeBroadHistoneGm12878CtcfStdAlnRep0_VS_wgEncodeBroadHistoneGm12878ControlStdAlnRep0.bb
```

Convert the bigBed file to a BED file using `bigBedToBed`:

```bash
$ bigBedToBed spp.optimal.wgEncodeBroadHistoneGm12878CtcfStdAlnRep0_VS_wgEncodeBroadHistoneGm12878ControlStdAlnRep0.bb Gm12878.CTCF.narrowPeak
```

Select the top 500 peaks, based on the signalValue column of the `narrowPeak` format, as input:

```bash
$ sort -k7gr Gm12878.CTCF.bed | head -n 500 > Gm12878.CTCF.top500.narrowPeak
```

Note that the top 500 peaks are just for the sake of the tutorial. Normally you would use a much larger sample (or all peaks) as input for `gimme motifs`.

Now, the ENCODE peak coordinates are based on hg19 so we need to install the hg19 genome. For a UCSC genome, this is just a matter of running `genomepy`.

```bash
$ genomepy install hg19 UCSC --annotation
```

This will take some time. The genome sequence will be downloaded and indexed, ready for use with GimmeMotifs.

Having both an index genome and an input file, we can run `gimme motifs`.

```bash
$ gimme motifs Gm12878.CTCF.top500.narrowPeak -g hg19 -n gimme.CTCF
```

Once again, this will take some time. When `gimme motifs` is finished you can view the results in a web browser. `gimme.CTCF/motif_report.html` should look a lot like this. This is what an almost perfect motif looks like, with a ROC AUC close to 1.0.

### 4.4.2 Scan for known motifs

**Note:** `gimme scan` can be used to identify motif locations. If you’re just interested in identifying enriched motifs in a data set, try `gimme roc`.

To scan for known motifs, you will need a set of input sequences and a file with motifs. By default, `gimme scan` uses the motif database that comes included, which is based on clustered, non-redundant motifs from CIS-BP. For input sequences you can use either a BED file, a FASTA file or a file with regions in `chr:start-end` format. You will also need to specify the genome, which can either be a genome installed with `genomepy` or a FASTA file. The genome sequence will be used to retrieve sequences, if you have specified a BED or region file, but also to determine a reasonable motif-specific threshold for scanning. The default genome can be specified in the configuration file.

We will use the file `Gm12878.CTCF.top500.narrowPeak` that was used for de novo motif search above for known motifs. While `gimme motifs` automatically extends regions from the center of the input regions, `gimme scan` uses the regions as specified in the file. This means we will have to change the size of the regions to 200 nucleotides. Depending on the type and quality of your input data, you can of course make this smaller or larger.

```bash
$ cat Gm12878.CTCF.top500.narrowPeak | awk '{print $1 "t" $2 + $10 - 100 "t" $2 + -$10 + 100}' > Gm12878.CTCF.top500.w200.bed
```

OK, let’s scan:
The first time you run `gimme scan` for a specific combination of motif database, genome, input sequence length and FPR (which is 0.01 by default) it will determine a motif-specific cutoff based on random genome background sequences. This will take a while. However, results will be cached for future scanning.

To get a BED file with the genomic location of motif matches add the `-b` argument:

```bash
$ gimme scan Gm12878.CTCF.top500.w200.bed -g hg19 -b > result.scan.bed
```

By default, `gimme scan` gives at most one match per sequence for each motif, if the score of the match reaches a certain threshold.

For a very simple summary, we can just have a look at the most abundant motifs:

```bash
$ cut -f4 result.scan.bed | sort | uniq -c | sort -n | tail
50 E2F_Average_31
58 C2H2_ZF_Average_123
58 MBD_Average_1
58 THAP_finger_M1541_1.01
59 Unknown_Average_5
67 Ets_Average_70
72 C2H2_ZF_M0401_1.01
72 CxxC_M0548_1.01
118 E2F_Average_27
394 C2H2_ZF_Average_200
```

In this case, the most abundant motif is the CTCF motif.

The specified false positive rate (FPR), with a default of 0.01, determines the motif-specific threshold that is used for scanning. This means that the expected rate of occurrence, determined by scanning random genomic sequences, is 1%. Based on the FPR, you can assume that any motif with more than 1% matches is enriched. However, for a more robust measure of motif significance use `gimme roc`, which is further explained below. This command will give the enrichment, but also the ROC AUC and recall at 10% FDR and other useful statistics.

For many applications, it is useful to have motif occurrences as a table.

```bash
$ gimme scan Gm12878.CTCF.top500.w200.bed -g hg19 -t > table.count.txt
```

This will result in a tab-separated table with counts. Same defaults as above, at most one match per sequence per motif. Alternatively, `gimme scan` can report the score of best match, regardless of the value of this score.

```bash
$ gimme scan Gm12878.CTCF.top500.w200.bed -g hg19 -T > table.score.txt
$ head table.score.txt | cut -f1-10
# GimmeMotifs version 0.10.1b2
# Input: Gm12878.CTCF.top500.w200.bed
# Motifs: /home/simon/anaconda3/share/gimmemotifs/motif_databases/gimme.vertebrate.v3.1.pwm
# FPR: 0.01 (hg19)
AP-2_Average_26 AP-2_Average_17 AP-2_Average_27 AP-2_Average_15 AP-2_M5965_1.01 ARID_BRIGHT_Average_1 ARID_BRIGHT_M0104_1.01 ARID_BRIGHT_Average_3 ARID_BRIGHT_M5966_1.01
chr11:190037-190237 3.315682 5.251773 5.852259 6.986044 -0.032952 -1.058302 -4.384525 1.989879 -13.872373
chr14:106873577-106873777 3.485541 5.315545 3.867055 1.129976 -3.305211 -1.392656 2.726421 -8.660561
```

(continues on next page)
4.4.3 Find differential motifs

The gimme maelstrom command can be used to compare two or more different experiments. For instance, ChIP-seq peaks for multiple factors, ChIP-seq peaks of the same factor in different cell lines or tissues, ATAC-seq peaks or expression data.

The input can be in one two possible formats. In both cases the genomic location should be present as chrom:start-end in the first column. The first option is a two-column format and looks like this:

<table>
<thead>
<tr>
<th>loc</th>
<th>cluster</th>
<th>chr15:49258903-49259103</th>
<th>NK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>chr10:72370313-72370513</td>
<td>NK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chr4:40579259-40579459</td>
<td>NK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chr10:82225678-82225878</td>
<td>Monocytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chr5:134237941-134238141</td>
<td>T-cells</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chr5:58858731-58858931</td>
<td>B-cells</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chr20:24941608-24941808</td>
<td>NK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chr5:124203316-124203316</td>
<td>NK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chr17:4009476-40094676</td>
<td>Erythroblast</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chr17:28659327-28659527</td>
<td>T-cells</td>
</tr>
</tbody>
</table>

This can be the result of a clustering analysis, for instance.

The second option looks like this:

<table>
<thead>
<tr>
<th>loc</th>
<th></th>
<th>chr12:93507547-93507747</th>
<th>Monocytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>chr7:38236460-38236660</td>
<td>3.11846121722</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chr10:21357147-21357347</td>
<td>1.0980120443</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chr6:115521512-115521712</td>
<td>0.46247786632</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chr2:9735908-97360008</td>
<td>1.5016209256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chr16:1668454-16684749</td>
<td>0.2338338577502</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chrX:138964544-138964744</td>
<td>0.330000689312</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chr2:186923973-186924173</td>
<td>0.430448401897</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chrX:113834470-113834670</td>
<td>0.560122313347</td>
</tr>
</tbody>
</table>

This is a tab-separated table, with a header describing the experiments. The values can be (log-transformed) read counts, expression values or other measurements.

By default, gimme maelstrom will run in ensemble mode, where it will combine the results from different classification and regression methods and statistical tests through rank aggregation. The only arguments necessary are the
input file, the genome and an output directory.

Here, we will run maelstrom on a dataset that is based on Corces et al. The example file hg19.blood.most_variable.1k.txt contains normalized ATAC-seq read count data for several hematopoietic cell types: Monocytes, CD4+ and CD8+ T cells, NK cells, B cells and erythrocytes. This is a subset of the data and contains only the 1000 most variable peaks (highest standard deviation). There is also a larger file, that contains more regions hg19.blood.most_variable.10k.txt and that will also take longer to run.

```
$ gimme maelstrom hg19.blood.most_variable.1k.txt hg19 maelstrom.blood.1k.out
```

There output directory contains several files:

```
$ ls maelstrom.blood.1k.out
```

The two motif files, motif.count.txt.gz and motif.score.gz contain the motif scan results. The activity.*.out.txt files are tables with the results of the individual methods. The main result is final.out.csv, which integrates all individual methods in a final score. This score represents the combined result of multiple methods. The individual results from different methods are ranked from high-scoring motif to low-scoring motif and then aggregated using the rank aggregation method from ‘Kolde, 2012<https://www.ncbi.nlm.nih.gov/pubmed/22247279>‘. The score that is shown is the -log10(p-value), where the p-value (from the rank aggregation) is corrected for multiple testing. This procedure is then repeated with the ranking reversed. These are shown as negative values.

The file gimme.maelstrom.report.html contains a graphical summary of this file that can be opened in your web browser.
The following Python snippet will create a heatmap of the results.

```python
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_table("maelstrom.blood.1k.out/final.out.csv", index_col=0)
m2f = pd.read_table("/home/simon/git/gimmemotifs/motif_databases/gimme.vertebrate.v3.˓
˓
˓
˓→1.motif2factors.txt", index_col=0)
m2f.factors = m2f.factors.str.slice(0,50)
df = df.join(m2f).set_index("factors")
df = df["Mono", "CD4", "CD8", "Bcell", "Nkcell", "Ery"]

cm = sns.clustermap(df[np.any(abs(df) >= 6, 1)], figsize=(4,15))
plt.setp(cm.ax_heatmap.yaxis.get_majorticklabels(), rotation=0);
plt.savefig("maelstrom.blood.1k.out/heatmap.png")
```

This will show a heatmap like this:
4.4. Tutorials
We see that the expected motifs for different cell types are identified. GATA/LMO2 for Erythrocytes, LEF/TCF for T cells (ie. Wnt signaling), EBF1 and PAX5 for B cells and so on. The RUNX motif is only identified in CD8+ T cells and not for CD4+ T cells, which recapitulates a known mechanism in CD4- versus CD8-positive T cell differentiation. It is kind of tricky to get the seaborn clustermapper to use reasonable dimensions by default, so play around with the figsize parameter to get it to work. Keep in mind that this shows only the most relevant motifs (-log10 p-value cutoff of 6), there are more relevant motifs. A file with more regions, hg19.blood.most_variable.10k.txt for this example, will usually yield better results.

### 4.4.4 Compare two sets with de novo motifs

gimme motifs

combine: gimme cluster

scan

#### 4.4.5 Motif enrichment statistics

You can use gimme roc to compare motifs or to identify relevant known motifs for a specific input file.

Let’s get some known motifs for one of the example files, TAp73alpha.fa. First, we need to define a background. To get random genomic sequences with a matched GC% content:

```
$ gimme background random.gc.fa gc -g hg19 -n 500 -l 200 -i TAp73alpha.fa
```

This will create a FASTA file with 500 sequences of 200 nucleotides, that has a GC% distribution similar to TAp73alpha.fa. Now we can run gimme roc:

```
$ gimme roc TAp73alpha.fa random.gc.fa -r TAp73alpha.roc
```

This will create an output directory with two files (and a dir with motif logos).

```
$ ls TAp73alpha.roc
gimme.roc.report.html gimme.roc.report.txt logos
```

The file **gimme.roc.report.html** is a graphical report that can be opened in your web browser. It should look something like this.
The columns are sortable (click on the header) and the full list of factors that can bind to this motif can be obtained by hovering over the text.

The file `gimme.roc.report.txt` is a text report of the same results. If you don’t need the graphical result you can leave out the `-r` argument in which case the text output will be printed to `stdout`.

What’s in the file?

```
$ head -n 1 TAp73alpha.roc/gimme.roc.report.txt | tr '\t' ' ' 
Motif                     # matches # matches background P-value log10 P-value corrected P-value ROC AUC Enr. at 1% FPR Recall at 10% FDR
TP33,TP3,TP3,TRP3,TRP3,TRP3,TRP3... 453 5 5.6e-91 90.3 3.36e-88 0.918 45.3 0.982
TRP3                      237 5 2.57e-50 18.0 8.01e-37 0.816 23.7 0.923
TRP3,TRP3,TP3,TRP3,TRP3    46  5 7.14e+05 4.11 3.00e+23 0.611 4.6 0.559
RUNX1,RUNX2,RUNX3          103  5 8.03e+14 10.1 1.0e+11 1.691 1e+11 10.3 3.387
TCFCP2,TCFCP2,TCFCP2,...   70  5 2.68e+13 6.98 1.62e+06 0.667 7 0.243
CBFB                      148  5 7.89e+22 21.1 1.64e+19 0.656 14.8 0.215
TRP3                      47  5 5.39e+03 4.25 9.00e+20 0.626 4.7 0.055
RUNX1,RUNX2,RUNX3          58  5 1.42e+06 5.85 8.32e+06 0.63 5.8 0.063
ATF3,BACH1,BACH2,BAFF,BAFF,ATF3,FO(... 96  7 4.62e+11 6.4 1.47e+09 0.689 6.8 0.182
SMAD2,SMAD3,SMAD9          42  5 0.000273 5.36 9.08e+00 0.693 4.2 0.144
BACH1,BACH2,MAF,MAF,MAF,... 53  5 7.76e+06 5.11 3.00e+32 0.689 5.3 0.053
ATF3,FO5,FOS,FO5,FOSL2,FOSL2,JUN(... 456 135 1.15e+07 9.84 6.51e+06 0.688 1.5 0
SMARCC1,SMARCC2            120 6 6.9e-16 13.2 1.07e+13 0.686 10 0.245
ATF3,ID2                  406 135 1.15e+07 9.04 6.51e+06 0.686 1.5 0
JUN,JUN,JUN,JUN.JUN       82  7 6.11e+09 2.01 3.44e+07 0.603 5.0 0.082
NFIB,NFIB,NFIB,NFIB       165 47 9.13e+04 4.04 3.00e+17 0.57 1.76 0
NFIB,NFIB,NFIB,NFIB       68  5 4.27e+07 7.37 2.06e+06 0.676 6.8 0.074
EFL,EFL,EFL,SFI,SFI,SFI,SPI,SPIC 888 303 1.85e+07 6.73 8.85e+06 0.551 1.13 0

```

The motif ID, the number of matches in the sample and in the background file, followed by five statistics: the enrichment p-value (hypergeometric/Fisher’s exact), the log-transformed p-value, the ROC area under curve (AUC), the enrichment compared to background set at 1% FPR and the recall at 10% FDR.

The ROC AUC is widely used, however, it might not always be the most informative. In situations where the background set is very large compared to the input set, it might give a more optimistic picture than warranted.

Let’s sort on the last statistic:

```
$ sort -t ',' -k6 -nr TAp73alpha.roc/gimme.roc.report.txt

```
$ sort -k8g TApa73alpha.roc/gimme.roc.report.txt | cut -f1,6,8 | tail

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bZIP_M0305_1.01</td>
<td>0.581</td>
<td>0.0820</td>
</tr>
<tr>
<td>SMAD_M5627_1.01</td>
<td>0.603</td>
<td>0.1440</td>
</tr>
<tr>
<td>Unknown_M6235_1.01</td>
<td>0.656</td>
<td>0.2350</td>
</tr>
<tr>
<td>Grainyhead_Average_6</td>
<td>0.687</td>
<td>0.2430</td>
</tr>
<tr>
<td>Myb_SANT_Average_7</td>
<td>0.586</td>
<td>0.2450</td>
</tr>
<tr>
<td>bZIP_Average_149</td>
<td>0.609</td>
<td>0.2520</td>
</tr>
<tr>
<td>Runt_Average_9</td>
<td>0.691</td>
<td>0.3670</td>
</tr>
<tr>
<td>p53_Average_10</td>
<td>0.811</td>
<td>0.5590</td>
</tr>
<tr>
<td>p53_M3568_1.01</td>
<td>0.816</td>
<td>0.6230</td>
</tr>
<tr>
<td>p53_Average_8</td>
<td>0.918</td>
<td>0.8820</td>
</tr>
</tbody>
</table>

Not surprisingly, the p53 family motif is the most enriched. In addition, we also get RUNX1 and AP1 motifs. The Grainyhead motif somewhat resembles the p53 motif, which could explain the enrichment. Let’s visualize this. This command will create two sequence logos in PNG format:

```bash
$ gimme logo -i p53_Average_8,Grainyhead_Average_6
```

The p53 motif, or p73 motif in this case, p53_Average_8.png:

![p53 motif](image)

And the Grainyhead motif, Grainyhead_Average_6:

![Grainyhead motif](image)

The resemblance is clear. This also serves as a warning to never take the results from a computational tool (including mine) at face value...
4.5 Command-line reference

In addition to `gimme motifs` the GimmeMotifs package contains several other tools that can perform the various substeps of GimmeMotifs, as well as other useful tools. Run them to see the options.

4.5.1 List of tools

- `gimme motifs`
- `gimme maelstrom`
- `gimme scan`
- `gimme roc`
- `gimme match`
- `gimme cluster`
- `gimme index`
- `gimme background`
- `gimme threshold`
- `gimme location`
- `gimme diff`
- `gimme logo`

4.5.2 Input formats

Most tools in this section take a file in PWM format as input. This is actually a file with Position Specific Scoring Matrices (PSSMs) containing frequencies. It looks like this:

```
>motif1
0.3611 0.0769 0.4003 0.1664
0.2716 0.0283 0.5667 0.1381
0.6358 0.0016 0.3344 0.0330
0.0016 0.9859 0.0016 0.0157
0.8085 0.0063 0.0502 0.1397
>motif2
0.2276 0.0157 0.0330 0.7284
0.0031 0.0016 0.9984 0.0016
0.0377 0.3799 0.0016 0.5856
0.0816 0.7096 0.0173 0.1962
0.1350 0.4035 0.0675 0.3987
```

The frequencies are separated by tabs, and in the order A,C,G,T.

4.5.3 Command: `gimme motifs`

**Quick example**

You can try GimmeMotifs with a small example dataset included in the examples directory, included with GimmeMotifs. This example does not require any additional configuration if GimmeMotifs is installed correctly.
Change to a directory where you have write permissions and run the following command (substitute the filename with the location of the file on your system):

```
gimme motifs /usr/share/gimmemotifs/examples/TAp73alpha.fa -n p73
```

The \texttt{-n} or \texttt{--name} option defines the name of the output directory that is created. All output files are stored in this directory.

Depending on your computer you may have to wait some minutes for your results. Once GimmeMotifs is finished you can open \texttt{p73/p73\_motif\_report.html} in your browser.

**Example: gimme motifs**

This example is the same as above, except it will start from a BED file. This example does require you to have \texttt{hg19} present and indexed. Change to a directory where you have write permissions and run the following command (substitute the filename with the location of the file on your system):

```
gimme motifs /usr/share/gimmemotifs/examples/TAp73alpha.bed -n example
```

The \texttt{-n} or \texttt{--name} option defines the name of the output directory that is created. All output files are stored in this directory.

Depending on your computer you may have to wait some minutes for your results. Once GimmeMotifs is finished you can open \texttt{example/example\_motif\_report.html} in your browser.

**Best practices and tips**

**GimmeMotifs is multi-threaded**

GimmeMotifs runs multi-threaded and uses all the CPU’s in the system. This means that all the programs will be run in parallel as much as possible. Of course some programs are still single-threaded, and will not benefit from this. Because GimmeMotifs uses all the available CPU’s it does not make much sense to start multiple GimmeMotifs jobs at the same time.

**Running time**

The running time of GimmeMotifs greatly depends on which tools you use for prediction and how large the dataset is. Some of the tools might take a very long time and two of them, GADEM is not added to the default tools because of this reason. You can always use them for an analysis (by specifying the \texttt{-t} command-line option), but it is recommended to only do this for a small dataset (say, less than 5000 peaks). Weeder in combination with the \texttt{xl} analysis can also take a very long time, so be prepared. In general a \texttt{small} analysis will be the quickest, and a \texttt{xl} analysis will be the slowest.

While GimmeMotifs is developed specifically for ChIP-seq datasets, most motif prediction tools are not. In practice this means that it does not make much sense to predict motifs on a large amount of sequences, as this will usually not result in higher quality motifs. Therefore GimmeMotifs uses an absolute limit for the prediction set. By default 20\% of the sequences are used as input for motif prediction, but with an absolute maximum. This is controlled by the \texttt{abs\_max} parameter in the configuration file, which is set to 1000 by default. In general, if you have a large amount of peaks, you can also consider to run GimmeMotifs on the top sequences of your input, for instance the 5000 highest peaks.

There are two options that you can use to control the running time of GimmeMotifs. First, you can set an absolute time limit with the \texttt{max\_time} option. This option (in hours) determines the maximum time used for motif prediction. If some programs take longer, the running jobs will be terminated, and the program will continue with all the motifs that
have been predicted so far. The other option is kind of an emergency button: when you think that GimmeMotifs has
been running long enough, you can press Ctrl+C once, and only once!. This will signal GimmeMotifs to terminate
the running jobs and continue with the analysis. Please note that this works almost always, but still, there is a small
cChance that program might be in a function where the Ctrl-C option screws up, and GimmeMotifs will not be able to
handle the result gracefully.

**Intermediate results**

GimmeMotifs produces a lot of intermediate results, such as all predicted motifs, fasta-files used for validation and so
on. These are deleted by default (as they can get quite large), but if you are interested in them, you can specify the \(-k\)
option.

**Running on FASTA files**

It is possible to run GimmeMotifs on a FASTA file as input instead of a BED file. This is detected automatically if your
inputfile is correctly formatted according to FASTA specifications. In this case it is not possible to generate a genomic
matched background, so only the random Markov background will be used. Please note that for best results, all the
sequences should be of the same length. This is not necessary for motif prediction, but the statistics and positional
preference plots will be wrong if sequences have different lengths. Also see the next section.

**Small input sets**

Keep in mind that GimmeMotifs is developed for larger datasets, where you have the luxury to use a large fraction of
your input for validation. So, at least several hundred sequences would be optimal. If you want to run GimmeMotifs
on a small input dataset, it might be worthwhile to increase the fraction used for prediction (with the \(-f\) argument.

**Detailed options for gimme motifs**

- **INPUTFILE**
  This is the only mandatory option. The inputfile needs to be in BED or FASTA format. BED-formatted files need
to contain at least three tab-separated columns describing chromosome name, start and end. The fourth column
is optional, if specified it will be used by MDmodule to sort the features before motif prediction. GimmeMotifs
will take the center of these features, and subsequently extend those to the width specified by the `width` argument (see below).
- **-n or -name**
  The name of your analysis. All outputfiles will be stored in a directory named as given by this parameter.
  By default this will be `gimmemotifs_dd_mm_yyyy`, where d,m and y are the current day, month and year
  respectively.
- **-a or -analysis**
  The size of motifs to look for: small (5-8), medium (5-12), large (6-15) or xl (6-20). The larger the motifs, the
  longer GimmeMotifs will run. The ‘xl’ can take a very long time!
- **-g or -genome**
  Name of the genome (index) to use. For instance, for the example in section indexing this would be `hg19`.
- **-s or -singlestrand**
  Only use the + strand for prediction (off by default).
• \(-f\) or \(-fraction\)
  This parameter controls the fraction of the sequences used for prediction. This 0.2 by default, so in this case a randomly chosen 20% of the sequences will be used for prediction. The remaining sequences will be used for validation (enrichment, ROC curves etc.). If you have a large set of sequences (ie. most ChIP-seq peak sets), this is fine. However, if your set is smaller, it might be worthwhile to increase this prediction fraction.

• \(-w\) or \(-width\)
  This is the width of the sequences used for motif prediction. Smaller sequences will result in a faster analysis, but you are of course limited by the accuracy of your data. For the tested ChIP-seq data sets 200 performs fine.

• \(-e\) or \(-enrichment\)
  All motifs should have an absolute enrichment of at least this parameter compared to background to be called significant.

• \(-p\) or \(-pvalue\)
  All motifs should have a pvalue of at most this parameter (hypergeometric enrichment compared to background) to be called significant.

• \(-b\) or \(-background\)
  Type of background to use. By default \texttt{random} (1st order Markov model, similar dinucleotide frequencies as your sequences) and \texttt{gc} (randomly chosen from the genome with a similar GC\% as your input sequences) are used.

• \(-l\) or \(-localization\_width\)
  Width used in the positional preference plots.

• \(-t\) or \(-tools\)
  A comma-separated list of all the motif prediction tools to use. By default all installed tools that are specified in the GimmeMotifs configuration file are used.

• \(-\text{max\_time}\)
  Time limit for motif prediction in hours. Use this to control the maximum number of hours that GimmeMotifs uses for motif prediction. After this time, all jobs that are still running will be terminated, and GimmeMotifs will continue with the motifs that are predicted so far.

4.5.4 Command: gimme maelstrom

This command can be used to identify differential motifs between two or more data sets. See the \texttt{maelstrom tutorial} for more details.

Positional arguments:

\begin{verbatim}
INPUTFILE          file with regions \textbf{and} clusters
GENOME             genome
DIR                output directory
\end{verbatim}

Optional arguments:

\begin{verbatim}
-h,  --help         show this help message \textbf{and} exit
-p PWMFILE,  --pwmfile PWMFILE
                    PWM file \textbf{with} motifs (default:
gimme.vertebrate.v3.1.pwm)
-m NAMES,  --methods NAMES
                    Run \textbf{with} specific methods
\end{verbatim}
The output scores of *gimme maelstrom* represents the combined result of multiple methods. The individual results from different methods are ranked from high-scoring motif to low-scoring motif and then aggregated using the rank aggregation method from ‘Kolde, 2012<https://www.ncbi.nlm.nih.gov/pubmed/22247279>’_. The score that is shown is the -log10(p-value), where the p-value (from the rank aggregation) is corrected for multiple testing. This procedure is then repeated with the ranking reversed. These are shown as negative values.

### 4.5.5 Command: *gimme scan*

Scan a set of sequences with a set of motifs, and get the resulting matches in GFF, BED or table format. If the FASTA header includes a chromosome location in `chrom:start-end` format, the BED output will return the genomic location of the motif match. The GFF file will always have the motif location relative to the input sequence.

A basic command would look like this:

```
$ gimme scan peaks.bed --genome hg38 -b > motifs.bed
```

The threshold that is used for scanning can be specified in a number of ways. The default threshold is set to a motif-specific 1% FPR by scanning random genomic sequences. You can change the FPR with the `-f` option and/or the set of sequences that is used to determine the FPR with the `-B` option.

For instance, this command would scan with thresholds based on 5% FPR with random genomic mouse sequences.

```
$ gimme scan input.fa --genome mm10 -f 0.05 -b > gimme.scan.bed
```

And this command would base a 0.1% FPR on the input file `hg38.promoters.fa`:

```
$ gimme scan input.fa -f 0.001 -B hg38.promoters.fa -b > gimme.scan.bed
```

Alternatively, you can specify the threshold as a single score. This score is relative and is based on the maximum and minimum possible score for each motif. For example, a score of 0.95 means that the score of a motif should be at least 95% of the (maximum score - minimum score). This should probably not be set much lower than 0.8, and should be generally at least 0.9-0.95 for good specificity. Generally, as the optimal threshold might be different for each motif, the use of the FPR-based threshold is preferred. One reason to use a single score as threshold is when you want a match for each motif, regardless of the score. This command would give one match for every motif for every sequence, regardless of the score.

```
$ gimme scan input.bed --genome hg38 --cutoff 0 -n 1 -b > matches.bed
```

Finally, *gimme scan* can return the scanning results in table format. The `-t` will yield a table with number of matches, while the `-T` will have the score of the best match.

### Positional arguments:

| INPUTFILE | inputfile (FASTA, BED, regions) |

### Optional arguments:

| `-g GENOME, --genome GENOME` | genome version |
| `-p PWMFILE, --pwmfile PWMFILE` | PWM file with motifs (default: gimme.vertebrate.v3.1.pwm) |
| `-f, --fpr` | FPR for motif scanning (default 0.01) |
| `-B, --bgfile` | background file for threshold |
| `-c, --cutoff` | motif score cutoff or file with cutoffs |
| `-n N, --nreport N` | report the N best matches |

(continues on next page)
4.5.6 Command: gimme roc

Given a sample (positives, peaks) and a background file (random sequences, random promoters or similar), gimme roc calculates several statistics and/or creates a ROC plot for motifs in an input PWM file. By default, all motifs will be used in the ROC plot, you can select one or more specific motifs with the \texttt{-i} option.

The basic command is as follows:

```
$ gimme roc input.fa bg.fa > statistics.txt
```

This will use the default motif database, and writes the statistics to the file \texttt{statistics.txt}.

Most likely you’ll want a graphical report. Add the \texttt{-r} argument to supply an output directory name. Once \texttt{gimme roc} finished, you’ll find a file called \texttt{gimme.roc.report.html} in this directory. Open it in your browser to get a graphical summary of the results.

Instead of a FASTA file you can also supply a BED file or regions. In this case you’ll need a genome file. A custom \texttt{.pwm} file can be supplied with the \texttt{-p} argument. For instance, the following command scans the input BED files with \texttt{custom\_motifs.pwm}:

```
$ gimme roc input.bed bg.bed -p custom\_motifs.pwm -g hg38 > statistics.txt
```

The statistics include the ROC area under curve (ROC\_AUC), the enrichment at 1% FPR and the recall at 10% FDR. To plot an ROC curve, add the \texttt{-o} argument. This command will plot the ROC curve for all the motifs that SPI1 can bind.

```
$ gimme roc input.fa bg.fa -i Ets\_Average\_110,Ets\_M1778\_1.01,Ets\_Average\_100,Ets\_Average\_93 -o roc.png > statistics.txt
```

Positional arguments:

- **FG\_FILE**: FASTA, BED or region file
- **BG\_FILE**: FASTA, BED or region file with background sequences

Optional arguments:

- \texttt{-h, \--help}: show this help message \textbf{and} exit
- \texttt{-r OUTDIR}: output dir for graphical report
- \texttt{-p PWMFILE}: PWM file \textbf{with} motifs (default: gimme.vertebrate.v3.1.pwm)
- \texttt{-g GENOME}: Genome (when input files are \textbf{not} in FASTA format)
- \texttt{-o FILE}: Name of output file \textbf{with} ROC plot (png, svg, ps, pdf)
- \texttt{-i IDS}: Comma-separated list of motif ids to plot \textbf{in} ROC (default \textbf{is all} ids)

4.5.7 Command: gimme match

Taking an input file with motifs, find the best matching file in another file of motifs (according to the WIC metric). If an output file is specified, a graphical output with aligned motifs will be created. However, this is slow for many motifs and can consume a lot of memory (see issue). It works fine for a few motifs at a time.
Positional arguments:

PWMFILE File with input pwms

Optional arguments:

-h, --help show this help message and exit
-d DBFILE File with pwms to match against (default: gimme.vertebrate.v3.1.pwm)
-o FILE Output file with graphical report (png, svg, ps, pdf)

4.5.8 Command: gimme cluster

Cluster a set of motifs with the WIC metric.

Positional arguments:

INPUTFILE Inputfile (PFM format)
OUTDIR Name of output directory

Optional arguments:

-h, --help show this help message and exit
-s Don't compare reverse complements of motifs
-t THRESHOLD Cluster threshold

4.5.9 Command: gimme index

Creates an index to use with GimmeMotifs. Use this command if your genome is not available on UCSC and you want to use it with GimmeMotifs. You should have a directory with FASTA files, one per chromosome. Note: this will change with a future version of GimmeMotifs.

Positional arguments:

FASTADIR Directory to place genome
GENOMEBUILD UCSC genome name

Optional arguments:

-h, --help show this help message and exit
-i DIR, --indexdir DIR Index dir (default <prefix>/share/gimmemotifs/genome_index)

4.5.10 Command: gimme background

Generate random sequences according to one of several methods:

- random - randomly generated sequence with the same dinucleotide distribution as the input sequences according to a 1st order Markov model
- genomic - sequences randomly chosen from the genome
- gc - sequences randomly chosen from the genome with the same GC% as the input sequences

4.5. Command-line reference 29
• **promoter** - random promoter sequences

The background types gc and random need a set of input sequences in BED or FASTA format. If the input sequences are in BED format, the genome version needs to be specified with `-g`.

### Positional arguments:

<table>
<thead>
<tr>
<th>FILE</th>
<th>outputfile</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>type of background sequences to generate</td>
</tr>
<tr>
<td></td>
<td>(random, genomic, gc, promoter)</td>
</tr>
</tbody>
</table>

### Optional arguments:

- `-h`, `--help` show this help message and exit
- `-i` `FILE` input sequences (BED or FASTA)
- `-f` `TYPE` output format (BED or FASTA)
- `-l` `INT` length of random sequences
- `-n` `NUMBER` number of sequence to generate
- `-g` `GENOME` genome version (not for `type 'random'`)
- `-m` `N` order of the Markov model (only for `type 'random'`, default 1)

#### 4.5.11 Command: gimme threshold

Create a file with motif-specific thresholds based on a specific background file and a specific FPR. The FPR should be specified as a float between 0.0 and 1.0. You can use this threshold file with the `-c` argument of `gimme scan`. Note that `gimme scan` by default determines an FPR based on random genomic background sequences. You can use this command to create the threshold file explicitly, or when you want to determine the threshold based on a different type of background. For instance, this command would create a file with thresholds for the motifs in `custom.pwm` with a FPR of 1%, based on the sequences in `promoters.fa`.

```
$ gimme threshold custom.pwm 0.05 promoters.fa > custom.threshold.txt
```

Positional arguments:

<table>
<thead>
<tr>
<th>PWMFILE</th>
<th>File with pwms</th>
</tr>
</thead>
<tbody>
<tr>
<td>FASTFILE</td>
<td>FASTA file with background sequences</td>
</tr>
<tr>
<td>FPR</td>
<td>Desired fpr</td>
</tr>
</tbody>
</table>

#### 4.5.12 Command: gimme location

Create the positional preference plots for all the motifs in the input PWM file. This will give best results if all the sequences in the FASTA-formatted inputfile have the same length. Keep in mind that this only makes sense if the sequences are centered around a similar feature (transcription start site, highest point in a peak, etc.). The default threshold for motif scanning is 0.95, see `gimme scan` for more details.

Positional arguments:

<table>
<thead>
<tr>
<th>PWMFILE</th>
<th>File with pwms</th>
</tr>
</thead>
<tbody>
<tr>
<td>FASTFILE</td>
<td>Fasta formatted file</td>
</tr>
</tbody>
</table>

Optional arguments:

- `-h`, `--help` show this help message and exit
- `-w` `WIDTH` Set width to W (default: determined from fastfile)
4.5.13 Command: gimme diff

This is a simple command to visualize differential motifs between different data sets. You are probably better of using gimme maelstrom, however, in some cases this visualization might still be informative. The input consists of a number of FASTA files, separated by a comma. These are compared to a background file. The last two arguments are a file with pwms and and output image. The gimme diff command then produces two heatmaps (enrichment and frequency) of all enriched, differential motifs. Reported motifs are at least 3 times enriched compared to the background (change with the -e argument) and have a minimum frequency in at least one of the input data sets of 1% (change with the -f argument). You can specify motif threshold with the -c argument (which can be a file generated with gimme threshold).

For a command like this...

```bash
$ gimme diff VEGT_specific.summit.200.fa,XBRA_specific.summit.200.fa,XEOMES_specific.summit.200.fa random.w200.fa gimme_diff_tbox.png -p tbox.pwm -f 0.01 -c threshold.0.01.txt
```

...the output will look like this (based on ChIP-seq peaks of T-box factors from Gentsch et al. 2013):

![Heatmaps showing enrichment and frequency](image)

The image layout is not always optimal. If you want to customize the image, you can either save it as a .svg file, or use the numbers that are printed to stdout. The columns are in the same order as the image, the row order may be different as these are clustered before plotting.

Note that the results might differ quite a lot depending on the threshold that is chosen! Compare for instance an FPR of 1% vs an FPR of 5%.

4.5. Command-line reference
Positional arguments:

<table>
<thead>
<tr>
<th>FAFILES</th>
<th>FASTA-formatted inputfiles OR a BED file with an identifier in the 4th column, for instance a cluster number.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGFAFILE</td>
<td>FASTA-formatted background file</td>
</tr>
<tr>
<td>PNGFILE</td>
<td>outputfile (image)</td>
</tr>
</tbody>
</table>

Optional arguments:

-h, --help  show this help message and exit
-p PWMFILE, --pwmfile PWMFILE
  PWM file with motifs (default: gimme.vertebrate.v3.1.pwm)
-c, --cutoff  motif score cutoff or file with cutoffs (default 0.9)
-e MINENR, --enrichment MINENR
  minimum enrichment in at least one of the datasets compared to background
-f MINFREQ, --frequency MINFREQ
  minimum frequency in at least one of the datasets
-g VERSION, --genome VERSION
  Genome version. Only necessary in combination with a BED file with clusters as inputfile.

4.5.14 Command: gimme logo

Convert one or more motifs in a PWM file to a sequence logo. You can optionally supply a PWM file, otherwise gimme logo uses the default. With the -i option, you can choose one or more motifs to convert.

This will convert all the motifs in CTCF.pwm to a sequence logo:

$ gimme logo -p CTCF.pwm

This will create logos for Ets_Average_100 and Ets_Average_109 from the default database.

$ gimme logo -i Ets_Average_100,Ets_Average_109

4.6 API documentation

4.7 Examples

4.7.1 Working with motifs

The Motif class stores motif information. There are several ways to create a Motif instance.

```python
from gimmemotifs.motif import Motif, read_motifs

# Read from file
with open("example.pwm") as f:
    motifs = read_motifs(f)

for motif in motifs:
    print(motif)
```
# Create from scratch
m = Motif([[[0,1,0,0],[0,0,1,0]]])
m.id = "CpG"
print(m)

CpG_CG

# Or from a consensus sequence
from gimmemotifs.motif import motif_from_consensus
ap1 = motif_from_consensus("TGASTCA")
print(ap1.to_pwm())

> TGASTCA
0.0001 0.0001 0.0001 0.9998
0.0001 0.0001 0.9998 0.0001
0.9998 0.0001 0.0001 0.0001
0.0001 0.4999 0.4999 0.0001
0.0001 0.9998 0.0001 0.0001
0.9998 0.0001 0.0001 0.0001

Read motifs from files in other formats.

with open("motifs.txt") as f:
    motifs = read_motifs(f, fmt="jaspar")

You can convert a motif to several formats.

with open("example.pwm") as f:
    motifs = read_motifs(f)

# pwm
print(motifs[0].to_pwm())

> AP1
0.4908 0.1862 0.2475 0.0755
0.0125 0.0102 0.0179 0.9594
0.0191 0.0151 0.9236 0.0422
0.9457 0.0349 0.0037 0.0158
0.0355 0.2714 0.6704 0.0228
0.0121 0.0023 0.0052 0.9804
0.0271 0.9665 0.0042 0.0022
0.9935 0.0018 0.0021 0.0027
0.0367 0.2994 0.1227 0.5412

# pfm
print(motifs[0].to_pfm())

> AP1
490.836673106 186.173418152 247.513020751 75.4768879912
12.547339755300001 10.155349184 17.9452120263 959.3520990339999

(continues on next page)
# consensus sequence
```python
print(motifs[0].to_consensus())
```

nTGAGTCAy

Some other useful tidbits.

```python
m = motif_from_consensus("NTGASTCAN")
print(len(m))
```

9

```python
# Trim by information content
m.trim(0.5)
print(m.to_consensus(), len(m))
```

TGAsTCA 7

```python
# Slices
print(m[:3].to_consensus())
```

TGA

```python
# Shuffle
random_motif = motif_from_consensus("NTGASTGAN").randomize()
print(random_motif)
```

random_snCTAGTAn

To convert a motif to an image, use `to_img()` function. Supported formats are png, ps and pdf.

```python
m = motif_from_consensus("NTGASTCAN")
m.to_img("apl.png", fmt="png")
```
For very simple scanning, you can just use a Motif instance. Let’s say we have a FASTA file called `test.fa` that looks like this:

```
>seq1
AAAAAAAAAAAAAAAAAAAAAAA
>seq2
CGCCGCTGAGTCACGCGCGCGCG
>seq3
TGASTCAAAAAAAAAATGASTCA
```

Now we can use this file for scanning.

```python
from gimmemotifs.motif import motif_from_consensus
from gimmemotifs.fasta import Fasta

f = Fasta("test.fa")
m = motif_from_consensus("TGAsTCA")
m.pwm_scan(f)
```

This returns a dictionary with the sequence names as keys. The value is a list with positions where the motif matches. Here, as the AP1 motif is a palindrome, you see matches on both forward and reverse strand. This is more clear when we use `pwm_scan_all()` that returns position, score and strand for every match.

```python
m.pwm_scan_all(f)
```

```
{'seq1': [], 'seq2': [(6, 9.02922042678255, 1), (6, 9.02922042678255, -1)], 'seq3': [(0, 8.331251500673487, 1), (16, 8.331251500673487, 1), (0, 8.331251500673487, -1), (16, 8.331251500673487, -1)]}
```

The number of matches to return is set to 50 by default, you can control this by setting the `nreport` argument. Use `scan_rc=False` to only scan the forward orientation.
m.pwm_scan_all(f, nreport=1, scan_rc=False)

{'seq1': [],
'seq2': [(6, 9.02922042678255, 1)],
'seq3': [(0, 8.331251500673487, 1)]}

While this functionality works, it is not very efficient. To scan many motifs in potentially many sequences, use the functionality in the scanner module. If you only want the best match per sequence, is a utility function called scan_to_best_match, otherwise, use the Scanner class.

from gimmemotifs.motif import motif_from_consensus
from gimmemotifs.scanner import scan_to_best_match

m1 = motif_from_consensus("TGAsTCA")
m1.id = "AP1"
m2 = motif_from_consensus("CGCG")
m2.id = "CG"
motifs = [m1, m2]

print("motif	pos	score")
result = scan_to_best_match("test.fa", motifs)
for motif, matches in result.items():
    for match in matches:
        print("{0}	{1}	{2}".format(motif, match[1], match[0]))

motif pos score
CG 0 -18.26379789133924
CG 0 5.554366880674296
CG 0 -7.743307225501047
AP1 0 -20.052563923836903
AP1 6 9.029486018303187
AP1 0 8.331550321011443

The matches are in the same order as the sequences in the original file.

While this function can be very useful, a Scanner instance is much more flexible. You can scan different input formats (BED, FASTA, regions), and control the thresholds and output.

As an example we will use the file Gm12878.CTCF.top500.w200.fa that contains 500 top CTCF peaks. We will get the CTCF motif and scan this file in a number of different ways.

from gimmemotifs.motif import default_motifs
from gimmemotifs.scanner import Scanner
from gimmemotifs.fasta import Fasta
import numpy as np

# Input file
fname = "examples/Gm12878.CTCF.top500.w200.fa"

# Select the CTCF motif from the default motif database
motifs = [m for m in default_motifs() if "CTCF" in m.factors]

# Initialize the scanner
s = Scanner()
s.set_motifs(motifs)

Now let's get the best score for the CTCF motif for each sequence.
GimmeMotifs Documentation, Release 0.13.1+0.g80a2acc.dirty

scores = [r[0] for r in s.best_score("examples/Gm12878.CTCF.top500.w200.fa")]
print("{}\t{:.2f}\t{:.2f}\t{:.2f}".format(
    len(scores),
    np.mean(scores),
    np.min(scores),
    np.max(scores)
))

500 10.61 1.21 14.16

In many cases you’ll want to set a threshold. In this example we’ll use a 1% FPR threshold, based on scanning randomly selected sequences from the ghg38 genome. The first time you run this, it will take a while. However, the thresholds will be cached. This means that for the same combination of motifs and genome, the previously generated threshold will be used.

# Set a 1% FPR threshold based on random hg38 sequence
s.set_threshold(fpr=0.01, genome="hg38")

# get the number of sequences with at least one match
counts = [n[0] for n in s.count("examples/Gm12878.CTCF.top500.w200.fa", nreport=1)]
print(counts[:10])

[1, 1, 1, 1, 1, 1, 1, 1]

# or the grand total of number of sequences with 1 match
print(s.total_count("examples/Gm12878.CTCF.top500.w200.fa", nreport=1))

[408]

# Scanner.scan() just gives all information
seqs = Fasta("examples/Gm12878.CTCF.top500.w200.fa")[:10]
for i,result in enumerate(s.scan(seqs)):
    seqname = seqs.ids[i]
    for m,matches in enumerate(result):
        motif = motifs[m]
        for score, pos, strand in matches:
            print(seqname, motif, score, pos, strand)

    chr11:190037-190237 C2H2_ZF_Average_200_CCAsyAGrkGGCr 13.4959558370929 143 -1
    chr11:190037-190237 C2H2_ZF_Average_200_CCAsyAGrkGGCr 10.821440417077262 22 -1
    chr11:190037-190237 C2H2_ZF_Average_200_CCAsyAGrkGGCr 10.658439190070851 82 -1
    chr14:106873577-106873777 C2H2_ZF_Average_200_CCAsyAGrkGGCr 14.16061638444734 120 -1
    chr14:106873577-106873777 C2H2_ZF_Average_200_CCAsyAGrkGGCr 13.72460285196088 83 -1
    chr14:106873577-106873777 C2H2_ZF_Average_200_CCAsyAGrkGGCr 11.450778540447134 27 -1
    chr14:106873577-106873777 C2H2_ZF_Average_200_CCAsyAGrkGGCr 10.037330832055455 7 -1
    chr14:106873577-106873777 C2H2_ZF_Average_200_CCAsyAGrkGGCr 8.998038360035828 159 -1
    chr14:106873577-106873777 C2H2_ZF_Average_200_CCAsyAGrkGGCr 8.668660161058972 101 -1
    chr14:106765204-106765404 C2H2_ZF_Average_200_CCAsyAGrkGGCr 14.16061638444734 145 -1
    chr14:106765204-106765404 C2H2_ZF_Average_200_CCAsyAGrkGGCr 13.848270770440264 185 -1
    chr14:106765204-106765404 C2H2_ZF_Average_200_CCAsyAGrkGGCr 13.668171128367552 165 -1
    chr14:106765204-106765404 C2H2_ZF_Average_200_CCAsyAGrkGGCr 12.785329839873164 27 -1
    chr14:106765204-106765404 C2H2_ZF_Average_200_CCAsyAGrkGGCr 11.886792072933595 126 -1
    chr14:106765204-106765404 C2H2_ZF_Average_200_CCAsyAGrkGGCr 11.25063146496227 67 -1
    chr15:22461178-22461378 C2H2_ZF_Average_200_CCAsyAGrkGGCr 14.16061638444734 28 -1
    chr15:22461178-22461378 C2H2_ZF_Average_200_CCAsyAGrkGGCr 14.16061638444734 185 -1

(continues on next page)

4.7. Examples 37
4.7.3 Finding de novo motifs

Let’s take the Gm12878.CTCF.top500.w200.fa file as example again. For a basic example we’ll just use two motif finders, as they’re quick to run.

```python
from gimmemotifs.denovo import gimme_motifs

peaks = "Gm12878.CTCF.top500.w200.fa"
outdir = "CTCF.gimme"
params = {
    "tools": ["Homer", "BioProspector"],
}

motifs = gimme_motifs(peaks, outdir, params=params)
```

```bash
2017-06-30 07:37:00,079 - INFO - starting full motif analysis
2017-06-30 07:37:00,082 - INFO - preparing input (FASTA)
2017-06-30 07:37:32,949 - INFO - starting motif prediction (medium)
2017-06-30 07:37:40,540 - INFO - BioProspector_width_5 finished, found 5 motifs
2017-06-30 07:37:41,308 - INFO - BioProspector_width_7 finished, found 5 motifs
2017-06-30 07:37:41,609 - INFO - BioProspector_width_6 finished, found 5 motifs
2017-06-30 07:37:42,003 - INFO - BioProspector_width_8 finished, found 5 motifs
2017-06-30 07:37:44,054 - INFO - Homer_width_5 finished, found 5 motifs
2017-06-30 07:37:45,201 - INFO - Homer_width_6 finished, found 5 motifs
2017-06-30 07:37:47,246 - INFO - Homer_width_7 finished, found 5 motifs
2017-06-30 07:37:50,503 - INFO - Homer_width_8 finished, found 5 motifs
2017-06-30 07:37:54,649 - INFO - BioProspector_width_9 finished, found 5 motifs
2017-06-30 07:37:56,169 - INFO - BioProspector_width_10 finished, found 5 motifs
2017-06-30 07:37:56,656 - INFO - Homer_width_9 finished, found 5 motifs
2017-06-30 07:37:59,313 - INFO - Homer_width_10 finished, found 5 motifs
2017-06-30 07:37:59,314 - INFO - all jobs submitted
2017-06-30 07:39:21,298 - INFO - predicted 60 motifs
```
This will basically run the same pipeline as the `gimme motifs` command. All output files will be stored in `outdir` and `gimme_motifs` returns a list of Motif instances. If you only need the motifs but not the graphical report, you can decide to skip it by setting `create_report` to `False`. Additionally, you can choose to skip clustering (cluster=False) or to skip calculation of significance (filter_significant=False). For instance, the following command will only predict motifs and cluster them.

```python
gimme_motifs(peaks, outdir,
             params=params, filter_significant=False, create_report=False)
```

4.7.4 Motif statistics

With some motifs, a sample file and a background file you can calculate motif statistics. Let’s say I wanted to know which of the p53-family motifs is most enriched in the file `TAp73alpha.fa`.

First, we’ll generate a GC%-matched genomic background. Then we only select p53 motifs.

```python
from gimmemotifs.background import MatchedGcFasta
from gimmemotifs.fasta import Fasta
from gimmemotifs.stats import calc_stats
from gimmemotifs.motif import default_motifs

sample = "TAp73alpha.fa"
basedir = MatchedGcFasta(sample, genome="hg19", number=1000)
motifs = [m for m in default_motifs() if any(f in m.factors for f in ["TP53", "TRP53", ...")]
stats = calc_stats(motifs, sample, background)

for k, v in stats["str(motifs[0])"].items():
    print(k, v)
```

A lot of statistics are generated and you will not always need all of them. You can choose one or more specific metrics with the additional `stats` argument.

4.7. Examples
metrics = ["roc_auc", "recall_at_fdr"]
stats = calc_stats(motifs, sample, bg, stats=metrics)

for metric in metrics:
    for motif in motifs:
        print("{} \t {} \t {:.2f}".format(motif.id, metric, stats[str(motif)][metric]))

p53_M5923_1.01 roc_auc 0.63
p53_M5922_1.01 roc_auc 0.64
p53_Average_10 roc_auc 0.83
p53_Average_8 roc_auc 0.93
p53_M3568_1.01 roc_auc 0.83
p53_M5923_1.01 recall_at_fdr 0.00
p53_M5922_1.01 recall_at_fdr 0.00
p53_Average_10 recall_at_fdr 0.24
p53_Average_8 recall_at_fdr 0.83
p53_M3568_1.01 recall_at_fdr 0.42

4.7.5 Maelstrom

4.8 Auto-generated

This part of the API documentation is not yet complete.

4.8.1 The Motif class

Module contain core motif functionality

class gimmemotifs.motif.Motif(pfm=None)
    Representation of a transcription factor binding motif.

Examples

>>> motif = Motif([[0,1,0,0], [0.5,0,0,0.5], [0,0,1,0]])
>>> print(motif.to_pwm())
> 0 1 0 0
0.5 0 0 0.5
0 0 1 0
>>> print(motif.to_consensus())
CwG

Methods

average_motifs(other, pos, orientation[...])
    Return the average of two motifs.
consensus_scan(fa)
    Scan FASTA with the motif as a consensus sequence.
hash()
    Return hash of motif.

Continued on next page
### Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ic_pos(row1[, row2])</code></td>
<td>Calculate the information content of one position.</td>
</tr>
<tr>
<td><code>information_content()</code></td>
<td>Return the total information content of the motif.</td>
</tr>
<tr>
<td><code>pcc_pos(row1, row2)</code></td>
<td>Calculate the Pearson correlation coefficient of one position compared to another position.</td>
</tr>
<tr>
<td><code>pfm_to_pwm(pfm[, pseudo])</code></td>
<td>Convert PFM with counts to a PFM with fractions.</td>
</tr>
<tr>
<td><code>pwm_max_score()</code></td>
<td>Return the maximum PWM score.</td>
</tr>
<tr>
<td><code>pwm_min_score()</code></td>
<td>Return the minimum PWM score.</td>
</tr>
<tr>
<td><code>pwm_scan(fa[, cutoff, nreport, scan_rc])</code></td>
<td>Scan sequences with this motif.</td>
</tr>
<tr>
<td><code>pwm_scan_all(fa[, cutoff, nreport, scan_rc])</code></td>
<td>Scan sequences with this motif.</td>
</tr>
<tr>
<td><code>pwm_scan_score(fa[, cutoff, nreport, scan_rc])</code></td>
<td>Scan sequences with this motif.</td>
</tr>
<tr>
<td><code>pwm_scan_to_gff(fa, gfffile[, cutoff, ...])</code></td>
<td>Scan sequences with this motif and save to a GFF file.</td>
</tr>
<tr>
<td><code>randomize()</code></td>
<td>Create a new motif with shuffled positions.</td>
</tr>
<tr>
<td><code>rc()</code></td>
<td>Return the reverse complemented motif.</td>
</tr>
<tr>
<td><code>score_kmer(kmer)</code></td>
<td>Calculate the log-odds score for a specific k-mer.</td>
</tr>
<tr>
<td><code>to_img(fname[, fmt, add_left, seqlogo, height])</code></td>
<td>Create a sequence logo using seqlogo.</td>
</tr>
<tr>
<td><code>to_meme()</code></td>
<td>Return motif formatted in MEME format</td>
</tr>
<tr>
<td><code>to_motevo()</code></td>
<td>Return motif formatted in MotEvo (TRANSFAC-like) format</td>
</tr>
<tr>
<td><code>to_pwm([precision, extra_str])</code></td>
<td>Return pwm as string.</td>
</tr>
<tr>
<td><code>to_transfac()</code></td>
<td>Return motif formatted in TRANSFAC format</td>
</tr>
<tr>
<td><code>trim([edge_ic_cutoff])</code></td>
<td>Trim positions with an information content lower than the threshold.</td>
</tr>
</tbody>
</table>

```
| ic | matrix_ic | max_ic | max_pcc | other_ic | other_ic_pos | pcc | randomize_dimer | to_consensus | to_consensusv2 | to_pwm | to_transfac | trim |
```

### average_motifs (other, pos, orientation, include_bg=False)

Return the average of two motifs.

Combine this motif with another motif and return the average as a new Motif object. The position and orientation need to be supplied. The pos parameter is the position of the second motif relative to this motif.

For example, take the following two motifs: Motif 1: CATGYT Motif 2: GGCTTGY

With position -2, the motifs are averaged as follows: xxCATGYT GGCTTGYx

**Parameters**

- `other` [Motif object] Other Motif object.
- `pos` [int] Position of the second motif relative to this motif.

---

4.8. Auto-generated
orientation [int] Orientation, should be 1 or -1. If the orientation is -1 then the reverse complement of the other motif is used for averaging.

include_bg [bool, optional] Extend both motifs with background frequencies (0.25) before averaging. False by default.

Returns

motif [motif object] New Motif object containing average motif.

consensus_scan (fa)
Scan FASTA with the motif as a consensus sequence.

Parameters

fa [Fasta object] Fasta object to scan

Returns


hash ()
Return hash of motif.

This is an unique identifier of a motif, regardless of the id.

Returns: hash : str

ic_pos (row1, row2=None)
Calculate the information content of one position.

Returns

score [float] Information content.

information_content ()
Return the total information content of the motif.

pcc_pos (row1, row2)
Calculate the Pearson correlation coefficient of one position compared to another position.

Returns

score [float] Pearson correlation coefficient.

pfm_to_pwm (pfm, pseudo=0.001)
Convert PFM with counts to a PFM with fractions.

Parameters

pfm [list] 2-dimensional list with counts.

pseudo [float] Pseudocount used in conversion.

Returns

pwm [list] 2-dimensional list with fractions.

pwm_max_score ()
Return the maximum PWM score.

Returns

score [float] Maximum PWM score.

pwm_min_score ()
Return the minimum PWM score.
Returns

score [float] Minimum PWM score.

**pwm_scan** *(fa, cutoff=0.9, nreport=50, scan_rc=True)*
Scan sequences with this motif.

Scan sequences from a FASTA object with this motif. Less efficient than using a Scanner object. By setting the cutoff to 0.0 and nreport to 1, the best match for every sequence will be returned. Only the position of the matches is returned.

Parameters

fa [Fasta object] Fasta object to scan.

cutoff [float, optional] Cutoff to use for motif scanning. This cutoff is not specifically optimized and the strictness will vary a lot with motif length.

nreport [int, optional] Maximum number of matches to report.

scan_rc [bool, optional] Scan the reverse complement. True by default.

Returns

matches [dict] Dictionary with motif matches. Only the position of the matches is returned.

**pwm_scan_all** *(fa, cutoff=0.9, nreport=50, scan_rc=True)*
Scan sequences with this motif.

Scan sequences from a FASTA object with this motif. Less efficient than using a Scanner object. By setting the cutoff to 0.0 and nreport to 1, the best match for every sequence will be returned. The score, position and strand for every match is returned.

Parameters

fa [Fasta object] Fasta object to scan.

cutoff [float, optional] Cutoff to use for motif scanning. This cutoff is not specifically optimized and the strictness will vary a lot with motif length.

nreport [int, optional] Maximum number of matches to report.

scan_rc [bool, optional] Scan the reverse complement. True by default.

Returns

matches [dict] Dictionary with motif matches. The score, position and strand for every match is returned.

**pwm_scan_score** *(fa, cutoff=0, nreport=1, scan_rc=True)*
Scan sequences with this motif.

Scan sequences from a FASTA object with this motif. Less efficient than using a Scanner object. By setting the cutoff to 0.0 and nreport to 1, the best match for every sequence will be returned. Only the score of the matches is returned.

Parameters

fa [Fasta object] Fasta object to scan.

cutoff [float, optional] Cutoff to use for motif scanning. This cutoff is not specifically optimized and the strictness will vary a lot with motif length.

nreport [int, optional] Maximum number of matches to report.

scan_rc [bool, optional] Scan the reverse complement. True by default.
Returns

matches [dict] Dictionary with motif matches. Only the score of the matches is returned.

\texttt{pwm\_scan\_to\_gff (fa, gfffile, cutoff=0.9, nreport=50, scan\_rc=True, append=False)}

Scan sequences with this motif and save to a GFF file.

Scan sequences from a FASTA object with this motif. Less efficient than using a Scanner object. By setting the cutoff to 0.0 and nreport to 1, the best match for every sequence will be returned. The output is save to a file in GFF format.

Parameters

\begin{itemize}
  \item \texttt{fa} [Fasta object] Fasta object to scan.
  \item \texttt{gfffile} [str] Filename of GFF output file.
  \item \texttt{cutoff} [float, optional] Cutoff to use for motif scanning. This cutoff is not specifically optimized and the strictness will vary a lot with motif length.
  \item \texttt{nreport} [int, optional] Maximum number of matches to report.
  \item \texttt{scan\_rc} [bool, optional] Scan the reverse complement. True by default.
  \item \texttt{append} [bool, optional] Append to GFF file instead of overwriting it. False by default.
\end{itemize}

\texttt{randomize ()}

Create a new motif with shuffled positions.

Shuffle the positions of this motif and return a new Motif instance.

Returns

\texttt{m} [Motif instance] Motif instance with shuffled positions.

\texttt{rc ()}

Return the reverse complemented motif.

Returns

\texttt{m} [Motif instance] New Motif instance with the reverse complement of the input motif.

\texttt{score\_kmer (kmer)}

Calculate the log-odds score for a specific k-mer.

Parameters

\begin{itemize}
  \item \texttt{kmer} [str] String representing a kmer. Should be the same length as the motif.
\end{itemize}

Returns

\texttt{score} [float] Log-odd score.

\texttt{to\_img (fname, fmt='PNG', add\_left=0, seqlogo=None, height=6)}

Create a sequence logo using seqlogo.

Create a sequence logo and save it to a file. Valid formats are: PNG, EPS, GIF and PDF.

Parameters

\begin{itemize}
  \item \texttt{fname} [str] Output filename.
  \item \texttt{fmt} [str, optional] Output format (case-insensitive). Valid formats are PNG, EPS, GIF and PDF.
  \item \texttt{add\_left} [int, optional] Pad motif with empty positions on the left side.
\end{itemize}
seqlogo [str] Location of the seqlogo executable. By default the seqlogo version that is included with GimmeMotifs is used.

height [float] Height of the image

to_meme()
    Return motif formatted in MEME format

    Returns
    m [str] String of motif in MEME format.

to_motefo()
    Return motif formatted in MotEvo (TRANSFAC-like) format

    Returns
    m [str] String of motif in MotEvo format.

to_pwm(\text{precision}=4, extra_str=''\text{)}
    Return pwm as string.

    Parameters
    \text{precision} [\text{int, optional, default 4}] Floating-point precision.
    \text{extra_str} [: \text{str, optional} Extra text to include with motif id line.

    Returns
    \text{motif\_str} [\text{str} Motif formatted in PWM format.

to_transfac()
    Return motif formatted in TRANSFAC format

    Returns
    m [str] String of motif in TRANSFAC format.

trim(\text{edge\_ic\_cutoff}=0.4)
    Trim positions with an information content lower than the threshold.

    The default threshold is set to 0.4. The Motif will be changed in-place.

    Parameters
    \text{edge\_ic\_cutoff} [\text{float, optional}] Information content threshold. All motif positions at the flanks with an information content lower than this will be removed.

    Returns
    m [Motif instance]

gimmemotifs.motif.default_motifs()
    Return list of Motif instances from default motif database.

gimmemotifs.motif.motif_from_align(align)
    Convert alignment to motif.

    Converts a list with sequences to a motif. Sequences should be the same length.

    Parameters
    \text{align} [\text{list}] List with sequences (A,C,G,T).

    Returns
    m [Motif instance] Motif created from the aligned sequences.
gimmemotifs.motif.motif_from_consensus(cons, n=12)
Convert consensus sequence to motif.

Converts a consensus sequences using the nucleotide IUPAC alphabet to a motif.

Parameters

cons [str] Consensus sequence using the IUPAC alphabet.

n [int, optional] Count used to convert the sequence to a PFM.

Returns

m [Motif instance] Motif created from the consensus.

gimmemotifs.motif.parse_motifs(motifs)
Parse motifs in a variety of formats to return a list of motifs.

Parameters

motifs [list or str] Filename of motif, list of motifs or single Motif instance.

Returns

motifs [list] List of Motif instances.

gimmemotifs.motif.read_motifs(infile=None, fmt='pwm', as_dict=False)
Read motifs from a file or stream or file-like object.

Parameters

infile [string or file-like object, optional] Motif database, filename of motif file or file-like object. If infile is not specified the default motifs as specified in the config file will be returned.


as_dict [boolean, optional] Return motifs as a dictionary with motif_id, motif pairs.

Returns

motifs [list] List of Motif instances. If as_dict is set to True, motifs is a dictionary.

4.8.2 Prediction of de novo motifs

De novo motif prediction.

This module contains functions to predict de novo motifs using one or more de novo motif tools. The main function is gimme_motifs, which is likely the only method that you’ll need from this module.

Examples

from gimmemotifs.denovo import gimme_motifs
peaks = “Gm12878.CTCF.top500.w200.fa” outdir = “CTCF.gimme” params = {
    “tools”: “Homer,BioProspector”, “genome”: “hg38”, }
motifs = gimme_motifs(peaks, outdir, params=params)
gimmemotifs.denovo.best_motif_in_cluster(single_pwm, clus_pwm, clusters, fg_fa, background, stats=None, metrics=('roc_auc', ‘recall_at_fdr’))

Return the best motif per cluster for a clustering results.

The motif can be either the average motif or one of the clustered motifs.
**Parameters**

- **single_pwm** [str] Filename of motifs.
- **clus_pwm** [str] Filename of motifs.
- **clusters** : Motif clustering result.
- **fg_fa** [str] Filename of FASTA file.
- **background** [dict] Dictionary for background file names.
- **stats** [dict, optional] If statistics are not supplied they will be computed.
- **metrics** [sequence, optional] Metrics to use for motif evaluation. Default are “roc_auc” and “recall_at_fdr”.

**Returns**

- **motifs** [list] List of Motif instances.

```python
gimmemotifs.denovo.create_background(bg_type, fafile, outfile, genome='hg18', width=200, nr_times=10, custom_background=None)
```

Create background of a specific type.

**Parameters**

- **bg_type** [str] Name of background type.
- **fafile** [str] Name of input FASTA file.
- **outfile** [str] Name of output FASTA file.
- **genome** [str, optional] Genome name.
- **width** [int, optional] Size of regions.
- **nr_times** [int, optional] Generate this times as many background sequences as compared to input file.

**Returns**

- **nr_seqs** [int] Number of sequences created.

```python
gimmemotifs.denovo.create_backgrounds(outdir, background=None, genome='hg38', width=200, custom_background=None)
```

Create different backgrounds for motif prediction and validation.

**Parameters**

- **outdir** [str] Directory to save results.
- **background** [list, optional] Background types to create, default is ‘random’.
- **genome** [str] Genome name (for genomic and gc backgrounds).
- **width** [int, optional] Size of background regions.

**Returns**

- **bg_info** [dict] Keys: background name, values: file name.

```python
gimmemotifs.denovo.filter_significant_motifs(fname, result, bg, metrics=None)
```

Filter significant motifs based on several statistics.

**Parameters**

- **fname** [str] Filename of output file were significant motifs will be saved.
- **result** [PredictionResult instance] Contains motifs and associated statistics.
bg  [str] Name of background type to use.

metrics  [sequence] Metric with associated minimum values. The default is 
((“max_enrichment”, 3), (“roc_auc”, 0.55), (“enr_at_f[tr”, 0.55))

Returns

motifs  [list] List of Motif instances.

gimmemotifs.denovo.gimme_motifs(inputfile, outdir, params=None, filter_significant=True, cluster=True, create_report=True)

De novo motif prediction based on an ensemble of different tools.

Parameters

inputfile  [str] Filename of input. Can be either BED, narrowPeak or FASTA.

outdir  [str] Name of output directory.

params  [dict, optional] Optional parameters.

filter_significant  [bool, optional] Filter motifs for significance using the validation set.

cluster  [bool, optional] Cluster similar predicted (and significant) motifs.

create_report  [bool, optional] Create output reports (both .txt and .html).

Returns

motifs  [list] List of predicted motifs.

Examples

```python
>>> from gimmemotifs.denovo import gimme_motifs
>>> gimme_motifs("input.fa", "motifs.out")
```

gimmemotifs.denovo.prepare_denovo_input_bed(inputfile, params, outdir)

Prepare a BED file for de novo motif prediction.

All regions to same size; split in test and validation set; converted to FASTA.

Parameters

inputfile  [str] BED file with input regions.


outdir  [str] Output directory to save files.

gimmemotifs.denovo.prepare_denovo_input_fa(inputfile, params, outdir)

Create all the FASTA files for de novo motif prediction and validation.

```
```

gimmemotifs.denovo.prepare_denovo_input_narrowpeak(inputfile, params, outdir)

Prepare a narrowPeak file for de novo motif prediction.

All regions to same size; split in test and validation set; converted to FASTA.

Parameters

inputfile  [str] BED file with input regions.


outdir  [str] Output directory to save files.
gimmemotifs.denovo.rename_motifs (motifs, stats=None)
    Rename motifs to GimmeMotifs_1..GimmeMotifs_N.
    If stats object is passed, stats will be copied.

4.8.3 Motif scanning

class gimmemotifs.scanner.Scanner (ncpus=None)
    scan sequences with motifs

    Methods

    best_match(seqs[, scan_rc])
        give the best match of each motif in each sequence
        returns an iterator of nested lists containing tuples:
        (score, position, strand)

    best_score(seqs[, scan_rc, normalize])
        give the score of the best match of each motif in each
        sequence returns an iterator of lists containing floats

    count(seqs[, nreport, scan_rc])
        count the number of matches above the cutoff returns
        an iterator of lists containing integer counts

    scan(seqs[, nreport, scan_rc, normalize])
        scan a set of regions / sequences

    set_background([fname, genome, length, nseq])
        Set the background to use for FPR and z-score calculations.

    set_genome(genome)
        set the genome to be used for:

    set_threshold([fpr, threshold])
        Set motif scanning threshold based on background
        sequences.

    total_count(seqs[, nreport, scan_rc])
        count the number of matches above the cutoff returns
        an iterator of lists containing integer counts

    set_meanstd

    set_motifs

    best_match (seqs, scan_rc=True)
        give the best match of each motif in each sequence returns an iterator of nested lists containing tuples:
        (score, position, strand)

    best_score (seqs, scan_rc=True, normalize=False)
        give the score of the best match of each motif in each sequence returns an iterator of lists containing floats

    count (seqs, nreport=100, scan_rc=True)
        count the number of matches above the cutoff returns an iterator of lists containing integer counts

    scan (seqs, nreport=100, scan_rc=True, normalize=False)
        scan a set of regions / sequences

    set_background (fname=None, genome=None, length=200, nseq=10000)
        Set the background to use for FPR and z-score calculations.

        Background can be specified either as a genome name or as the name of a FASTA file.

        Parameters

        fname [str, optional] Name of FASTA file to use as background.

        genome [str, optional] Name of genome to use to retrieve random sequences.
length  [int, optional] Length of genomic sequences to retrieve. The default is 200.

nseq  [int, optional] Number of genomic sequences to retrieve.

set_genome (genome)
set the genome to be used for:

• converting regions to sequences
• background for MOODS

set_threshold (fpr=None, threshold=None)
Set motif scanning threshold based on background sequences.

Parameters

fpr  [float, optional] Desired FPR, between 0.0 and 1.0.

threshold  [float or str, optional] Desired motif threshold, expressed as the fraction of the
difference between minimum and maximum score of the PWM. Should either be a float
between 0.0 and 1.0 or a filename with thresholds as created by ‘gimme threshold’.

total_count (seqs, nreport=100, scan_rc=True)
count the number of matches above the cutoff returns an iterator of lists containing integer counts
gimmemotifs.scanner.scan_to_best_match (fname, motifs, ncpus=None, genome=None, score=False)
Scan a FASTA file with motifs.
Scan a FASTA file and return a dictionary with the best match per motif.

Parameters

fname  [str] Filename of a sequence file in FASTA format.
motifs  [list] List of motif instances.

Returns

result  [dict] Dictionary with motif scanning results.

4.8.4 Maelstrom

4.8.5 Motif activity prediction

Module for motif activity prediction
class gimmemotifs.moap.BayesianRidgeMoap (scale=True, ncpus=None)

Methods

create(name[, ncpus])  Create a Moap instance based on the predictor name.
list_classification_predictors()  List available classification predictors.
list_predictors()  List available predictors.
list_regression_predictors()  List available regression predictors.
register_predictor(name)  Register method to keep list of predictors.
class gimmemotifs.moap.HypergeomMoap(*args, **kwargs)

Methods

create(name[, ncpus])  Create a Moap instance based on the predictor name.
lst_classification_predictors()  List available classification predictors.
lst_predictors()  List available predictors.
lst_regression_predictors()  List available regression predictors.
register_predictor(name)  Register method to keep list of predictors.

fit

class gimmemotifs.moap.LassoMoap(scale=True, kfs=4, alpha_stepsize=1.0, ncpus=None)

Methods

create(name[, ncpus])  Create a Moap instance based on the predictor name.
lst_classification_predictors()  List available classification predictors.
lst_predictors()  List available predictors.
lst_regression_predictors()  List available regression predictors.
register_predictor(name)  Register method to keep list of predictors.

fit

class gimmemotifs.moap.LightningClassificationMoap(scale=True, permute=False, ncpus=None)

Methods

create(name[, ncpus])  Create a Moap instance based on the predictor name.
lst_classification_predictors()  List available classification predictors.
lst_predictors()  List available predictors.
lst_regression_predictors()  List available regression predictors.
register_predictor(name)  Register method to keep list of predictors.

fit

class gimmemotifs.moap.LightningRegressionMoap(scale=True, cv=3, ncpus=None)

Methods

create(name[, ncpus])  Create a Moap instance based on the predictor name.
lst_classification_predictors()  List available classification predictors.
Continued on next page
Table 7 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list_predictors()</td>
<td>List available predictors.</td>
</tr>
<tr>
<td>list_regression_predictors()</td>
<td>List available regression predictors.</td>
</tr>
<tr>
<td>register_predictor(name)</td>
<td>Register method to keep list of predictors.</td>
</tr>
</tbody>
</table>

```python
class gimmemotifs.moap.MWUMoap (*args, **kwargs)

Methods

create(name[, ncpus]) Create a Moap instance based on the predictor name.

list_classification_predictors() List available classification predictors.

list_predictors() List available predictors.

list_regression_predictors() List available regression predictors.

register_predictor(name) Register method to keep list of predictors.
```

```python
class gimmemotifs.moap.Moap

Moap base class.

Motif activity prediction.

Attributes

name

Methods

create(name[, ncpus]) Create a Moap instance based on the predictor name.

list_classification_predictors() List available classification predictors.

list_predictors() List available predictors.

list_regression_predictors() List available regression predictors.

register_predictor(name) Register method to keep list of predictors.
```

```python
classmethod create (name, ncpus=None)

Create a Moap instance based on the predictor name.

Parameters

name [str] Name of the predictor (eg. Xgboost, BayesianRidge, …)

ncpus [int, optional] Number of threads. Default is the number specified in the config.

Returns

moap [Moap instance] moap instance.
```

```python
classmethod list_classification_predictors ()

List available classification predictors.
```
class method list_predictors()
    List available predictors.

class method list_regression_predictors()
    List available regression predictors.

class method register_predictor(name)
    Register method to keep list of predictors.

class gimmemotifs.moap.RFMoap(ncpus=None)

    Methods

create(name[, ncpus]) Create a Moap instance based on the predictor name.
list_classification_predictors() List available classification predictors.
list_predictors() List available predictors.
list_regression_predictors() List available regression predictors.
register_predictor(name) Register method to keep list of predictors.

class gimmemotifs.moap.XgboostRegressionMoap(scale=True, ncpus=None)

    Methods

create(name[, ncpus]) Create a Moap instance based on the predictor name.
list_classification_predictors() List available classification predictors.
list_predictors() List available predictors.
list_regression_predictors() List available regression predictors.
register_predictor(name) Register method to keep list of predictors.

gimmemotifs.moap.moap (inputfile, method='hypergeom', scoring=None, outfile=None, motiffile=None, pwmfile=None, genome=None, fpr=0.01, ncpus=None, subsample=None)
Run a single motif activity prediction algorithm.

Parameters

inputfile [str]: File with regions (chr:start-end) in first column and either cluster name in second column or a table with values.
scoring: str, optional: Either ‘score’ or ‘count’
outfile [str, optional]: Name of outputfile to save the fitted activity values.
motiffile [str, optional]: Table with motif scan results. First column should be exactly the same regions as in the inputfile.
**pwmfile** [str, optional] File with motifs in pwm format. Required when motiffile is not supplied.

**genome** [str, optional] Genome name, as indexed by gimme. Required when motiffile is not supplied

**fpr** [float, optional] FPR for motif scanning

**ncpus** [int, optional] Number of threads to use. Default is the number specified in the config.

**Returns**

pandas DataFrame with motif activity

### 4.8.6 Motif statistics

Calculate motif enrichment statistics.

```
gimmemotifs.stats.calc_stats(motifs, fg_file, bg_file, genome=None, stats=None, ncpus=None)
```

Calculate motif enrichment metrics.

**Parameters**

- **motifs** [str, list or Motif instance] A file with motifs in pwm format, a list of Motif instances or a single Motif instance.
- **fg_file** [str] Filename of a FASTA, BED or region file with positive sequences.
- **bg_file** [str] Filename of a FASTA, BED or region file with negative sequences.
- **genome** [str, optional] Genome or index directory in case of BED/regions.
- **stats** [list, optional] Names of metrics to calculate. See gimmemotifs.rocmetrics.__all__ for available metrics.
- **ncpus** [int, optional] Number of cores to use.

**Returns**

result [dict] Dictionary with results where keys are motif ids and the values are dictionary with metric name and value pairs.

```
gimmemotifs.stats.calc_stats_iterator(motifs, fg_file, bg_file, genome=None, stats=None, ncpus=None)
```

Calculate motif enrichment metrics.

**Parameters**

- **motifs** [str, list or Motif instance] A file with motifs in pwm format, a list of Motif instances or a single Motif instance.
- **fg_file** [str] Filename of a FASTA, BED or region file with positive sequences.
- **bg_file** [str] Filename of a FASTA, BED or region file with negative sequences.
- **genome** [str, optional] Genome or index directory in case of BED/regions.
- **stats** [list, optional] Names of metrics to calculate. See gimmemotifs.rocmetrics.__all__ for available metrics.
- **ncpus** [int, optional] Number of cores to use.

**Returns**

result [dict] Dictionary with results where keys are motif ids and the values are dictionary with metric name and value pairs.
gimmemotifs.stats.rank_motifs(stats, metrics=('roc_auc', 'recall_at_fdr'))
    Determine mean rank of motifs based on metrics.

gimmemotifs.stats.write_stats(stats, fname, header=None)
    write motif statistics to text file.

### 4.8.7 Motif comparison

Module to compare DNA sequence motifs (positional frequency matrices)

```python
class gimmemotifs.comparison.MotifComparer
    Class for motif comparison.

    Compare two or more motifs using a variety of metrics. Probably the best metric to compare motifs is seqcor. The implementation of this metric is similar to the one used in Grau (2015), where motifs are scored according to the Pearson correlation of the scores along sequence. In this case a de Bruijn of k=7 is used.

    Valid metrics are: seqcor - Pearson correlation of motif scores along sequence. pcc - Pearson correlation coefficient of motif PFMs. ed - Euclidean distance-based similarity of motif PFMs. distance - Distance-based similarity of motif PFMs. wic - Weighted Information Content, see van Heeringen 2011. chisq - Chi-squared similarity of motif PFMs. akl - Similarity based on average Kullback-Leibler similarity, see Mahony, 2011. ssd - Sum of squared distances of motif PFMs.

Examples

mc = MotifComparer()
    # Compare two motifs score, pos, strand = mc.compare_motifs(m1, m2, metric="seqcor")
    # Compare a list of motifs to another list of motifs mc.get_all_scores(motifs, dbmotifs, match, metric, combine)
    # Get the best match for every motif in a list of reference motifs mc.get_closest_match(motifs, dbmotifs=None)

Methods

compare_motifs(m1, m2[, match, metric, ...])  Compare two motifs.
get_all_scores(motifs, dbmotifs, match, ...)  Pairwise comparison of a set of motifs compared to reference motifs.
get_closest_match(motifs[, dbmotifs, match, ...])  Return best match in database for motifs.
```

```python
compare_motifs(m1, m2, match='total', metric='wic', combine='mean', pval=False)
    Compare two motifs.
```
The similarity metric can be any of seqcor, pcc, ed, distance, wic, chisq, akl or ssd. If match is ‘total’ the similarity score is calculated for the whole match, including positions that are not present in both motifs. If match is partial or subtotal, only the matching positions are used to calculate the score. The score of individual position is combined using either the mean or the sum.

Note that the match and combine parameters have no effect on the seqcor similarity metric.

**Parameters**

- **m1** [Motif instance] Motif instance 1.
- **m2** [Motif instance] Motif instance 2.
- **match** [str, optional] Match can be “partial”, “subtotal” or “total”. Not all metrics use this.
- **metric** [str, optional] Distance metric.
- **combine** [str, optional] Combine positional scores using “mean” or “sum”. Not all metrics use this.
- **pval** [bool, optional] Calculate p-value of match.

**Returns**

- **score, position, strand**

**get_all_scores** *(motifs, dbmotifs, match, metric, combine, pval=False, parallel=True, trim=None, ncpus=None)*

Pairwise comparison of a set of motifs compared to reference motifs.

**Parameters**

- **motifs** [list] List of Motif instances.
- **dbmotifs** [list] List of Motif instances.
- **match** [str] Match can be “partial”, “subtotal” or “total”. Not all metrics use this.
- **metric** [str] Distance metric.
- **combine** [str] Combine positional scores using “mean” or “sum”. Not all metrics use this.
- **pval** [bool, optional] Calculate p-value of match.
- **parallel** [bool, optional] Use multiprocessing for parallel execution. True by default.
- **trim** [float or None] If a float value is specified, motifs are trimmed used this IC cutoff before comparison.
- **ncpus** [int or None] Specifies the number of cores to use for parallel execution.

**Returns**

- **scores** [dict] Dictionary with scores.

**get_closest_match** *(motifs, dbmotifs=None, match='partial', metric='wic', combine='mean', parallel=True, ncpus=None)*

Return best match in database for motifs.

**Parameters**

- **motifs** [list or str] Filename of motifs or list of motifs.
- **dbmotifs** [list or str, optional] Database motifs, default will be used if not specified.
- **match** [str, optional]
- **metric** [str, optional]
combine [str, optional]
ncpus [int, optional] Number of threads to use.

Returns

closest_match [dict]
gimmemotifs.comparison.akl(p1, p2)
Calculates motif position similarity based on average Kullback-Leibler similarity.
See Mahony, 2007.

Parameters

Returns
score [float]
gimmemotifs.comparison.chisq(p1, p2)
Calculates motif position similarity based on chi-square.

Parameters

Returns
score [float]
gimmemotifs.comparison.seqcor(m1, m2, seq=None)
Calculates motif similarity based on Pearson correlation of scores.

Based on Kielbasa (2015) and Grau (2015). Scores are calculated based on scanning a de Bruijn sequence of
7-mers. This sequence is taken from ShortCAKE (Orenstein & Shamir, 2015). Optionally another sequence can
be given as an argument.

Parameters
m1 [Motif instance] Motif 1 to compare.
m2 [Motif instance] Motif 2 to compare.
seq [str, optional] Sequence to use for scanning instead of k=7 de Bruijn sequence.

Returns
score, position, strand
gimmemotifs.comparison.ssd(p1, p2)
Calculates motif position similarity based on sum of squared distances.

Parameters

Returns
score [float]
4.9 FAQ

4.9.1 Sorry, motif prediction tool [X] is not supported

If *gimme motifs* does not recognize a motif tools that should be installed, you can update the configuration file. This file is likely located at ~/.config/gimmemotifs/gimmemotifs.cfg.

Edit this file and update the following lines under the [params] section:

```
available_tools = MDmodule, MEME, MEMEWEB, Weeder, GADEM, MotifSampler, Trawler, Improbizer, BioProspector, Posmo, ChIPMunk, AMD, Homer
```

Add the tool that you want to the available_tools parameter. Keep in mind the exact upper-/lower-case combination that GimmeMotifs uses. By updating the tools parameter you can set the tools that *gimme motifs* uses by default. This can always be changed at the command-line.

In addition, you might also have to update the binary location. Either update this section if it exists or add it. For instance, to set this information for MEME:

```
[MEME]
bin = /home/simon/anaconda2/envs/gimme/gimmemotifs/tools/meme.bin
dir = /home/simon/anaconda2/envs/gimme/gimmemotifs/tools
```

The dir variable usually doesn’t matter and you can set it the same directory as where the binary is located.

4.9.2 I get motifs that have differential scores in gimme maelstrom, however, the number is not different across clusters

The different methods use different ways to rank the motifs. The hypergeometric test is the only one that uses motif counts. All the other methods use the PWM logodds score of the best match. While the counts may not be different across clusters, the scores most likely are.

4.9.3 I have upgraded GimmeMotifs and now it doesn’t find my genome

The genome index in GimmeMotifs has changed, see upgradegenome_.

4.9.4 I cannot run gimme index anymore

The genome index in GimmeMotifs has changed, see upgradegenome_.

4.9.5 I get ‘RuntimeError: Invalid DISPLAY variable’

The default matplotlib configuration expects a display. Probably you are running GimmeMotifs on a server without an X server. There are several ways to solve it.
Option 1

Change the matplotlib configuration. Find the path of the matplotlib installation of your current environment (make sure to activate the environment you use for GimmeMotifs first).

```bash
$ python -c "import matplotlib; print(matplotlib.__file__)"
/home/simon/anaconda2/envs/gimme3/lib/python3.5/site-packages/matplotlib/__init__.py
```

So, matplotlib is in /home/simon/anaconda2/envs/gimme3/lib/python3.5/site-packages/matplotlib/. Now you can edit `<MATPLOT_DIR>/mpl-data/matplotlibrc`. In my example case this would be:

```
/home/simon/anaconda2/envs/gimme3/lib/python3.5/site-packages/matplotlib/mpl-data/matplotlibrc
```

Change the line

```plaintext
backend : Qt5Agg
```

to

```plaintext
backend : Agg
```

You can also put a matplotlibrc file in $HOME/.config/matplotlib.

Option 2

Run GimmeMotifs via `xvfb-run`. If this program is installed, you can simply run GimmeMotifs in a virtual X server environment.

For example:

```bash
$ xvfb-run gimme motifs [args]
```

4.9.6 I get a KeyError when running gimme maelstrom

You get an error like this:

```python
File "pandas/_libs/index.pyx", line 132, in pandas._libs.index.IndexEngine.get_loc
    (...pandas/_libs/index.c:5280)
File "pandas/_libs/index.pyx", line 154, in pandas._libs.index.IndexEngine.get_loc
    (...pandas/_libs/index.c:5126)
File "pandas/_libs/hashtable_class_helper.pxi", line 1210, in pandas._libs.hashtable.PyObjectHashTable.get_item (pandas/_libs/hashtable.c:20523)
File "pandas/_libs/hashtable_class_helper.pxi", line 1218, in pandas._libs.hashtable.PyObjectHashTable.get_item (pandas/_libs/hashtable.c:20477)
  KeyError: '5'
```

This is a bug in gimme maelstrom. The column headers can’t be numbers. Change this to a word, for instance `cluster5` or `col5`.
4.10 Acknowledgments

We are grateful to Waseem Akhtar, Robert Akkers, Max Koeppel, Evelyn Kouwenhoven, Marion Lohrum, Leonie Smeenk and Jo Zhou for providing data and feedback during GimmeMotifs development. Also we would like to thank Stefanie Bartels, Adalberto Costessi, Joost Martens and Nagesha Rao for testing and helpful discussion. Of course GimmeMotifs by itself wouldn’t be able to do anything, if there wasn’t such a number of excellent tools available. Therefore, a big thanks to all the authors of the motif prediction programs for making their software publicly available and allowing me to distribute them with GimmeMotifs! In addition, I would like to thank Wolfgang Lugmayr and Aaron Statham for various bugfixes.
Python Module Index

g

gimmemotifs.comparison, 55

gimmemotifs.denovo, 46

gimmemotifs.moap, 50

gimmemotifs.motif, 40

gimmemotifs.scanner, 49

gimmemotifs.stats, 54
Index

A
akl() (in module gimmemotifs.comparison), 57
average_motifs() (gimmemotifs.motif.Motif method), 41

B
BayesianRidgeMoap (class in gimmemotifs.moap), 50
best_match() (gimmemotifs.scanner.Scanner method), 49
best_motif_in_cluster() (in module gimmemo-
tifs.denovo), 46
best_score() (gimmemotifs.scanner.Scanner method), 49

C
calc_stats() (in module gimmemotifs.stats), 54
calc_stats_iterator() (in module gimmemo-
tifs.stats), 54
chisq() (in module gimmemotifs.comparison), 57
compare_motifs() (gimmemotifs.comparison.MotifComparer method), 55
consensus_scan() (gimmemotifs.motif.Motif method), 42
count() (gimmemotifs.scanner.Scanner method), 49
create() (gimmemotifs.moap.Moap class method), 52
create_background() (in module gimmemo-
tifs.denovo), 47
create_backgrounds() (in module gimmemo-
tifs.denovo), 47

D
default_motifs() (in module gimmemotifs.motif), 45

F
filter_significant_motifs() (in module gimmemotifs.denovo), 47

G
get_all_scores() (gimmemotifs.comparison.MotifComparer method), 56
get_closest_match() (gimmemotifs.comparison.MotifComparer method), 56
gimme_motifs() (in module gimmemo-
tifs.denovo), 48
gimmemotifs.comparison (module), 55
gimmemotifs.denovo (module), 46
gimmemotifs.moap (module), 50
gimmemotifs.motif (module), 40
gimmemotifs.scanner (module), 49
gimmemotifs.stats (module), 54

H
hash() (gimmemotifs.motif.Motif method), 42

I
ic_pos() (gimmemotifs.motif.Motif method), 42
information_content() (gimmemotifs.motif.Motif method), 42

L
LassoMoap (class in gimmemotifs.moap), 51
LightningClassificationMoap (class in gim-
memotifs.moap), 51
LightningRegressionMoap (class in gimmemo-
tifs.moap), 51
list_classification_predictors() (gim-
memotifs.moap.Moap class method), 52
list_predictors() (gimmemotifs.moap.Moap class method), 52
list_regression_predictors() (gimmemotifs.
moap.Moap class method), 53

M
Moap (class in gimmemotifs.moap), 52
moap() (in module gimmemotifs.moap), 53
Motif (class in gimmemotifs.motif), 40
motif_from_align() (in module gimmemotifs.motif), 45
motif_from_consensus() (in module gimmemotifs.motif), 45
MotifComparer (class in gimmemotifs.comparison), 55
MWUMoap (class in gimmemotifs.moap), 52

P
parse_motifs() (in module gimmemotifs.motif), 46
pcc_pos() (gimmemotifs.motif.Motif method), 42
pfm_to_pwm() (gimmemotifs.motif.Motif method), 42
prepare_denovo_input_bed() (in module gimmemotifs.denovo), 48
prepare_denovo_input_fa() (in module gimmemotifs.denovo), 48
prepare_denovo_input_narrowpeak() (in module gimmemotifs.denovo), 48
pwm_max_score() (gimmemotifs.motif.Motif method), 42
pwm_min_score() (gimmemotifs.motif.Motif method), 42
pwm_scan() (gimmemotifs.motif.Motif method), 43
pwm_scan_all() (gimmemotifs.motif.Motif method), 43
pwm_scan_score() (gimmemotifs.motif.Motif method), 43
pwm_scan_to_gff() (gimmemotifs.motif.Motif method), 44

R
randomize() (gimmemotifs.motif.Motif method), 44
rank_motifs() (in module gimmemotifs.stats), 54
rc() (gimmemotifs.motif.Motif method), 44
read_motifs() (in module gimmemotifs.motif), 44
register_predictor() (gimmemotifs.moap.Moap class method), 53
rename_motifs() (in module gimmemotifs.denovo), 48
RFMoap (class in gimmemotifs.moap), 53

S
scan() (gimmemotifs.scanner.Scanner method), 49
scan_to_best_match() (in module gimmemotifs.scanner), 50
Scanner (class in gimmemotifs.scanner), 49
score_kmer() (gimmemotifs.motif.Motif method), 44
seqcor() (in module gimmemotifs.comparison), 57
set_background() (gimmemotifs.scanner.Scanner method), 49
set_genome() (gimmemotifs.scanner.Scanner method), 50
set_threshold() (gimmemotifs.scanner.Scanner method), 50
ssd() (in module gimmemotifs.comparison), 57

T
to_img() (gimmemotifs.motif.Motif method), 44
to_meme() (gimmemotifs.motif.Motif method), 45
to_motef() (gimmemotifs.motif.Motif method), 45
to_pwm() (gimmemotifs.motif.Motif method), 45
to_transfac() (gimmemotifs.motif.Motif method), 45
total_count() (gimmemotifs.scanner.Scanner method), 50
trim() (gimmemotifs.motif.Motif method), 45

W
write_stats() (in module gimmemotifs.stats), 55

X
XgboostRegressionMoap (class in gimmemotifs.moap), 53