MNLT - Mariusz Nowacki Lab Tools

Sebastian Bechara

August 2024

This is a tiny documentation how to use the different scripts. It will contain a rough breakdown of the methods used in the scripts and how to use them.

The toolset itself consists of several “main” scripts and a few utility scripts. The “main” scripts comprise some common analyses and a few not so common ones.

Setup

This deals with setting up everything. Since python is an interpreted scripting language it does not need to be compiled to run. If everything is configured properly on your system, it should technically run out of the box.
That said most of the scripts will still use some programs / modules that you’ll need to install beforehand. In the following is a short description on how to do that on MAC and on Linux (tested for MACOS High Sierre 10.13.4 and Ubuntu 20.04). Windows won’t work as it uses a different directory structure and most bioinformatics program won’t work there anyway.

Dependencies

These are the different programs or python modules that you need to have in your path (environment variable) to be able to run everything.

- Python version >= 3.8 - Scriting language
- Python: Numpy - Python module
- Python: Pandas - Python module
- Python: Matplotlib - Python module
- Python: Seaborn - Python module
- Python: BioPython - Python module
- Python: SciPy - Python module
- Python: scikit-learn - Python module
- Python: UMAP - Python module
- Python: statsmodels - Python module
- HiSat2 - Alignment program
- samtools - sam/bam file manipulation

These are requirements for ParTIES. You’ll need it to get cryptic and alternative excision events. The software itself is written in Perl (another scripting language), which is also interpreted, so no need for compiling.

- PERL version >= 5.30.0
- PERL - Statistics::R - Statistical library. You will nedd to have R installed for this to work
- PERL - Parallel::ForkManager - For multithreading

- PERL - Bio::GFF3::LowLevel - BioPerl
- PERL - Bio::DB::Sam - BioPerl
- bowtie2 - Alignment program
- velvet
- RepeatMasker - Detects and masks repeats
- BLAT - Alignment program
- muscle - Multiple alignment program

How to get it

You can obtain a copy by either copying a tar-archive, that I left on the server here: XXX to your system and then type `tar -zxvf MNLT.tar.gz`. Or you can clone/wget the github repository by one of the following commands:

```
## clone it
git clone https://github.com/SebastianBechara/MNLT.git

## wget it
wget https://github.com/SebastianBechara/MNLT/archive/main.tar.gz
tar -zxvf main.tar.gz
mv MNLT-main MNLT
```

Ubuntu

In Ubuntu (and by proxy Debian and any Ubuntu based Linux distributions) you can use the apt package manager. Most things are available there. Python modules are downloaded with the Python own package manager pip. And Perl modules are installed via cpan. Here you'll get a short guide on how to install stuff via both.

Python modules

We'll start with Python and its modules. Technically Python should be installed on Linux machines.
Open the terminal by shortcut ctrl+alt+t or via the sidebar Then check if Python is installed:

```
python3 --version
```

This should display the installed version of python if any. If python is not installed it will tell you something like this:

```
Command 'python3' not found, did you mean:

.
.
.

Try: sudo apt install <deb name>
```

If you get this message you will have to install it. We'll use apt to do that

```
sudo apt update && sudo apt update -y    ## updates the package manager
sudo apt install python3                 ## installs python
```

You can then verify it with `python3 --version`. If it gives you the version now you have it successfully installed
Python modules are installed via pip. Technically you could use:

```
pip install <package>
```

But there are some problems with pip since quite a while, and they don't really seem to address it (as far as I checked) so I would advise to use this instead:

```
python3 -m pip install <package>
```

This will essentially call pip via python, which is a bit safer and tends to have less problems. Sometimes you need to add `--user` to allow it access to certain parts of your system. So, the commands for the above modules would be:

```
python3 -m pip install numpy pandas matplotlib seaborn biopython scipy \
scikit-learn umap-learn [--user]
```

Other programs and PERL

HiSat2, samtools and a couple of others can be installed like so:

```
sudo apt install hisat2 samtools bowtie2 velvet muscle
```

RepeatMasker and BLAT need to be compiled separately, as they are not on the apt package-manager at the time of writing this. Theoretically you won't need any of these (repeatmasker, BLAT, velvet, muscle) to run the MNTL wrapper, as MILORD only needs bowtie2. So, everything will work fine if you just bowtie2 installed.
For Perl you can check if it is installed by default by typing `perl --version`, if this returns a version you should be fine. You will also have a new enough version (>= 5.8) if you're not running everything on a potato.
To install the Perl modules we'll use cpan, but first we'll need an ancient version of samtools to make Bio::DB::Sam work. The version we need is 0.1.18 and the newest is 1.20, just to give you an idea 0.1.18 is from 2011!
To get it, navigate to the location you would like to compile it and type the following into your terminal:

```
mkdir samtools && cd samtools            ## you can ommit that if you wish
wget https://sourceforge.net/projects/samtools/files/samtools/0.1.18\
/samtools-0.1.18.tar.bz2
tar -jxvf samtools-0.1.18.tar.bz2
cd samtools-0.1.18
make "CFLAGS=-g -Wall -O2 -fPIC"
```

If you don't have R installed yet, you can do that before going ahead with the Perl modules. Go to their website: https://cran.r-project.org/. Pick your OS and follow the instructions.
Now we can install the Perl modules using cpan:

```
cpan install Parallel::ForkManager
cpan install Statistics::R
cpan install Bio::GFF3::LowLevel
cpan install Bio::DB::Sam
```

For some modules you'll need to confirm a few things by typing `y` in the terminal when it prompts you. For Bio::DB::Sam you will also need to supply the path to these files: `bam.h`, `libbam.a`, which is the directory where you compiled the old samtools version. If you get a prompt that some other module is missing install that too. The prompt will usually tell you which is missing and how to install it. If not, Perl modules usually have this syntax `something::something`, sometimes also with more than one part `a::b::c`.
If I didn't forget anything you should be ready for setting up MNLT.

MAC

For this to work your MAC computer will need to be as up-to-date as possible. Otherwise, you won't be able to install certain things out of forced incompatibility reasons on Apples side. You will also need sudo rights to install a few things. If you're installing it on a private MAC then you should know what your password is. If you're installing it on a uni MAC however, you'll need access to the local admin Jan and Bashi have set up on the different machines. You can ask either of them, they'll then generate you temporary local-admin password. Tell them what it's for and for how long you'll need it (let's say a week to be on the safe side).
For MAC you'll need to download homebrew, and a few additional software that is not available on homebrew. Homebrew is a so-called package manager, think of it as an extra step to monitor updates and software related stuff.
You can install it with this command:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew\
/install/HEAD/install.sh)"
```

The script will guide you through the process. You will be prompted to input a so called sudo password. This is essentially the admin password of the computer you're running right now. Depending on what it will need to install dependency-wise it will ask you a couple of times to input it.
To check if it is installed you can type in: `brew --version`; you should get an output similar to this one: `Homebrew 4.3.13`. Obviously there may be some version changes, but that should be fine. Just to get started with homebrew, we'll install wget:

```
brew install wget
```

For some programs you'll also need the xcode dev tools, for which you need an own account with Apple. Go to the Apple developer website and if you do not have an account create one. I don't really know if any Apple account will work, feel free to try. Once you're in, navigate to 'More' (it should be at the top right) and search for "xcode 15.4", this is the stable version of xcode for macOS 14.6 (Sonoma). Click on 'View Details' and download the 'Xcode 15.4.xip'. Once downloaded click on the .xip file to extract it, this happens completely automatically. Once it's done a file will appear name something like 'Xcode', this you'll have to drag and drop to the applications directory. To finish the initialisation, you should probably want to install system updates (Not sure if you really need that, it was recommended by a couple of people online). This will reboot your machine. Once it is on and running again search for xcode with cmd+space and execute it. This will guide you through additional software installations if deemed necessary. Now it should be ready.
You'll also need conda, well specifically miniconda. Lucky for you we can just download it with Homebrew:

```
brew install miniconda
```

Python and modules

Again, as with Ubuntu (or any other Linux distro) Python should be preinstalled. You can check that (analogue to other linux distros) by typing in this in the terminal:

```
python3 --version
```

This should return you the version of the installed python. It would be best if you have >=3.8, as that is the version I used to write everything.
To install all neccessary modules use this, as with ubuntu:

```
python3 -m pip install numpy pandas matplotlib seaborn biopython scipy \
scikit-learn umap-learn [--user]
```

Other programs and PERL

We'll start with programs other that Perl. Some of these are found and can be install with homebrew:

```
brew install bowtie2 samtools
```

Some with conda:

```
conda install bioconda::hisat2 bioconda::velvet muscle
```

Technically those can be installed from source, but that may end up in a lot of troubleshooting. When installing with conda, you will need to open the conda environment. I you're doing that for the first time you have to type `conda init` into your terminal, close and reopen it. Now you can `conda activate`, which will put `(base)` at the beginning of your terminal line. Now all programs installed in conda are usable. You may need to reinstall the python modules if you installed them in your system and not your conda environment.
For Perl you will essentially need a few different modules to get ParTIES working. We kind off only need the MILORD submodule, but you still need to install everything else.
First, you'll need to get R. Go to their website: https://cran.r-project.org/. Pick your OS (MAC) now you'll have to download the right version depending on your cpu (either intel or apple-intern).

For the Perl modules:
DISCLAIMER: I didn't get everything running on a MAC. Essentially it's only this module: Bio::DB::Sam. Olivier Arnaiz, who is the original author of ParTIES, did never try to install it on MACs. I have also written to Estienne, when and/or if he answers I'll modify this documentation and reupload it on github.
Theoretically you will also need to install an ancient version of samtools v<=0.1.18, which theoretically should work the same as in Ubuntu:

```
mkdir samtools && cd samtools             ## you can ommit that if you wish
wget https://sourceforge.net/projects/samtools/files/samtools/0.1.18\
/samtools-0.1.18.tar.bz2
tar -jxvf samtools-0.1.18.tar.bz2
cd samtools-0.1.18
make "CFLAGS=-g -Wall -O2 -fPIC"
```

As already said I didn't get this to work on a MAC system (14.5 - Sonoma). Seemingly Perl cannot find the `sam.h` file, even though it is there. I assume some kind of compatibility error or some permissions conflicts, but I am not familiar enough with the quirk of macOS.

Perl module can be installed in a few different ways. If you want to dig into how to do it otherwise feel free. We'll be using cpan in the following.

Type this into your terminal:

```
cpan install Parallel::ForkManager
cpan install Statistics::R
cpan install Bio::GFF3::LowLevel
cpan install Bio::DB::Sam
```

Final setup

To finish setting up the whole program run the setup file:

```
chmod u+x setup
./setup
```

This will set permissions for the user to be able to execute files. And it will also set the MNLT home directory to the path. Then you can call it from everywhere.
Now run the check_I script to see what you already have installed and what is still missing. As already mentioned, these programs are not essential: velvet, RepeatMasker, BLAT and MUSCLE.

When you are in the active directory of package type

```
./check_I
```

This will show you a list with dependencies, and it will tell you if they are `OK` or `Not found`. You can then run `.check_II`, this will go once through each major script and run them on a test dataset. It may take a while, and it uses multiple threads so be aware if you're on an older machine with less than 5 core (which amounts to 10 threads). Once this is done, you can compare the files generated in TEST directory with the ones int the `testData/testOut` directory. They should be identical.
If everything ran without any error messages you have it up and running.

Tooloverview & Usage

Overview

The whole tool is written as a modular program. Every script can be executed by itself, or via the MNLT-wrapper.
This is the directory overview:

```
MNLT
├── bin
├── docs
├── res
│   ├── anno
│   ├── ref
│   │   └── hisat2
│   │       ├── klebs
│   │       ├── L4440
│   │       ├── mito
│   │       ├── new_para_MIC
│   │       ├── para_MAC
│   │       ├── para_MIC
│   │       └── para_rRNA
│   └── seqs
│       └── genomes
├── testData
│   ├── sRNA
│   └── testOut
│       └── MILORD
├── tpp
│   └── ParTIES
│       ├── conf
│       ├── example
│       ├── lib
│       │   └── PARTIES
│       └── utils
└── utils
```

The program contains 7 submodules and a bunch of utility scripts.

The 7 main scripts are:

Module / Scripts	Function
IRS_calc	As the name suggests, this calculates IRSs based on either trimmed and subsampled reads or on a BAM file.
EWMA_Freq	This can create two plots. Firstly, it calculates a moving average for the IRSs based on the length of the associated IES. And secondly it gets you base compositions of the first 3 bases after the initial TA
afterParties	This is an implementation or rather a wrapper of Estienne's correlation matrices. In short it will calculate pairwise correlations between samples using IRSs
Clustering	An alternate approach to Estienne's correlations. Clustering by dimensional

	reduction is used to infer similarities between samples based on IRSs
sRNA_hist	This generates the typical sRNA distribution histograms
MILORD	A wrapper for the MILORD submodule of ParTIES. It will detect excision events, be it IES or alternative/cryptic excision
afterMILORD	This produces the cryptic / alternative excision histograms

These utility scripts under `utils` are essentially scripts that I thought may be useful but are not necessarily essential.
Here is a list on scripts:

```
utils
├── boundarySmallRNAs.py
├── codon_opt.py
├── CountsRNAnucs.py
├── get_motive.py
├── gff_fasta_to_GenBank.py
├── IESexcissionScript.py
├── plotHist.py
├── pol23.py
├── returnORFs.py
├── sRNAhist_w_boundaries.py
└── statsForAssemblies.py
```

They are also not guaranteed to work, as they were only run by me. I didn't generalise them, but I did some minor cleaning, meaning removing outcommented stuff and changing hardcoded bits. That also means that they are not tested on different machines, so no guarantee whatsoever if you want to use these. They should all work. If not, they can be used as future reference.
Here a short explanation for each scripts function:

Script	Function
boundarySmallRNAs.py	This checks sRNAs that sit on IES::MDS boundaries
codon_opt.py	This does some simple codon optimisation for adapting sequences or for recodonisation
CountsRNAnucs.py	This gets nucleotide composition from sRNAs. Don't remember why I wrote this. May be useful for someone in the future
get_motive.py	This checks regular expression patterns in a provided nucleotide / protein file
gff_fasta_to_Genbank.py	Converts GFF and fasta file to genbank format
IESexcissionScript.py	Script I used to infer IES excision in long reads with ONT sequencing (Iris's RPC paper)
plotHist.py	Script to get the typical IRS histograms. Mainly for continuity reasons
plo23.py	Script I used to figure out if Sarah's PolII/III pull downs had IES::IES junctions and concatemers associated to them. The concatemer detection part is very bare bones, I didn't refine it as we also couldn't find a discernible lot of IES::IES junctions
returnORFs.py	Gets ORFs in provided fasta file
sRNAhist_w_boundaries.py	sRNA histograms with integrated sRNA boundary calls. You'll need to change some hardcoded paths if you want to run that
statsForAssemblies.py	Calculates some common stats for assemblies. I never got around to

	actually finish it, so you may need to copy paste some functions into an interactive python session

The resources directory `res` contains a bunch of files needed for proper functioning, and something that are generally useful. They include reference genomes under `res/seqs/genomes`, some annotation files under `res/anno/` and some references for hisat2 under `res/ref/hisat2`
The `testData` directory contains a bunch of files that the `check_II` script uses for testing the whole program as for example the sRNA reads. It also contains the `testOut` directory, which contains an example set of files that the program is supposed to generate. You should use these to check if everything worked well. In this case the newly generated TEST directory will have the file that you want to check against the ones in `testOut`.
`tpp` includes a locally bundled version of ParTIES, that I know works. There were different versions of MILORD, without going into too much detail, this version here is the one that works best (has been tested by Estienne and Aditi)

The submodules

The following will include a tiny intro to each module and give some useful pointers.

IRS_calc

As already mentioned, this will calculate the IES retention scores following the method described here: Wilkes, Arnaiz, and Sperling, 2016, Bioinformatics.
What the script does is to map reads once to the MAC genome and once to the MAC+IES genome, and then get the read that span the TA-scar in the MAC (IES^-) or that overlap IESs in the MAC+IES genome (IES^+). The IRS is then calculated like so:

$$IRS = \frac{IES^+}{IES^+ + IES^-}$$

. The Script takes paired or unpaired reads or a BAM file as input. The reads should be (quality) trimmed and subsampled before using them, usually 15 Mio randomly sampled reads are used.
It will output a TSV file containing the IRS per IES and one that contains the read coverage per IES for IES^+ and IES^-

Usage:

```
MNLT IRS_calc --kd <Name of your KD/sample> --fwd <forward reads> --rev
<reverse reads>
```

All arguments:

Argument	Function
`-kd` or `--kd`	This is the name of your KD or your sample. Try to give it a sensible name, not Sample1 or smth similar
`-fwd` or `--fwd`	This is you forward reads. As mentioned, trimmed and subsampled
`-rev` or `--rev`	Same as before only for reverse reads
`-s` or `--soma_sam`	You technically also give it an already mapped SAM file. Just make sure that you did map them without allowing for splicing or big deletions/inserts

-g or --germ_sam	Same as above only this time the reference is the MAC+IES instead of the MAC genome
-single or --single	You can also run the whole thing with single reads if you wish. These also need to be trimmed a sampled
-na or --na	If this is set IESs with low support (<5) will be recorded as NA instead of a 0
-h or --help	Display the help message

EWMA_Freq

This will generate the plots that you can see here: Swart et al, 2017, F1000. EWMA stands for exponentially weighted moving average. Essentially an EWMA is a modified moving average (MA). These are normally used to find trends over time, with a timeseries covering stock prices for example. The MA 'smooths out' data by averaging a set number of data points (over a set timeframe for example 5 days). This will give you a smooth trend line but depending on the number of observations you are using to calculate you MA it may get smoother or 'spikier'. Also, if you datapoints are not evenly spread then the trendline will have 'holes'. The exponentially weighted MA (EWMA) is smoother than the MA as it includes a weighting factor for prior observations that decreases exponentially in strength over time (or datapoints). More recent data will have a higher impact than older data. This also gets rid of 'holes' in the MA as it does not try to average over nothing, but it will average over the weighting factor.
The other functionality of this script is to create nucleotide frequency plots, which can be also found in the same paper. These show the frequency of the first three nucleotides after the initial TA of IESs of a specific size.
The script has three required arguments: `-s` or `--switch`, which tells it what you want (either EWMA, frequency or both plots); `-i` or `--IRS`, which specifies the TSV file containing the IRSs; `-k` or `--KDs`, which tells it which KDs you're interested in.
It will output the specified plots.

Usage:

```
MNLT EWMA_Freq --switch <EWMA | FREQ | BOTH> --IRS <TSV containing IRSs> --
KDs <KD names as present in the IRS file>
```

All arguments:

Argument	Function
-s or --switch	This tells the script what to do: EWMA for the EWMA calculation; FREQ for the frequency calculation or BOTH if you want both plots
-i or --IRSs	This specifies the path to the TSV-file containing the IRSs
-k or --KDs	Here you specify which KDs to regard in order to generate the plots. If you wish to look at multiple at one you can provide the names separated by a whitespace. The names need to be exactly as in the TSV-file, they are case-sensitive
-a or --IES_annotation	This specifies a custom IES annotation file in GFF3 format. If nothing is specified it will use the on supplied in the res directory. These are the standard IES-annotations.

-p or --peaks	The peak sizes are needed for the nucleotide-frequency calculation. These peak sizes are the same as the IES-peak sizes in the IES distribution histogram. Provide them separated by a whitespace, ordered and multiples of 2. Peaks will be inferred by input, e.g. if input = "15 25 98 1110 2000 4000" you will get three peaks with (15,25), (98,1110),(2000,4000). Default are (26,31),(44,52)&(54,62)
-o or --outDir	Specify the path to the directory in which you want to save the plots. If it does not exist it will be created. Default is the current working directory
-h or --help	Displays the help message

afterParties

This calls Estienne's afterParTIES script that calculates correlations between samples. You can pick between spearman and pearson correlations. The default is spearman as pearson is meant for normally distributed data, and IRSs are by definition not normal, more like beta-distributed but I won't go into the statistics on that. I won't explain correlations, as almost everyone has an idea what they are. This script needs 4 arguments that are mandatory: `--input_file`, that specifies IRSs in a TSV-file; `--experiment_columns`, this tells the script what KDs to use for calculating correlations; `--control_columns`, this tells it where the control (EV) experiments are; and `--output_matrix_image`, which specifies the path of the output picture.

Usage:

```
MNLT afterParties --input_file <TSV file containing IRSs> --
experiment_columns <Columns of KDs to use> --control_columns <Control columns
to use> --output_matrix_image <path of output>
```

All arguments:

Argument	Function
--input_file	TSV file containing IRSs
--eyperiment_columns	This specifies the experiment (KD) columns to use for the calculations. Be aware these are zero-based. Meaning if you specify something like this: 2 3 5 7, you actually specified the 3rd, 4th, 6th and 8th columns. Essentially the computer starts to count at 0 and not 1.
--control_columns	This specifies the control (EV) columns to use for the calculations. As for the experiment columns, be aware these are zero based
--output_matrix_image	This specifies the where to save the image. You can choose the format by the file ending. So `/home/<user>/corrMat.svg` saves the figure created in your (user) home directory under the name corrMat as a svg file
--use_pearson	Add this option if Pearson's correlation is to be used instead of Spearman's correlation. Again, in the case of IRSs this does not make sense
--alpha	Alpha is the cut-off from which something is considered significant; A p-value of 0.01 with an alpha of 0.05 is considered significant. Is used

	for p-value testing of Spearman's correlation coefficient (default: 0.01); a Bonferroni correction is applied to this before testing, and when p > alpha/(number of hypotheses), a '^' character is shown next to the calculated correlation coefficient in the upper diagonal matrix (meaning not significant). For example, for 10 experimental knockdowns the number of hypotheses is 45.
--deactivate_ODR	Use this flag to turn of ODR (orthogonal distance regression; the hexbins in the lower triangle) for a speedup (default: False)
--image_resolution	Change the image resolution in DPI. (default: 400)
--base_font_size	Change font size (default: 14)
--font_style	Change font style (default: italic); options = normal, italic or oblique
--regression_line_width	Thickness of regression lines in the hexbins of the lower triangle (default: 1.2)
--show_colorbar	Show colour-bar scale of hexagonal bins
--show_axis_labels	Show labels on axes
--histogram_max	Maximum value of y-axis of the matrix diagonal's histograms (default: 9000)
--hexbin_vmin	Minimum value of hexagonal bin counts (log10 scale) (default: 1)
--hexbin_vmax	Maximum value of hexagonal bin counts (log10 scale) (default: 1000)
--verbose	This displays the ODR calculation output
--help	Displays the help message

Clustering

This will run my alternative to Estienne's correlation matrices. This approach is based on clustering by dimensionality reduction using three different methods: PCA, tSNE and UMAP. I will try to keep the statistics as little as possible while explaining.

Dimensionality reduction is, as the name suggests, a (group of) method(s) that try to visualise a high dimensional dataset in two or three dimensions based on the observations in the dimensions to reduce. So in the context of the TSV-file containing the IRS values, every individual IES is an observation and the different KDs are the dimensions of the data-set.

Implemented are principal component assays (PCA), t-distributed neighbour embedding (tSNE) and uniform manifold approximation and projection (UMAP). PCA is a linear dimensionality reduction method, whereas tSNE and UMAP are non-linear, meaning they do not directly use any parameters pertaining to a normal distribution.

A PCA is based on the variance and the covariance in between the dimensions of a data set. In short: eigenvectors with eigenvalues are calculated from the variance/covariance matrix of a dataset. These Eigenvectors are the principal components (PCs) and the eigenvalues tell you how much of the variance of the original dataset they are able to explain. The PCs are ordered by the amount of variance they capture. Most of the time, the first 2 to 3 PCs are looked at to infer similarities in observations or dimensions (groups).

tSNE and UMAP are not based on a direct parameter (variance and covariance are both linear parameters).

Instead of calculating a linear project of the data to find PCs, tSNE is designed to preserve the structure of the data on a local level. tSNE calculates the distance for each data point to all others and uses a t-distribution to group them together. The similarity is calculated in the high dimensional space and then stochastically embedded into the low dimensional space while minimising a target variable (to keep this brief I'll omit what actually happens).

In essence UMAP utilises a similar approach and can be seen as the "successor" to tSNE. Instead of calculating probabilities and checking them under the assumption of a distribution, UMAP constructs graphs in both low and high dimensional spaces while also minimising a certain target variable. The construction of the lowdimensional graph is also of a stochastic nature. You can control how to preserve the structure of the data (local vs. global) by specifying certain parameters.
As tSNE and UMAP contain stochastic steps, results may vary when repeated and not setting a RNG-seed. In contrast PCA will always give you the same results as it is based on calculating linear parameters. But PCA will only give you sensible results when your data is somewhat following a normal distribution. As such tSNE and UMAP will "outperform" PCA in certain situations (like IES clustering). Outperforming in this case means revealing structures in the data.
As mentioned tSNE and UMAP have a bunch of parameters that control how the data is structured in lower dimensions.
Without any additional flags this submodule will calculate a PCA for the specified IRSs on KD level, meaning it will cluster the KDs and not the IESs.
It is able to output 3D plots and animated 3D plots. Meaning if using PCA, it will plot the first 3 dimensions.

Usage:

```
MNLT Clustering -i <input TSV w/ IRSs>
```

All arguments:

Argument	Function
`-i` or `--IRS`	TSV-file containing the IRSs. A up-to-date file is bundled with this program in `MNLT/res/anno/All_IRSs.tsv`
`-o` or `--outPath`	Path to where to save the results. If it does not exist it will be created. (default: current working directory)
`-k` or `--KDs`	Specify the KD and control experiments you are interested in. If multiple, provide them separated by whitespace. The spelling needs to be exactly as in the IRS file header. Be aware that if you're interested in clustering KDs you need a certain amount to have a meaningful result. The more the better. (default: all KDs)
`-f` or `--format`	Specify the format in which to save pictures. Available formats: SVG (default), PNG, JPG
`-D` or `--plot3D`	Set this flag if you want 3-dimensional plot (e.g. first 3 PCs). This will override `-d` or `--dimensions` to 3. (default: False)
`-a` or `--anim`	If this flag is set, you will get an animated gif in addition to the 3-dimensional plot. It will rotate once around each axis. If set it will overwrite/set the following flags: plot3d & dimensions (default: False)
`-m` or `--method`	Specify which method to use. Available methods: PCA (default), tSNE & UMAP
`-c` or `--clusterIES`	If this flag is set, the script will also calculate the specified method on IES level additionally to the KD level, meaning IESs will be clustered (default: False)
`-s` or `--outScree`	If this is set the script will generate so-called scree plots. These are barplots representing how much variance each PC is explaining.

	Obviously, these will only be generated for PCA (default: False)
-O or --outTSV	Set this flag if you wish to have a TSV file containing the coordinates to the scatter plots of the low dimensional data (default: False)
-l or --loadings	If this is set, you can add the eigenvalues of the eigenvectors to the scatter plots. This will tell which dimension (KD) contributes in which direction (PC). Right now, it won't give you much additional information, maybe in future, when more KD data is available (default: False)
-d or --dimensions	Specify the dimensions to represent. Essentially only 2 or 3 will make sense, as you can only sensibly visualise those. PCA will by definition always calculate all PCs, so this will only affect the scree plot and/or the 3D plot (& animation). For tSNE and UMAP this would specify the dimensions in which to represent your original data. As before only 2 or 3 will be sensible. But you can have them also represented in more. You won't get a plot though (default: 2)
-r or --randomState	Specify a seed for tSNE and UMAP. As both methods have a stochastic element in their embedding/projection, setting the same number (INT) will ensure reproducibility. This means that if you redo it 100 times without a seed you may get 100 slightly different plots, but if all of them have a set seed, they are all identical (default: None)
-p or --perplexity	[tSNE] This is one of the parameters that control the function of the algorithm. Perplexity is essentially the minimum number of neighbours to consider to form a group/cluster. This is tightly linked to the variance in the calculated probabilities (t-distribution). A starting point would be a value between 5 and 50 (default: 30)
-n or --neighbours	[UMAP] This is one of the parameters that control the function of the algorithm. This controls how many neighbours to consider to build the graphs for projection. In simple terms it controls where the focus of the low dimensionally represented structures lie, meaning focusing on local data structures (as in tSNE) vs. focusing on global structures. Typically, this is a number between 2 and 100; where a low number focuses on preserving local and a high number focuses on preserving global structures (default: 15)
-M or --min_dist	[UMAP] This is one of the parameters that control the function of the algorithm. This defines the minimum distance between data-points in the low dimensional space. Essentially, this specifies how far apart the data point / clusters are projected into low dimensions. Typically, this is a float between 0.001 and 0.99. This lets you control the 'resolution' if you so will. Here a low number will result in densely packed data points, whereas a high number will result in more spaced placing of data point (default: 0.1)
-h or --help	Displays the help message
-v or --verbose	Will toggle a more verbose terminal output

Tips on usage:

As already mentioned, tSNE and UMAP have a stochastic element and parameters that alter how the reduction takes place. As perplexity and the number of neighbours both directly impact how clusters are formed, you need to take some heed when running a tSNE or UMAP. Clusters may not show you what you want or just show something that is plain wrong. Since you can directly tell the algorithm how far to

space points in the UMAP, also distance between clusters gets meaningless. Both tSNE and UMAP are useful to find clusters, but you cannot infer how they behave to one another. You can however compare clusters to each other with PCA, e.g. if these 5 data points are near those data point, they are more similar than those very far away. This works because PCA is based on actual variance and not distance in between data points. The problem with PCA is that it won't give any useful info if your data is not somewhat normally distributed, which is the case with IRSs (as mentioned before more beta than normal). Also having quite a high number of zeros or missing values really does not help in a parametrical approach.
To make this short:

- PCA is actually comparable between clusters (if you get any)
- tSNE and UMAP are good at finding non-linear structures
- Clusters are not really comparable with each other in tSNE and UMAP

If you are only interested in the clustering of KDs, PCA will work fine. Just be aware that KDs with lots of zeros and NAs will be very near to each other, regardless of actual similarity.
Side note:
If anyone reading this has statistical knowledge and interest in continuing that stuff; I have seen some interesting behaviour in IES clustering. If combined with a classification technique like gaussian mixture models you might be able to classify IESs and retrace steps to find which KD has direct effect on IESs. This could also be used to infer similarities in KDs over individual IES classification.

sRNA_hist

This creates the typical sRNA histograms, as seen in nearly every publication of this lab. It can generate two different plots: simple & complex. Simple in this context means that the plot only shows sRNAs that aligned to the MAC, IESs, other sequences and the rest is labelled as unknown. The complex version is essentially the same only that here the other category is further split into: Vector, OES, Mitochondria, Klebsiella and rRNA.
The only required argument is a directory that contains the uncompressed FastQ files trimmed split and binned by size in the file name.

Usage:

```
MNLT sRNA_hist -d <directory w/ reads>
```

All arguments:

Argument	Function
-d or --inDir	Directory containing sRNA sequencing data. They need to be trimmed and split by size, which has to be reflected in the name e.g. xxx-15bp.fastq (see `--sep`). Fasteris normally supplies them already binned in the inserts directory
-min or --MinSize	Minimum sRNA (insert) size to use (default = 15)
-max or --MaxSize	Maximum sRNA (insert) size to use (default = 35)
-o or --outDir	Name for the directory that is going to be generated. If "." then working directory is used as out directory (default)
-t or --title	Title of the plot. If not set, base filename will be displayed
-p or --plotComp	Whether to have the complex or simple plot, or both. Specify by one of these: "simple", "complex", "both" (default: simple)

-s or --sep	Separator to be used to infer sRNA size by filename. If the file name is like so: xxx-15bp.fastq, 'bp' is the separator used. If it is this: xxx-15nts.fastq, then 'nts' should be specified with this flag. Also you will need to have a '-' in front of the size number
-th or --threads	Specify the number of threads to use while mapping data (default = 10). This may be a bit high for normal machines, you'll have to check how many cores you have. 1 core = 2 threads
-k or keep_tmp	Specify whether to keep tmp directory. Contains intermediate sam files (removed by default)
-h or --help	Displays the help message

MILORD

This is a wrapper for the ParTIES submodule MILORD. MILORD is designed to find excisions in an alignment file (BAM) given a reference genome. For best use you will need to map your reads with bowtie2 using the `--very-sensitive-local` flag. The MILORD module will then extract and realign the unmapped parts of your reads from the BAM file in order to identify deletions. An excision event is called when the newly mapped position is unique and coherent in the context of the initial alignment. The MILORD submodule requires mandatory 3 arguments: `-i` or `--inBam`, the BAM file containing information to your initial alignment; `-o` or `--outDir`, the directory where the output will be written to disk; and `-g` or `--genome`, the genome used as reference in your initial alignment (since you're interested in IESs that would/should have been the MAC+IES genome). The output of MILORD is a new GFF3 file containing information about the excision events, and two log files for debugging.

Usage:

```
MNLT MILORD -i <alignment file in BAM format> -o <output directory> -g
<genome to use as reference>
```

All arguments:

Argument	Function
-i or --inbam	Specify an alignment file in BAM format against the genome you wish to find excision events on. Use bowtie2 with `--very-sensitive-local` flag set
-o or --outDir	Specify the output directory where MILORD will save all generated files. It will create a new directory there called MILORD
-g or --genome	The genome which was used as reference in the original bowtie2 alignment
-m or --minSize	Defines the minimum size for a deletion / excision event to be reported (default = 5)
-M or --maxSize	Defines the maximum size for a deletion / excision event to be reported (default = 10k)
-I or --IES	A custom IES annotation file in GFF3 format can be specified. If not set it will use the standard IES annotations bundled with this program
-f or --flankSeq	MILORD also gives you the context / environment of the excision event. Here you can specify the length of the flanking sequence to be reported (default = 15)

-F or --force	If a MILORD directory already exists, you can force the program to overwrite it
-t or --threads	Specify number of threads to use (default = 6)
-h or --help	Displays the help message

afterMILORD

This submodule processes the MILORD output, more specifically the GFF3 file. It will extract the information from this file and create the typical cryptic and alternative excision plots. If wished it can also output a TSV file, containing the specific locations of the excision events.
It requires one mandatory argument: `-g` or `--GFF`, which specifies the path to the file produced by MILORD.

Usage:

```
MNLT afterMILORD -g <GFF3 output file from MILORD >
```

All arguments:

Argument	Function
-g or --GFF	The path to the GFF3 file generated by MILORD. It contains the information necessary to generate the alternative / cryptic excision plots
-c or --cutOff	Specify cut-off for excision event. Some excision events have weird support coverage. MILORD give you the read coverage for the variant (the excision) and the reference (so no excision). The main problem is that it also reports excision with only 1 read that supports the variant vs 60 read that support the reference. Which may hint at some background level of excision or some other artifact. For that I have implemented a simple cut-off based both support coverages. If $\frac{variance}{variance+reference}$ > cut-off, the excision event is regarded as real. (default = 0.1)
-o or --outPath	Path to where everything is to be saved. If it does not exist it will be created. (default is current working directory)
-f or --format	Specify the format of the picture. Available formats: SVG (default), PNG, JPG
-t or --TSV	If this flag is set you'll get TSV files containing information on locations of all the excision events. You will get 3 separate files for IESs, alternative and cryptic excision
-h or --help	Displays the help message