OncodriveFML Documentation

Release 2.1.2

BBGLab

Mar 08, 2019

OncodriveFML

Contents:

CHAPTER 1

OncodriveFML

Distinguishing the driver mutations from somatic mutations in a tumor genome is one of the major challenges of cancer research. This challenge is more acute and far from solved for non-coding mutations. OncodriveFML is a method designed to analyze the pattern of somatic mutations across tumors in both coding and non-coding genomic regions to identify signals of positive selection, and therefore, their involvement in tumorigenesis. We described the method and illustrated its usefulness to identify protein coding genes, promoters, untranslated regions, intronic splice regions, and lncRNAs-containing driver mutations in several malignancies in Mularoni et al., Genome Biology 2016.

To use OncodriveFML check its website or download the source code from our git repository.

OncodriveFML is a project developed by the Barcelona Biomedical Genomics Lab.

We are a research group integrated in the Institute for Research Biomedicine in Barcelona, which is part of the Barcelona Institute of Science and Technology. Our lab is located at the Barcelona Science Park.

Our main research interest is the computational study of cancer at the genomic level.

Check the README file to find infomation about licensing and installation.

Run the example for a quick check of the installation.

CHAPTER 2

How it works

This section will try to give an overview of how OncodriveFML carries on the analysis.

2.1 The command line interface

By typing `oncodrivefml -h` you will have a brief description of how to use OncodriveFML:

Options:

-i, --input MUTATIONS_FILE Variants file [required] (*see format*)

-e, --elements ELEMENTS_FILE Genomic elements to analyse [required] (*see format*)

-t, --type Type of genomic elements file [required]:

- *coding*: the files corresponds to coding regions
- *noncoding*: the file corresponds to noncoding regions

See *details about the command line interface* to find more information about this option.

-s, --sequencing Type of sequencing [required]:

- *wgs*: whole genome sequencing
- *wes*: whole exome sequencing
- *targeted*: targeted sequencing

See *details about the command line interface* to find more information about this option.

-o, --output OUTPUT_FOLDER Output folder. Default to regions file name without extensions.

-c, --configuration CONFIG_FILE Configuration file. Default to 'oncodrivefml_v2.conf' in the current folder if exists or to ~/.bbglab/oncodrivefml_v2.conf if not.

--samples-blacklist SAMPLES_BLACKLIST Remove these samples when loading the input file.

--no-indels Discard indels in your analysis

--generate-pickle Run OncodriveFML to generate pickle files that could speed up future executions and exit.

--debug Show more progress details

--version Show the version and exit.

-h, --help Show this message and exit.

If you prefer to call OncodriveFML from a Python script, you can download the source code, install it and call the `main()` function.

Note: You might have notice that the `main()` function accepts less parameters than the command line interface. This is because the command line interface modifies some parameters in the configuration, while calling directly the Python code does not. Check *what is modified by the command line interface*.

This implies that you should adapt the *configuration file* to your needs.

2.2 The files

2.2.1 Input files

OncodriveFML makes use of three files:

Variants Also named as input. This file contains the observed mutations for the analysis.

Regions File containing the regions for the analysis. Only mutations that fall in these regions are analysed and only the genomic positions defined in this file are used for the simulation.

You can define your own regions file based on your criteria. You can check an example of a regions file downloading our example.

> **Warning:** It is not recommended to mix coding and non-coding regions in your regions file. In fact this will likely produce artifacts in the results as coding and non-coding regions of the genome have a very different functional impact scores. A good set of genomic regions should include elements that share biological functions (e.g. CDS, UTRs, promoters, enhancers, etc.).

Check the *formats for the input files*.

Configuration The configuration file is also a key part of the run, and understanding how to adapt it to your needs is important. Check *this section* to find more details about it.

2.2.2 Output files

Find information about the output *output files* section.

2.3 Workflow

1. The first thing that is done by OncodriveFML is to load the configuration file and to create the output folder if it does not exist.

   Note: If you have not provided any output folder, OncodriveFML will create one in the current directory with the same name as the elements file (without extension).

   If the output folder exits, OncodriveFML checks whether a file with the expected output name exits and, if so, it does not run.

2. The regions file is loaded, and a tree with the intervals is created. This tree is used to find which mutations fall in the regions being analysed.
3. Loads the mutations file and keeps only the ones that fall into the regions being analysed.
4. Computes the signature (see the *signature* section).
5. Analyses each region separately (only the ones that have mutations). In each region the analysis is as follow:
   1. Computes the score of each of the observed mutations.
   2. Simulates the same number of mutations in the segments of the region under analysis. Save the scores of each of the simulated mutations. The simulation is done several times.
   3. Applies a predefined function to the observed scores and to each of the simulated groups of scores. Counts how many times the simulated value is higher than, or equal to, the observed.
   4. From these counts, computes a P-value by dividing the counts by the number of simulations performed.

   You can find more details in the *analysis section*.
6. Joins the results and performs a multiple test correction. The multiple test correction is only done for regions with mutations from at least two samples.
7. Creates the *output files*.
8. Checks that the output file does not contain missing or repeated genomic regions.

CHAPTER 3

Files

3.1 File formats

Note: All the files can be compressed using GZIP (extension ".gz"), BZIP2 (extension ".bz2") or LZMA (extension ".xz")

3.1.1 Input file format

The variants file is a text file with, at least, 5 columns separated by a tab character (the header is required, but the order of the columns can change):

- Column CHROMOSOME: Chromosome. A number between 1 and 22 or the letter X or Y (upper case)
- Column POSITION: Mutation position. A positive integer.
- Column REF: Reference allele[1].
- Column ALT: Alternate allele[1].
- Column SAMPLE: Sample identifier. Any alphanumeric string.
- Column CANCER_TYPE: Cancer type. Any alphanumeric string. Optional.
- Column SIGNATURE: User defined signature categories. Any alphanumeric string. Optional.

3.1.2 Regions file format

The regions file is a text file with, at least, 4 columns separated by a tab character (the column order must be preserved):

- Column 1 [CHROMOSOME]: Chromosome. A number between 1 and 22 or the letter X or Y (upper case)

[1] The alleles consist on a single letter or a set of letters using A, C, G or T (upper case). Single Nucleotide Variants are indentified because both, REF and ALT contain only one letter. In Multi-Nucleotide Variants REF and ALT columns contain a set of letters of the same length. Insertions use – in the REF and a set of letters as ALT while deletions contain the set of deleted characters in the REF and – in the ALT columns.

- Column 2 [START]: Start position. A positive integer.
- Column 3 [STOP]: End position. A positive integer.
- Column 4 [STRAND]: Strand: + for positive, − for negative, . for unknown.
- Column 5 [ELEMENT]: Element identifier.
- Column 6 [SEGMENT]: Segment identifier. Optional column.
- Column 7 [SYMBOL]: Symbol, a different identifier for the element that will also be printed in the output file. Optional column.

3.1.3 Output file format

OncodriveFML generates a tabulated file with the results with the extension ".tsv".

Check the *output section* to find a detailed description regarding the output.

CHAPTER 4

Configuration

The method behaviour can be modified through a configuration file.

Warning: Using the command line interface overwrites some setting in the configuration file. Check how the command line interface changes the configuration in the *command line interface* section.

Check the `oncodrivefml_v2.conf.template` that is included in the package to find an example of the configuration file.

This section will explain each of the parameters in the configuration file:

4.1 Genome

```
[genome]
# Build of the reference genome
# Currently human genomes supported: hg19, hg38 and hg18
# mouse genomes: c3h and mm10
```

The genome section makes reference to the reference genome used by OncodriveFML.

The reference genome has been obtained from http://hgdownload.cse.ucsc.edu/downloads.html.

Currently, only `HG19` is fully supported. Use `build = 'hg19'` to use it.

There is a partial support for `HG18` and `HG38`. The support is only partial because the values for the position and alterations of the stops in the these genomes have not been computed yet. If you want to run OncodriveFML with any of these genomes, make sure you do not use the `stop` method for the indels (*ref*).

Warning: If you decide to use a reference genome other than `HG19`, make sure that the scores file you use is compatible with it.

4.2 Signature

```
# "full" : Use a 192 matrix with all the possible signatures



# Choose the classifier (categorical value for the signature:
# The classifier is a column in the dataset and must be one of these:
# classifier = 'SIGNATURE'
# classifier = 'SAMPLE'
# classifier = 'CANCER_TYPE'


# Include/exclude MNP mutations in the signature computation


# Choose if the signature must be computed using the whole cohort or
# only the elements that fall into the regions you are analysing:



# None: do not correct (comment the option)
```

The signature represents the probability of a certain nucleotide to mutate taking into account its context[1].

You can choose one of the following options for the signature:

- To not use any signature, which is equivalent to assume that all changes have equal probability to happen: `method = 'none'`. This approach is recommended for small datasets.
- OncodriveFML can also compute the signatures using the provided dataset. This option contains a set of parameters that you can use to decide how this computation is done.
  - Select one of the methods to compute the signatures from the dataset: `method = 'full'` to count each mutation once and `method = 'complement'` to collapse complementary mutations.

    Note: The option `method = 'bysample'` is equivalent to `method = 'complement'` but forces the classifier (see below) to be `SAMPLE`.

  - The classifier parameter indicates which column from the mutations file is used to group the mutations when computing the signatures. E.g. grouping by `SAMPLE` generates one signature for each sample. Only `SAMPLE`, `CANCER_TYPE` and `SIGNATURE` columns can be used.
  - You can decide to use only SNP (`include_mnp = False`) or also use MNP mutations (`include_mnp = True`).
  - You can choose between using only the mutations that are mapped to the regions under analysis (`only_mapped_mutations = True`) or use all the mutations (SNPs and optionally MNPs) in the dataset.
  - The signatures can be corrected by the frequencies of sites. If you do not specify anything, OncodriveFML will not correct the signatures. Use `normalize_by_sites = 'whole_genome'` or `'wgs'` to correct by the frequencies in the whole genome. Use `normalize_by_sites = 'whole_exome'` or `'wes'` or `'wxs'` to correct by the frequencies in the exome. If you have specified `only_mapped_mutations = True`, then the correction will be done by the frequencies of

[1] Previous and posterior nucleotides

trinuceotides found in the regions under analysis, as long as you indicate one of the above mentioned values.

Note: The frequencies have been computed for genome build `HG19`. If you want to check the values, use the *bgdata package*.

- The recommended approach is to use your own signatures. OncodriveFML has the option `method = 'file'` to load precomputed signatures from a file. This option requires a few additional parameters:
  - `path`: path to the file containing the signature
  - `colum_ref`: column that contains the reference triplet
  - `column_alt`: column that contains the alternate triplet
  - `column_probability`: column that contains the probability

Warning: Probabilities must sum to one.

4.3 Score

The score section is used to know which scores are going to be used.

```
[score]
# Path to score file

# 'pack': binary format

# Column that has the chromosome

# If the chromosome has a prefix like 'chr'. Example: chrX chr1 ...

# Column that has the position

# Column that has the reference allele

# Column that has the alternative allele

# Column that has the score value
# element = 6

# Minimum number of stops per element to infer a for the stops using the mean of all␣
↪scores

# Function to infer the value of the stops in an element using the mean (x is the␣
↪mean value of the scores)
```

The scores should be a file that for a given position, in a given chromosome, gives a value to every possible alteration.

Some of the parameters in this section are optional, while others are mandatory.

- `file` is a string and represents the path to the scores file.
- `format = 'tabix'` indicates that the file is a tab separated file compressed with bgzip. This means that a .tbi index file should be present in the same location. The other option currently supported is `format =`

'pack' which is a binary format we have implemented to reduce the file size. Thus, if you want to use your own file, use the tabix format.

- chr column in the file where the chromosome is indicated.
- chr_prefix: when querying the tabix file for a specif chromosome OncodriveFML only uses the number of the chromosome or 'X' or 'Y'. If the tabix file requires a prefix before the chromosome, use this option. For instance, if the chromosomes in the tabix file are labeled as chr1, chr2, .., chrY, set this option to: chr_prefix = 'chr'. If this is not the case, use an empty string: chr_prefix = ''.
- pos column that indicates the position of the scored alteration in the chromosome.
- ref column that contains the reference allele. It is optional.
- alt column that contains the alternate allele. It is optional. If is not specified, it is assumed that the 3 possible changes have the same score.
- score column that contains the score.
- element column that contains the element identifier. It is optional. If it is provided and the value does not match with the one from the regions, these scores are discarded.

OncodriveFML uses two additional parameters, which are related only to the stop method for *computing the indels*.

- When analysing a certain gene, OncodriveFML might need to score an indel according to the value of the stops in the gene. It might happen that the number of stops is 0 or is below a certain threshold. In such cases, OncodriveFML uses the function specified in this parameter to assign a score from the mean value of all the stops in the gene.
- When analysing a certain gene, OncodriveFML gets all the scores associated with the mutations that produce a stop in that gene. minimum_number_of_stops indicates the minimum number of stops that a gene is required to have in order to avoid using the function above.

4.4 Statistic

The statistic section is related to the configuration of the analysis

```
# Arithmetic mean


# Do not use/use MNP mutations in the analysis


# Minimum sampling

# Maximum sampling

# Sampling chunk (in millions)

# Minimum number of observed (if not reached, keeps computing)
```

There a different parameters you can configure:

- method represents the type of operation that is applied to observed and simulated scores before comparing them. The arithmetic mean (method = 'amean') and the geometric mean (method = 'gmean') are supported. The recommended one is the arithmetic mean.
- In some cases, you might be interested in performing the analysis per sample. This means that all the mutations that come from the same sample are reduced to a single score. This score can be the maximum (per_sample_analysis = 'max'), the arithmetic mean (per_sample_analysis = 'amean') or

the geometric mean (`per_sample_analysis = 'gmean'`) of all the mutation's scores that come from the sample sample. Comment this option if you are not interested in this type of analysis.

- MNP mutations can optionally be included in the analysis. Use `discard_mnp = False` to include them and `discard_mnp = True` to discard them.

OncodriveFML includes a few more parameters that are related to how many simulations are performed.

- `sampling` represents the minimum number of simulations to be performed.
- `sampling_max` represents the maximum number of simulations to be performed.
- `sampling_chunk` represents the maximum size (in millions) that a single process can handle. This value is used to keep the memory usage within certain limits.

Note: With a value of 100, each process takes less than 4 GB of RAM. We have not considered the memory taken by the main process.

- `sampling_min_obs` represents the minimum number of observations[2]. When it is reached, no more simulations are performed.

4.4.1 Indels

The indels subsection of statistic contains the configuration for the analysis of indels.

```
[[indels]]
# Include/exclude indels from your analysis



# Treat them as a set of substitutions and take the maximum


# Number of consecutive times the indel appears to consider it falls in a repetitive␣
↪region
# Looking from the indel position and in the direction of the strand


# Indels simulated as substitutions take into account signature or not


# Use exomic probabilities of frameshift indels in the dataset for the simulation



# Arithmetic mean
```

OncodriveFML accepts various parameters related to the indels:

- The main option is `include`, which indicates whether to include indels in the anlysis or not. Use `include = True` to include indels and `include = False` to exclude them.
- OncodriveFML can simulate indels in two ways. `method = 'max'` simulates indels as a set of substitutions. `method = 'stop'` simulates indels as stops. This option is recommended for simulating indels in coding regions. Check the *analysis of indels* section to find more details.

[2] An observation is counted when a simulated value, after applying the function in `method` to the simulated scores, is higher than the result of applying the same function to the observed scores.

- OncodriveFML discards indels that fall in repetitive regions. OncodriveFML considers that an indel is in a repetitive region when the same sequence of the indel appears consecutively in a genomic element a certain number of times (or even more) following the direction of the strand. The maximum number of consecutve repetitions can be set with the `max_consecutive` option. OncodriveFML will not discard any indel due to repetitive regions if you set `max_consecutive = 0`.
- Indels that are simulated as substitutions[3] can be simulated assigning to all the positions of the genomic element under analysis the same probability to be mutated. Alternatively the probability of each position to be mutated can depend on the mutational signature. For instance if the signature is represented by the cancer type, indels coming from a breast cancer dataset will be simulated with the signature of that cancer type. Indels do not contribute to the signature of a cancer type, therefore through this option you can decide whether indels should be simulated following the mutational signature or not. Use `simulate_with_signature = True` to use the signature or `simulate_with_signature = False` to simulate indels with the same probabilities.
- `gene_exomic_frameshift_ratio` is a flag that indicates OncodriveFML which mutations influence the *probabilities* for frameshift indels and substitutions. When `gene_exomic_frameshift_ratio = False` the probabilities are taken from the mapped mutations discarding those whose length is multiple of 3. Note that in order to work properly, this option should be set when the regions file corresponds to coding regions. If `gene_exomic_frameshift_ratio = True`, the probabilities are taken from the observed mutations rate in each region. This option is harmless when `method = 'max'`.
- The *observed* score of an indel that is computed with the `method = 'stop'` option is related to the score of the stops in its gene. You can decide how this relation is by choosing a function that is applied to all stops scores in the gene. E.g. `stops_function = 'mean'` associates the indel to a value that is equal to the mean of all stop scores in the gene. The options you can choose are: - `'mean'` for arithmetic mean - `'median'` for the median - `'random'` for a random value between the maximum and the minimum - `'random_choice'` for choosing a random value between all the possible ones

4.5 Settings

To configure the system where the analysis is performed OncodriveFML includes the setting section:

```
[settings]
# Number of cores to use in the analysis
# Comment this option to use all avaliable cores
```

Use the `cores` option to indicate how many cores to use. You can comment this option in order to use all the available cores.

Note: OncodriveFML works on shared memory systems using the `multiprocessing` module.

4.6 Logging

The logging section is used to configure the logging system of OncodriveFML.

OncodriveFML uses the `logging` module. In particular it loads the configuration file into a dictionary and passes this section to `dictConfig()`.

You can change this section to other compatible configurations to fit your needs.

[3] All indels are simulated as substitutions when `method = 'max'`. Indels that are in-frame are also simulated as substitutions when `method = 'stop'`.

All the logs are done using a logger named `oncodrivefml`. The logging system can be configured through the logging section of the *configuration file*.

Warning: If OncodriveFML detects that the run has already been calculated, the warning informing the user uses the root logger.

OncodriveFML does override the configuration in two ways:

- If the `debug` flag is set, the console logger level is set to `DEBUG`. Otherwise, it is set to `INFO`.
- If one of the handlers is named `file`, its filename is set to `<mutations file name>__log.txt` and saved in the same folder as the OncodriveFML output.

CHAPTER 5

Analysis

This sections explains how OncodriveFML compute the scores for the observed mutations and how mutations are simulated.

The analysis is done for each element independently. The same number of observed mutations is simulated within the element, taking only the positions indicated in the regions file.

5.1 Observed

5.1.1 Single Nucleotide Polymorphism (SNP)

SNP mutations are the simplest to compute. To score them, OncodriveFML get the score for the corresponding alteration in the position of the mutation.

If there is not a score for that particular change, the mutation is ignored[1].

5.1.2 Multi Nucleotide Polymorphism (MNP)

MNP mutations are considered as set of SNPs. The observed value is the maximum value of all the changes produced by the MNP.

MNPs are ignored[1] when none of the changes it introduces has a score.

5.1.3 Insertion or deletion (INDEL)

Indels are scored in two different ways: as stops or as substitutions.

[1] When an observed mutation is ignored it means that it cannot be assigned a score, and thus it does not contribute to the observed scores and in the simulation the number of mutations simulated is one less for that region.

As stops Indels are scored as stops in the analysis of coding regions and if their length is not a multiple of 3. In coding regions, a frameshift indel might cause, somewhere in the gene, a stop. This is why OncodriveFML uses this approach. The way OncodriveFML scores this type of indels is taking all the stop scores[2] in the gene under analysis and applying a user defined function to them. In some cases, OncodriveFML can infer a value for the scores of the stops using the mean score of all mutations in the gene. See the *configuration of indel* section for further information.

As substitutions Indels that fall in non-coding regions or in-frame indels in coding regions are considered as a set of substitutions. Similarly to MNP mutations, the changes produced by the indel are computed as a set of SNPs mutation and OncodriveFML assigns the indel the maximum score of those changes. In an insertion, the reference genome is compared with the indel in the direction of the strand. In a deletion, the reference genome is compared with itself but shifted a number of position equal to the length of the indel in the direction of the stand. Only the changes produced in the length of the indel are considered.

Note: If none of the changes produced by the indel has a score, the indel is ignored[1].

Indels with a length higher than 20 nucleotides are ignored[1].

5.2 Simulated

The same number of mutations that are observed and have a score are simulated.

To perform the simulation two arrays are computed:

- One contains the scores of all possible single nucleotide substitutions that can occur within the segments of the element under analysis. Additionally, this array also contains the values of the stops in this region.
- The other array contains the probabilities of each of those changes.

Using the probability array, a random sampling of the scores array is done to obtain the simulated scores.

5.2.1 Probabilities

The probability array is computed taking into account different parameters.

The probability associated to any of the stop scores is:

$$p = \frac{1}{n_{stops}} * p_{frameshiftindel}$$

where $p_{frameshiftindel} + p_{subs} = 1$, and n_{stops} is the number of stop scores for that gene.

$p_{frameshiftindel}$ represents the probability of simulating a frameshift indel in that gene, and p_{subs} represents the probability of simulating a substitution.

- If the analysis type is selected to be `max` (see *configuration*) $p_{frameshiftindel} = 0$.
- If the analysis type is `stop` (see *configuration*), OncodriveFML assumes you are analysing coding regions. For coding regions, the probability of simulating a frameshift indel depends on amonwhether you are analysing using the whole cohort percentages or only the mutations observed in each gene.
  - When using *exomic frameshift probabilities* OncodriveFML computes how many indels you observe, and how many of those fall into the region you are analysing (which should be coding). Among the mapped

[2] The package BgData includes the precomputed position and alteration of the stops for the HG19 genome build. OncodriveFML makes use of it.

indels OncodriveFML distinguishes between frameshift and in-frame indels. The ratio of frameshift indels against the total amount of mutations is used to compute $p_{frameshiftindel}$.

- When using the probabilities taken from the gene:

$$p_{frameshiftindel} = \frac{n_{observedframeshiftindels}}{n_{observedmutations}}$$

where $n_{observedframeshiftindels}$ is the number of observed frameshift indels and $n_{observedmutations}$ is the number of observed mutations.

The probabilities associated with the substitutions are:

$$p = p_{subs} * \frac{\sum_s p_s * f_s}{n_{substitutions}}$$

where `s` represents each of the signatures found in the gene in the observed mutations, p_s is the probability of a particular mutation to occur given the `s` signature, $n_{substitutions}$ is the total number of substitutions, and f_s is the relative frequency of a particular signature `s` in the gene.

However, if you are not using any signature (see *singature configuration*):

$$p = p_{subs}/n_{substitutions}$$

where $n_{substitutions}$ is the amount of substitutions in the gene.

CHAPTER 6

Signature

The signature is an array that assigns a probability to a single nucleotide mutation taking into account its context[1]. It represents the chance of a certain mutation to occur within a context.

Check the different options for the signature in the *configuration file*. In short, you can choose between not using any signature, using your own signature or computing the signature from the mutations file. Additionally, signatures can be grouped into different categories (such as the sample).

The signature array is computed by counting, for each Single Nucleotide Polymorphism, the reference and alternated triplets.

Note: OncodriveFML also uses the MNP mutations to compute the signature, by treating them as a set of separate SNPs. You can enable or disable this behaviour with the `include_mnp` option in the *configuration file*.

The counts are then divided by the total number of counts to generate a frequency of triplets. For a mutation i the frequency is $f_i = \frac{m_i}{M}$ where $M = \sum_j m_j$, and m_i represent the number of times that the mutation i with its context[1] has been observed.

Optionally, the signature can be corrected taking into account the frequency of trinucleotides in the reference genome. OncodriveFML introduces this feature because the distribution of triplets is not expected to be constant. When using the command line interface, OncodriveFML does this correction automatically according to the value passed in the flag `--sequencing` (you can list all the options *using the help*).

6.1 Reasoning behind the correction

Let's first take the conditional probability of a mutation (with contectx[1]) to occur given the number of those triplets in the region: $p_i = p(m = i|T_i) = \frac{m_i}{T_i}$.

Then, the normalized frequency of the mutation i is: $\overline{f_i} = \frac{m_i/T_i}{\sum_j m_j/T_j}$.

[1] The context is formed by the previous and posterior nucleotides.

The results can be adapted in case our inputs are not absolute values but relative frequencies. f_i is the frequency of mutations and t_i the frequency of nucleotides:

$$f_i = \frac{m_i}{\sum_j m_j}; t_i = \frac{T_i}{\sum_j T_j} \simeq \frac{T_i}{N}$$

(N is the number of nucleotides, $\sum_j T_j = N - 2 \cdot s$, where s is the number of segments)

Then:

$$\overline{f_i} = \frac{f_i/t_i}{\sum_j f_j/t_j}$$

Proof:

$$\frac{f_i/t_i}{\sum_j f_j/t_j} = \frac{\frac{\frac{m_i}{\sum_j m_j}}{\frac{T_i}{\sum_j T_j}}}{\sum_k \frac{\frac{m_k}{\sum_j m_j}}{\frac{T_k}{\sum_j T_j}}} = \frac{\frac{m_i}{T_i} \cdot \frac{\sum_j T_j}{\sum_j m_j}}{\sum_k (\frac{m_k}{T_k} \cdot \frac{\sum_j T_j}{\sum_j m_j})} = \frac{\frac{m_i}{T_i} \cdot \frac{\sum_j T_j}{\sum_j m_j}}{\frac{\sum_j T_j}{\sum_j m_j} \cdot \sum_k \frac{m_k}{T_k}} = \frac{m_i/T_i}{\sum_k m_k/T_k}$$

CHAPTER 7

Output

OncodriveFML generates 3 output files:

- A `.tsv` with the analysis results
- A `.png` image with the most significant genes labeled.
- A `.html` interactive plot which can be used to search for specific genes.

7.1 Naming

All the 3 files generated by OncodriveFML have the same name. They only differ in the extension. The name given to the files is the same as the name of the mutations file followed by `-oncodrivefml` and the extension.

7.2 The `.tsv` file

This tabulated file is the most important (as the others are just plots using the data in this one) and contains the results of the analysis.

In the file, the following columns can be found:

index Gene ID from Ensembl

MUTS number of mutations found in the dataset for that gene

MUTS_RECURRENCE number of mutations that do not occur in the same position

SAMPLES number of mutated samples in the gene

P_VALUE times that the observed value is higher than or equal to the expected value, divided by the number of randomizations

Q_VALUE `pvalue` corrected using the Benjamini/Hochberg correction (for samples with at least 2 `samples_mut`)

P_VALUE_NEG times that the observed value is lower than or equal to the expected value, divided by the number of randomizations

Q_VALUE_NEG `pvalue_neg` corrected using the Benjamini/Hochberg correction (for samples with at least 2 `samples_mut`)

SNP number of mutations that are Single Nucleotide Polymorphisms

MNP number of mutations that are Multi Nucleotide Polymorphisms (two or more)

INDELS number of mutations that are insertions or deletions

SYMBOL HGNC Symbol

7.3 The plots

Both plots (`.png` and `.html`) represent the same. They are similar to Q-Q plots where in the Y axis the $-log10$ of the computed P-values are represented (sorted) and in the X axis the $-log10$ of the expected P-values are reported (sorted).

The expected P-values represent the null distribution: $-log10(i/N)$ where $i \in [1, N]$ and `N` represents the number of computed P-values.

Note: The P-values of OncodriveFML are always > 0, even when all the simulated functional impact scores are lower than the observed functional impact score. In this case, a pseudocount is added.

The genomic elements that have a lighter color in the plot are the ones for which the number of mutated sample does not reach the minimum required to perform the multiple test correction.

All the genomic regions above the red line in the plot represent those with a Q-value below 0.1. The ones between the green line and the red line are the ones with a Q-value between 0.25 and 0.1.

CHAPTER 8

Behind the scenes

This section will point out some parts which might be interesting if you are running OncodriveFML yourself.

8.1 Command line interface

The command line interface of OncodriveFML overwrites some of the parameters in the configuration file.

> **Warning:** This overwrite is performed regardless the parameter is set or not in the configuration file.

The *following table* shows the modifications introduced in the *indels configuration parameters* by the **`--type`** flag:

Table 1: Effects of –type

Value	Effect in configurtion of indels
coding	`method = 'stop'`
noncoding	`method = 'max'`

The flag **`--no-indels`** also affects the *indels configuration parameters*. Particularly, it has effect on the `include` option. The use of this flag discards the analysis of indels by setting `include = False`, while not using it includes indels (`include = True`).

The *table below* shows the effects of the **`--sequencing`** flag in the *signature configuration*:

Table 2: Effects of –sequencing

Value	Effect in signature
wgs	`normalize_by_sites = 'whole_genome'`
wes	`normalize_by_sites = 'whole_exome'`
targeted	`normalize_by_sites = None`

Finally, the use of the **`--debug`** flag sets the level of the *console* handler in the *logging section* to `'DEBUG'`.

8.2 Pickle files

OncodriveFML can create and use intermediate files to speed up computations that use the same files.

The regions file is loaded using the BgParsers library, so the cache of that file is out of the scope of OncodriveFML. In short, the file will be cached the first time you use it and rebuild if you change its name or content.

There are 2 other items for which OncodriveFML can create or use a cache-like files to speed up future executions. Those files are saved in (or loaded from) the same folder as the mutations file. However, the systems is not as sophisticated as the BgParsers and may lead to few issues. To generate these cache-like files you need to run OncodriveFML with the **`--generate-pickle`** option (you can list all the options *using the help*).

Warning: Using this option can speed up computations as some steps can be replaced by a single file read. However, changes in the input files are not noticed by these pickle files unless you rename them. Thus we recommend its use only to advanced users that understand the process.

8.2.1 Mutations

One of the pickle files that can be created contains a dictionary with the mutations mapped to the genomic elements being analysed and some other useful metadata (such as the number of indels or SNP mutations). This file, named `<mutations file>+__mapping__+<elements file>`, is helpful to skip the steps of loading and mapping mutations. If this file is in the same location as the mutations file, OncodriveFML loads it as long as it does not receive any file with blacklisted samples.

8.2.2 Signature

The other pickle file created is the signature pickle. It is only created for signature methods: `full` and `complement` Its name is: `<mutations file>+_signature_+<method>+_+<classifier>`. See *signature configuration* for more details (methods, classifiers, etc.) about the signature.

If this file is located in the same directory as the mutations file, OncodriveFML loads it as long as it does not receive any file with blacklisted samples and the `only_mapped_mutations` option is not used (see *signature configuration*).

8.3 BgData

OncodriveFML uses external data retrieved using the BgData package. You can download and check this data yourself. If you want to use different data, you can download the source code and modify the code to use your own data.

8.3.1 Reference genome

As March 2017 BgData includes three reference genomes: *HG18*, *HG19* and *HG38*.

```
bgdata datasets genomereference hg19
```

If you want to use a different genome, you need to modify the code in the *oncodrivefml.signature* module.

8.3.2 Signature correction

BgData includes the counts of the triplets in whole exome and whole genome.

```
bgdata datasets exomesignature hg19

bgdata datasets genomesignature hg19
```

Those counts are used to compute the trinucleotides frequencies and to perform signature correction (find more details in the *signature* section and in the *signature configuration*).

8.3.3 Gene stops

OncodriveFML also uses a tabix file that contains the positions and the alterations of the gene stops.

```
bgdata datasets genestops hg19
```

CHAPTER 9

Caveats

MNP mutations contribute to the signatures as a set of independent SNPs mutations. This means that the calculation of the signatures is made with a higher number of mutations compared to the observed substitutions being analysed because OncodriveFML simulates a MNP mutation as a single SNP mutation.

If the scores files lacks scores for some positions or certain alterations, OncodriveFML ignores them.

If, for any reason, your signatures lack certain triplets (probability equal to 0) that are the only ones present in certain region, OncodriveFML will not compute a P-value for that region.

OncodriveFML statistical power is limited by the number of simulations performed in each regions. You can increase the number of simulations, but be aware that the time cost is exponential.

Indels do not contribute to the signatures. You can simulate indels as substitutions and perform the simulations taking the signatures into account, but be aware that the signatures are not calculated considering indels.

On the other hand, if you choose to not use the signatures with the indels, their probability is the inverse of the number of distinct trinucleotides for all the regions multiplied by three (there are 3 possible changes). Typically the value should be 1/192, and that is the default value OncodriveFML uses. If OncodriveFML corrects the signatures, it obtains the number of distinct triplets from the correction.

Depending on the values of `sampling_min_obs` and `sampling_chunk` in the configuration file the number of simulations performed for a particular genomic element can differ.

CHAPTER 10

oncodrivefml

10.1 oncodrivefml package

10.1.1 Submodules

10.1.2 oncodrivefml.compute module

oncodrivefml.compute.**gmean**(*a*)

oncodrivefml.compute.**gmean_weighted**(*vectors*, *weights*)

oncodrivefml.compute.**random_scores**(*num_samples*, *sampling_size*, *background*, *signature*, *statistic_name*)

10.1.3 oncodrivefml.config module

This module contains code related with the configuration file (see *Configuration*).

Additionally, it includes other file realted code, specially from `bgconfig`.

oncodrivefml.config.**load_configuration**(*config_file*, *override=None*)
Load the configuration file and checks the format.

Parameters **config_file** – configuration file path

Returns configuration as a `dict`

Return type `bgconfig.BGConfig`

oncodrivefml.config.**possible_extensions = ['.gz', '.xz', '.bz2', '.tsv', '.txt']**
Some expected extensions

oncodrivefml.config.**remove_extension_and_replace_special_characters**(*file_path*)
Modifies the name of a file by removing any extension in *possible_extensions* and replacing any character in *special_characters* for −.

Parameters **file_path** – path to a file

Returns file name modified

Return type str

oncodrivefml.config.**special_characters = ['.', '_']**
Some special characters

10.1.4 oncodrivefml.indels module

This module contains all utilities to process insertions and deletions.

Currently 3 methods have been implemented to compute the impact of the indels.

1. As a set of substitutions ('max'):

   The indel is treated as set of substitutions. It is used for non-coding regions

   The functional impact of the observed mutation is the maximum of all the substitutions. The background is simulated as substitutions are.

2. As a stop ('stop'):

   The indel is expected to produce a stop in the genome, unless it is a frame-shift indel. It is used for coding regions.

   The functional impact is derived from the function impact of the stops of the gene. The background is simulated also as stops.

class oncodrivefml.indels.**Indel**(*scores*, *strand*)
Bases: object

Methods to compute the impact of indels for the observed and the background

Parameters

- **scores** (*Scores*) – functional impact per position
- **signature** (*dict*) – see *signature*
- **signature_id** (*str*) – classifier for the signatures
- **method** (*str*) – identifies which method to use to compute the functional impact (see *methods*)
- **strand** (*str*) – if the element being analysed has positive, negative or unknown strand (+,-,.)

compute_scores(*reference*, *alternation*, *initial_position*, *size*)
Compute the scores of all substitution between the reference and altered sequences

Parameters

- **reference** (*str*) – sequence
- **alternation** (*str*) – sequence
- **initial_position** (*int*) – position where the indel occurs
- **size** (*int*) – number of position to look

Returns Scores of the substitution in the indel. nan when it is not possible to compute a value.

Return type list

get_background_indel_scores_as_stops()

Returns Values of the stop scores of the gene

Return type list

get_background_indel_scores_as_substitutions_without_signature()
Return the values of scores of all possible substitutions :returns: list.

get_indel_score_from_stop(*mutation*)
Compute the indel score as a stop

A function is applied to the values of the scores in the gene

Parameters **mutation** (*dict*) – a mutation object as in *here*

Returns Score value. nan if is not possible to compute it

Return type float

get_indel_score_max_of_subs(*mutation*)
Compute the score of an indel by treating each alteration as a substitution.

Parameters **mutation** (*dict*) – a mutation object as in *here*

Returns Maximum value of all substitutions

Return type float

get_mutation_sequences(*mutation*, *size*)
Get the reference and altered sequence of the indel along the window size

Parameters

- **mutation** (*dict*) – a mutation object as in *here*
- **size** (*int*) – window length

Returns Reference and alternated sequences

Return type tuple

static is_frameshift(*size*)

Parameters **size** (*int*) – length of the indel

Returns bool. Whether the size is multiple of 3 (in the frames have been enabled in the configuration)

is_in_repetitive_region(*mutation*)
Check if an indel falls in a repetitive region

Looking in the window with the indel in the middle, check if the same sequence of the indel appears at least a certain number of times specified in the configuration. The window where to look has twice the size of the indel multiplied by the number of times already mentioned.

Parameters **mutation** (*dict*) – a mutation object as in *here*

Returns Whether the indel falls in a repetitive region or not

Return type bool

not_found(*mutation*)

class oncodrivefml.indels.**StopsScore**(*funct_type*)
Bases: object

choose(*x*)

function(*x*)

mean(*x*)

median(*x*)

random(*x*)

oncodrivefml.indels.**init_indels_module**(*indels_config*)
Initialize the indels module

Parameters **indels_config** (*dict*) – configuration of how to compute the impact of indels

10.1.5 oncodrivefml.load module

This module contains the methods used to load and parse the input files: elements and mutations

elements (**dict**) contains all the segments related to one element. The information is taken from the elements_file. Basic structure:

```
{ element_id:
    [
        {
        'CHROMOSOME': chromosome,
        'START': start_position_of_the_segment,
        'STOP': end_position_of_the_segment,
        'STRAND': strand (+ -> positive | - -> negative)
        'ELEMENT': element_id,
        'SEGMENT': segment_id,
        'SYMBOL': symbol_id
        }
    ]
}
```

mutations (**dict**) contains all the mutations for each element. Most of the information is taken from the mutations_file but the *element_id* and the *segment* that are taken from the **elements**. More information is added during the execution. Basic structure:

```
{ element_id:
    [
        {
        'CHROMOSOME': chromosome,
        'POSITION': position_where_the_mutation_occurs,
        'REF': reference_sequence,
        'ALT': alteration_sequence,
        'SAMPLE': sample_id,
        'ALT_TYPE': type_of_the_mutation,
        'CANCER_TYPE': group to which the mutation belongs to,
        'SIGNATURE': a different grouping category,
        }
    ]
}
```

mutations_data (**dict**) contains the *mutations dict* and some metadata information about the mutations. Currently, the number of substitutions and indels. Basic structure:

```
{
    'data':
        {
            `mutations dict`_
```

(continues on next page)

(continued from previous page)

```
        },
    'metadata':
        {
            'snp': amount of SNP mutations
            'mnp': amount of MNP mutations
            'mnp_length': total length of the MNP mutations
            'indel': amount of indels
        }
}
```

oncodrivefml.load.**build_regions_tree**(*regions*)

Generates a binary tree with the intervals of the regions

Parameters **regions** (*dict*) – segments grouped by *elements*.

Returns

for each chromosome, it get one `IntervalTree` which is a binary tree. The leafs are intervals [low limit, high limit) and the value associated with each interval is the `tuple` (element, segment). It can be interpreted as:

```
{ chromosome:
    (start_position, stop_position +1): (element, segment)
}
```

Return type dict of `IntervalTree`

oncodrivefml.load.**load_and_map_variants**(*variants_file*, *elements_file*, *blacklist=None*, *save_pickle=False*)

From the elements and variants file, get dictionaries with the segments grouped by element ID and the mutations grouped in the same way, as well as some information related to the mutations.

Parameters

- **variants_file** – mutations file (see `OncodriveFML`)
- **elements_file** – elements file (see `OncodriveFML`)
- **blacklist** (*optional*) – file with blacklisted samples (see `OncodriveFML`). Defaults to None. If the blacklist option is passed, the mutations are not loaded from a pickle file.
- **save_pickle** (`bool`, optional) – save pickle files

Returns

mutations and elements

Elements: *elements dict*

Mutations: *mutations data dict*

Return type tuple

The process is done in 3 steps:

1. `load_regions()`
2. *`build_regions_tree()`*.
3. each mutation (*`load_mutations()`*) is associated with the right element ID

oncodrivefml.load.**load_mutations**(*file*, *blacklist=None*, *metadata_dict=None*)

Parsed the mutations file

Parameters

- **file** – mutations file (see OncodriveFML)
- **metadata_dict** (*dict*) – dict that the function will fill with useful information
- **blacklist** (*optional*) – file with blacklisted samples (see OncodriveFML). Defaults to None.

Yields One line from the mutations file as a dictionary. Each of the inner elements of *mutations*

10.1.6 oncodrivefml.main module

10.1.7 oncodrivefml.mtc module

Module containing functions related to multiple test correction

oncodrivefml.mtc.**multiple_test_correction**(*results*, *num_significant_samples=2*)

Performs a multiple test correction on the analysis results

Parameters

- **results** (*dict*) – dictionary with the results
- **num_significant_samples** (*int*) – mininum samples that a gene must have in order to perform the correction

Returns DataFrame. DataFrame with the q-values obtained from a multiple test correction

10.1.8 oncodrivefml.scores module

This module contains the methods associated with the scores that are assigned to the mutations.

The scores are read from a file.

Information about the stop scores.

As of December 2016, we have only measured the stops using CADD1.0.

The stops of a gene retrieved only if there are ast least 3 stops in the regions being analysed. If not, a formula is applied to derived the value of the stops from the rest of the values.

Note: This formula was obtained using the CADD scores of the coding regions. Using a different regions or scores files will make the function to return totally nonsense values.

---

class oncodrivefml.scores.**PackScoresReader**(*conf*)

Bases: object

BIT_TO_REF = {(0, 0, 0): '?', (0, 0, 1): 'T', (0, 1, 0): 'A', (0, 1, 1): 'C', (1,

SCORE_ALT = {'A': 'CGT', 'C': 'AGT', 'G': 'ACT', 'T': 'ACG'}

SCORE_ORDER = {'A': {'C': 0, 'G': 1, 'T': 2}, 'C': {'A': 0, 'G': 1, 'T': 2}, 'G': {'A'

STRUCT_SIZE = 6

get(*chromosome*, *start*, *stop*, **args*, ***kwargs*)

unpack(*block*)

exception oncodrivefml.scores.**ReaderError**(*msg*)
Bases: Exception

exception oncodrivefml.scores.**ReaderGetError**(*chr*, *start*, *stop*)
Bases: *oncodrivefml.scores.ReaderError*

class oncodrivefml.scores.**ScoreValue**(*ref*, *alt*, *value*, *ref_triplet*, *alt_triplet*)
Bases: tuple

Tuple that contains the reference, the alteration, the score value and the triplets

Parameters

- **ref** (*str*) – reference base
- **alt** (*str*) – altered base
- **value** (*float*) – score value of that substitution
- **ref_triplet** (*str*) – reference triplet
- **alt_triplet** (*str*) – altered triplet

alt
Alias for field number 1

alt_triplet
Alias for field number 4

ref
Alias for field number 0

ref_triplet
Alias for field number 3

value
Alias for field number 2

class oncodrivefml.scores.**Scores**(*element: str*, *segments: list*, *config: dict*)
Bases: object

Parameters

- **element** (*str*) – element ID
- **segments** (*list*) – list of the segments associated to the element
- **config** (*dict*) – configuration

scores_by_pos
for each positions get all possible changes, and for each change the triplets

```
{ position:
    [
        ScoreValue(
            ref,
            alt_1,
            value,
            ref_triplet,
            alt_triple
        ),
        ScoreValue(
            ref,
            alt_2,
```

(continues on next page)

(continued from previous page)

```
            value,
            ref_triplet,
            alt_triple
        ),
        ScoreValue(
            ref,
            alt_3,
            value,
            ref_triplet,
            alt_triple
        )
    ]
}
```

Type dict

get_all_positions() → List[int]
Get all positions in the element

Returns list of positions

Return type list of int

get_score_by_position(*position: int*) → List[oncodrivefml.scores.ScoreValue]
Get all ScoreValue objects that are asocated with that position

Parameters **position** (*int*) – position

Returns list of all ScoreValue related to that positon

Return type list of *ScoreValue*

get_stop_scores()
Get the scores of the stops in a gene that fall in the regions being analyzed

class oncodrivefml.scores.**ScoresTabixReader**(*conf*)
Bases: object

get(*chromosome*, *start*, *stop*, *element=None*)

oncodrivefml.scores.**init_scores_module**(*conf*)

oncodrivefml.scores.**null**(*x*)

oncodrivefml.scores.**stop_function**(*x*)

10.1.9 oncodrivefml.signature module

This module contains information related with the signature.

The signature is a way of assigning probabilities to certain mutations that have some relation amongst them (e.g. cancer type, sample...).

This relation is identified by the **signature_id**.

The `classifier` parameter in the *configuration* of the signature specifies which column of the mutations file (`MUTATIONS_HEADER`) is used as the identifier for the different signature groups. If the column does not exist the `classifier` itself is used as value for the *signature_id*.

The probabilities are taken only from substitutions. For them, the two bases that surround the mutated one are taken into account. This is called the triplet. For a certain mutation in a position *x* the reference triplet is the base in the reference genome in position *x-1*, the base in *x* and the base in the *x+1*. The altered triplet of the same mutation is equal for the bases in *x-1* and *x+1* but the base in *x* is the one observed in the mutation.

signature (`dict`)

```
{ signature_id:
    {
        (ref_triplet, alt_triplet): prob
    }
}
```

`oncodrivefml.signature.`**`change_ref_build`**(*build*)

Modify the default build fo the reference genome

Parameters **`build`** (*`str`*) – genome reference build

`oncodrivefml.signature.`**`chunkizator`**(*iterable*, *size=1000*)

Creates chunks from an iterable

Parameters

- **`iterable`** –
- **`size`** (*`int`*) – elements in the chunk

Returns list. Chunk

`oncodrivefml.signature.`**`collapse_complementaries`**(*signature*)

Add to the amount of a certain pair (ref_triplet, alt_triplet) the amount of the complementary.

Parameters **`signature`** (*`dict`*) – { (ref_triplet, alt_triplet): amount }

Returns { (ref_triplet, alt_triplet): new_amount }. New_amount is the addition of the amount for (ref_triplet, alt_triplet) and the amount for (complementary_ref_triplet, complementary_alt_triplet)

Return type dict

`oncodrivefml.signature.`**`compute_regions_signature`**(*elements*, *cores*)

Counts triplets in the elements

Parameters

- **`elements`** –
- **`cores`** (*`int`*) – cores to use

Returns `collections.Counter`. Counts of the triplets in the elements

`oncodrivefml.signature.`**`compute_signature`**(*signature_function*, *classifier*, *collapse=False*, *include_mnp=False*)

Gets the probability of each substitution that occurs for a certain signature_id.

Each substitution is identified by the pair (reference_triplet, altered_triplet).

The signature_id is taken from the mutations field corresponding to the classifier.

Parameters

- **`signature_function`** – function that yields one mutation each time
- **`classifier`** (*`str`*) – passed to *`load_mutations()`* as parameter `signature_classifier`.

- **collapse** (*bool*) – consider one substitutions and the complementary one as the same. Defaults to True.
- **include_mnp** (*bool*) – use MNP mutation in the signature computation or not

Returns

probability of each substitution (measured by the triplets) grouped by the signature_classifier

```
{ signature_id:
    {
        (ref_triplet, alt_triplet): prob
    }
}
```

Return type dict

Warning: Only substitutions (MNP are optional) are taken into account

oncodrivefml.signature.**correct_signature_by_triplets_frequencies**(*signature*, *triplets_frequencies*)

Normalized de signature by the frequency of the triplets

Parameters

- **signature** (*dict*) – see *signature*
- **triplets_frequencies** (*dict*) – {triplet: frequency}

Returns dict. Normalized signature

oncodrivefml.signature.**count_valid_trinucleotides**(*trinucleotides_dict*)

Count how many trinucleotides are valid

Parameters **trinucleotides_dict** (*dict*) – trinucleotides counts

Returns int. Valid trinucleotides

oncodrivefml.signature.**get_alternate_signature**(*line*)

Parameters **line** (*dict*) – contains the previous base, the alteration and the next base

Returns triplet with the central base replaced by the alteration indicated in the line

Return type str

oncodrivefml.signature.**get_build**()

oncodrivefml.signature.**get_normalized_frequencies**(*signature*, *triplets_frequencies*)

Divides the frequency of each triplet alteration by the frequency of the reference triplet to get the normalized signature

Parameters

- **signature** (*dict*) – {(ref_triplet, alt_triplet): counts}
- **triplets_frequencies** (*dict*) – {triplet: frequency}

Returns dict. Normalized signature

oncodrivefml.signature.**get_ref**(*chromosome*, *start*, *size=1*)

Gets a sequence from the reference genome

Parameters

- **chromosome** (*str*) – chromosome
- **start** (*int*) – start position where to look
- **size** (*int*) – number of bases to retrieve

Returns str. Sequence from the reference genome

oncodrivefml.signature.**get_ref_triplet**(*chromosome*, *start*)

Parameters

- **chromosome** (*str*) – chromosome identifier
- **start** (*int*) – starting position

Returns 3 bases from the reference genome

Return type str

oncodrivefml.signature.**get_reference_signature**(*line*)

Parameters **line** (*dict*) – contatins the chromosome and the position

Returns triplet around certain positions

Return type str

oncodrivefml.signature.**is_valid_trinucleotides**(*trinucleotide*)

Check if a trinucleotide has a nucleotide distinct than A, C, G, T :param trinucleotide: triplet :type trinucleotide: str

Returns bool.

oncodrivefml.signature.**load_signature**(*signature_config*, *signature_function*, *trinucleotides_counts=None*, *load_pickle=None*, *save_pickle=False*)

Computes the probability that certain mutation occurs.

Parameters

- **signature_config** (*dict*) – information of the signature (see *configuration*)
- **signature_function** – function that yields one mutation each time
- **trinucleotides_counts** (dict, optional) – counts of trincleotides used to correct the signature
- **load_pickle** (str, optional) – path to the pickle file
- **save_pickle** (str, optional) – path to pickle file

Returns

probability of each substitution (measured by the triplets) grouped by the signature_id

```
{ signature_id:
    {
        (ref_triplet, alt_triplet): prob
    }
}
```

Return type dict

Before computing the signature, it is checked whether a pickle file with the signature already exists or not.

oncodrivefml.signature.**load_trinucleotides_counts**(*region*)

Get the trinucleotides counts for a precomputed region: whole exome or whole genome

Parameters **region** (*str*) – whole genome or whole exome

Returns dict. Counts of the different trinucleotides

oncodrivefml.signature.**ref_build = 'hg19'**
Build of the Reference Genome

oncodrivefml.signature.**reverse_complementary_sequence**(*seq*)

Parameters **seq** (*str*) – sequence of bases

Returns complementary sequence

Return type str

oncodrivefml.signature.**sum2one_dict**(*signature_counts*)
Associates to each key (tuple(reference_tripet, altered_triplet)) the value divided by the total amount

Parameters **signature_counts** (*dict*) – pair key-amount {(ref_triplet, alt_triplet): value}

Returns pair key-(amount/total_amount)

Return type dict

oncodrivefml.signature.**triplet_counter_executor**(*elements*)
For a list of regions, get all the triplets present in all the segments

Parameters **elements** (list of list) – list of lists of segments

Returns collections.Counter. Count of each triplet in the regions

oncodrivefml.signature.**triplets**(*sequence*)

Parameters **sequence** (*str*) – sequence of nucleotides

Yields str. Triplet

oncodrivefml.signature.**yield_mutations**(*mutations*)
Yields one mutation each time from a list of mutations

Parameters **mutations** (*dict*) – *mutations*

Yields Mutation

10.1.10 oncodrivefml.stats module

This modules contains different statistical methods used to compare the observed and the simulated scores

class oncodrivefml.stats.**ArithmeticMean**
Bases: object

static calc(*values*)
Computes the arithmetic mean

Parameters **values** (list, array) – array of values

Returns mean value

Return type float

static calc_observed(*values*, *observed*)
Measure how many times the mean of the values is higher than the mean of the observed values

Parameters

- **values** (array) – m x n matrix with scores (m: number of randomizations; n: number of mutations)

- **observed** (list, array) – n size vector with the observed scores (n: number of mutations)

Returns

the number of times that the mean value of a randomization is greater or equal than the mean observed valu
(as int) and the number of times that the mean value of a randomization is equal or lower than the mean observed value (as int).

Return type tuple

class oncodrivefml.stats.**ArithmeticMeanHeteroscedasticScores**
Bases: object

static calc_observed(*values*, *observed*)

class oncodrivefml.stats.**GeometricMean**
Bases: object

The geometric mean used is not the standard.

$$(\prod_{i=1}^{n}(x_i+1))^{1/n} - 1 = \sqrt[n]{(x_1+1)(x_2+1)\cdots(x_n+1)} - 1$$

static calc(*values*)
Computes the geometric mean of a set of values.

Parameters **values** (list, array) – set of values

Returns geometric mean (array): geometric mean by columns (if the input is a matrix)

Return type (float)

static calc_observed(*values*, *observed*)

Measure how many times the geometric mean of the values is higher than the geometric mean of the observed values

Parameters

- **values** (array) – m x n matrix with scores (m: number of randomizations; n: number of mutations)
- **observed** (list, array) – n size vector with the observed scores (n: number of mutations)

Returns

the number of times that the mean value of a randomization is greater or equal than the mean observed valu
(as int) and the number of times that the mean value of a randomization is equal or lower than the mean observed value (as int).

Return type tuple

class oncodrivefml.stats.**Maximum**
Bases: object

static calc(*values*)

static calc_observed(*values*, *observed*)

10.1.11 oncodrivefml.store module

This module contains the methods used to store the results.

3 different types of output are available:

- **tsv** file
- **png** graph: uses the *tsv* file and matplotlib
- **html** graph: uses the *tsv* file and bokeh

class oncodrivefml.store.**QQPlot**(*input_file*, *cutoff=True*, *rename_fields=None*, *extra_fields=None*)

Bases: object

Parameters

- **input_file** – tsv file with the data
- **cutoff** (*bool*) – add cutoffs to the figure
- **rename_fields** (*dict*) – column names from the input file can be renamed providing a dictionary {old_name : new_name}
- **extra_fields** (*list*) – list of column names that want to be passed to the figure data. Need for example to search by them.

add_search_widget(*fields*)

Add text input for each field.

Parameters **fields** (str or list) – list of fields to do a search.

add_tooltip()

Adds tooltip to show the parameters of each glyph in the figure

add_tooltip_enhanced()

The tooltip is shown via JavaScript to avoid been block in areas with a high density of points

show(*output_path*, *showit=True*, *notebook=False*)

Show the figure

Parameters

- **output_path** – file where to store the figure
- **showit** (*bool*) – the figure is displayed (widgets and the like are not shown) or is fully saved. Defaults to True.
- **notebook** (*bool*) – if is is called form a notebook or not. Defaults to False.

oncodrivefml.store.**add_symbol**(*df*)

oncodrivefml.store.**eliminate_duplicates**(*df*)

oncodrivefml.store.**store_html**(*input_file*, *output_path*)

Create the QQPlot and save it.

Parameters

- **input_file** – tsv filw with the data
- **output_path** – file where to store the graph
- **showit** (*bool*) – defaults to False. See *show()*.

oncodrivefml.store.**store_png**(*input_file*, *output_file*, *showit=False*)

Creates a figure from the resutls.

Parameters

- **input_file** – tsv file with the results
- **output_file** – file where to store the figure
- **showit** (*bool*) – calls `show()` before returning. Defaults to False.

`oncodrivefml.store.`**store_tsv**(*results*, *result_file*)
Saves the results in a tsv file sorted by pvalue

Parameters

- **results** (`DataFrame`) – results of the analysis
- **result_file** – file where to store the results

10.1.12 oncodrivefml.utils module

This module contains some useful methods

`oncodrivefml.utils.`**defaultdict_list**()
Shortcut

Returns `defaultdict` of `list`

`oncodrivefml.utils.`**executor_run**(*executor*)
Method to call the run method

Parameters **executor** (`ElementExecutor`) –

Returns `run()`

`oncodrivefml.utils.`**exists_path**(*path*)

`oncodrivefml.utils.`**loop_logging**(*iterable*, *size=None*, *step=1*)
Loop through an iterable object displaying messages using `info()`

Parameters

- **iterable** –
- **size** (*int*) – Defaults to None.
- **step** (*int*) – Defaults to 1.

Yields The iterable element

10.1.13 oncodrivefml.walker module

10.1.14 oncodrivefml.walker_cython module

10.1.15 Module contents

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

I

L

M

N

O

P

Q

R

S

T

U

V

Y