MutEnricher: Tutorial

Code version: 1.0.0

Anthony R. Soltis

June 26, 2018

1. Description

MutEnricher is a computational toolset designed to perform somatic mutation enrichment analysis of both coding and non-coding regions from whole genome sequencing (WGS) data. MutEnricher contains two distinct modules for performing these analyses: `coding` and `noncoding`. Both modules use the negative binomial test to determine gene/region somatic mutation enrichment significance. Background mutation rates for each gene/region are determined via one of three distinct user-selected methods.

2. Requirements & Installation

Python and packages

This software has been explicitly tested with Python versions 2.7.12 and 2.7.13, though compatibility with versions >= 2.7 is expected. Compatibility with Python versions > 3 has not been examined, but may be considered for future releases. Compatibility with versions < 2.7 is likely possible, though untested. The library argparse (https://pypi.python.org/pypi/argparse), which is part of the standard Python library for versions >= 2.7, must be installed (in addition to the modules described below) if attempting use with versions < 2.7. Other packages and/or updates may also be necessary when running with older versions.

Beyond the basic standard Python library, several easily accessible Python packages must be installed (if not already present) for successful execution (some version restrictions noted):

- NumPy (http://www.numpy.org/)
- SciPy (https://scipy.org/)
- Cython (http://docs.cython.org/en/latest/src/quickstart/install.html)
- cyvcf2 (http://brentp.github.io/cyvcf2/)
- pysam (>=0.8.1, tested with >= 0.10.0) (http://pysam.readthedocs.io/en/latest/)

The easiest way to obtain these additional packages is to use the Python package manager pip. Python versions >= 2.7.9 include pip by default; otherwise, this package can be installed separately on your system (https://pip.pypa.io/en/stable/installing/).

Cythonize math functions code

This package utilizes “cythonized” code to enhance speed for some numerical functions. The user needs to compile these functions with his or her own version of Python/Cython to successfully run the programs:

1. From the package install directory, enter the math_funcs sub-directory (`cd math_funcs`).

2. Run the following command to cythonize the code:
   1. `python setup.py build_ext --inplace`

These steps will produce the file `math_funcs.so` in the math_funcs directory.

Additional package utilities

We provide additional utilities (in the `utilities` sub-directory) that can be used to help generate feature covariate files for use with the programs. Currently, this directory includes the Python scripts:

- `get_gene_covariates.py`
- `get_region_covariates.py`

These scripts use the same input gene/region files (GTF or BED files, more on these later) to calculate sequence (and other) covariates for the genes/regions considered in the analyses. Further usage instructions for these optional scripts are described in more detail below.

Additional suggested external command-line utilities

We recommend that users install the HTSlib utilities bcftools, bgzip, and tabix (see http://www.htslib.org/download/). These tools are extremely useful for interfacing with and manipulating variant call format (VCF) files.

3. Tutorial

This package performs somatic mutation recurrence analysis in two distinct fashions:

- *coding analysis*, which tests for enrichment above background of non-silent somatic mutations in protein coding genes
- *non-coding analysis*, which tests for enrichment above background of somatic mutations in user-defined non-coding regions of the genome (e.g. promoters, UTRs, enhancers, etc.)

Both tools are accessible through the main package driver script **`mutEnricher.py`**. To see the main help page, use **`python mutEnricher.py -h`**. To see the detailed help pages for the two analysis types, use `python` **`mutEnricher.py`** `<command> -h`, where `<command>` is either `coding` or `noncoding`.

Example data for test execution is provided in the `example_data` sub-directory. There are three additional sub-directories within containing example data:

1. `annotation_files` – includes an example GTF file and BED file of coding gene promoters.
2. `covariates` – includes example covariate and covariate weights files
3. `precomputed_apcluster` – includes pre-computed gene/region covariate clustering results for use/testing.
4. `vcfs` – 100 example somatic mutation files (with tabix-indexed .tbi files)

Additional files include:

- `nonsilent_terms.txt`
- `quickstart_commands.txt`
- `vcf_files.txt`

The example somatic VCF files were generated by randomly assigning “somatic mutations” throughout the genome at a global frequency of 2 mutations per megabase, and thus **DO NOT** represent true whole genome somatic mutation data. Included with these randomly distributed “mutations” are known cancer mutations in the coding regions of the genes KRAS and TP53 (coding examples) and in the promoter of the TERT gene (non-coding example). These mutations were included at target cohort frequencies of 30%, 90%, and 40%, respectively, and serve as known true positives for testing.

3.1. Inputs

3.1.1. Somatic mutations

The main inputs to MutEnricher are per-sample somatic mutation calls. These should be provided as **sorted, bgzipped, and tabix-indexed VCF files** for both analysis types (though other options are described later). Such files can be generated from a variety of somatic mutation callers comparing tumor versus normal DNA, though discussion of such tools is outside the scope of this tutorial. Post-filtering of somatic variants for quality is generally necessary to remove weak or potential false positive variants. MutEnricher considers only PASS variants from the individual VCFs; no other per-variant filtering is performed. ***It is highly recommended that users provide VCF files filtered for only the most reliable variant calls to prevent false discoveries***. Several programs, including bcftools, are capable of performing filtering by a variety of variables.

Somatic VCFs for each sample are provided with an **input tab-delimited text file** (one sample per row). Each row has two columns:

- The VCF file path (e.g. /path/to/vcfs/sample1.vcf.gz)
- A unique name for the sample (e.g. sample1)

An error is thrown if the sample names provided are not unique or if any of the VCF files listed are not found on the system.

For non-coding analyses, no specific VCF annotations are necessary as somatic variants are considered based on genomic location only. ***Variant annotation is necessary for coding analyses, however***. If a non-silent term is found (e.g. nonsynonymous, frameshift, etc.), the variant is recorded as non-silent; otherwise, it is recorded as background. VCFs may be annotated during calling (depending on workflows), added later through various programs (e.g. bcftools or ANNOVAR [http://annovar.openbioinformatics.org/en/latest/]), or added manually. These annotations must be **present in the INFO field** of each VCF. For example, a nonsynonymous variant in an exon of the TP53 gene may be annotated in the INFO field with ANNOVAR as:

SOMATIC;Gene.refGene=TP53;ExonicFunc.refGene=nonsynonymous_SNV;Func.refGene=exonic

Here, the gene name is encoded in the “Gene.refGene” field and the relevant effect on the gene is encoded in the “ExonicFunc.refGene” field.

In another example:

SOMATIC;Gene.refGene=TP53;ExonicFunc.refGene=.;Func.refGene=splicing

Here, the variant occurs adjacent to an exon boundary and potentially modulates gene splicing. This information is contained in the “Func.refGene” field.

Users can define what terms are considered non-silent in a few ways using the `--anno-type` option. If using somatic VCFs generated and annotated by Illumina pipelines (e.g. annotated by the Illumina annotation engine), users can use pre-defined terms by setting this option to "illumina" (this is the default). Alternatively, if VCFs are annotated with ANNOVAR using RefGene data, pre-defined terms can be accessed by setting this option to "annovar." Alternatively, users can provide a tab-delimited text file containing these terms with this same option (e.g. `--anno-type=/path/to/nonsilent_terms.txt`). An example of such a file is:

```
Gene    Gene.refGene    x
Effect  ExonicFunc.refGene      frameshift_deletion
Effect  ExonicFunc.refGene      frameshift_insertion
Effect  ExonicFunc.refGene      frameshift_substitution
Effect  ExonicFunc.refGene      nonframeshift_deletion
Effect  ExonicFunc.refGene      nonframeshift_insertion
Effect  ExonicFunc.refGene      nonframeshift_substitution
Effect  ExonicFunc.refGene      nonsynonymous_SNV
Effect  ExonicFunc.refGene      stopgain
Effect  ExonicFunc.refGene      stoploss
Effect  Func.refGene    splicing
```

The **first column** entry of each row must be either **"Gene" or "Effect"** - this signifies which INFO fields are to be searched when extracting gene name and gene effect information, respectively. There should only be one "Gene" row, but multiple "Effect" rows are possible. The **second column** defines the field(s) to be searched from the INFO column of each VCF. For "Effect," multiple fields can be considered (e.g. ExonicFunc.refGene and Func.refGene are scanned in the above example). The **third column** defines the actual non-silent terms. For "Gene," this field is not considered (e.g. can be blank or contain a placeholder). **Note** also that gene names in the "Gene" field of each VCF should match the nomencalture used in the appropriate `genefield` from the input GTF file (see below).

The coding analysis code allows users to alternatively input coding variants in mutation annotation format (MAF, see specifications at https://wiki.nci.nih.gov/display/TCGA/Mutation+Annotation+Format+(MAF)+Specification). Non-silent terms following MAF specification are used if this input type is used.

NOTE: the non-coding analysis code does not currently accept MAF files.

3.1.2. Genes and non-coding regions

For coding analyses, gene models must be provided with a gene transfer format (GTF) file. Such files are available from several online sources (e.g. the UCSC table browser from https://genome.ucsc.edu). Gene names are read using the `gene_id` field by default (the field choice can be modified with the `--gene-field` option) and exon, coding sequences (CDS), and introns are determined and consolidated into one composite gene in cases of multi-isoform transcript models. As a filtering step, any genes that are annotated to multiple loci in the genome (e.g. the same gene name present on two different chromosomes) are removed from further consideration.

Users may also restrict the program to testing only genes in the GTF that are present in an "include" list. This is accomplished by providing a text file with gene names (one per line) with the `-g` or `--gene-list` option.

For non-coding analyses, regions for testing (e.g. promoters) are provided with a simple input BED file (see specification https://genome.ucsc.edu/FAQ/FAQformat.html#format1). The first three columns are required (chromosome, 0-based start position, 1-based end position). Any text information provided in the fourth column is used as the name for the region (e.g. TERT_promoter) and can be useful for identifying regions in the output. If the fourth column is empty, default names are created.

It is recommended that the input BED file contain sorted and non-overlapping intervals. To accomplish and/or check this, the user can run a simple BEDTools command:

```
bedtools sort -i input.bed | bedtools merge -c 4 -o collapse > output.bed
```

Here, `input.bed` and `output.bed` are the original input and the sorted, merged output BED files, respectively. The first command sorts the records by position and the second merges any overlapping intervals into one. The options `-c 4` and `-o collapse` instruct bedtools to join the entries in the fourth column (i.e. region names) by a delimiter (comma in this case).

3.1.3. Covariate and covariate weights files

As described earlier, users can optionally cluster features by genomic covariates. To perform this clustering, an input covariates file must be supplied (with the `-c` or `--covariates-file` option). The provided file should be tab-delimited with a single header line. The first column contains the feature name, which is either the gene name (coding analysis) or a region string (non-coding) analysis. The latter is of the format <chromosome>:<1-based start>-<1-based end>, e.g. chr10:100-200, and should match the regions in the input BED file (note the 1-based start differs from the 0-based start in the BED!). The remaining columns contain values for the feature covariates, named with their respective column labels in the header line.

As mentioned earlier, we include two utilities to assist with creating these input files. For coding analyses, users can run `get_gene_covariates.py`, which takes the GTF needed for the coding analysis and an indexed genome seqeuence file (e.g. for hg19):

```
python get_gene_covariates.py /path/to/genes.gtf /path/to/genome.fa
```

The above run will use the gene models included in the GTF and calculate the full gene length, coding length, and GC and CpG contents of the underlying DNA sequence. For coding analysis, the lengths reported are converted to per kilobase and $\log_2$ transformed. For example:

```
Gene     full_length     coding_length   GC      CpG
A1BG     2.74286855102   0.570462931026  0.658   0.672
A1CF     6.43073688093   0.929790997719  0.469   0.315
A2M      5.60056711218   2.14469902515   0.493   0.291
A2ML1    5.76314595815   2.12598165385   0.497   0.286
A3GALT2  3.84126870254   0.0285691521968 0.668   0.873
A4GALT   4.86740230573   0.0827025893302 0.637   0.682
A4GNT    3.11603199345   0.0285691521968 0.507   0.275
AAAS     3.82507326084   0.711935356979  0.575   0.281
AACS     6.28479171873   1.01149563884   0.557   0.528
```

For non-coding analysis, users can run `get_region_covariates.py` in very similar fashion:

```
python get_region_covariates.py /path/to/regions.bed /path/to/genome.fa
```

The only difference in this base run (other than the function name) is the use of the regions BED file instead of the GTF file used above. This also reports the region length (in basepairs with no transformation) and GC and CpG contents:

```
Region  length  GC      CpG
chr1:68549-69749        1201    0.356   0.315
chr1:367128-368328      1201    0.416   0.155
chr1:621365-622565      1201    0.415   0.136
chr1:860150-861350      1201    0.689   0.917
chr1:894437-896273      1837    0.655   0.943
chr1:900935-902135      1201    0.676   0.471
chr1:917271-918471      1201    0.642   0.502
chr1:935112-936399      1288    0.731   0.962
chr1:947901-949101      1201    0.318   0.828
```

Additional information can be supplied to both functions in one of two formats:

1. One or multiple "table" files, which are tab-delimited with one header each. The first column is reserved for the gene or region name (and should match the features provided in the input GTF/BED file), while the remaining columns contain the desired covariate information. For example, a user could supply a table of gene expression levels from one or more samples to control for gene expression level in the analysis. *Data supplied in this way is not transformed or re-scaled and any genes (regions) in the input GTF (BED) file not containing information from these tables are reported as NA.*
2. Replication timing information from Repli-seq data, supplied as bgzipped and tabix-indexed BED/bedgraph files (using the `--repliseq-fns` option). The input to this option is a two-column tab-delimited text file with no header with columns 1) Repli-seq dataset file path and 2) sample name. The code will search the genomic coordinates of the genes/regions in these files and report an average value (in case of multiple overlaps) from the data within. If no data is found in the immediate vicinity, the code will expand its search window from the feature midpoint until it finds a valid interval. If no overlapping intervals are found after the expanded search, "None" is reported. Such data is available online from varioius sources (e.g. ENCODE https://www.encodeproject.org/search/?type=Experiment&assay_title=Repli-seq) and can be converted to bedgraph format with various tools.

Example runs:

```
python get_gene_covariates.py /path/to/genes.gtf /path/to/genome.fa -t
/path/to/RNA_1.txt -t /path/to/RNA_2.txt
```

The above will produce the same gene covariate information as above, but with additional columns for the information provided in the two additional tables `RNA_1.txt` and `RNA_2.txt`.

```
python get_gene_covariates.py /path/to/genes.gtf /path/to/genome.fa -t
/path/to/RNA_1.txt --repliseq-fns /path/to/repliseq.txt
```

The above will produce the same gene covariate information as above, but with additional columns for the information provided in the table as well as the replication timing information from the file paths included in the text file.

In addition, this code can be parallelized across multiple processors with the `-p` or `--processors` option. For coding analysis, users can also restrict the list of genes to include in the output with the `-g` or `--gene-list` option.

Along with the covariate files themselves, users can also define weights to each covariate that are used in the feature-wise similarity calculations. This allows users to increase or decrease the emphasis of specific covariates. Such weights should be provided as a simple two-column tab-delimited text file with no header line and columns:

1. Covariate name – this should match the headers in the matching covariates file
2. The weight value

Weights are numeric values and can be on an arbitrary scale. The enrichment analysis code will automatically re-scale the supplied weights such that they sum to 1. Larger values represent more weight, while smaller values should reflect lower weighted covariates.

Example covariate and covariate weights files are included in the `example_data/covariates` sub-direcotry.

3.2. Run options

After all desired input files are prepared, the user is now ready to execute either the coding analysis portion of the package or the non-coding portion. Throughout the remaining tutorial, we refer to run protocols using the example files in the `example_data` sub-directory. We assume the working directory for the analysis is from this directory.

3.2.1. Coding analysis

The most basic coding analysis run can be executed with:

```
python ../mutEnricher.py coding
annotation_files/ucsc.refFlat.20170829.no_chrMY.gtf vcf_files.txt
```

This calls the main driver script and tells it to run the `coding` module. To explicitly define the output directory of the analysis (default is current working directory) as well as to define a prefix to the output analysis files:

```
python ../mutEnricher.py coding
annotation_files/ucsc.refFlat.20170829.no_chrMY.gtf vcf_files.txt -o
/path/to/output/directory --prefix my_analysis
```

The default run options here tell the code to consider only global gene background frequencies when performing enrichment analyses. Also, the code expects non-silent annotations in the default format, which is currently set to “illumina” for this variant caller. To run with a different set of annotation terms relevant to your data:

```
python ../mutEnricher.py coding
annotation_files/ucsc.refFlat.20170829.no_chrMY.gtf vcf_files.txt --anno-type
nonsilent_terms.txt
```

If running the test data, the user should run the above option as the example somatic VCFs were annotated with RefGene information using ANNOVAR.

If the user wishes to run the analysis using local background frequencies for the genes, run the analysis with the --`use-local` option:

```
python ../mutEnricher.py coding
annotation_files/ucsc.refFlat.20170829.no_chrMY.gtf vcf_files.txt --anno-type
nonsilent_terms.txt --use-local
```

If the user wishes to run the analysis with covariate information, one run option is:

```
python mutEnricher.py coding annotation_files/ucsc.refFlat.20170829.no_chrMY.gtf
vcf_files.txt --anno-type nonsilent_terms.txt -c
ucsc.refFlat.20170829.no_chrMY.covariates.txt -w
ucsc.refFlat.20170829.no_chrMY.covariate_weights.txt
```

Here, a covariates file and an associated covariate weights file is supplied. By default for coding analysis, affinity propagation is run considering all genes. While affinity propagation is indeed capable of handling large numbers of data points, this run can be quite slow as all pairwise similarities between genes must be calculated and written to a file. o reduce the problem complexity and to take advantage of parallelization, the user can split the similarity computations by contigs (e.g. by chromosomes) with the --`by-contig` option. This tells the program to cluster only genes on the same chromosome and use these clusters in the downstream analysis. This can dramatically reduce run time while still grouping genes by similar characteristics. Also, MutEnricher contains two implementations of affinity propagation that the user can select: `fast` and `slow`. The `fast` algorithm, as the name implies, is faster but consumes more memory than `slow`. An example run with clustering is:

```
python ../mutEnricher.py coding
annotation_files/ucsc.refFlat.20170829.no_chrMY.gtf vcf_files.txt --anno-type
nonsilent_terms.txt -c ucsc.refFlat.20170829.no_chrMY.covariates.txt -w
ucsc.refFlat.20170829.no_chrMY.covariate_weights.txt --by-contig -p 10
```

Note that the `-p` or --`processors` sets the number of processors the code can use during execution. The default value is 1, but this can always be set higher if resources are available. When running with covariate information and splitting by contigs, similarity calculations and subsequent affinity propagation runs are split across the available processors. Additional steps in the analysis also benefit from parallelization.

If the user has already run the analysis and computed covariates, but would like to run the workflow with new samples or re-run the same analysis with different parameters (outside of those that influence the covariate clustering), he or she may provide pre-computed covariate clusters to the code. This is accomplished with the --`precomputed-covars` option, which accepts a path to already computed clustering output. For example:

```
python ../mutEnricher.py coding
annotation_files/ucsc.refFlat.20170829.no_chrMY.gtf vcf_files.txt --anno-type
nonsilent_terms.txt -p 10 --precomputed-covars /path/to/apcluster_genes
```

When running coding analysis with covariate clustering, an output directory `apcluster_genes` will be created in the main output directory and houses the covariate information. This path can be supplied in subsequent runs.

An additional feature of the coding analysis code is the ability to accept MAF format files. Typically, such files are generated from experiments of somatic variation using whole exome sequencing. To supply a MAF file (e.g. `mutations.maf`), the user should use the `--use-maf` option and supply a place-holder character (e.g. -) for the VCF file list input:

```
python ../mutEnricher.py coding
annotation_files/ucsc.refFlat.20170829.no_chrMY.gtf - --anno-type
nonsilent_terms.txt -p 10 --maf mutations.maf
```

If the data contained in the MAF was generated using whole exome sequencing or a related methodology, the assumptions used in the background calculations will be violated due to the reduced coverage space. In such cases, the user can set the `--exome-only` option to tell the program to only consider exonic sequence coordinates for genes.

```
python ../mutEnricher.py coding
annotation_files/ucsc.refFlat.20170829.no_chrMY.gtf - --anno-type
nonsilent_terms.txt -p 10 --maf mutations.maf --exome-only
```

With MAF input, the default (global) and covariate clustering background methodologies are available. However, given that the local background method depends on the ability to scan regions in and outside the gene limits, this option is not as of now available with MAF input.

NOTE: The p-value distributions with MAF input can be highly conservative when using the covariate clustering background rate method. Users may wish to try the global background method if this is encountered. Alternatively, users can adjust the mutations considered in the background to only silent mutations with the `--bg-vars-type` flag (i.e. `--bg-vars-type silent`). This latter option will produce less conservative p-values, though they may now be too optimistic. Users can run several versions of the analysis and select the most appropriate for the problem at hand.

3.2.2. Non-coding analysis

The most basic run of the non-coding analysis tool is very similar to that of the coding analysis. *The major difference is the requirement of an input BED file as opposed to a GTF:*

```
python ../mutEnricher.py noncoding
annotation_files/ucsc.refFlat.20170829.promoters_up1kb_down200.no_chrMY.bed
vcf_files.txt -o /path/to/output/directory --prefix my_noncoding_analysis
```

Overall, there are fewer options associated with the non-coding analysis. However, those that are present follow the same structure as in the coding analysis. All background options available for the coding analysis are available here.

A few subtle differences when running with covariate clustering:

- The default behavior is to split the analysis by chromosome. This reduces the problem size and takes advantage of parallelization. This is generally useful, if not necessary, as the potential space of non-coding regions is much greater than that of coding genes, depending on how the

user defines such features.
- If using pre-computed covariates, the run procedure is the same as for coding analysis. Be sure to point to the correct clustering output directory (/path/to/apcluster_regions) as before.

3.2.3. Other run options

The below options apply to both coding and non-coding analysis methods.

Mutations used in background calculations

- `--bg-vars-type`
  - Option to control definition of "background." Options are `'all'`, which instructs MutEnricher to consider both silent and non-silent mutations during its background estimation procedures (the default). Alternatively, users can adjust this definition by using the `'silent'` option with this flag, which instructs MutEnricher to only consider silent mutations in the background calculations.

Parallelization

- `-p` or `--processors`
  - Option to set the number of run processors.

Mappable regions filtering

- `-m` or `--mappable-regions`
  - Not all regions of the genome are uniquely mappable by most short read based whole genome/exome sequencing technologies. Users can optionally supply a BED file (bgzipped and tabix-indexed) of mappable genomic regions (e.g. based on known callable regions or user-calculated data from protocol benchmarks) that is used to restrict the analysis space. Variants existing only in genic/non-coding regions overlapping these callable regions are considered and the lengths used in the enrichment calculations are adjusted accordingly.

GTF and gene input options (coding analysis)

- `--gene-field`
  - Explicitly define the field name in the input GTF containing the desired gene IDs. The default is `gene_id`, but users can adjust to suit their needs. **NOTE**: the format of the IDs in the selected field should be consistent with what was used to annotate the VCF files (e.g. gene symbols).
- `-g` or `--gene-list`
  - Provide a simple text file of gene Ids/symbols to which the coding analysis should be restricted. For instance, this could be a list of genes believed to be expressed in the particular tumor tissue type. Useful for avoiding analysis of non- or lowly-expressed genes, reducing problem size, and reducing multiple hypothesis testing burden.

Covariate clustering

- `--min-rclust-size`
  - During covariate clustering, features grouped together are used for background estimation. In some cases, singletons or very small clusters can be generated. In such cases, a local background can be calculated to better estimate this value for features that are more or less

"distinct." This option takes an integer value and tells the programs to calculate a local background for all features belonging to a cluster with fewer than this many members.

- `--ap-iters`
  - This parameter controls the maximum number of affinity propagation iterations. Iterations continue until convergence is attained or after this many iterations are run. In cases where the clustering does not converge, the program will automatically re-run the algorithm with a slightly perturbed value for the self-similarity paramter. This value can be set higher to run more iterations, but is not necessarily recommended (default is 1000).
- `--ap-convits`
  - This parameter sets the number of convergence iterations (i.e. the number of consecutive iterations for which the exemplars remain constant) required before terminating affinity propagation runs.
- `--ap-algorithm`
  - Choose between one of two implementations of the affinity propagation algorithm: 'slow' or 'fast'. As the names imply, 'fast' is a faster implementation, but requires more memory for use. The 'slow' version produces the same results with a moderately slower implementation, but uses less memory.

"Hotspots"

- `-d` or `--hotspot-distance`
  - Set maximum distance (in base pairs) between two variants for consideration as part of a hotspot. Default value is 50.
- `--min-hs-vars`
  - Minimum number of variants that must be present in a candidate hotspot for statistical testing. Default is 3.

Use only SNPs in analysis

- `--snps-only`
  - Set this flag to only consider SNPs in analysis (i.e. skip indels).

Variant blacklist

- --blacklist
  - Provide a tab-delimited text file with no header containing blacklist variants, i.e. variants that are potential false positives that should not be considered in analysis. The required columns are: 1) chromosome, 2) position (1-based index), 3) reference sequence, 4) alternate. The format of reference/alternate should match that of your input VCFs. ***NOTE**: this option is only recommended for a small set of variants (e.g. tens to a few hundreds) as large numbers of variants can cause memory issues if using several parallel processors. In cases where large numbers of variants potentially should be removed, we suggest removing these directly from the input VCFs prior to running the analyses.

3.3 Outputs

3.3.1 Coding analysis

Four output files are generated from a successful coding analysis run:

1. <prefix>_gene_enrichments.txt
2. <prefix>_hotspot.txt

3. <prefix>_gene_data.pkl
4. <prefix>.params.log

The first text file is the main output file for the overall gene significance levels. There are a number of output fields describing the numbers of non-silent and silent variants per gene, the affected samples, the particular variants, significance levels, etc. A simple view from the example output:

```
cat example_gene_enrichments.txt | cut -f1,3,10 | head
Gene    num_nonsilent   FDR_BH
TP53    93      1.28e-184
KRAS    29      1.46e-50
MYOF    7       0.796
CTSH    3       0.796
SSUH2   3       0.796
MBTPS1  4       0.796
RPS24   3       0.796
PALMD   3       0.796
TMEM39A 3       0.796
```

Displayed above are the columns containing the gene names, number of non-silent mutations, and the overall FDR-corrected significance level. Note the two spiked in coding examples TP53 and KRAS indeed show up as highly significant in the output.

The second output file describes the results of the hotspot enrichment analysis. As example:

```
cat example_hotspot.txt | cut -f1-5,9-11
Gene    hotpsot num_mutations   hotspot_length  effective_length        BH_qval
num_samples     position_counts
KRAS    chr12:25398284-25398285 29      2       200     2.8e-127        29
25398284_12;25398285_17
TP53    chr17:7577120-7577120   26      1       100     4.56e-119       26
7577120_26
TP53    chr17:7576853-7576853   25      1       100     2.56e-117       25
7576853_25
TP53    chr17:7578555-7578555   23      1       100     5.25e-106       23
7578555_23
TP53    chr17:7577574-7577574   19      1       100     1.73e-87        19
7577574_19
```

In the above output, the second column provides the coordinates for the tested hotspot, the third the number of mutations, the fourth the length of the hotspot, the fifth the “effective” length (i.e. the hotspot length multiplied by the number of samples), the sixth the FDR-corrected significance level, the seventh the number of mutated samples, and the last the hotspot positions with the numbers of mutations observed at these positions from the sample cohort. Note these hotspots are anticipated given the nature of the spiked-in variants.

The third output file is a Python pickle object that contains the information loaded from the input VCF files along with various calculated information. Technically, the data structure contained is a list of Gene objects defined by the main coding analysis code. A user can browse this data if desired in Python. For example, if the user wishes to find what samples possess non-silent *KRAS* mutations, from python (or ipython):

```
import sys,os
import cPickle
sys.path.insert(0,'/path/to/install/directory') # change to user's directory
sys.path.insert(0,'/path/to/install/directory/math_funcs')
from coding_enrichment import Gene
genes = cPickle.load(open('example_gene_data.pkl'))
for g in genes:
      if g.name == 'KRAS':
            for s in g.mutations_by_sample:
                  print s,g.mutations_by_sample[s]['nonsilent']
```

The output from this displays the sample names followed by a list of non-silent mutations:

```
sample_40 ['25398285_C_A']
sample_27 ['25398284_C_A']
sample_43 []
sample_22 ['25398285_C_A']
sample_45 ['25398284_C_A']
sample_70 ['25398285_C_A']
sample_62 ['25398285_C_A']
sample_49 ['25398284_C_A']
sample_67 ['25398285_C_A']
sample_65 ['25398284_C_A']
sample_8 ['25398285_C_A']
sample_6 ['25398284_C_A']
sample_5 ['25398284_C_A']
sample_3 ['25398285_C_A']
sample_93 []
sample_91 ['25398285_C_A']
sample_97 ['25398285_C_A']
sample_96 ['25398285_C_A']
sample_29 ['25398284_C_A']
sample_12 ['25398285_C_A']
sample_15 ['25398284_C_A']
sample_14 ['25398284_C_A']
sample_31 ['25398285_C_A']
sample_56 ['25398285_C_A']
sample_75 ['25398284_C_A']
sample_50 ['25398285_C_A']
sample_26 ['25398285_C_A']
sample_39 ['25398284_C_A']
sample_81 ['25398285_C_A']
sample_88 ['25398285_C_A']
sample_89 ['25398284_C_A']
```

3.3.2 Non-coding analysis

Similar to the coding analysis, non-coding analysis produces four output files:

1. <prefix>_region_WAP_enrichments.txt
2. <prefix>_hotspot.txt
3. <prefix>_region_data.pkl
4. <prefix>.params.log

The first output is again the main output. For the overall regional significance level, a Fisher's

combined p-value for the overall region significance and a “weighted average proximity” (WAP) permuted p-value is reported. This value is further corrected for multiple hypotheses. As example:

```
cat example_region_WAP_enrichments.txt | cut -f1-3,8-12 | head
Region  region_name     num_mutations   region_pval     WAP     WAP_pval
Fisher_pval     FDR_BH
chr17:7578391-7579591   TP53_2  23      2.74e-19        0       1       0       0
chr5:1294824-1296024    TERT    43      6.6e-40 8.2     0.821   0       0
chr19:56825576-56827173 ZSCAN5A_2,ZSCAN5A_5,ZSCAN5A_1,ZSCAN5A_3 3       0.943
1.8     8.2e-05 0.00081 1
chr11:118753588-118754788       CXCR5_2 3       0.212   0.8     0.0205  0.028   1
chr1:47426232-47427432  CYP4X1_2        3       0.725   0.88    0.0153  0.061   1
chr17:38136232-38137432 PSMD3   3       0.0231  1.1e-256        0.712   0.084   1
chr10:91403351-91404551 PANK1_1 2       0.0848  -1      -1      0.0848  1
chr10:99495978-99497180 ZFYVE27_2,ZFYVE27_1     2       0.085   -1      -1
0.085   1
chr10:121410124-121411324       BAG3    2       0.102   -1      -1      0.102   1
```

Columns 1 and 2 describe the region coordinates and name, respectively; the latter is taken from the fourth column of the input BED file (if provided; otherwise a default value is chosen). The number of mutations in the region, the overall region p-value, the WAP statistic, the permuted WAP p-value, Fisher p-value, and FDR-corrected significance level are reported in the remaining columns displayed above. Note the TERT promoter mutations that were specifically spiked-in show the most significant overall regional p-value (the TP53 mutations derive mostly from overlapping definitions of promoter versus coding region). Note also that the -1 values in the WAP and WAP_pval columns reflect cases where this value was not actually calculated as a result of too few variants in the region (this is controlled by the `--min-hs-vars` option, as with the formal hotspot analysis).

The output formats for the remaining two output files are similar to those produced by the coding analysis.

APPENDIX

Help pages for code:

- mutEnricher.py coding

```
usage: python mutEnricher.py coding [-h] [-o OUTDIR] [--prefix PREFIX]
                                    [--gene-field GENEFIELD] [-g GENE_LIST]
                                    [--bg-vars-type BG_VARS_TYPE] [--maf MAF]
                                    [--exome-only] [--anno-type TTYPE]
                                    [-m MAP_REGIONS] [-p NPROCESSORS]
                                    [--snps-only] [-c COV_FN] [-w WEIGHTS_FN]
                                    [--by-contig] [--use-local]
                                    [--min-clust-size MIN_CLUST_SIZE]
                                    [--precomputed-covars COV_PRECOMP_DIR]
                                    [-d MAX_HS_DIST]
                                    [--min-hs-vars MIN_CLUST_VARS]
                                    [--blacklist BLACKLIST_FN]
                                    [--ap-iters AP_ITERS]
                                    [--ap-convits AP_CONVITS]
                                    [--ap-algorithm AP_ALG]
                                    genes.gtf vcfs_list.txt

positional arguments:
  genes.gtf             Input GTF file (Required).
  vcfs_list.txt         Input VCFs list file (Required). Required columns:
                        file path, sample name. NOTE: sample names must be
                        unique for each sample!

optional arguments:
  -h, --help            show this help message and exit
  -o OUTDIR, --outdir OUTDIR
                        Provide output directory for analysis. (default: ./)
  --prefix PREFIX       Provide prefix for analysis. (default:
                        mutation_enrichment)
  --gene-field GENEFIELD
                        Provide field name from input GTF containing gene
                        name/id information. (default: gene_id)
  -g GENE_LIST, --gene-list GENE_LIST
                        Provide list of genes to which analysis should be
                        restricted (one gene per-line in text file). Analysis
                        will only considers genes from GTF file that are
                        present in this list. Default behavior is to query all
                        coding genes present in input GTF. (default: None)
  --bg-vars-type BG_VARS_TYPE
                        Select which variants should be counted in background
                        rate calculations. Choices are: 'all' and 'silent'. If
                        'all' is selected, all variants (silent + non-silent)
                        are counted in background calculations. If 'silent' is
                        selected, only silent mutations count towards
                        background. (default: all)
  --maf MAF             Instead of VCF list file, provide MAF (mutation
                        annotation format) file with mutation information.
                        Gene information (e.g. lengths) are computed from
                        input GTF. Genes not present by genefield in GTF (read
```

```
                      from first column of MAF) are skipped. (default: None)
--exome-only          If using exome-based data, choose this flag to only
                      consider exonic coordinates of genes for background
                      estimates. Default behavior is to consider full gene
                      length (exons + introns) in calculations. (default:
                      False)
--anno-type TTYPE     Select annotation type for determining non-silent
                      somatic variants. Alternatively, provide tab-delimited
                      input text file describing terms for use. Valid
                      default options are: illumina, annovar. If providing
                      text file, must include one term per row with 3
                      columns: 1) String that is either "Gene" or "Effect"
                      to denote field with gene name or gene effect,
                      respectively; 2) value from VCF INFO field for code to
                      search for matching gene name or non-silent effect; 3)
                      valid terms (can be left blank for "Gene" row). If MAF
                      input is used, this option is ignored and default MAF
                      terms are used. (default: illumina)
-m MAP_REGIONS, --mappable-regions MAP_REGIONS
                      Provide BED file of mappable genomic regions (sorted
                      and tabix-indexed). If provided, only portions of
                      regions from input file overlapping these mappable
                      regions will be used in analsyis. Region lengths are
                      also adjusted for enrichment calculations. (default:
                      None)
-p NPROCESSORS, --processors NPROCESSORS
                      Set number of processors for parallel runs. (default:
                      1)
--snps-only           Set this flag to tell program to only consider SNPs in
                      analysis. Default is to consider all variant types.
                      (default: False)
-c COV_FN, --covariates-file COV_FN
                      Provide covariates file. Format is tab-delimited text
                      file, with first column listing gene name according to
                      gene_id field in input GTF. Header should contain
                      covariate names in columns 2 to end. (default: None)
-w WEIGHTS_FN, --covariate-weights WEIGHTS_FN
                      Provide covariates weight file. Format is tab-
                      delimited file (no header) with: covariate name,
                      weight. Weights are normalized to sum=1. If not
                      provided, uniform weighting of covariates is assumed.
                      (default: None)
--by-contig           Use this flag to perform clustering on genes by contig
                      (i.e. by chromosome). This speeds computation of gene
                      clusters. If not set, clusters are computed using all
                      genes in same run. (default: False)
--use-local           Use this flag to tell program to use local gene
                      background rate instead of global background rate. If
                      covariate files or pre-computed covariates supplied,
                      this option is ignored. (default: False)
--min-clust-size MIN_CLUST_SIZE
                      Set minimum number of covariate cluster members.
                      Regions belonging to a cluster with only itself or
                      less than this value are flagged and a local
                      background around the region is calculated and used
                      instead. (default: 3)
```

```
  --precomputed-covars COV_PRECOMP_DIR
                        Provide path to pre-computed covariate clusters for
                        regions in input BED file. (default: None)
  -d MAX_HS_DIST, --hotspot-distance MAX_HS_DIST
                        Set maximum distance between mutations for candidate
                        hotspot discovery. (default: 50)
  --min-hs-vars MIN_CLUST_VARS
                        Set minimum number of mutations that must be present
                        for a valid candidate hotspot. (default: 3)
  --blacklist BLACKLIST_FN
                        Provide a blacklist of specific variants to exclude
                        from analysis. Blacklist file format is tab-delimited
                        text file with four required columns: contig
                        (chromosome), position (1-indexed), reference base,
                        alternate base. (default: None)
  --ap-iters AP_ITERS   Set maximum number of AP iterations before re-
                        computing with alternate self-similarity. (default:
                        1000)
  --ap-convits AP_CONVITS
                        Set number of convergence iterations for AP runs (i.e.
                        if exemplars remain constant for this many iterations,
                        terminate early). This value MUST be smaller than the
                        total number of iterations. (default: 50)
  --ap-algorithm AP_ALG
                        Select between one of two versions of AP clustering
                        algorithm: 'slow' or 'fast'. The 'fast' version is
                        faster in terms of runtime but consumes more memory
                        than 'slow'. (default: fast)
```

- mutEnricher.py noncoding

```
usage: python mutEnricher.py noncoding [-h] [-o OUTDIR] [--prefix PREFIX]
                                       [-m MAP_REGIONS] [-p NPROCESSORS]
                                       [--snps-only] [-c COV_FN]
                                       [-w WEIGHTS_FN] [--use-local]
                                       [--min-rclust-size MIN_RCLUST_SIZE]
                                       [--precomputed-covars COV_PRECOMP_DIR]
                                       [-d MAX_HS_DIST]
                                       [--min-hs-vars MIN_CLUST_VARS]
                                       [--blacklist BLACKLIST_FN] [--no-wap]
                                       [--ap-iters AP_ITERS]
                                       [--ap-convits AP_CONVITS]
                                       [--ap-algorithm AP_ALG]
                                       regions.bed vcfs_list.txt

positional arguments:
  regions.bed           Input regions BED file (Required). Required columns:
                        contig, 0-based start, 1-based end. A name for the
                        region can be supplied in the 4th column.
  vcfs_list.txt         Input VCFs list file (Required). Required columns:
                        file path, sample name. NOTE: sample names must be
                        unique for each sample!

optional arguments:
  -h, --help            show this help message and exit
```

```
-o OUTDIR, --outdir OUTDIR
                      Provide output directory for analysis. (default: ./)
--prefix PREFIX       Provide prefix for analysis. (default:
                      mutation_enrichment)
-m MAP_REGIONS, --mappable-regions MAP_REGIONS
                      Provide BED file of mappable genomic regions (sorted
                      and tabix-indexed). If provided, only portions of
                      regions from input file overlapping these mappable
                      regions will be used in analsyis. Region lengths are
                      also adjusted for enrichment calculations. (default:
                      None)
-p NPROCESSORS, --processors NPROCESSORS
                      Set number of processors for parallel runs. (default:
                      1)
--snps-only           Set this flag to tell program to only consider SNPs in
                      analysis. Default is to consider all variant types.
                      (default: False)
-c COV_FN, --covariates-file COV_FN
                      Provide covariates file. Format is tab-delimited text
                      file, with first column listing regions in format
                      <contig>:<1-based start>-<1-based end> and remaining
                      columns with values. Header should contain covariate
                      names in columns 2 to end. (default: None)
-w WEIGHTS_FN, --covariate-weights WEIGHTS_FN
                      Provide covariates weight file. Format is tab-
                      delimited file (no header) with: covariate name,
                      weight. Weights are normalized to sum=1. If not
                      provided, uniform weighting of covariates is assumed.
                      (default: None)
--use-local           Use this flag to tell program to use local region
                      background rate instead of global background rate. If
                      covariate files or pre-computed covariates supplied,
                      this option is ignored. (default: False)
--min-rclust-size MIN_RCLUST_SIZE
                      Set minimum number of covariate cluster members.
                      Regions belonging to a cluster with only itself or
                      less than this value are flagged and a local
                      background around the region is calculated and used
                      instead. (default: 3)
--precomputed-covars COV_PRECOMP_DIR
                      Provide path to pre-computed covariate clusters for
                      regions in input BED file. (default: None)
-d MAX_HS_DIST, --hotspot-distance MAX_HS_DIST
                      Set maximum distance between mutations for candidate
                      hotspot discovery. (default: 50)
--min-hs-vars MIN_CLUST_VARS
                      Set minimum number of mutations that must be present
                      for a valid candidate hotspot. (default: 3)
--blacklist BLACKLIST_FN
                      Provide a blacklist of specific variants to exclude
                      from analysis. Blacklist file format is tab-delimited
                      text file with four required columns: contig
                      (chromosome), position (1-indexed), reference base,
                      alternate base. (default: None)
--no-wap              Select flag to skip weighted average proximity (WAP)
                      procedure. (default: False)
```

```
  --ap-iters AP_ITERS   Set maximum number of AP iterations before re-
                        computing with alternate self-similarity. (default:
                        1000)
  --ap-convits AP_CONVITS
                        Set number of convergence iterations for AP runs (i.e.
                        if exemplars remain constant for this many iterations,
                        terminate early). This value MUST be smaller than the
                        total number of iterations. (default: 50)
  --ap-algorithm AP_ALG
                        Select between one of two versions of AP clustering
                        algorithm: 'slow' or 'fast'. The 'fast' version is
                        faster in terms of runtime but consumes more memory
                        than 'slow'. (default: fast)
```

- **utilities/get_gene_covariates.py**

```
usage: python get_gene_covariates.py <GTF> <genome.fa> [options]

Create gene covariates file for genes in GTF from sequence features and
external data

positional arguments:
  genes.gtf             Input GTF file (Required).
  genome                Indexed genome fasta file (Required).

optional arguments:
  -h, --help            show this help message and exit
  -o OUTNAME, --outname OUTNAME
                        Provide output filename. (default:
                        ./gene_covariates.txt)
  -t TABLES, --table-files TABLES
                        Provide one or more additional table files with
                        covariate info for genes. Use as -t file1 -t file2 ...
                        -t fileN. Each table file must be tab-delimited with
                        one column header. First column is reserved for gene
                        name, remaining are for information. Names for each
                        covariate are read from corresponding header line.
                        Provided values are not adjusted and genes in GTF with
                        no information for one or more covariates are set to
                        "NA". (default: [])
  --repliseq-fns REPLISEQ_FNS
                        Provide a file with paths to RepliSeq data (for
                        replication timing information). This file should be
                        tab-delimited with no header in the format: file_path
                        sampleID. The file paths should point to bed/bedgraph
                        files compressed with bgzip and indexed with tabix.
                        This program will extract a replication timing value
                        from each file for each gene by scanning overlapping
                        intervals in the files. For multiple intersecting
                        intervals, the average value is taken. If no data
                        exists in the immediate gene vicinity, wider windows
                        around the gene are scanned until a value is
                        determined; otherwise, "None" is reported. (default:
                        None)
  -p NPROCESSORS, --processors NPROCESSORS
```

```
                        Set number of processors for parallel runs. (default:
                        1)
  -g GENE_LIST, --gene-list GENE_LIST
                        Provide list of genes to which analysis should be
                        restricted (one gene per-line in text file). Analysis
                        will only considers genes from GTF file that are
                        present in this list. Default behavior is to query all
                        coding genes present in input GTF. (default: None)
```

utilities/get_gene_covariates.py

```
usage: python get_region_covariates.py <regions.bed> <genome.fa> [options]

Create region covariates file for regions in BED file from sequence features
and external data

positional arguments:
  regions.bed           Input BED file of regions (Required).
  genome                Indexed genome fasta file (Required).

optional arguments:
  -h, --help            show this help message and exit
  -o OUTNAME, --outname OUTNAME
                        Provide output filename. (default:
                        ./region_covariates.txt)
  -t TABLES, --table-files TABLES
                        Provide one or more additional table files with
                        covariate info for regions. Use as -t file1 -t file2
                        ... -t fileN. Each table file must be tab-delimited
                        with one column header. First column is reserved for
                        region string, remaining are for information. Names
                        for each covariate are read from corresponding header
                        line. Provided values are not adjusted and regions in
                        BED with no information for one or more covariates are
                        set to "NA". (default: [])
  --repliseq-fns REPLISEQ_FNS
                        Provide a file with paths to RepliSeq data (for
                        replication timing information). This file should be
                        tab-delimited with no header in the format: file_path
                        sampleID. The file paths should point to bed/bedgraph
                        files compressed with bgzip and indexed with tabix.
                        This program will extract a replication timing value
                        from each file for each region by scanning overlapping
                        intervals in the files. For multiple intersecting
                        intervals, the average value is taken. If no data
                        exists in the immediate region vicinity, wider windows
                        around each are scanned until a value is determined;
                        otherwise, "None" is reported. (default: None)
  -p NPROCESSORS, --processors NPROCESSORS
                        Set number of processors for parallel runs. (default:
                        1)
```