**IVAC-easyfuse documentation**

03.12.2018; Urs Lahrmann

**Brief summary**

In the following, we describe a pipeline for the prediction of highly reliable (i.e. high precision, low sensitivity) peptide sequences surrounding a breakpoint of two “fused” RNA transcripts. Starting from a set of RNA-Seq reads, multiple different fusion prediction tools are called in order to generate a set of fusion candidates. The candidates from the different tools are then merged and further analyzed and annotated. Finally, a predictive model is applied in order to rank the fusions based on their reliability.

**Short descriptions of the pipeline classes**

**processing.py**

This is the main script of the pipeline and its default starting point. Provided fastq files are paired (single end processing is not supported) and sample ids created from their filenames (or from the patient identifier listed in the demux\_stats.csv in case the “icam\_run” flag is used). For each sample, an output folder tree and command line strings for all programs based on the selection of programs and defined resources in the config are created. Processing of the individual samples is performed in individual slurm jobs. Furthermore, samples are “registered” to the "samples.csv" file for progress monitoring.

Parameter list:

|  |  |  |
| --- | --- | --- |
| Parameter | Description | Mandatory? |
| -i / --input | Path to fastq folder or fastq file to process | Y |
| -o / --output | Path to the folder where results shall be generated | Y |
| -u / --userid | User identifier for slurm and stats timing | Y |
| -c / --config | Path to the config file (see config.ini) | Y |
| --partitions | Partition list for queueing via slurm | N |
| --tool\_support | Minimal number of tools which must predict a fusion event in order to consider it as correct | N |
| --read\_filter | Whether pre-filtering of likely chimeric reads shall be performed in advance | N |
| --version | Return the version string and exit (see version.py) | N |
| --icam\_run | Triggers processing tailored to icam data (see desc above) | N |
| --overwrite | Force overwriting of existing results | N |

**iomethods.py**

This is just a container for different class/static methods which are required by different scripts in the pipeline. These methods are used to:

1) Create folder

2) Grant specific permissions for files/directories (superfluous for any windows file system)

3) Get fastq files and a derived sample id from a specified input directory

4) Get icam data from searching a directory tree

5) Get tumor content estimates from mymut log files (currently not in use)

6) Return the basename of a file path string

**scheduling.py**

This script is just a container for different static methods which are required by different scripts in the pipeline. These methods are used to:

1) Create an sbatch script and sending it to slurm

2) Execute a command w/o slurm on the shell with a call to subprocess

3) Like (2) but with keeping the stdout in a variable

4) Retrieving a list of currently running job ids based on a lookup of a specified job name

**logger.py**

This script provides convenience methods for writing to a log file. Each log record is prefixed with a timestamp (YYYY-MM-DD HH:MM:SS) and one of four log-level string (0:NONE, 1:DEBUG, 2: INFO, 3:ERROR). When running the easyfuse pipeline, a single log file will be created for all samples during the analysis setup (see processing.py) which contains the overall processing information. During data post processing (see fetchdata.py) another log file will be generated for each sample individually. Besides these log files which contain information specific to easyfuse, all output (standard and error) from the independent slurm scripts (see scheduling.py) is further stored in their separate logs.

**fetchdata.py**

Like the main processing script (processing.py), the main function of this script is to prepare and organize in- and outputs of different analysis scripts: (1) Fusion prediction parsing and collection (see fusiontoolparser.py), (2) fusion sequence generation and annotation (see GetFusionSequence.R), (3) re-quantification of the fusion prediction (see requantify.py) and (4) de novo assembly of the (partial) fusion transcript (see denovoassembly.py)

Parameter list:

|  |  |  |
| --- | --- | --- |
| Parameter | Description | Mandatory? |
| -i / --input | Path to the results folder of a sample | Y |
| -s / --sample | Sample identifier | Y |
| -c / --config | Path to the config file (see config.ini) | Y |
| --fq1 | First read file (R1) for re-quantification analysis | N |
| --fq2 | Second read file (R1) for re-quantification analysis | N |
| --tool\_support | Minimal number of tools which must predict a fusion event in order to consider it as correct | N |

**liftover.py**

**fusionreadfilter.py**

Reads line wise through a name-sorted bam file and classify each read pair as either (1) chimeric as defined by star, (2) multi-mapped, (3) unmapped, (4) discordantly mapped, (5) soft-clipped or (6) normal. If the pair is classified as (1|3|4|5) it is written to an output bam file. The counts of each group are recorded and written to a log file, together with two QC values indicating that all lines have been processed as expected, once processing is completed

|  |  |  |
| --- | --- | --- |
| Parameter | Description | Mandatory? |
| -i / --input | Path to a star-chimeric-aligned, name-sorted, unmapped-included bam file | Y |
| -o / --output | Prefix of the output bam containing filtered alignments only | Y |

**fusiontoolparser.py**

This script parses the individual outputs of different fusion prediction tools and writes the Detected\_Fusions.csv file. This contains the easyfuse fusion identifier (FGID), the fusion pair, the genomic breakpoint positions, the number of junction reads supporting the fusion, the number of spanning reads supporting the fusion, the sample id and the tool which predicted the fusion. Because every prediction tool has its specific output format, each prediction tool has its own parser. Currently implemented are parser for: Fusioncatcher, STAR-Fusion, MapSplice2, Starchip, Infusion and Pizzly.

|  |  |  |
| --- | --- | --- |
| Parameter | Description | Mandatory? |
| -i / --input | Path to the “scratch” folder of a sample | Y |
| -o / --output | Path to the output folder where results shall be saved | Y |
| -s / --sample | Sample identifier | Y |
| -t / --toolcutoff | Minimal number of tools which must predict a fusion event in order to consider it as correct | Y |
| -f / --fusionlist | A list of fusion prediction tools that where run on the sample | Y |
| -l / --logger | Path to the log file | Y |

**requantify.py**

Reads line wise through a coordinate-sorted bam file and classify each primary read pair as either (1) junction\_ft / junction\_wt if either one of the reads is overlapping the breakpoint region on the fusion transcript or on the respective wildtype background (2) spanning\_ft / spanning\_wt if the reads are mapping to different sides of the breakpoint without overlapping the breakpoint itself.

|  |  |  |
| --- | --- | --- |
| Parameter | Description | Mandatory? |
| -i / --input | Path to a coordinate-sorted bam file | Y |
| -o / --output | Path to an output file where results shall be written to | Y |
| -d / --bp\_distance | Threshold of bases around the breakpoint for junction/spanning counting | N |

**denovoassembly.py**

\*\*\* so far, this script is not automatically called within the pipeline and needs to be started separately \*\*\*

For each fusion pair, a local de novo assembly of all reads mapping to both fusion genes is performed and the assembled transcripts afterwards compared against a blast database of all human transcriptome sequences. Finally, blast results are parsed in order to separate the fusion transcripts from the wildtype background. For fast access of gene boundaries, a pre-parsed gtf file is required which has to be generated once with reformatgtf.py. For fast parsing of the blast output, the reference blast database must be created from a pre-parsed fasta file which has to be generated once with reformatfasta.py.

|  |  |  |
| --- | --- | --- |
| Parameter | Description | Mandatory? |
| -i / --input | Input list of fusion genes | Y |
| -c / --config | Path to the config file (see config.ini) | Y |
| -b / --bam | The bam file of the first mapping containing all reads | Y |
| -o / --outdir | Path to the output directory | Y |

**reformatgtf.py**

Helper script for denovoassembly.py to generate a file providing fast access to gene coordinates. The input must be a gtf containing the hgnc gene symbol as “gene\_name” in the comments field (e.g. ensembl gtf). The output file than contains the gene symbol, chromosome id, gene start and gene stop in a tab-delimited form.

**reformatfasta.py**

Helper script for denovoassembly.py. The script must be run on an ensembl whole transcriptome fasta file. The header of each transcript is changed into the following format: “>(gene\_symbol)\_(ensembl\_gene\_id)\_(ensemble\_transcript\_id)” (w/o the brackets)

**fusionranking.py**

**GTFtoCSV.R**

Helper script to generate csv file with gene annotation information from a gtf file (e.g. ensemble gene annotation file, can be obtained from <https://www.ensembl.org/info/data/ftp/index.html>). The generated csv file can be pre-generated and is a required input for GetFusionSequence.R.

**GetFusionSequence.R**

|  |  |  |
| --- | --- | --- |
| Parameter | Description | Mandatory? |
| -i / --input | Input list of fusion genes | Y |
| -f / --fasta\_genome\_dir | Path to the fasta files containing genomic sequences (e.g. hg38) | Y |
| -e / --ensembl\_csv | csv file with gene annotation information | Y |
| -o / --outout\_file | Path to output file | Y |
| --cis\_near\_distance | Maximal distance to annotate a fusion gene as cis\_near (default = 1000000) | N |
| --genomic\_seq\_len | Genomic sequence that is obtained for annotation of breakpoints outside of genes | N |
| -- context\_seq\_len | around the fusion breakpoint | N |

This script annotates predicted fusion breakpoints based on a given gene annotation file and generates the exon structure of all potential fusion transcripts. Furthermore, the sequence for the fusion transcript is obtained and translated into the peptide. The output is saved in a csv file and fasta files containing transcript, context and peptide sequences.

**model\_prediction.R**

|  |  |  |
| --- | --- | --- |
| Parameter | Description | Mandatory? |
| -i / --input | Input list of fusion genes | Y |
| -c / --context\_seq\_file | Path to the context csv file containing annotation information | Y |
| -q / --requantification\_file | file containing information on junction and spanning reads per context sequence | Y |
| -qc / --qc\_table\_file | Path to fastqc output table | Y |
| -t / --tool\_state\_file | Containing information on which fusion detection tools were used | Y |
| -m / --model\_file | File containing the prediction model that should be applied | Y |
| -o / --outout\_file | Path to output file | Y |

This script reads in all available information on predicted fusion events and applies a prediction model to assign for each event and prediction value and a prediction status (positive/negative) based on a given PPV threshold.

**stats.py**

**config.py**

This script provides methods to retrieve information from the main configuration file (see config\_default.ini).

**config\_default.ini**

This file contains configuration parameters required for the proper function of all classes. The parameters are organized in blocks with their respective header in square brackets. During a normal run, the user eventually needs to update information in the “general” block only. When the pipeline is being setup the first time, all blocks should at least be revised for their correctness.

1. [general]

The used defines here which tools to run within the pipeline and which versions of references to use. Currently, only ensembl and hg38 are listed in the default config, but any other combination will also work, like “ucsc, mm10” for mouse data or “my\_genome, my\_transcriptome” for individual references.

1. [resources]

The number of CPUs and the amount of RAM reserved by slurm for the individual processes is defined here.

1. [commands]

The command string to call a program (this is actually the string one would also use to call a program in the terminal

1. [references]

Full path to references file, like genome/transcriptome fasta sequences and the gene annotations

1. [indices]

Full path to genome indices required by individual programs

1. [otherFiles]

Full path to program specific config files

1. [easyfuse\_helper]

Full path to some helper scripts/files which are required for the pipeline itself but not directly for the fusion prediction

**samples.py**

**statscollect.py**

**dependency\_versions.txt**

**versioncontrol.py**

**version.py**

Exit code 99 is used for user input errors

Exit code 100 is used for Slurm and subprocess call errors