HTSinfer

Release 0.10.0

Zavolan Lab

Feb 06, 2024

MODULES

CHAPTER

ONE

HTSINFER

1.1 htsinfer package

HTSinfer project root

1.1.1 Submodules

1.1.2 htsinfer.cli module

Command-line interface client.

htsinfer.cli.**main**() → None

Entry point for CLI executable.

htsinfer.cli.**parse_args**() → Namespace

Parse CLI arguments.

Returns
Parsed CLI arguments.

htsinfer.cli.**setup_logging**(*verbosity: str = 'INFO'*) → None

Configure logging.

Parameters
verbosity – Level of logging verbosity.

1.1.3 htsinfer.exceptions module

Custom exceptions.

exception htsinfer.exceptions.**CutadaptProblem**

Bases: Exception

Exception raised when running cutadapt commands.

exception htsinfer.exceptions.**FileProblem**

Bases: Exception

Exception raised when file could not be opened or parsed.

exception htsinfer.exceptions.**InconsistentFastqIdentifiers**

Bases: Exception

Exception raised when inconsistent FASTQ sequence identifiers were ecountered.

exception htsinfer.exceptions.**KallistoProblem**

Bases: Exception

Exception raised when running kallisto index and quant commands.

exception htsinfer.exceptions.**MetadataWarning**

Bases: Exception

Exception raised when metadata could not be determined.

exception htsinfer.exceptions.**SamFileProblem**

Bases: Exception

Exception raised when an invalid sam file is encountered.

exception htsinfer.exceptions.**StarProblem**

Bases: Exception

Exception raised when running STAR index and quant commands.

exception htsinfer.exceptions.**TranscriptsFastaProblem**

Bases: Exception

Exception raised when an invalid transcripts fasta file is passed.

exception htsinfer.exceptions.**UnknownFastqIdentifier**

Bases: Exception

Exception raised when a FASTQ sequence identifier of unknown format was ecountered.

exception htsinfer.exceptions.**UnsupportedSampleSourceException**

Bases: Exception

Exception raised when taxonomy ID is not supported.

exception htsinfer.exceptions.**WorkEnvProblem**

Bases: Exception

Exception raised when the work environment could not be set up or cleaned.

1.1.4 htsinfer.get_library_source module

Infer library source from sample data.

class htsinfer.get_library_source.**GetLibSource**(*config:* Config)

Bases: object

Determine the source of FASTQ sequencing of a single- or paired-end seguencing library.

Parameters
config – Container class for all arguments used in inference and results produced by the class.

Attrubutes:

paths: Tuple of one or two paths for single-end and paired end library
files.

transcripts_file: File path to an uncompressed transcripts file in
FASTA format. Expected to contain |-separated sequence identifier lines that contain an organism short name and a taxon identifier in the fourth and fifth columns, respectively. Example sequence identifier: *rpl-13|ACYPI006272|ACYPI006272-RA|apisum|7029*

out_dir: Path to directory where output is written to. tmp_dir: Path to directory where temporary output is written to. min_match_pct: Minimum percentage of reads that are consistent with a

given source in order for it to be considered as the to be considered the library's source.

min_freq_ratio: Minimum frequency ratio between the first and second
most frequent source in order for the former to be considered the library's source.

tax_id: Taxonomy ID of the sample source.

`create_kallisto_index`() → Path

Build Kallisto index from FASTA file of target sequences.

Returns
Path to Kallisto index.

Raises
KallistoProblem – Kallisto index could not be created.

`evaluate`() → *ResultsSource*

Infer read source.

Returns
Source results object.

`get_source`(*fastq: Path*, *index: Path*) → *Source*

Determine source of a single sequencing library file.

Parameters

- **fastq** – Path to FASTQ file.
- **index** – Path to Kallisto index.

Returns
Source of library file.

`static get_source_expression`(*kallisto_dir: Path*) → DataFrame

Return percentages of total expression per read source.

Parameters
kallisto_dir – Directory containing Kallisto quantification results.

Returns

Data frame with columns *source_ids* (a tuple of source short name
and taxon identifier, e.g., *("hsapiens", 9606)*) and *tpm*, signifying the percentages of total expression per read source. The data frame is sorted by total expression in descending order.

Raises
FileProblem – Kallisto quantification results could not be processed.

`static get_source_name`(*taxon_id: int*, *transcripts_file: Path*) → str

Return name of the source organism, based on tax ID.

Parameters

- **taxon_id** – Taxonomy ID of a given organism.
- **transcripts_file** – Path to FASTA file containing transcripts.

Returns
Short name of the organism belonging to the given tax ID.

Raises
- *FileProblem* – Could not process input FASTA file.
- *UnsupportedSampleSourceException* – Taxon ID is not supported.

run_kallisto_quantification(*fastq: Path*, *index: Path*) → Path

Run Kallisto quantification on individual sequencing library file.

Parameters
- **fastq** – Path to FASTQ file.
- **index** – Path to Kallisto index.

Returns
Path to output directory.

Raises
KallistoProblem – Kallisto quantification failed.

1.1.5 htsinfer.get_library_stats module

Infer read orientation from sample data.

class htsinfer.get_library_stats.**GetLibStats**(*config:* Config)

Bases: object

Determine library statistics of a single- or paired-end sequencing library.

Parameters
config – Container class for all arguments used in inference and results produced by the class.

paths

Tuple of one or two paths for single-end and paired end library files.

tmp_dir

Path to directory where temporary output is written to.

evaluate() → *ResultsStats*

Infer read statistics.

Returns
Statistics results object.

static fastq_get_stats_read_length(*fastq: Path*) → Tuple[int, int, float, int, int]

Get number of records in a FASTQ file.

Parameters
fastq – Path to FASTQ file.

Returns
Tuple of minimum and maximum read lengths in input file.

Raises
FileProblem – Could not process FASTQ file.

1.1.6 htsinfer.get_library_type module

Infer mate information from sample data.

class htsinfer.get_library_type.**GetFastqType**(*path: Path*)

Bases: object

Determine type (single/paired) information for an individual FASTQ sequencing library.

Parameters
path – File path to read library.

path

File path to read library.

seq_ids

List of sequence identifier prefixes of the provided read library, i.e., the fragments up until the mate information, if available, as defined by a named capture group `prefix` in a regular expression to extract mate information.

seq_id_format

The sequence identifier format of the read library, as identified by inspecting the first read and matching one of the available regular expressions for the different identifier formats.

result

The current best guess for the type of the provided library.

Examples

```
>>> lib_type = GetFastqType(
...     path="tests/files/first_mate.fastq"
... ).evaluate()
<OutcomesType.first_mate: 'first_mate'>
```

evaluate() → None

Decide library type.

Raises
NoMetadataDetermined – Type information could not be determined.

class htsinfer.get_library_type.**GetLibType**(*config:* Config, *mapping:* Mapping)

Bases: object

Determine type (single/paired) information for a single or a pair of
FASTQ sequencing libraries.

Args:

config: Container class for all arguments used in inference
and results produced by the class.

Attributes:
path_1: Path to single-end library or first mate file. path_2: Path to second mate file. results: Results container for storing library type information for

the provided files, as well as the mate relationship between the two files, if applicable.

Examples:

```
>>> GetLibType(
...     path_1="tests/files/first_mate.fastq"
... ).evaluate()
ResultsType(file_1=<OutcomesType.single: 'single'>, file_2=<OutcomesTyp
```

e.not_available: 'not_available'>, relationship=<OutcomesTypeRelationship.not_a vailable: 'not_available'>)

```
>>> GetLibType(
...     path_1="tests/files/first_mate.fastq",
...     path_2="../tests/test_files/second_mate.fastq",
... ).evaluate()
ResultsType(file_1=<OutcomesType.first_mate: 'first_mate'>, file_2=<Out
```

comesType.second_mate: 'second_mate'>, relationship=<OutcomesTypeRelationship.s plit_mates: 'split_mates'>)

('first_mate', 'second_mate', 'split_mates')

class AlignedSegment

Bases: object

Placeholder class for mypy "Missing attribute" error in _compare_alignments(), the actual object used is pysam.AlignedSegment class.

evaluate() → None

Decide type information and mate relationship.

1.1.7 htsinfer.get_read_layout module

Infer adapter sequences present in reads.

class htsinfer.get_read_layout.**GetAdapter3**(*path: Path*, *adapter_file: Path*, *out_dir: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/htsinfer/checkouts/sta*
min_match_pct: float = 0.1, *min_freq_ratio: float = 2*)

Bases: object

Determine 3' adapter sequence for an individual FASTQ library.

Parameters

- **path** – File path to read library.
- **adapter_file** – Path to text file containing 3' adapter sequences (one sequence per line) to scan for.
- **out_dir** – Path to directory where output is written to.
- **min_match_pct** – Minimum percentage of reads that contain a given adapter Minimum percentage of reads that contain a given adapter sequence in order for it to be considered as the library's 3'-end adapter.
- **min_freq_ratio** – Minimum frequency ratio between the first and second most frequent adapter in order for the former to be considered as the library's 3'-end adapter.

path

File path to read library.

adapter_file

Path to text file containing 3' adapter sequences (one sequence per line) to scan for.

out_dir

Path to directory where output is written to.

min_match_pct

Minimum percentage of reads that contain a given adapter Minimum percentage of reads that contain a given adapter sequence in order for it to be considered as the library's 3'-end adapter.

min_freq_ratio

Minimum frequency ratio between the first and second most frequent adapter in order for the former to be considered as the library's 3'-end adapter.

adapters

List of adapter sequences.

trie

Trie data structure of adapter sequences.

adapter_counts

Dictionary of adapter sequences and corresponding count percentages.

result

The most frequent adapter sequence in FASTQ file.

Examples

```
>>> GetAdapter3(
...     path_1="tests/files/sra_sample_2.fastq",
...     adapter_file="data/adapter_fragments.txt",
... ).evaluate()
<"AAAAAAAAAAAAAA">
```

evaluate() → None

Search for adapter sequences and validate result confidence constraints.

class htsinfer.get_read_layout.**GetReadLayout**(*config:* Config)

Bases: object

Determine the adapter sequence present in the FASTQ sequencing libraries.

Parameters

config – Container class for all arguments used in inference and results produced by the class.

path_1

Path to single-end library or first mate file.

path_2

Path to second mate file.

adapter_file

Path to text file containing 3' adapter sequences (one sequence per line) to scan for.

out_dir

Path to directory where output is written to.

min_match_pct

Minimum percentage of reads that contain a given adapter sequence in order for it to be considered as the library's 3'-end adapter.

min_freq_ratio

Minimum frequency ratio between the first and second most frequent adapter in order for the former to be considered as the library's 3'-end adapter.

results

Results container for storing adapter sequence information for the provided files.

Examples

```
>>> GetReadLayout(
...     path_1="tests/files/sra_sample_2.fastq",
...     adapter_file="data/adapter_fragments.txt",
... ).evaluate()
ResultsLayout(
    file_1=<Layout().adapt_3: "AAAAAAAAAAAAAAA">,
    file_2=<Layout().adapt_3: None>,
)
>>> GetReadLayout(
...     path_1="tests/files/sra_sample_1.fastq",
...     path_2="tests/files/sra_sample_2.fastq",
...     adapter_file="data/adapter_fragments.txt",
...     min_match_pct=2,
...     min_freq_ratio=1,
... ).evaluate()
ResultsLayout(
    file_1=<Layout().adapt_3: "AAAAAAAAAAAAAAA">,
    file_2=<Layout().adapt_3: "AAAAAAAAAAAAAAA">,
)
```

evaluate() → None

Decide adapter sequence.

get_poly_a(*file=PosixPath('.')*) → float

Run cutadapt and parse report

1.1.8 htsinfer.get_read_orientation module

Infer read orientation from sample data.

class htsinfer.get_read_orientation.**GetOrientation**(*config:* Config, *mapping:* Mapping)

Bases: object

Determine library strandedness and relative read orientation of a single- or paired-end seguencing library.

Parameters

config – Container class for all arguments used in inference and results produced by the class.

paths

Tuple of one or two paths for single-end and paired end library files.

library_type

ResultsType object with library type and mate relationship.

library_source

ResultsSource object with source information on each library file.

transcripts_file

File path to an uncompressed transcripts file in FASTA format.

tmp_dir

Path to directory where temporary output is written to.

threads_star

Number of threads to run STAR with.

min_mapped_reads

Minimum number of mapped reads for deeming the read orientation result reliable.

min_fraction

Minimum fraction of mapped reads required to be consistent with a given read orientation state in order for that orientation to be reported. Must be above 0.5.

evaluate() → *ResultsOrientation*

Infer read orientation.

Returns
Orientation results object.

static get_frequencies(**items: Any*) → Dict[Any, float]

Get frequencies of arguments as fractions of the number of all arguments.

Parameters
***items** – Items to get frequencies for.

Returns
Dictionary of arguments and their frequencies.

process_alignments(*star_dirs: List[Path]*) → *ResultsOrientation*

Determine read orientation of one or two single-ended or one paired-end sequencing library.

Parameters
star_dirs – List of one or two paths to STAR output directories.

Returns
Read orientation state of library or libraries.

process_paired(*sam: Path*) → *ResultsOrientation*

Determine read orientation of a paired-ended sequencing library.

Parameters
sam – Path to SAM file.

Returns

Read orientation state of each mate and orientation state
relationship of library.

process_single(*sam: Path*) → *StatesOrientation*

Determine read orientation of a single-ended sequencing library. :param sam: Path to SAM file.

Returns
Read orientation state of library.

Raises
Sam file could not be processed. –

static sum_dicts(**dicts: Dict[Any, float]*) → Dict[Any, float]

Sum of dictionaries with numeric values.

Parameters
***dicts** – Dictionaries to sum up.

Returns
Dictionary with union of keys of input dictionaries and all values added up.

1.1.9 htsinfer.htsinfer module

Main module.

class htsinfer.htsinfer.**HtsInfer**(*config:* Config)

Bases: object

Determine sequencing library metadata.

Parameters
config – Container class for all arguments used in inference and results produced by the class.

config

Container class for all arguments used in inference and results produced by the class.

run_id

Random string identifier for HTSinfer run.

state

State of the run; one of *RunStates*.

clean_up()

Clean up work environment.

evaluate()

Determine library metadata.

get_library_source() → *ResultsSource*

Determine library source.

Returns
Library source results.

get_library_stats()

Determine library statistics.

get_library_type()

Determine library type.

get_read_layout()

Determine read layout.

get_read_orientation()

Determine read orientation.

prepare_env()

Set up work environment.

print()

Print results to STDOUT.

process_inputs()

Process and validate inputs.

1.1.10 htsinfer.mapping module

Mapping FASTQ's and managing the outputs of STAR.

class htsinfer.mapping.**Mapping**(*config:* Config)

Bases: object

Map FASTQ file/s and manage outputs.

Parameters

path – Path to FASTQ file.

path_1

Path to single-end library or first mate file.

path_2

Path to second mate file.

Raises

FileProblem – The input file could not be parsed or the output file could not be written.

create_star_index(*fasta: Path*, *chr_bin_bits: int = 18*, *index_string_size: int = 5*) → Path

Prepare STAR index.

Parameters

- **fasta** – Path to FASTA file of sequence records to create index from.
- **index_string_size** – Size of SA pre-indexing string, in nucleotides.

Returns

Path to directory containing STAR index.

Raises

StarProblem – STAR index could not be created.

evaluate()

Align FASTQ files to reference transcripts with STAR.

static generate_star_alignments(*commands: Dict[Path, List[str]]*) → None

Align reads to index with STAR.

Parameters

commands – Dictionary of output paths and corresponding STAR commands.

Raises

StarProblem – Generating alignments failed.

static get_fasta_size(*fasta: Path*) → int

Get size of FASTA file in total nucleotides.

Parameters
fasta – Path to FASTA file.

Returns
Total number of nucleotides of all records.

Raises
FileProblem – Could not open FASTA file for reading.

static get_star_chr_bin_bits(*ref_size: int*, *transcripts: Path*) → int

Get size of bins for STAR genome storage.

Parameters

- **ref_size** – Size of genome/transcriptome reference in nucleotides.
- **transcripts** – Path to filtered FASTA transcripts file.

Returns
Number of bins for genome storage.

static get_star_index_string_size(*ref_size: int*) → int

Get length of STAR SA pre-indexing string.

Cf. https://github.com/alexdobin/STAR/blob/51b64d4fafb7586459b8a61303e40beceeead8c0/doc/STARmanual.pdf

Parameters
ref_size – Size of genome/transcriptome reference in nucleotides.

Returns
Size (in nucleotides) of SA pre-indexing string.

prepare_star_alignment_commands(*index_dir: Path*) → Dict[Path, List[str]]

Prepare STAR alignment commands.

Input FASTQ files are assumed to be sorted according to reference names or coordinates, the order of input reads is kept with the option "PairedKeepInputOrder", no additional sorting of aligned reads is done.

Parameters
index_dir – Path to directory containing STAR index.

Returns
Dictionary of output paths and corresponding STAR commands.

subset_transcripts_by_organism() → Path

Filter FASTA file of transcripts by current sources.

The filtered file contains records from the indicated sources.
Typically, this is one source. However, for if two input files were supplied that are originating from different sources (i.e., not from a valid paired-ended library), it may be from two different sources. If no source is supplied (because it could not be inferred), no filtering is done.

Returns
Path to filtered FASTA file.

Raises
FileProblem – Could not open input/output FASTA file for reading/writing.

1.1.11 htsinfer.models module

Data models.

class htsinfer.models.**Args**(*, *path_1: Path = PosixPath('.')*, *path_2: Path | None = None*, *out_dir: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/htsinfer/checkouts/stable/docs/api/results_*, *tmp_dir: Path = PosixPath('/tmp/tmp_htsinfer')*, *cleanup_regime: CleanupRegimes = CleanupRegimes.DEFAULT*, *records: int = 1000000*, *threads: int = 1*, *tax_id: int | None = None*, *transcripts_file: Path = PosixPath('.')*, *read_layout_adapter_file: Path = PosixPath('.')*, *read_layout_min_match_pct: float = 0.1*, *read_layout_min_freq_ratio: float = 2*, *lib_source_min_match_pct: float = 2*, *lib_source_min_freq_ratio: float = 2*, *lib_type_max_distance: int = 1000*, *lib_type_mates_cutoff: float = 0.85*, *read_orientation_min_mapped_reads: int = 20*, *read_orientation_min_fraction: float = 0.75*, *path_1_processed: Path = PosixPath('.')*, *path_2_processed: Path | None = None*, *t_file_processed: Path = PosixPath('.')*)

Bases: `BaseModel`

Configuration model for CLI arguments.

Parameters

- **path_1** – Path to single-end library or first mate file.
- **path_2** – Path to second mate file.
- **out_dir** – Path to directory where output is written to.
- **tmp_dir** – Path to directory where temporary output is written to.
- **cleanup_regime** – Which data to keep after run concludes; one of *CleanupRegimes*.
- **records** – Number of input file records to process; set to *0* to process all records.
- **threads** – Number of threads to run STAR with.
- **tax_id** – Taxonomy ID of the sample source.
- **transcripts_file** – File path to transcripts FASTA file.
- **read_layout_adapter_file** – Path to text file containing 3' adapter sequences to scan for (one sequence per line).
- **read_layout_min_match_pct** – Minimum percentage of reads that contain a given adapter in order for it to be considered as the library's 3'-end adapter.
- **read_layout_min_freq_ratio** – Minimum frequency ratio between the first and second most frequent adapter in order for the former to be considered as the library's 3'-end adapter.
- **lib_source_min_match_pct** – Minimum percentage of reads that are consistent with a given source in order for it to be considered as the to be considered the library's source.
- **lib_source_min_freq_ratio** – Minimum frequency ratio between the first and second most frequent source in order for the former to be considered the library's source.
- **read_orientation_min_mapped_reads** – Minimum number of mapped reads for deeming the read orientation result reliable.
- **read_orientation_min_fraction** – Minimum fraction of mapped reads required to be consistent with a given read orientation state in order for that orientation to be reported. Must be above 0.5.

`path_1`

Path to single-end library or first mate file.

Type
pathlib.Path

`path_2`

Path to second mate file.

Type
pathlib.Path | None

`out_dir`

Path to directory where output is written to.

Type
pathlib.Path

`run_id`

Random string identifier for HTSinfer run.

`tmp_dir`

Path to directory where temporary output is written to.

Type
pathlib.Path

`cleanup_regime`

Which data to keep after run concludes; one of *CleanupRegimes*.

Type
htsinfer.models.CleanupRegimes

`records`

Number of input file records to process.

Type
int

`threads`

Number of threads to run STAR with.

Type
int

`transcripts_file`

File path to transcripts FASTA file.

Type
pathlib.Path

`read_layout_adapter_file`

Path to text file containing 3' adapter sequences to scan for (one sequence per line).

Type
pathlib.Path

`read_layout_min_match_pct`

Minimum percentage of reads that contain a given adapter in order for it to be considered as the library's 3'-end adapter.

Type
float

`read_layout_min_freq_ratio`

Minimum frequency ratio between the first and second most frequent adapter in order for the former to be considered as the library's 3'-end adapter.

Type
float

`lib_source_min_match_pct`

Minimum percentage of reads that are consistent with a given source in order for it to be considered as the to be considered the library's source.

Type
float

`lib_source_min_freq_ratio`

Minimum frequency ratio between the first and second most frequent source in order for the former to be considered the library's source.

Type
float

`lib_type_max_distance`

Upper limit on the difference in the reference sequence coordinates between the two mates to be considered as coming from a single fragment. (Used only when sequence identifiers do not match)

Type
int

`lib_type_mates_cutoff`

Minimum fraction of mates that can be mapped to compatible loci and are considered concordant pairs / all mates. (Used only when sequence identifiers do not match)

Type
float

`read_orientation_min_mapped_reads`

Minimum number of mapped reads for deeming the read orientation result reliable.

Type
int

`read_orientation_min_fraction`

Minimum fraction of mapped reads required to be consistent with a given read orientation state in order for that orientation to be reported. Must be above 0.5.

Type
float

`path_1_processed`

Path to processed *path_1* file.

Type
pathlib.Path

`path_2_processed`

Path to processed *path_2* file.

Type
pathlib.Path | None

t_file_processed

Path to processed *transcripts_file* file.

Type
pathlib.Path

state

State of the run; one of *RunStates*.

results

Results container for storing determined library metadata.

cleanup_regime: *CleanupRegimes*

lib_source_min_freq_ratio: float

lib_source_min_match_pct: float

lib_type_mates_cutoff: float

lib_type_max_distance: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'cleanup_regime': FieldInfo(annotation=CleanupRegimes, required=False, default=<CleanupRegimes.DEFAULT: 'default'>), 'lib_source_min_freq_ratio': FieldInfo(annotation=float, required=False, default=2), 'lib_source_min_match_pct': FieldInfo(annotation=float, required=False, default=2), 'lib_type_mates_cutoff': FieldInfo(annotation=float, required=False, default=0.85), 'lib_type_max_distance': FieldInfo(annotation=int, required=False, default=1000), 'out_dir': FieldInfo(annotation=Path, required=False, default=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/htsinfer/checkouts/stable/docs/api/results_htsinfer')), 'path_1': FieldInfo(annotation=Path, required=False, default=PosixPath('.')), 'path_1_processed': FieldInfo(annotation=Path, required=False, default=PosixPath('.')), 'path_2': FieldInfo(annotation=Union[Path, NoneType], required=False), 'path_2_processed': FieldInfo(annotation=Union[Path, NoneType], required=False), 'read_layout_adapter_file': FieldInfo(annotation=Path, required=False, default=PosixPath('.')), 'read_layout_min_freq_ratio': FieldInfo(annotation=float, required=False, default=2), 'read_layout_min_match_pct': FieldInfo(annotation=float, required=False, default=0.1), 'read_orientation_min_fraction': FieldInfo(annotation=float, required=False, default=0.75), 'read_orientation_min_mapped_reads': FieldInfo(annotation=int, required=False, default=20), 'records': FieldInfo(annotation=int, required=False, default=1000000), 't_file_processed': FieldInfo(annotation=Path, required=False, default=PosixPath('.')), 'tax_id': FieldInfo(annotation=Union[int, NoneType], required=False), 'threads': FieldInfo(annotation=int, required=False, default=1), 'tmp_dir': FieldInfo(annotation=Path, required=False, default=PosixPath('/tmp/tmp_htsinfer')), 'transcripts_file': FieldInfo(annotation=Path, required=False, default=PosixPath('.'))}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

out_dir: Path

path_1: Path

path_1_processed: Path

path_2: Path | None

path_2_processed: Path | None

read_layout_adapter_file: Path

read_layout_min_freq_ratio: float

read_layout_min_match_pct: float

read_orientation_min_fraction: float

read_orientation_min_mapped_reads: int

records: int

t_file_processed: Path

tax_id: int | None

threads: int

tmp_dir: Path

transcripts_file: Path

class htsinfer.models.**CleanupRegimes**(*value*)

Bases: Enum

Enumerator of cleanup regimes.

DEFAULT = 'default'

KEEP_ALL = 'keep_all'

KEEP_NONE = 'keep_none'

KEEP_RESULTS = 'keep_results'

class htsinfer.models.**Config**(*, *args: ~htsinfer.models.Args = Args(path_1=PosixPath('.'), path_2=None, out_dir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/htsinfer/checkouts/stable/*
tmp_dir=PosixPath('/tmp/tmp_htsinfer'), cleanup_regime=<CleanupRegimes.DEFAULT: 'default'>, records=1000000, threads=1, tax_id=None, transcripts_file=PosixPath('.'), read_layout_adapter_file=PosixPath('.'), read_layout_min_match_pct=0.1, read_layout_min_freq_ratio=2, lib_source_min_match_pct=2, lib_source_min_freq_ratio=2, lib_type_max_distance=1000, lib_type_mates_cutoff=0.85, read_orientation_min_mapped_reads=20, read_orientation_min_fraction=0.75, path_1_processed=PosixPath('.'), path_2_processed=None, t_file_processed=PosixPath('.')), *results: ~htsinfer.models.Results = Results(library_stats=ResultsStats(file_1=Stats(read_length=ReadLength(min=None, max=None, mean=None, median=None, mode=None)), file_2=Stats(read_length=ReadLength(min=None, max=None, mean=None, median=None, mode=None))), library_source=ResultsSource(file_1=Source(short_name=None, taxon_id=None), file_2=Source(short_name=None, taxon_id=None)), library_type=ResultsType(file_1=<StatesType.not_available: None>, file_2=<StatesType.not_available: None>, relationship=<StatesTypeRelationship.not_available: None>), read_orientation=ResultsOrientation(file_1=<StatesOrientation.not_available: None>, file_2=<StatesOrientation.not_available: None>, relationship=<StatesOrientationRelationship.not_available: None>), read_layout=ResultsLayout(file_1=Layout(adapt_3=None, polyA_frac=None), file_2=Layout(adapt_3=None, polyA_frac=None)))*)

Bases: `BaseModel`

Configuration model for CLI arguments and inference results.

Parameters

- **args** – Container class for CLI arguments.
- **results** – Container class for aggregating results from the different inference functionalities.

args

Container class for CLI arguments.

Type
htsinfer.models.Args

results

Container class for aggregating results from the different inference functionalities.

Type
htsinfer.models.Results

args: ***Args***

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'args': FieldInfo(annotation=Args, required=False, default=Args(path_1=PosixPath('.'), path_2=None, out_dir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/htsinfer/checkouts/stable/docs/api/results_htsinfer'), tmp_dir=PosixPath('/tmp/tmp_htsinfer'), cleanup_regime=<CleanupRegimes.DEFAULT: 'default'>, records=1000000, threads=1, tax_id=None, transcripts_file=PosixPath('.'), read_layout_adapter_file=PosixPath('.'), read_layout_min_match_pct=0.1, read_layout_min_freq_ratio=2, lib_source_min_match_pct=2, lib_source_min_freq_ratio=2, lib_type_max_distance=1000, lib_type_mates_cutoff=0.85, read_orientation_min_mapped_reads=20, read_orientation_min_fraction=0.75, path_1_processed=PosixPath('.'), path_2_processed=None, t_file_processed=PosixPath('.'))), 'results': FieldInfo(annotation=Results, required=False, default=Results(library_stats=ResultsStats(file_1=Stats(read_length=ReadLength(min=None, max=None, mean=None, median=None, mode=None)), file_2=Stats(read_length=ReadLength(min=None, max=None, mean=None, median=None, mode=None))), library_source=ResultsSource(file_1=Source(short_name=None, taxon_id=None), file_2=Source(short_name=None, taxon_id=None)), library_type=ResultsType(file_1=<StatesType.not_available: None>, file_2=<StatesType.not_available: None>, relationship=<StatesTypeRelationship.not_available: None>), read_orientation=ResultsOrientation(file_1=<StatesOrientation.not_available: None>, file_2=<StatesOrientation.not_available: None>, relationship=<StatesOrientationRelationship.not_available: None>), read_layout=ResultsLayout(file_1=Layout(adapt_3=None, polyA_frac=None), file_2=Layout(adapt_3=None, polyA_frac=None))))}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

results: *Results*

class htsinfer.models.**Layout**(*, *adapt_3: str | None = None*, *polyA_frac: float | None = None*)

Bases: BaseModel

Read layout of a single sequencing file.

Parameters

- **adapt_3** – Adapter sequence ligated to 3'-end of sequence.
- **polyA_frac** – Fraction of reads containing polyA tails.

adapt_3

Adapter sequence ligated to 3'-end of sequence.

Type
str | None

polyA_frac

Fraction of reads containing polyA tails.

Type
float | None

adapt_3: str | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapt_3': FieldInfo(annotation=Union[str, NoneType], required=False), 'polyA_frac': FieldInfo(annotation=Union[float, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

polyA_frac: float | None

class htsinfer.models.**LogLevels**(*value*)

Bases: Enum

Log level enumerator.

CRITICAL = 50

DEBUG = 10

ERROR = 40

INFO = 20

WARN = 30

WARNING = 30

class htsinfer.models.**ReadLength**(**, min: int | None = None, max: int | None = None, mean: float | None = None, median: int | None = None, mode: int | None = None*)

Bases: BaseModel

Read length of a sequencing file.

Parameters

- **min** – Minimum read length.
- **max** – Maximum read length.
- **mean** – Mean of read lengths.
- **median** – Median of read lengths.
- **mode** – Mode of read length.

min

Minimum read length.

Type
int | None

max

Maximum read length.

Type
int | None

mean

Mean of read lengths.

Type
float | None

median

Median of read lengths.

Type
int | None

mode

Mode of read length.

Type
int | None

max: int | None

mean: float | None

median: int | None

min: int | None

mode: int | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'max': FieldInfo(annotation=Union[int, NoneType], required=False), 'mean': FieldInfo(annotation=Union[float, NoneType], required=False), 'median': FieldInfo(annotation=Union[int, NoneType], required=False), 'min': FieldInfo(annotation=Union[int, NoneType], required=False), 'mode': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class htsinfer.models.**Results**(*, *library_stats: ~htsinfer.models.ResultsStats = ResultsStats(file_1=Stats(read_length=ReadLength(min=None, max=None, mean=None, median=None, mode=None)), file_2=Stats(read_length=ReadLength(min=None, max=None, mean=None, median=None, mode=None)))*, *library_source: ~htsinfer.models.ResultsSource = ResultsSource(file_1=Source(short_name=None, taxon_id=None), file_2=Source(short_name=None, taxon_id=None))*, *library_type: ~htsinfer.models.ResultsType = ResultsType(file_1=<StatesType.not_available: None>, file_2=<StatesType.not_available: None>, relationship=<StatesTypeRelationship.not_available: None>)*, *read_orientation: ~htsinfer.models.ResultsOrientation = ResultsOrientation(file_1=<StatesOrientation.not_available: None>, file_2=<StatesOrientation.not_available: None>, relationship=<StatesOrientationRelationship.not_available: None>)*, *read_layout: ~htsinfer.models.ResultsLayout = ResultsLayout(file_1=Layout(adapt_3=None, polyA_frac=None), file_2=Layout(adapt_3=None, polyA_frac=None))*)

Bases: BaseModel

Container class for aggregating results from the different inference functionalities.

Parameters

- **library_type** – Library type inference results.
- **library_source** – Library source inference results.
- **orientation** – Read orientation inference results.
- **read_layout** – Read layout inference results.
- **type** – Library type inference results.
- **source** – Library source inference results.
- **read_orientation** – Read orientation inference results.
- **read_layout** – Read layout inference results.

library_source: *ResultsSource*

library_stats: *ResultsStats*

library_type: *ResultsType*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'library_source': FieldInfo(annotation=ResultsSource, required=False, default=ResultsSource(file_1=Source(short_name=None, taxon_id=None), file_2=Source(short_name=None, taxon_id=None))), 'library_stats': FieldInfo(annotation=ResultsStats, required=False, default=ResultsStats(file_1=Stats(read_length=ReadLength(min=None, max=None, mean=None, median=None, mode=None)), file_2=Stats(read_length=ReadLength(min=None, max=None, mean=None, median=None, mode=None)))), 'library_type': FieldInfo(annotation=ResultsType, required=False, default=ResultsType(file_1=<StatesType.not_available: None>, file_2=<StatesType.not_available: None>, relationship=<StatesTypeRelationship.not_available: None>)), 'read_layout': FieldInfo(annotation=ResultsLayout, required=False, default=ResultsLayout(file_1=Layout(adapt_3=None, polyA_frac=None), file_2=Layout(adapt_3=None, polyA_frac=None))), 'read_orientation': FieldInfo(annotation=ResultsOrientation, required=False, default=ResultsOrientation(file_1=<StatesOrientation.not_available: None>, file_2=<StatesOrientation.not_available: None>, relationship=<StatesOrientationRelationship.not_available: None>))}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

read_layout: *ResultsLayout*

read_orientation: *ResultsOrientation*

class htsinfer.models.**ResultsLayout**(*, *file_1:* Layout = *Layout(adapt_3=None, polyA_frac=None)*, *file_2:* Layout = *Layout(adapt_3=None, polyA_frac=None)*)

Bases: BaseModel

Container class for read layout of a sequencing library.

Parameters

- **file_1** – Adapter sequence present in first file.
- **file_2** – Adapter sequence present in second file.

file_1

Adapter sequence present in first file.

Type
htsinfer.models.Layout

file_2

Adapter sequence present in second file.

Type
htsinfer.models.Layout

file_1: *Layout*

file_2: *Layout*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'file_1': FieldInfo(annotation=Layout, required=False, default=Layout(adapt_3=None, polyA_frac=None)), 'file_2': FieldInfo(annotation=Layout, required=False, default=Layout(adapt_3=None, polyA_frac=None))}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class htsinfer.models.**ResultsOrientation**(*, *file_1:* StatesOrientation | *None* = *StatesOrientation.not_available*, *file_2:* StatesOrientation | *None* = *StatesOrientation.not_available*, *relationship:* StatesOrientationRelationship | *None* = *StatesOrientationRelationship.not_available*)

Bases: BaseModel

Container class for aggregating library orientation.

Args:

file_1: Read orientation of first file. file_2: Read orientation of second file. relationship: Orientation type relationship between the provided files.

file_1

Read orientation of first file.

Type

htsinfer.models.StatesOrientation | None

file_2

Read orientation of second file.

Type

htsinfer.models.StatesOrientation | None

file_1: *StatesOrientation* | None

file_2: *StatesOrientation* | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'file_1': FieldInfo(annotation=Union[StatesOrientation, NoneType], required=False, default=<StatesOrientation.not_available: None>), 'file_2': FieldInfo(annotation=Union[StatesOrientation, NoneType], required=False, default=<StatesOrientation.not_available: None>), 'relationship': FieldInfo(annotation=Union[StatesOrientationRelationship, NoneType], required=False, default=<StatesOrientationRelationship.not_available: None>)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

relationship: *StatesOrientationRelationship* **| None**

class htsinfer.models.**ResultsSource**(*, *file_1:* Source = *Source(short_name=None, taxon_id=None)*, *file_2:* Source = *Source(short_name=None, taxon_id=None)*)

Bases: BaseModel

Container class for aggregating library source.

Parameters

- **file_1** – Library source of the first file.
- **file_2** – Library source of the second file.

file_1

Library source of the first file.

Type
htsinfer.models.Source

file_2

Library source of the second file.

Type
htsinfer.models.Source

file_1: *Source*

file_2: *Source*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'file_1': FieldInfo(annotation=Source, required=False, default=Source(short_name=None, taxon_id=None)), 'file_2': FieldInfo(annotation=Source, required=False, default=Source(short_name=None, taxon_id=None))}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class htsinfer.models.**ResultsStats**(*, *file_1:* Stats = *Stats(read_length=ReadLength(min=None, max=None, mean=None, median=None, mode=None))*, *file_2:* Stats = *Stats(read_length=ReadLength(min=None, max=None, mean=None, median=None, mode=None))*)

Bases: BaseModel

Container class for aggregating library statistics information.

Parameters

- **file_1** – Library statistics for the first file.
- **file_2** – Library statistics for the second file.

file_1

Library statistics for the first file.

Type
htsinfer.models.Stats

file_2

Library statistics for the second file.

Type
htsinfer.models.Stats

file_1: *Stats*

file_2: *Stats*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'file_1': FieldInfo(annotation=Stats, required=False, default=Stats(read_length=ReadLength(min=None, max=None, mean=None, median=None, mode=None))), 'file_2': FieldInfo(annotation=Stats, required=False, default=Stats(read_length=ReadLength(min=None, max=None, mean=None, median=None, mode=None)))}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class htsinfer.models.**ResultsType**(*, *file_1:* StatesType | *None* = *StatesType.not_available*, *file_2:* StatesType | *None* = *StatesType.not_available*, *relationship:* StatesTypeRelationship | *None* = *StatesTypeRelationship.not_available*)

Bases: BaseModel

Container class for aggregating library type and mate relationship information.

Parameters

- **file_1** – Library type of the first file.
- **file_2** – Library type of the second file.
- **relationship** – Type/mate relationship between the provided files.

file_1

Library type of the first file.

Type
htsinfer.models.StatesType | None

file_2

Library type of the second file.

Type
htsinfer.models.StatesType | None

relationship

Type/mate relationship between the provided files.

Type
htsinfer.models.StatesTypeRelationship | None

file_1: *StatesType* **| None**

file_2: *StatesType* **| None**

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'file_1': FieldInfo(annotation=Union[StatesType, NoneType], required=False, default=<StatesType.not_available: None>), 'file_2': FieldInfo(annotation=Union[StatesType, NoneType], required=False, default=<StatesType.not_available: None>), 'relationship': FieldInfo(annotation=Union[StatesTypeRelationship, NoneType], required=False, default=<StatesTypeRelationship.not_available: None>)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

relationship: *StatesTypeRelationship* **| None**

class htsinfer.models.**RunStates**(*value*)

Bases: IntEnum

Enumerator of run states and exit codes.

ERROR = 2

OKAY = 0

WARNING = 1

class htsinfer.models.**SeqIdFormats**(*value*)

Bases: Enum

An enumeration.

class htsinfer.models.**Source**(*, *short_name: str | None = None*, *taxon_id: int | None = None*)

Bases: BaseModel

Library source of an individual sequencing file.

Parameters

- **short_name** – Library source short name, e.g., "hsapiens".
- **taxon_id** – Library source taxon identifer, e.g., *9606*.

short_name

Library source short name, e.g., "hsapiens".

Type
str | None

taxon_id

Library source taxon identifer, e.g., *9606*.

Type
int | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'short_name': FieldInfo(annotation=Union[str, NoneType], required=False), 'taxon_id': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

short_name: str | None

taxon_id: int | None

class htsinfer.models.**StatesOrientation**(*value*)

Bases: Enum

Enumerator of read orientation types for individual library files. Cf. https://salmon.readthedocs.io/en/latest/library_type.html

not_available

Orientation type information is not available for a given file, either because no file was provided, the file could not be parsed, an orientation type has not yet been assigned.

stranded_forward

Reads are stranded and come from the forward strand.

stranded_reverse

Reads are stranded and come from the reverse strand.

unstranded

Reads are unstranded.

not_available = None

stranded_forward = 'SF'

stranded_reverse = 'SR'

unstranded = 'U'

class htsinfer.models.**StatesOrientationRelationship**(*value*)

Bases: Enum

Enumerator of read orientation type relationships for paired-ended libraries. Cf. https://salmon.readthedocs.io/en/latest/library_type.html

inward_stranded_forward

Mates are oriented toward each other, the library is stranded, and first mates come from the forward strand.

inward_stranded_reverse

Mates are oriented toward each other, the library is stranded, and first mates come from the reverse strand.

inward_unstranded

Mates are oriented toward each other and the library is unstranded.

not_available

Orientation type relationship information is not available, likely because only a single file was provided or because the orientation type relationship has not been or could not be evaluated.

inward_stranded_forward = 'ISF'

inward_stranded_reverse = 'ISR'

inward_unstranded = 'IU'

not_available = None

class htsinfer.models.**StatesType**(*value*)

Bases: Enum

Possible outcomes of determining the sequencing library type of an individual FASTQ file.

file_problem

There was a problem with opening or parsing the file.

first_mate

All of the sequence identifiers of the processed file counts indicate that the library represents the first mate of a paired-end library.

mixed_mates

All of the sequence identifiers of the processed file include mate information. However, the file includes at least one record for either mate, indicating that the library represents a mixed mate library.

not_available

Library type information is not available for a given file, either because no file was provided, the file could not be parsed, a library type has not yet been assigned, the processed file contains records with sequence identifiers of an unknown format, of different formats or that are inconsistent in that they indicate the library represents both a single-ended and paired-ended library at the same time.

second_mate

All of the sequence identifiers of the processed file indicate that the library represents the second mate of a paired-end library.

single

All of the sequence identifiers of the processed file indicate that the library represents a single-end library.

first_mate = 'first_mate'

first_mate_assumed = 'first_mate_assumed'

mixed_mates = 'mixed_mates'

not_available = None

second_mate = 'second_mate'

second_mate_assumed = 'second_mate_assumed'

single = 'single'

class htsinfer.models.**StatesTypeRelationship**(*value*)

Bases: Enum

Possible outcomes of determining the sequencing library type/mate relationship between two FASTQ files.

not_available

Mate relationship information is not available, likely because only a single file was provided or because the mate relationship has not yet been evaluated.

not_mates

The library type information of the files is not compatible, either because not a pair of first and second mate files was provided, or because the files do not compatible sequence identifiers.

split_mates

One of the provided files represents the first and the other the second mates of a paired-end library.

not_available = None

not_mates = 'not_mates'

split_mates = 'split_mates'

class htsinfer.models.**Stats**(*, *read_length:* ReadLength = *ReadLength(min=None, max=None, mean=None, median=None, mode=None)*)

Bases: BaseModel

Library statistics of an individual sequencing file.

Parameters

read_length – Tuple of mininimum, maximum, mean, median and mode of lengths of reads in library.

read_length

Tuple of mininimum, maximum, mean, median and mode of lengths of reads in library.

Type

htsinfer.models.ReadLength

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'read_length': FieldInfo(annotation=ReadLength, required=False, default=ReadLength(min=None, max=None, mean=None, median=None, mode=None))}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

read_length: *ReadLength*

1.1.12 htsinfer.subset_fastq module

FASTQ subsetting, extraction and validation.

class htsinfer.subset_fastq.**SubsetFastq**(*path: Path*, *out_dir: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/htsinfer/checkouts/stable/*
records: int = 0)

Bases: object

Subset, uncompress and validate a FASTQ file.

Parameters

- **path** – Path to FASTQ file.
- **out_dir** – Path to directory where output is written to.
- **records** – Number of input file records to process; set to *0* to process all records.

path

Path to FASTQ file.

out_dir

Path to directory where output is written to.

records

Number of input file records to process.

out_path

Path for uncompressed, filtered *path* file.

n_processed

Total number of processed records.

Raises

FileProblem – The input file could not be parsed or the output file could not be written.

process()

Uncompress, subset and validate files.

1.1.13 htsinfer.utils module

Utilities used across multiple HTSinfer modules.

htsinfer.utils.**convert_dict_to_df**(*dic: Dict*, *col_headers: Tuple[str, str] | None = None*, *sort: bool = False*, *sort_by: int = 0*, *sort_ascending: bool = True*) → DataFrame

Convert dictionary to two-column data frame.

Parameters

- **dic** – Dictionary to convert.
- **col_headers** – List of column headers. Length MUST match number of dictionary keys/data frame columns.
- **sort** – Whether the resulting data frame is supposed to be sorted.
- **sort_by** – Column index used for sorting. Ignored if *sort* is *False*.
- **sort_ascending** – Whether the data frame is supposed to be sorted in ascending order. Ignored if *sort* is *False*.

Returns

Data frame prepared from dictionary.

Raises

ValueError – Raised if number of provided column headers does not match the number of data frame columns.

htsinfer.utils.**validate_top_score**(*vector: List[float]*, *min_value: float = 2*, *min_ratio: float = 2*, *accept_zero: bool = True*, *rev_sorted: bool = True*) → bool

Validates whether (1) the maximum value of a numeric list is equal to or higher than a specified minimum value AND (2) that the ratio of the first and second highest values of the list is higher than a specified minimum ratio.

If the passed list/vector does NOT contain at least two items, the function returns *False*.

Parameters

- **vector** – List of numbers.
- **min_value** – Minimum value required in first row of *column_index* for validation to pass.
- **min_ratio** – Minimum ratio of first and second rows of *column_index* required for validation to pass.
- **accept_zero** – Whether to accept a top score (i.e., return *True*) if the second highest value in the provided list is zero. If not set to *True*, *False* is returned in these cases.
- **rev_sorted** – Whether the list of numbers is sorted in descencing numeric order.

Returns

Whether data frame *data* satisfies the *min_value* and *min_ratio* constraints for value in column *column_index*.

Raises

ValueError – Raised if one of the list items can not be interpreted as a number.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

h

INDEX

H

I

K

L

M

N

O

P

R

S

T

U

V

W