protfasta Documentation

protfasta

Mar 16, 2023

Contents:

protfasta is a simple, robust parser for working with FASTA files. It is pure python and has no external package dependencies other than Python language modules.

It contains two distinct components:

1. A Python API for reading and writing FASTA files, which includes a collection of santization functions. This makes it easy to write code that reads/writes FASTA files.
2. A command line tool (`pfasta`) that allows manipulation of FASTA files directly from the command line.

This documentation provides an overeview of both components.

CHAPTER 1

Why did you build protfasta? Don't you have better things to do with your time?

This is a reasonable question...

Working with protein-based FASTA files is at the heart of a lot of what the Holehouse lab does. We had previously used a few different existing parsers but had found limitations with respect to certain features. Part of this came from the fact that many FASTA parsers can work with nucleotide or protein data. Given our bread and butter is protein sequences, we decided to build a parser explicitly for working with proteins. We also wanted the ability to deal with FASTA files with duplicate entries. Not necessarily because this is 'good', but for processing reasons being able to deal with this in-code is easier than having to sanitize ahead of time.

We built `pfasta` as a compact tool for working with FASTA files at the command-line level. In particular, the ability to filter FASTA files by sequence length, correct/remove sequence with invalid amino acids, and do various other things lends `pfasta` as a useful first tool in our informatics pipelines.

CHAPTER 2

Will protfasta work with nucleotide-based FASTA files?

In principle yes, but none of our testing suites are set up to rigerously explore this. However, there's no reason it shouldn't, although it may be less efficient that some other tools such as the excellent pyfaidx.

CHAPTER 3

Bugs and help

If you find any bugs or have feature requests please raise an issue on our Github page. **protfasta** uses a continous integration suite for the main package, and **pfasta** has a set of local tests that are run upon updates.

A note: protfasta was built to work with Python 3.7 or higher. However, it does - in principle, work with Python 3.6 but there may be some oddities. We strongly recommend using 3.7.6 or higher.

CHAPTER 4

How to cite **protfasta**

For now please just cite the Github repository (including the date accessed). We are planning on putting out a short biorxiv paper (not submitting to a journal) for a DOI-ed reference, at which point this documentation will be updated accordingly.

4.1 Installation

protfasta has been tested on Linux and macOS. It should also work on Windows but we haven't tested it there yet.

protfasta can be downloaded and installed directly from PyPI using **pip**:

```
pip install protfasta
```

If this has worked, the **pfasta** tool should be available from the command-line

```
pfasta --help
```

And you're done. This also means you can now `import` and use **protfasta** in your Python workflow.

4.2 pfasta

pfasta is a command-line tool for working with FASTA files to filter and sanitize them based on various criterion. This includes:

- Filtering out sequences that contain invalid amino acids
- Take sequences that contain invalid characters and replace/fix them
- Filter a set of sequences by a maximum and/or minimum sequence length
- Sub-sample a set of sequences for building a reduced set of randomly selected sequences

At it's basline, **pfasta** takes a single sequence file as input and writes a new output sequence. There are a series of flags that can be applied, as outlined in the Usage section below.

4.2.1 Usage

```
pfasta <flags> filename.fasta
```

```
-o <output filename> (default: output.fasta)
   Define the name of the output FASAT file

--non-unique-header
  Flag that, if provided allows multiple FASTA records to have identical headers

--duplicate-record (default: fail)
  Flag that provides a keyword that defines how duplicate FASTA records are dealt␣
↪with.
  Options are:
      fail   : throws an exception and exits the parsing
      ignore : duplicate records are retained
      remove : duplicate records are removed

--duplicate-sequence (default: fail)
  Flag that provides a keyword that defines how duplicate sequences are dealt with.
  Options are:
      fail   : throws an exception and exits the parsing
      ignore : duplicate sequences are retained
      remove : duplicate sequences are removed

--invalid-sequence (default: fail)
  Flag that provides a keyword that defines how invalid sequences are dealt with.
  Options are:
      fail                : throws an exception and exits the parsing
      ignore              : invalid sequences are retained
      remove              : invalid sequences are removed
      convert-all         : invalid residues are converted according to the standard␣
↪conversion table
                            (shown below) but if OTHER invalid residues are found an␣
↪exception is raised
                            B->N,    U->C,    X->G,    Z->Q,    '*'->'',    '-'->''
      convert-res         : invalid residues are converted according to the standard␣
↪conversion table
                            with the exception of sequence-alignment gaps ('-')
      convert-all-ignore  : invalid residues are converted according to the standard␣
↪conversion table,
                            and if OTHER invalid residues are found they are ignored
      convert-res-ignore  : invalid residues are converted according to the standard␣
↪conversion table,
                            with the exception of the sequence-aligment gap ('-')␣
↪character, but
                            if OTHER invalid residues are found they are ignored

--number-lines (default: 60)
  Flag that defines the number of lines in the output FASTA file

--shortest-seq-lines (default: None)
  Flag that defines a filter that sets the shortest sequence returned

--longest-seq-lines (default: None)
  Flag that defines a filter that sets the longest sequence returned
```

(continues on next page)

(continued from previous page)

```
--random-subsample (default: None)
  Flag that defines the number of randomly sub-sampled sequences. Allows a test FASTA␣
↪file to be
  generated as a sub-set for testing analysis pipelines

--print-statistics
  Flag that, if provided, means statistics about the FINAL set of sequences written

--no-outputfile
  Flag that, if provided, means NO outputfile is generated.

--silent
  Flag that, if provided, means pfasta generates ZERO output to STDOUT
```

4.3 Examples

It's often easies to see how to use code through some well-worked examples. Here we provide some simple examples that illustrate how **protfasta** can be used to read and write FASTA files.

4.3.1 `read_fasta` examples

Some possible exampls of reading FASTA files using protfasta:

Example 1: Simple read in FASTA file

```
import protfasta

sequences = protfasta.read_fasta('inputfile.fasta')
```

Example 2: Simple read in FASTA file and ignore duplicate FASTA records and return a nested-list of residues

```
import protfasta

sequences = protfasta.read_fasta('inputfile.fasta',
                                 expect_unique_header=False,
                                 return_list=True,
                                 duplicate_record_action='ignore')
```

Example 3: Read in FASTA file and correct invalid residues using standard error correction dictionary

```
import protfasta

sequences = protfasta.read_fasta('inputfile.fasta',
                                 invalid_sequence_action='convert')
```

Example 4: Read in FASTA file and correct invalid residues using a custom dictionary

```
import protfasta

CD = {'U':'G', '-':''}
sequences = protfasta.read_fasta('inputfile.fasta',
                                 invalid_sequence_action='convert',
                                 correction_dictionary=CD)
```

Example 5: Read in FASTA file quickly without error checking By default **protfasta performs a bunch of sanity checking. In general this probably doesn't need to be done every time if you KNOW a file is safe. To cancel any sanity checking and read in at maximum efficiency the following options can be provided:

```
import protfasta

sequences = protfasta.read_fasta('inputfile.fasta',
                                 invalid_sequence_action='ignore',
                                 duplicate_record_action='ignore',
                                 duplicate_sequence_action='ignore',
                                 expect_unique_header=False)
```

4.3.2 `write_fasta` examples

```
# input example using a sequence dictionary
import protfasta

sequence_in = {'seq1': 'MEEPQSDPSVEPPLS', 'seq2': 'DEAPRMPEAAPPVAPA'}
protfasta.write_fasta(sequence_in, 'example.fasta')
```

```
# input example using a sequence list
import protfasta

sequence_in = [['seq1','MEEPQSDPSVEPPLS'], ['seq2', 'DEAPRMPEAAPPVAPA']]
protfasta.write_fasta(sequence_in, 'example.fasta')
```

4.4 read_fasta

`read_fasta` is a one-stop-shop for reading in FASTA files! Customizable keywords allow a variety of sanitizing functions which include:

- Ignore, remove, or convert sequences with invalid amino acid characters (B/U/X/*/-)
- Ignore or remove duplicate sequences or duplicate FASTA records
- Alternatively, allow duplicate sequences, headers, and FASTA records (something most other parsers do not)
- Arbitrary conversion of amino acids via a customizable `correction_dictionary`

Once parsed, `read_fasta` returns either a dictionary of header-to-sequence values or a nested list, where each sub-list contains two elements (header, sequence).

For usage examples see the *Examples* page. Full documentation is shown below.

4.4.1 Documentation

`protfasta.`**`read_fasta`**(*filename*, *expect_unique_header=True*, *header_parser=None*, *check_header_parser=True*, *duplicate_sequence_action='ignore'*, *duplicate_record_action='fail'*, *invalid_sequence_action='fail'*, *alignment=False*, *return_list=False*, *output_filename=None*, *correction_dictionary=None*, *verbose=False*)

`read_fasta` is the main one of of only two user-facing functions associated with **protfasta**. It is designed as a catch-all function for reading in a FASTA file, performing sanitization, and returning a list or dictionary of sequences and their associated headers.

There are a number of parameters which can be included, but as one might expect the simplest usage is just

```
>>> x = read_fasta(filename)
```

This will read in the file associated with filename and return a dictionary, where the keys are the FASTA file headers and the values are the amino acid sequences associated with each.

Note that as of python 3.7 the order in which one adds items to a dictionary is guaranteed to be the order in which they're retrieved, so cycling through the resulting dictionary will in fact allow you to cycle through in order.

In addition to this simple usage, there are a number of keywords which are described in depth below and allow additional processing to be complete.

There is an order of options in which sanitization occurs:

1. File is read in, custom headers are parsed, and unique headers are tested (if `expect_unique = True`)
2. Check for duplicate records and respond appropriately (**optional**)
3. Check for duplicate sequences and respond appropriately (**optional**)
4. Invalid sequences dealt with (**optional**)
5. Final set of sequences/headers written to a new FASTA file (**optional**)
6. Dictionary/list returned to user.

Understanding there is a specific order is important when considering what options to pass. If a set of options are incompatible, this will be caught before the file is read.

Parameters

- **expect_unique_header** (*bool*) – [**Default = True**] Should the function expect each header to be unique? In general this is true for FASTA files, but this is strictly not guarenteed. If this is set to True and a duplicate header is found then this means an error will be thrown. If it's set to false duplicate headers are dealt with, although for this to work `return_list` must also be set to True. Note that this won't happen automatically to avoid the scenario where you expect a dictionary to return and actually get a list.
- **header_parser** (*function*) – [**Default = None**] `header_parser` allows a user-defined function that will be fed the FASTA header and whatever it returns will be used as the actual header as the files are parsed. This can be useful if you know your FASTA header has a consistent format that you want to take advantage of. A function provided here MUST **(1)** Take a single input argument (the header string) and **(2)** Return a single string. When parsing this function the following test is applied, unless `check_header_parser` is set to false.

  ```
  >>> return_string = header_parser('this test string should work')
  ```

  Where `return_string` is tested to be a string. The function will show an exception if this test fails and `check_header_parser` is set to true.
- **check_header_parser** (*bool*) – [**Default = True**] Flag which - if set to false - will not test if the header_parser function returns a valid string. This may lead to unexpected header values if the passed header_parser function is not well defined.
- **duplicate_record_action** (`'ignore'`, `'fail'`, `'remove'`) – [**Default = 'fail'**] Selector that determines how to deal with duplicate entries. Note that duplicate records refers to entries in the fasta file where both the sequence and the header are identical. duplicate_record_action is only relevant keyword when expect_unique_header is False. Options are as follows:

  - `ignore` - duplicate entries are allowed and ignored
  - `fail` - duplicate entries cause parsing to fail and throw an exception
  - `remove` - duplicate entries are removed, so there's only one copy of any duplicates
- **duplicate_sequence_action** (`'ignore'`, `'fail'`, `'remove'`) – [**Default = 'ignore'**] Selector that determines how to deal with duplicate sequences. This completely ignores the header and simply asks is two sequences are duplicated (or not).
  - `ignore` - duplicate sequences are allowed and ignored
  - `fail` - duplicate sequences cause parsing to fail and throw an exception
  - `remove` - duplicate sequences are removed, so there's only one copy of any duplicates (1st instance kept)
- **invalid_sequence_action** (`'ignore'`, `'fail'`, `'remove'`, `'convert'`, `'convert-ignore'`, ``'convert-remove'``) – [**Default = 'fail'**] Selector that determines how to deal with invalid sequences. If `convert` or `convert-ignore` are chosen, then conversion is completed with either the standard conversion table (shown under the `correction_dictionary` documentation) or with a custom conversion dictionary passed to `correction_dictionary`. Options are as follows:
  - `ignore` - invalid sequences are completely ignored
  - `fail` - invalid sequence cause parsing to fail and throw an exception
  - `remove` - invalid sequences are removed
  - `convert` - invalid sequences are convert
  - `convert-ignore` - invalid sequences are converted to valid sequences and any remaining invalid residues are ignored
  - `convert-remove` - invalid sequences are converted to valid sequences where possible, and any remaining sequences with invalid residues are removed
- **alignment** (*bool*) – [**Default = False**] Flag which - if set to true - the Fasta file is treated as containing alignments (with dashes) such that '-' characters are not treated as invalid or converted. Works in concert with other flags.
- **return_list** (*bool*) – [**Default = False**] Flag that tells the function to return a list of 2-mer lists (where position 0 is the header and position 1 the sequence). If you have duplicate identical headers which you want to deal with, this is required.
- **output_filename** (*string*) – [**Default = None**] If you are performing sanitization of the input file it is often useful to write out the actual set of sequences you'll be analyzing, so you have a persistent copy of this data for further analysis later on. If you provide a string to output filename it will cause a new FASTA file to be written with the final set of sequences returned.
- **correction_dictionary** (*dict*) – [**Default = None**] **protfasta** can automatically correct non-standard amino acids to standard amino acids using the `invalid_sequence` keyword. This is useful if downstream analysis assumes/requires fully standard amino acids. This is also useful for removing '-' from aligned sequences. The standard conversions used are:
  - `B -> N`
  - `U -> C`
  - `X -> G`
  - `Z -> Q`
  - `" " -> <empty string>` (i.e. a whitespace character)

- `*` -> <empty string>
- `-` -> <empty string>

However, if alternative definitions are needed they can be passed via the `correction_dictionary` keyword. The `correction_dictionary` should be a dictionary that maps sequences characters to some other character (ideally valid amino acid characters). In principle this could be used to perform arbitrary coarse-graining if a sequence...

- **verbose** (*bool*) – [**Default = False**] If set to True, **protfasta** will print out information as it works its way through reading and parsing FASTA files. This can be useful for diagnosis.

Returns

- *Return type is *list* or *dict**
- If `return_list` is set to `True` then the function returns a list of lists. In each sublist contains two elements, where the first is the FASTA record header and the second the sequence. The order of FASTA records will match the order they were read in from the FASTA file. If `return_list` is `False` then the function returns a dictionary where the keys are the FASTA record heades and the values are the sequences. NOTE the order of keys will match the order that the FASTA file was read in IF the Python version is 3.7 or higher.

4.5 write_fasta

`write_fasta` allows for standardized FASTA files to be written to disk using a dictionary or a list of two-position lists. Note that it is possible to automatically write a FASTA file when reading in using the `output_filename` keyword in the function `read_fasta`.

Unlike `read_fasta`, `write_fasta` has a relatively limited set of options, which are documented below.

For usage examples see the *Examples* page.

4.5.1 Documentation

`protfasta.`**`write_fasta`**(*fasta_data*, *filename*, *linelength=60*, *verbose=False*, *append_to_fasta=False*)

Simple function that takes a dictionary of key to sequence values and writes out a valid FASTA file. No return type, but writes a file to disk according to the location defined by the variable `filename`.

Parameters

- **fasta_data** (*dict or list*) – If a dictionary is passed then keys must be identifiers and the values are amino acid sequences. If a list is passed it must be a list where each element contains two sub-elements, a header, and a sequence.
- **filename** (*string*) – Filename to write to. Should end with *.fasta* or *.fa* but this is not enforced.
- **linelength** (*int*) – [**Default = 60**] Length of line to be written for sequence (note this does not effect the header line. 60 is default used by UniProt. If set to 0, None or False no line-length limit is used. Note `linelength` must be > 5.
- **append_to_fasta** (*bool*) – Whether to append to a fasta file that already exists. If this is set to True, if the file does not exist, protfasta will create a new file. However, if the file does exist, protfasta will simply append additional fasta entries to the existing file. Default=False

Returns No return value is provided but a new FASTA file is written to disk

Return type None

Python Module Index

p

Index