abstar Documentation

Release 0.3.4

Bryan Briney

Aug 25, 2022

getting started

1 getting started 3

2 usage 7

3 about 13

4 related projects 15

5 Index 17

abstar is a core component of the ab[x] toolkit for antibody sequence analysis. abstar performs V(D)J germline gene assignment and primary sequence annotation and can readily scale from a single sequence to billions of sequences.

CHAPTER 1

getting started

1.1 Overview

With technical breakthroughs in the throughput and read-length of next-generation sequencing platforms, antibody repertoire sequencing is becoming an increasingly important tool for detailed characterization of the immune response to infection and immunization. Accordingly, there is a need for open, scalable software for the genetic analysis of repertoire-scale antibody sequence data.

We built abstar to be a modular component of these analyses. abstar is engineered to be highly flexible, capable of processing a single sequence or billions of sequences and scaling from individual laptops to large clusters of cloud computing instances.

1.1.1 Workflows

In addition to V(D)J germline gene assignment and primary antibody sequence annotation, abstar contains utilities for sequence acquisition, pre-processing, and database import. abstar also exposes a high-level public API to many of the core functions, which allows users to easily construct *custom analysis workflows* using multiple abstar utilities as well as other third-party tools. To ease integration of abstar into currently existing antibody analysis pipelines based on IMGT, abstar can optionally produce output that mimics the IMGT-HighV/Quest Summary output file.

1.1.2 File formats

abstar accepts standard FASTA or FASTQ files and produces, by default, JSON-formatted output. This output format allows us to build the output using data structures that match the way we process data programatically. JSON is also easily importable into NoSQL databases like MongoDB. We have found NoSQL databases to be very well suited for performing downstream analyses of antibody repertoire data, as the flexible schema allows for easy updating of sequence records with additional annotation information. Although additional data can be added to relational databases, querying this data often involves joining tables, which can require significant optimization for very large datasets.

1.1.3 Scalability

Cloud computing has dramatically changed the landscape of high-performance computing (HPC), and has allowed small academic labs to 'rent' access to computational resources that would have been previously far outside their budget. abstar is tightly integrated with abcloud, which provides tools for launching, configuring and managing clusters of compute instances on Amazon's Elastic Compute Cloud (EC2). Using the Celery distributed task queue, jobs are distributed to worker nodes and processed in parallel.

In order to maintain compatability with alternate cloud computing platforms with minimal effort, an abstar Docker image is also provided.

1.2 Install

The easiest way to install abstar locally (on OSX or Linux) is to use pip:

```
$ pip install abstar
```

If you don't have pip, the Anaconda Python distribution contains pip along with a ton of useful scientific Python packages and is a great way to get started with Python.

abstar does not run natively on Windows, but Windows users can run abstar with Docker:

```
$ docker pull briney/abstar
$ docker run -it briney/abstar
```

Stable and development versions of abstar can also be downloaded from Github. You can manually install the latest development version of abstar with:

```
$ git clone https://github.com/briney/abstar
$ cd abstar/
$ python setup.py install
```

Note: If installing manually via setup.py and you don't already have scikit-bio installed, you may get an error when setuptools attempts to install scikit-bio. This can be fixed by first installing scikit-bio with pip:

```
$ pip install scikit-bio
```

and then retrying the manual install of abstar. Starting with version 0.5, scikit-bio dropped support for Python 2.7, so install scikit-bio on Python 2.7 with:

```
$ pip install scikit-bio<=0.4.2
```

1.2.1 Requirements

- Python 2.7 or 3.5+
- abutils
- biopython
- celery
- pymongo

- pytest
- scikit bio

1.2.2 Optional dependencies

Several optional abstar components have additional dependencies:

- `abstar.preprocessing` requires FASTQC, cutadapt and sickle
- sequence merging requires PANDAseq
- downloading data from BaseSpace requires the BaseSpace Python SDK
- `batch_mongoimport` requires MongoDB

If using Docker, all of the the optional dependencies are included.

CHAPTER 2

usage

2.1 Commandline Use

Running abstar from the command line is reasonably simple, even for users with minimal experience with command-line applications. In the most basic case, with a single input file of human antibody sequences:

```
$ abstar -i /path/to/mydata.fasta -t /path/to/temp/ -o /path/to/output/
```

abstar will process all sequences contained in `mydata.fasta` and the results will be written to `/path/to/output/mydata.json`. If either (or both) of `/path/to/temp/` or `/path/to/output/` don't exist, they will be created.

If you have a directory of FASTA/Q-formatted files for abstar to process, you can pass a directory via `-i` and all files in the directory will be processed:

```
$ abstar -i /path/to/input/ -t /path/to/temp/ -o /path/to/output/
```

For input directories that contain paired FASTQ files that need to be merged prior to processing, passing the `-m` flag instructs abstar to merge paired files with PANDAseq:

```
$ abstar -i /path/to/input/ -t /path/to/temp/ -o /path/to/output/ -m
```

The merged reads will be deposited into a `merged` directory located in the parent directory of the input directory. By default, abstar will use PANDAseq's `simple_bayesian` merging algorithm, although alternate merging algorithms can be selected with `--pandaseq-algo`.

For data generated with Illumina sequencers, abstar can directly interface with BaseSpace to download raw sequencing data. In order for abstar to connect to BaseSpace, you need BaseSpace access token. The easiest way to do this is to set up a BaseSpace developer account following these instructions. Once you have your credentials, you can generate a BaseSpace credentials file by running:

```
$ make_basespace_credfile
```

and following the instructions.

When downloading data from BaseSpace, you obviously don't have an input directory of data for abstar to process (since that data hasn't been downloaded yet). Instead of providing input, output and temp directories, you can just pass abstar a project directory using `-p` and abstar will create all of the necessary subdirectories within the project directory. Running abstar with the `-b` option indicates that input data should be downloaded from BaseSpace:

```
$ abstar -p /path/to/project_dir/ -b
```

A list of available BaseSpace projects will be displayed and you can select the appropriate project. If downloading data from BaseSpace, `-m` is assumed and paired-end reads will be merged.

abstar uses a human germline database by default, but germline databases are also provided for macaque, mouse and rabbit. To process macaque antibody sequences (from BaseSpace):

```
$ abstar -p /path/to/project_dir/ -b -s macaque
```

2.2 API Examples

abstar and abutils both expose a public API containing many of the core functions. This makes it reasonably straightforward to build custom pipelines that include several abstar/abutils components or integrate these tools with third-party tools. A few simple examples are shown below.

2.2.1 Case #1

Sequencing data consists of an Illumina MiSeq run on human samples, with the raw data stored in BaseSpace (project ID: 123456789). Samples are indexed, so each sample will be downloaded from BaseSpace as a separate pair of read files. We'd like to do several things:

- get a FASTQC report on the raw data
- remove adapters
- quality trim
- get another FASTQC report on the cleaned data
- merge paired reads
- annotate with abstar

```
import os

import abstar
from abstar.utils import basespace, pandaseq

PROJECT_DIR = '/path/to/project'
PROJECT_ID = '123456789'

# download data from BaseSpace
bs_dir = os.path.join(PROJECT_DIR, 'raw_data')
basespace.download(bs_dir, project_id=PROJECT_ID)

# FASTQC on the raw data
fastqc1_dir = os.path.join(PROJECT_DIR, 'fastqc-pre')
abstar.fastqc(bs_dir, output=fastqc1_dir)

# adapter trimming
```

(continues on next page)

(continued from previous page)

```
adapter_dir = os.path.join(PROJECT_DIR, 'adapter_trimed')
adapters = '/path/to/adapters.fasta'
abstar.adapter_trim(bs_dir, output=adapter_dir, adapter_both=adapters)

# quality trimming
quality_dir = os.path.join(PROJECT_DIR, 'quality_trimed')
abstar.quality_trim(adapter_dir, output=quality_dir)

# FASTQC on the cleaned data
fastqc2_dir = os.path.join(PROJECT_DIR, 'fastqc-post')
abstar.fastqc(quality_dir, output=fastqc2_dir)

# read merging
merged_dir = os.path.join(PROJECT_DIR, 'merged')
pandaseq.run(quality_dir, merged_dir)

# run abstar
temp_dir = os.path.join(PROJECT_DIR, 'temp')
json_dir = os.path.join(PROJECT_DIR, 'json')
abstar.run(input=merged_dir,
           temp=temp_dir,
           output=json_dir)
```

2.2.2 Case #2

Sequencing data is a directory of single-read FASTQ files that have already been quality/adapter trimmed. We'd like to do the following:

- get a FASTQC report
- annotate with abstar
- import the JSONs into a MongoDB database named `MyDatabase`

Our FASTQ file names are formatted as: `SampleNumber-SampleName.fastq`, which means the abstar output file name would be `SampleNumber-SampleName.json`. We'd like the corresponding MongoDB collection to just be named `SampleName`.

```
import os

import abstar
from abstar.utils import mongoimport

PROJECT_DIR = '/path/to/project'
FASTQ_DIR = '/path/to/fastqs'

MONGO_IP = '123.45.67.89'
MONGO_PORT = 27017
MONGO_USER = 'MyUsername'
MONGO_PASS = 'Secr3t'

# FASTQC on the input data
fastqc_dir = os.path.join(PROJECT_DIR, 'fastqc')
abstar.fastqc(FASTQ_DIR, output=fastqc_dir)

# run abstar
```

(continues on next page)

(continued from previous page)

```
temp_dir = os.path.join(PROJECT_DIR, 'temp')
json_dir = os.path.join(PROJECT_DIR, 'json')
abstar.run(input=FASTQ_DIR,
           temp=temp_dir,
           output=json_dir)

# import into MongoDB
mongoimport.run(ip=MONGO_IP,
                port=MONGO_PORT
                user=MONGO_USER,
                password=MONGO_PASS,
                input=json_dir,
                db='MyDatabase'
                delim1='-',
                delim2='.')
```

2.2.3 Case #3

Now we'd like to use abstar as part of an analysis script in which sequence annotation isn't the primary output. In the previous examples, we started with raw(ish) sequence data and ended with either a directory of JSON files or a MongoDB database populated with abstar output. In this case, we're going to start with a MongoDB database, query that database for some sequences, and generate the unmutated common ancestor (UCA). We'd like to annotate the UCA sequence inline (as part of the script) so that we can do world-changing things with the annotated UCA later in our script. For simplicity's sake, we're querying a local MongoDB database that doesn't have authentication enabled, although `abutils.utils.mongodb` can work with remote MongoDB servers that require authentication.

```
import abstar

from abutils.utils import mongodb
from abutils.utils.sequence import Sequence

DB_NAME = 'MyDatabase'
COLLECTION_NAME = 'MyCollection'

def get_sequences(db_name, collection_name):
    db = mongodb.get_db(db_name)
    c = db[collection]
    seqs = c.find({'chain': 'heavy'})
    return [Sequence(s) for s in seqs]

def calculate_uca(sequences):
    #
    # code to calculate the UCA sequence, as a string
    #
    return uca

# get sequences, calculate the UCA
sequences = get_sequences(DB_NAME, COLLECTION_NAME)
uca_seq = calculate_uca(sequences)

# run abstar on the UCA, returns an abutils Sequence object
uca = abstar.run(['UCA', uca_seq])

# do amazing, world-changing things with the UCA
```

(continues on next page)

(continued from previous page)

```
# ...
# ...
# ...
```

2.3 API Reference

2.3.1 core

core

2.3.2 assigner

base assigner

BLASTn assigner

2.3.3 preprocess

preprocess

2.3.4 helper utilities

helper utilities

abstar.utils.basespace

abstar.utils.mongoimport

abstar.utils.pandaseq

CHAPTER 3

about

3.1 License

3.2 News

CHAPTER 4

related projects

CHAPTER 5

Index

- modindex
- search