nucleic

Release 0.6.3

Nov 15, 2018

Contents

```
pip install nucleic
```

CHAPTER 1

Features

- Model DNA and variant alleles within their local context using an elegant API
- Combine single nucleotide variants into spectrums of mutagenesis
- Fetch COSMIC signatures of mutation as well as other published signatures
- SVG plotting functions for displaying single nucleotide variant spectrums

1.1 Tutorial

1.1.1 Nucleotides

The class `DNA` is an IUPAC valid sequence of non-degenerate DNA nucleotides. For the purposes of the tutorial we will assume single nucleotide sequences.

```
>>> from nucleic import DNA
>>> DNA("A").is_purine()
True
```

1.1.2 Creating Variant Alleles

```
>>> DNA("A").to("C")
Variant(ref=DNA("A"), alt=DNA("C"), context=DNA("A"))
```

By default, the context of the variant is assigned to the reference base, although a larger context can be set. The context must be symmetrical in length about the base substitution otherwise an error will be raised.

```
>>> DNA("A").to("C").within("TAG")
Variant(ref=DNA("A"), alt=DNA("C"), context=DNA("TAG"))
```

Unless the chemical process for the base substitution is known, it is useful to represent all base substitutions in a canonical form, with either a purine or pyrimidine as the reference base.

```
>>> DNA("A").to("C").within("TAG").with_pyrimidine_ref()
Variant(ref=DNA("T"), alt=DNA("G"), context=DNA("CTA"))
```

A complete example showing the creation of a notation-normalized `Variant` from strings only:

```
>>> ref, alt, context = DNA("A"), DNA("C"), DNA("TAG")
>>> snv = ref.to(alt).within(context).with_pyrimidine_ref()
>>> snv.is_transversion()
True
```

Each `Variant` has a color associated with it for a uniform color palette.

```
>>> snv.color_stratton()
'#EDBFC2'
```

1.1.3 Single Nucleotide Variant Spectrums

A `SnvSpectrum` can be initialized by specifying the size of the local context and the reference notation.

```
>>> from nucleic import SnvSpectrum, Notation
>>> spectrum = SnvSpectrum(k=3, notation=Notation.pyrimidine)
>>> spectrum
SnvSpectrum(k=3, notation=Notation.pyrimidine)
```

Record observations by accessing the `SnvSpectrum` like a Python dictionary.

```
spectrum[snv] += 2
```

Note: this is shorthand for `spectrum.counts[snv] += 2`.

If you have a vector of counts, or probabilities, then you can directly build a `SnvSpectrum` as long as the data is listed in the correct alphabetic order of the `SnvSpectrum` keys.

```
>>> vector = [6, 5, 2, 2, 3, 8]
>>> # SnvSpectrum.from_iterable(vector, k=1, notation=Notation.pyrimidine).counts
```

1.1.4 Working with Probability

Many spectra are produced from whole-genome or whole-exome sequencing experiments. Spectra must be normalized to the _kmer_ frequencies in the target study. Without normalization, no valid spectrum comparison can be made between data generated from different target territories or species.

By default each `nucleic.Variant` is given a weight of 1 and calling `nucleic.SnvSpectrum.mass_as_array()` will simply give the proportion of `nucleic.Variant` counts in the `nucleic.SnvSpectrum`. After weights are set to the observed *k*-mer counts or frequency of the target territory, calling `SnvSpectrum.mass()` will compute a true normalized probability mass.

All weights can be set with assignment *e.g.*: `spectrum.context_weights["ACA"] = 23420`.

```
>>> # spectrum.mass()
```

k-mer counts can be found with `skbio.DNA.kmer_frequencies()` for large targets.

1.1.5 Fetching COSMIC Signatures

Download the published COSMIC signatures of mutational processes in human cancer:

```
>>> from nucleic import fetch_cosmic_signatures
>>> signatures = fetch_cosmic_signatures()
```

1.1.6 Plotting Spectrums

Spectra with `k=3` in either `pyrimidine` or `purine` reference notation can be plotted using a style that was first used in Alexandrov *et. al.* in 2013 (PMID: 23945592). Both `nucleic.Variant` raw counts (`kind="count"`) or their probabilities (`kind="mass"`) can be plotted.

The figure and axes are returned to allow for custom formatting.

```
from nucleic.figures import plot_stratton_spectrum

cosmic_signatures = fetch_cosmic_signatures()

fig, (ax_main, ax_cbar) = plot_stratton_spectrum(cosmic_signatures["Signature 1"],␣
↪kind="mass")
fig, (ax_main, ax_cbar) = plot_stratton_spectrum(cosmic_signatures["Signature 14"],␣
↪kind="mass")
```

1.2 API Reference

1.2.1 Submodules

nucleic module

nucleic.constants module

nucleic.constants.**DNA_IUPAC_NONDEGENERATE = 'ACGT'**
The non-degenerate IUPAC DNA bases.

nucleic.constants.**STRATTON_SNV_COLOR = {'A→C': '#EDBFC2', 'A→G': '#97D54C', 'A→T': '#CBC**
The colors of all single nucleotide variants used in Stratton *et. al.* papers.

nucleic.constants.**DEFAULT_SNV_COLOR = {'A→C': '#D53E4F', 'A→G': '#FC8D59', 'A→T': '#FEE0**
The colors of all single nucleotide variants.

nucleic.constants.**LONGFORM_LABEL = {'A→C': 'A:T→C:G', 'A→G': 'A:T→G:C', 'A→T': 'A:T→T:**
A mapping between shortform canonical single nucleotides and longform.

nucleic.figures module

nucleic.figures.**GridSpec**
alias of `nucleic.figures.Grid`

nucleic.figures.**plot_stratton_spectrum**(*spectrum: nucleic.SnvSpectrum*, *kind: str = 'count'*, *title: str = ''*) → Tuple[toyplot.canvas.Canvas, Tuple[toyplot.canvas.Canvas.cartesian, toyplot.canvas.Canvas.cartesian]]

Plot the trinucleotide spectrum of mutation.

Parameters

- **spectrum** – single nucleotide variants in trinucleotide contexts.
- **kind** – whether to plot data as counts or as a probability mass.
- **title** – the plot title.

Note: The spectrum must be of pyrimidine notation.

nucleic.sequence module

nucleic.sequence.**dna_kmers**(*k: int = 3*) → Generator[[str, None], None]

Return the cartesian product of all DNA substrings of length *k*.

Parameters **k** – Length of of the DNA substring.

Yields Cartesian product of all DNA substrings of length *k*.

Examples

```
>>> list(dna_kmers(1))
['A', 'C', 'G', 'T']
>>> len(list(dna_kmers(3)))
64
```

nucleic.sequence.**hamming_circle**(*string: str, n: int, alphabet: List[str]*) → Generator[[str, None], None]

Find strings, of a given alphabet, with a distance of *n* away from a string.

Examples

```
>>> sorted(hamming_circle('abc', n=0, alphabet='abc'))
['abc']
>>> sorted(hamming_circle('abc', n=1, alphabet='abc'))
['aac', 'aba', 'abb', 'acc', 'bbc', 'cbc']
>>> sorted(hamming_circle('aaa', n=2, alphabet='ab'))
['abb', 'bab', 'bba']
```

nucleic.util module

class nucleic.util.**DictMostCommonMixin**

Give any *dict-like* object a most common method.

Examples

```
>>> class MyDict(DictMostCommonMixin, dict):
...     def __init__(self, *args, **kwargs):
...         super().__init__(*args, **kwargs)
>>> mapping = MyDict({'sample-1': 2, 'sample-2': 10})
>>> mapping.most_common()
[('sample-2', 10), ('sample-1', 2)]
>>> mapping.most_common(n=1)
[('sample-2', 10)]
```

most_common(*n: Optional[int] = None*) → List[Tuple[Any, Any]]

List the *n* most common elements and their counts.

Method returns items from the most common to the least. If *n* is `None`, then list all element counts.

Parameters **n** – The *n* most common items to return, optional.

class nucleic.util.**DictNpArrayMixin**

Make any *dict-like* object methods return `numpy.ndarray` by default.

Examples

```
>>> class MyDict(DictNpArrayMixin, dict):
...     def __init__(self, *args, **kwargs):
...         super().__init__(*args, **kwargs)
>>> mapping = MyDict({'sample-1': 2})
>>> mapping.keys()
array(['sample-1'], dtype='<U8')
>>> mapping.values()
array([2])
```

keys() → numpy.ndarray

Return this dictionary's keys as a `numpy.ndarray`.

values() → numpy.ndarray

Return this dictionary's values as a `numpy.ndarray`.

class nucleic.util.**DictPrettyReprMixin**

Make any *dict-like* object pretty print when `DictPrettyReprMixin.__repr__()` is called.

Examples

```
>>> class AReallyLongDictName(DictPrettyReprMixin, dict):
...     def __init__(self, *args, **kwargs):
...         super().__init__(*args, **kwargs)
>>> AReallyLongDictName({
...     'ScientificObservation1': 1,
...     'ScientificObservation2': 2,
...     'ScientificObservation3': 3,
...     'ScientificObservation4': 4})
AReallyLongDictName({'ScientificObservation1': 1,
                     'ScientificObservation2': 2,
                     'ScientificObservation3': 3,
                     'ScientificObservation4': 4})
```

1.3 How to Contribute

Pull requests, feature requests, and issues welcome! The complete test suite is configured through `Tox`:

```
cd nucleic
pip install tox
tox  # Run entire dynamic / static analysis test suite
```

List all environments with:

```
tox -av
using tox.ini: .../nucleic/tox.ini
using tox-3.1.2 from ../tox/__init__.py
default environments:
py36      -> run the test suite with (basepython)
py36-lint -> check the code style
py36-type -> type check the library
py36-docs -> test building of HTML docs

additional environments:
dev       -> the official sample_sheet development environment
```

To run just one environment:

```
tox -e py36
```

To pass in positional arguments to a specified environment:

```
tox -e py36 -- -x tests/test_sample_sheet.py
```

Python Module Index

n

Index