HYEENNA Documentation

Release 0.0.1

Andrew Bennett

Jul 19, 2019

Contents

HYENNA provides estimators for information theoretic quantities as well as a series of algorithms and analysis tools implemented in pure python.

CHAPTER 1

Installation

For now, HYEENNA is only available to install from source. To do so, clone HYEENNA with:

git clone https://github.com/arbennett/HYEENNA.git

Then navigate to the HYEENNA directory and install with:

python setup.py install

CHAPTER 2

Usage

HYEENNA provides nearest neighbor based estimators for

- Shannon Entropy (Single and multivariate cases)
- Mutual Information
- Conditional Mutual Information
- KL Divergence
- Transfer Entropy
- Conditional Transfer Entropy

CHAPTER 3

Examples

We provide several example notebooks in the notebooks directory.

CHAPTER 4

Documentation

See the full documentation at https://hyeenna.readthedocs.io

CHAPTER 5

References

CHAPTER 6

Sitemap

6.1 API Reference

This page provides an auto-generated summary of HYENNA's API. For more details and examples, refer to the rest of the documentation.

6.1.1 Estimators

hyeenna.estimators.**conditional_entropy**(*X: numpy.array*, *Y: numpy.array*, *k: int = 5*) → float

Computes the conditional Shannon entropy of a sample of a random variable X given another sample of a random variable Y using an adaptation of the KL and KSG estimators

Parameters

- **X** (*np.array*) – Sample from a random variable
- **Y** (*np.array*) – Sample from a random variable
- **k** (*int, optional*) – Number of neighbors to use in estimation

Returns **cent** – estimated conditional entropy

Return type float

References

hyeenna.estimators.**conditional_mutual_info**(*X: numpy.array*, *Y: numpy.array*, *Z: numpy.array*, *k: int = 5*) → float

Compute the conditional mutual information

Parameters

- **X** (*np.array*) – Sample from random variable X
- **Y** (*np.array*) – Sample from random variable Y

- **Z** (*np.array*) – Sample from random variable Z
- **k** (*int, optional*) – Number of neighbors to use in estimation

Returns

Return type estimated conditional mutual information

References

hyeenna.estimators.**conditional_transfer_entropy**(*X: numpy.array*, *Y: numpy.array*, *Z: numpy.array*, *tau: int = 1*, *omega: int = 1*, *nu: int = 1*, *k: int = 1*, *l: int = 1*, *m: int = 1*, *neighbors: int = 5*, ***kwargs*) → float

Compute the transfer entropy from a source variable, X, to a target variable, Y, conditioned on other variables contained in Z.

Parameters

- **X** (*np.array*) – Source sample from a random variable X
- **Y** (*np.array*) – Target sample from a random variable Y
- **Z** (*np.array*) – Conditioning variable(s).
- **tau** (*int (default: 1)*) – Number of timestep lags for the source variable
- **omega** (*int (default: 1)*) – Number of timestep lags for the target variable conditioning
- **nu** (*int (default: 1)*) – Number of timestep lags for the source variable conditioning
- **k** (*int (default: 1)*) – Width of window for the source variable.
- **l** (*int (default: 1)*) – Width of window for the target variable conditioning.
- **m** (*int (default: 1)*) – Width of window for the source variable conditioning.
- **neighbors** (*int (default: K)*) – Parameter controlling the number of neighbors to use in estimation.
- ****kwargs** – Other arguments (undocumented, for internal usage)

Returns **conditional_transfer_entropy** – Computed via conditional_mutual_info

Return type float

References

hyeenna.estimators.**entropy**(*X: numpy.array*, *k: int = 5*) → float

Computes the Shannon entropy of a random variable X using the KL nearest neighbor estimator.

The formula is given by: $$ hat{H}(X) = psi(N) - psi(k) + log(C_d) + d langle log(epsilon)

angle

$$

where

- N is the number of samples
- k is the number of neighbors

- $psi is the digamma function
- $

angle cdot angle$ is the mean

- $epsilon_i$ is the 2 times the distance to the k^{th} nearest neighbor.

X: np.array Sample from a random variable

k: int, optional Number of neighbors to use in estimation

ent: float estimated entropy

hyeenna.estimators.**kl_divergence**(*P: numpy.array*, *Q: numpy.array*, *k: int = 5*)

Compute the KL divergence

Parameters

- **P** (*np.array*) – Sample from random variable P
- **Q** (*np.array*) – Sample from random variable Q
- **k** (*int, optional*) – Number of neighbors to use in estimation

Returns

Return type estimated KL divergence D(P|Q)

References

hyeenna.estimators.**marginal_neighbors**(*X: numpy.array*, *R: numpy.array*, *metric='chebyshev'*) → list

Number of neighbors within a certain radius

hyeenna.estimators.**mi_local_nonuniformity_correction**(*X*, **args*, *k: int = 5*, *alpha=1.05*, ***kwargs*)

Compute the local nonuniformity correction factor for strongly dependent variables. This correction is calculated based on the structure of the space of k-nearest neighbors. The volume of the hyper-rectangle of the maximum-norm bounding box for the k-nearest neighbor estimation is compared to that of the hyper-rectangle bounding the principal components of the covariance matrix of the k-nearest neighbor locations.

Parameters

- **X** (*np.array*) – A sample from a random variable
- ***args** (*List[np.array]*) – Samples from random variables
- **k** (*int, optional*) – Number of neighbors to use in estimation.
- **alpha** (*float, optional*) – Sensitivity parameter for filtering non-dependent volumes
- ****kwargs** (*np.array*) – Samples from random variables

Returns **lnc** – The correction factor to be subtracted from the mutual information

Return type float

References

Estimation of Mutual Information for Strongly Dependent Variables. Retrieved from https://arxiv.org/abs/1411.2003v3

hyeenna.estimators.**multi_mutual_info**(*X: numpy.array*, **args*, *k: int = 5*, ***kwargs*) → float

Computes the multivariate mututal information of several random variables using the KSG nearest neighbor estimator.

The formula is given by: $$ hat{I}(X_1,...,X_m) = (m-1)cdotpsi(N) + psi(k) -

rac{m-1}{k}

- langle psi(n_{X_1} +1) + ... + psi(n_{X_m} +1)

angle

$$

where

- N is the number of samples
- m is the number of variables
- k is the number of neighbors
- $psi is the digamma function
- $

angle cdot angle$ is the mean

- $

_i$ is the number of points within the distance of

the k^{th} nearest neighbor when projected into the subspace spanned by i.

X: np.array A sample from a random variable

***args: List[np.array]** Samples from random variables

k: int, optional Number of neighbors to use in estimation.

****kwargs: np.array** Samples from random variables

mi: float The mutual information

hyeenna.estimators.**mutual_info**(*X: numpy.array*, *Y: numpy.array*, *k: int = 5*) → float

Computes the Mututal information of two random variables, X and Y, using the KSG nearest neighbor estimator.

The formula is given by: $$ hat{I}(X,Y) = psi(N) + psi(k) -

rac{1}{k}

- langle psi(n_X +1) + psi(n_Y +1)

angle

$$

where

- N is the number of samples
- k is the number of neighbors
- $psi is the digamma function
- $

angle cdot angle$ is the mean

- $

_i$ is the number of points within the distance of

the k^{th} nearest neighbor when projected into the subspace spanned by i.

X: np.array A sample from a random variable

Y: np.array A sample from a random variable

k: int, optional Number of neighbors to use in estimation.

mi: float The mutual information

hyeenna.estimators.**nearest_distances**(*X: numpy.array*, *Y: numpy.array = None*, *k: int = 5*, *metric='chebyshev'*) → list

Distance to the kth nearest neighbor

hyeenna.estimators.**nearest_distances_vec**(*X: numpy.array*, *Y: numpy.array = None*, *k: int = 5*, *metric='chebyshev'*) → numpy.array

Find vector distance to all k nearest neighbors

hyeenna.estimators.**transfer_entropy**(*X: numpy.array*, *Y: numpy.array*, *tau: int = 1*, *omega: int = 1*, *k: int = 1*, *l: int = 1*, *neighbors: int = 5*, ***kwargs*) → float

Compute the transfer entropy from a source variable, X, to a target variable, Y.

Parameters

- **X** (*np.array*) – Source sample from a random variable X
- **Y** (*np.array*) – Target sample from a random variable Y
- **tau** (*int (default: 1)*) – Number of timestep lags for the source variable
- **omega** (*int (default: 1)*) – Number of timestep lags for the target variable conditioning
- **k** (*int (default: 1)*) – Width of window for the source variable.
- **l** (*int (default: 1)*) – Width of window for the target variable conditioning.
- **neighbors** (*int (default: K)*) – Parameter controlling the number of neighbors to use in estimation.
- ****kwargs** – Other arguments (undocumented, for internal usage)

Returns **transfer_entropy** – Computed via conditional_mutual_info

Return type float

References

6.1.2 Analysis

hyeenna.analysis.**estimate_info_transfer_network**(*varlist: list*, *names: list*, *tau: int = 1*, *omega: int = 1*, *nu: int = 1*, *k: int = 1*, *l: int = 1*, *m: int = 1*, *condition: bool = True*, *nruns: int = 10*, *sample_size: int = 3000*) → pandas.core.frame.DataFrame

Compute the pairwise transfer entropy for a list of given variables, resulting in an information transfer network.

Parameters

- **varlist** (*list*) – List of given variable data
- **names** (*list*) – List of names corresponding to the data given in *varlist*
- **tau** (*int (default=1)*) – Lag value for source variables
- **omega** (*int (default=1)*) – Lag value for conditioning target variable history
- **nu** (*int (default=1)*) – Lag value for conditioning source variable histories
- **k** (*int (default=1)*) – Window length for source variables (applied to the same variable as the *tau* parameter)
- **l** (*int (default=1)*) – Window length for target variable histories (applied to the same variable as the *omega* parameter)
- **m** (*int (default=1)*) – Window length for source conditioning variables (applied to the same variable as the *nu* parameter)
- **condition** (*bool (default=False)*) – Whether to condition on all variables, or just the target variable history.
- **nruns** (*int (default=10)*) – Number of samples to compute for each connection. The median value is reported.
- **sample_size** (*int (default=3000)*) – Size of samples to take during estimation of transfer entropy.

Returns **df** – Dataframe representing the information transfer network. Both rows and columns are populated with the given *names*.

Return type pd.DataFrame

hyeenna.analysis.**estimate_timescales**(*X: numpy.ndarray*, *Y: numpy.ndarray*, *lag_list: list*, *window_list: list*, *sample_size: int = 5000*) → pandas.core.frame.DataFrame

Compute the transfer entropy (TE) over a range of lag counts and window sizes.

Parameters

- **X** (*np.array*) – Source data
- **Y** (*np.array*) – Target data
- **lag_list** (*list*) – A list enumerating the lag counts to compute TE with
- **window_list** (*list*) – A list enumerating the window widths to compute TE with
- **sample_size** (*int*) – Number of samples to use when computing TE

Returns **out** – A dataframe containing the computed transfer entropies for every combination of lag and window given in the input parameters

Return type pd.DataFrame

hyeenna.analysis.**estimator_stats**(*estimator: callable*, *data: dict*, *params: dict*, *nruns: int = 10*, *sample_size: int = 3000*) → dict

Compute some statistics about a given estimator.

Parameters

- **estimator** (*callable*) – The estimator to compute statistics on. Suggested to be from the HYEENNA library.
- **data** (*dict*) – Input data to feed into the estimator
- **params** (*dict*) – Parameters to feed into the estimator
- **nruns** (*int (default: 10)*) – Number of times to run the estimator.
- **sample_size** (*int (default 3000)*) – Size of sample to draw from *data* to feed into the estimator

Returns **stats** – A dictionary containing sample statistics along with the actual results from each run of the estimator.

Return type dict

hyeenna.analysis.**shuffle_test**(*estimator: callable*, *data: dict*, *params: dict*, *confidence: float = 0.99*, *nruns: int = 10*, *sample_size: int = 3000*) → dict

Compute a one tailed Z test against a sample of shuffled surrogates.

Parameters

- **estimator** (*callable*) – The estimator to compute statistics on. Suggested to be from the HYEENNA library.
- **data** (*dict*) – Input data to feed into the estimator
- **params** (*dict*) – Parameters to feed into the estimator
- **confidence** (*float (default: 0.99)*) – Confidence level to conduct the test at.
- **nruns** (*int (default: 10)*) – Number of times to run the estimator.
- **sample_size** (*int (default: 3000)*) – Size of sample to draw from *data* to feed into the estimator

Returns **stats** – A dictionary with statistics from the standard *estimator_stats* function along with statistics computed on the shuffled surrogates. Most importantly are the 'test_value' and 'significant' keys, which are the value to perform the test on, along with whether the test result was significantly significant at the given confidence level.

Return type dict

6.1.3 Plotting

class hyeenna.plot.**NumpyEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Credit: https://stackoverflow.com/questions/26646362/numpy-array-is-not-json-serializable

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

Python Module Index

h

Index