Manual for `entropy_estimators.py`

Entropy Estimation Library for Discrete Distributions

Fermín Moscoso del Prado Martín (`fm611@cam.ac.uk`)

July 28, 2025

0.1 Introduction

The `entropy_estimators.py` library provides a suite of entropy estimation techniques for discrete probability distributions, including both classical and advanced Bayesian estimators. These methods are useful in linguistics, ecology, information theory, neuroscience, and any domain where quantifying uncertainty or diversity is required.

0.2 Installation and Dependencies

This library requires the following Python packages:

- `numpy`
- `scipy`
- `mpmath`

Install them using `pip install numpy scipy mpmath`.

0.3 Overview of Estimators

The following entropy estimation techniques are implemented:

Method	Description
MLE	Maximum Likelihood Estimator
JSE	James-Stein Shrinkage Estimator
CAE	Chao-Shen Estimator
CWJ	Chao-Wang-Jost Estimator
NBRS	Nemenman-Bialek-de Ruyter Estimator
NSB	Nemenman-Shafee-Bialek Estimator

0.4 API Reference

0.4.1 Main Function

Entropy

```
Entropy(counts, method="MLE", K=None, base=2)
```

Parameters:

- `counts`: Array or list of non-negative integer frequencies for each observed type.
- `method`: String specifying the estimator to use. One of: `"MLE"`, `"JSE"`, `"CAE"`, `"CWJ"`, `"NBRS"`, `"NSB"`.
- `K`: (Optional) Total number of possible types (used in some estimators).

- `base`: (Optional) Logarithm base for output entropy (`2` for bits, `np.e` for nats, `10` for bans).

Returns: Estimated entropy (float).

0.4.2 Estimator Details

Maximum Likelihood (MLE)

The standard plug-in estimator using observed frequencies:

$$\hat{H}_{\text{MLE}} = -\sum_{i=1}^{k} \frac{n_i}{N} \log \frac{n_i}{N}$$

where n_i is the count of type i, and $N = \sum_i n_i$.

James-Stein Shrinkage (JSE)

Adjusts empirical probabilities by shrinking toward the uniform distribution, reducing small-sample bias.

Coverage-Adjusted Estimator (CAE)

A coverage-adjusted entropy estimator, correcting for unseen types:

$$\hat{H}_{\text{CS}} = -\sum_{i=1}^{k} \tilde{p}_i \log \tilde{p}_i / \left[1 - (1 - \tilde{p}_i)^N\right]$$

where $\tilde{p}_i$ are the Good-Turing adjusted probability estimate,

$$\tilde{p}_i = \left(1 - \frac{f_1}{N}\right) \frac{n_i}{N},$$

with f_1 being the number of hapax legomena observed.

Chao-Wang-Jost (CWJ)

Refines Chao-Shen to improve performance for sparse and undersampled data. Probably the most accurate estimator available.

NBRS

A Bayesian estimator using a Dirichlet prior to robustly estimate entropy when many types are rare or unseen. Use with care, as it tends to dramatically overestimate.

NSB

Integrates over Dirichlet priors, providing state-of-the-art entropy estimates even for heavy-tailed and undersampled distributions. This estimator is very good, but its results are very dependent on the value of K, that needs to be chosen a priori.

0.5 Utility Functions

`sample_frequencies(M, n)`: Samples n items from a frequency array M without replacement.

`dict_to_ndarray(d)`: Converts a dictionary with tuple keys and integer values to a multidimensional array and index mapping.

0.6 Other Methods

`EntropyML(counts)`: Computes the MLE entropy.

`FreqShrink(counts)`: Returns the James-Stein shrinkage probabilities.

`JamesSteinShrink(counts)`: James-Stein entropy estimate.

`ChaoShen(counts)`: Chao-Shen entropy. (i.e., CAE)

`ChaoWangJost(counts)`: Chao-Wang-Jost entropy.

`NBRS(counts)`: NBRS entropy.

`JS_KullbackLeibler(counts1, counts2)`: KL divergence between two count vectors using shrinkage smoothing.

`JS_JensenShannon(counts1, counts2)`: Jensen-Shannon divergence between two distributions.

0.7 NSB Estimator Details

The NSB estimator numerically integrates over Dirichlet priors to infer entropy and its variance. For details, see:

> Ilya Nemenman, Fariel Shafee, and William Bialek, "Entropy and Inference, Revisited", *arXiv:physics/0108025*

0.8 Example Usage

0.8.1 Computing Entropies

```
import numpy as np
from entropy_estimators import Entropy

counts = [10, 4, 1, 0, 5, 0]
print("MLE entropy:", Entropy(counts, method="MLE"))
print("James-Stein entropy:", Entropy(counts, method="JSE"))
print("Chao-Shen entropy:", Entropy(counts, method="CAE"))
print("NSB entropy (with K=10):", Entropy(counts, method="NSB", K=10))
```

0.8.2 Example: KL Divergence and Jensen-Shannon Distance

The library also provides functions for computing divergence measures between two discrete distributions, given as count vectors. These use James-Stein shrinkage for robust estimation.

Kullback-Leibler (KL) Divergence

KL divergence measures how one distribution diverges from another, and is defined as:

$$D_{\mathrm{KL}}(P\|Q) = \sum_i p_i \log \frac{p_i}{q_i}$$

where p_i and q_i are probabilities for type i in the two distributions.

```
from entropy_estimators import JS_KullbackLeibler

# Example count vectors for two distributions
counts1 = [10, 4, 1, 0, 5, 0]
counts2 = [8, 2, 0, 2, 4, 1]

kl_div = JS_KullbackLeibler(counts1, counts2)
print("KL divergence (from counts1 to counts2):", kl_div)
```

Jensen-Shannon (JS) Distance

The Jensen-Shannon distance is a symmetric and smoothed version of KL divergence, defined as:

$$D_{\mathrm{JS}}(P,Q) = \sqrt{\frac{1}{2}D_{\mathrm{KL}}(P\|M) + \frac{1}{2}D_{\mathrm{KL}}(Q\|M)}$$

where $M = \frac{1}{2}(P+Q)$ is the average distribution. We can also smooth it using the James-Stein shrinkage probabilities.

```
from entropy_estimators import JS_JensenShannon

js_dist = JS_JensenShannon(counts1, counts2)
print("Jensen-Shannon distance between counts1 and counts2:", js_dist)
```

Interpretation:

- KL divergence is not symmetric: $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$.
- Jensen-Shannon distance is symmetric and always finite. It is often used as a metric for the similarity of probability distributions.

0.8.3 Downsampling

This library provides utilities for handling frequency distributions over multiple categorical variables, using dictionaries with tuple keys and functions to convert these to arrays suitable for entropy estimation.

Converting a Dictionary of Joint Frequencies to a 2D Matrix

Suppose you have observed counts for pairs of categorical labels (for example, bigram counts in language, or joint counts of two experimental conditions). You can store these in a dictionary:

```
from entropy_estimators import dict_to_ndarray, sample_frequencies
import numpy as np

# Example: Counts for pairs (A,b), (A,c), (B,c), etc.
pair_counts = {
    ('A', 'b'): 10,
    ('A', 'c'): 4,
    ('B', 'b'): 7,
    ('B', 'c'): 3,
    ('C', 'a'): 5
}

# Convert dictionary to a 2D NumPy array and label mappings
array2d, index_maps = dict_to_ndarray(pair_counts)

print("Bidimensional frequency matrix:")
print(array2d)

print("Label-to-index maps for each dimension:")
print(index_maps)
```

This produces a matrix (NumPy array) of frequencies, along with index mappings from original labels to matrix indices for each axis.

Downsampling from a Multidimensional Distribution

You can use `sample_frequencies` to simulate a sample of a given size drawn without replacement from the observed distribution:

```
# Downsample 10 observations from the full matrix
downsampled = sample_frequencies(array2d, 10)
print("Downsampled matrix:")
print(downsampled)
```

This will return a matrix with the same shape as `array2d`, containing the sampled counts.

Interpretation

This workflow is especially useful when you wish to:

- Analyze joint or conditional distributions over two (or more) categorical variables.

- Downsample large observed contingency tables for simulation or robustness checks.
- Convert real-world data in dictionary form into array formats compatible with entropy estimators.

Tip: For higher-dimensional cases, the same approach works with tuple keys of length greater than two.

0.9 References

- Chao, A. and Shen, T.-J. (2003). Nonparametric estimation of Shannon's index of diversity when there are unseen species in sample. *Environmental and Ecological Statistics*, 10(4), 429–443.
- Chao, A., Wang, Y.-T., & Jost, L. (2013). Entropy and the species accumulation curve: a novel entropy estimator via discovery rates of new species. *Methods in Ecology and Evolution*, 4(11), 1091–1100.
- Nemenman, I., Shafee, F., & Bialek, W. (2002). Entropy and inference, revisited. In *Advances in Neural Information Processing Systems* (pp. 471–478).
- Nemenman, I., Bialek, W., & de Ruyter van Steveninck, R. (2004). Entropy and information in neural spike trains: Progress on the sampling problem. *Physical Review E*, 69(5), 056111.
- Hausser, J. and Strimmer, K. (2009). Entropy inference and the James-Stein estimator, with application to nonlinear gene association networks. *Journal of Machine Learning Research*, 10, 1469–1484.