SoftwareX 14 (2021) 100686

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

ennemi: Non-linear correlation detection with mutual information

Petri Laarne [a,*], Martha A. Zaidan [a,c], Tuomo Nieminen [a,b]

[a] *Institute for Atmospheric and Earth System Research/Physics, Faculty of Science, P.O. Box 64, FI-00014 University of Helsinki, Finland*
[b] *Institute for Atmospheric and Earth System Research/Forest Sciences, Faculty of Agriculture and Forestry, P.O. Box 27, FI-00014 University of Helsinki, Finland*
[c] *Joint International Research Laboratory of Atmospheric and Earth System Sciences, School of Atmospheric Sciences, Nanjing University, Nanjing 210023, China*

A R T I C L E   I N F O

Article history:
Received 20 August 2020
Received in revised form 2 March 2021
Accepted 3 March 2021

MSC:
62B10
94A17

Keywords:
Information theory
Mutual information
Correlation analysis
Variable selection

A B S T R A C T

We present ennemi, a Python package for correlation analysis based on mutual information (MI). MI is a measure of relationship between variables. Unlike Pearson correlation it is valid also for non-linear relationships, yet in the linear case the two are equivalent. The effect of other variables can be removed like with partial correlation, with the same equivalence. These features make MI a better correlation measure for exploratory analysis of many variable pairs. Our package provides methods for common correlation analysis tasks using MI. It is scalable, integrated with the Python data science ecosystem, and requires minimal configuration.

Code metadata

Current code version	1.0.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-20-00028
Code Ocean compute capsule	N/A
Legal Code License	MIT License
Code versioning system used	Git
Software code languages, tools, and services used	Python 3
Compilation requirements, operating environments & dependencies	Python 3.6+, NumPy 1.17.5+, SciPy 1.4+
Link to developer documentation/manual	https://polsys.github.io/ennemi/
Support email for questions	petri.laarne@helsinki.fi

1. Motivation and significance

During the past few decades, a rapidly increasing amount of measurement data has become available in various fields of science. This allows us to discover unexpected relationships between measured variables. On the flip side, the number of variable pairs has also grown significantly. In order to focus the exploratory (hypothesis-generating) data analysis on the most promising relationships, an automated process is necessary. The most popular correlation coefficient, Pearson's r, is not a sufficient measure for this purpose, since many natural phenomena are non-linear.

Mutual information (MI) is an information-theoretic measure of dependency between two random variables [1]. It is defined as the difference between marginal and joint *entropies*, or equivalently as

$$I(X;Y) = \int_{\mathbb{R}^2} p(x,y) \log\left(\frac{p(x,y)}{p(x)p(y)}\right) d(x,y), \tag{1}$$

where p denotes the respective probability densities. The definition is analogous for discrete variables. In addition, *conditional MI*, denoted by $I(X;Y|Z)$, is defined by conditioning on a third, possibly multidimensional, variable. It is analogous to partial

* Corresponding author.
E-mail address: petri.laarne@helsinki.fi (P. Laarne).

correlation. In both cases, zero MI implies independence between the two variables.

Mutual information ranges over $[0, \infty]$, but can be normalized to a correlation coefficient in $[0, 1]$ by

$$\rho_{I(X;Y)} = \sqrt{1 - \exp(-2\,I(X;Y))}. \tag{2}$$

If (X, Y) is normally distributed, then $\rho_{I(X;Y)}$ equals the Pearson correlation coefficient between the two variables. The same equivalence applies between conditional MI and partial correlation coefficient.

Furthermore, MI is invariant under monotonic transformations of variables. This means that the MI correlation coefficient of a non-linear model (X, Y) matches the Pearson correlation of the linearized model $(f(X), g(Y))$. General conditions for f and g are described in [2, Thm. 1.6.3].

MI has been used for correlation analysis across fields, such as in bioinformatics [3] and atmospheric sciences [4]. Specialized software includes MDEntropy [5] for molecular dynamics and IDTxl [6] for network inference. Machine learning packages such as scikit-learn [7] apply MI as part of feature selection. Self-contained packages such as infotheory [8] or pycit [9] are somewhat technical to use.

Our goal is to present MI as a correlation measure, a term well understood by practitioners. Hence we focus on compatibility with existing workflows and on variety of use cases rather than variety of algorithms. Our package, named *ennemi*, is distinguished by the following features:

- Simple, documented interface for common analysis workflows,
- Integration with the Python data science ecosystem,
- Support for discrete–continuous and (multi-dimensional) conditional MI,
- Minimal exposure of technical details,
- Good and scalable runtime performance.

There are three common methods for estimating MI from continuous data: binning/discretization (used by infotheory), kernel density estimation [10], and k-nearest neighbor search [11] (scikit-learn, pycit). We have chosen the last one because it performs well and requires only a single parameter k, the effect of which is relatively small [11]. The other methods are sensitive to bin/kernel width [e.g. 12, Figure 2]. We also use the algorithm variants for discrete–continuous MI [12] and conditional MI [13].

2. Software description

The implementation of ennemi consists of two parts: the algorithms and the public interface. The latter module implements all the distinguishing features: parallelism, lags, masking and optional Pandas data type support. This split greatly simplifies the algorithms module, which only handles raw NumPy `ndarray` variables and independent estimation tasks. The algorithms are straightforward implementations of the referenced articles.

2.1. Functionalities

The primary methods of ennemi are `estimate_mi` and `pairwise_mi`. The first compares a variable against one or more variables, whereas the second does pairwise comparisons between a set of variables. Data may be passed in as Python lists, `ndarray`s or Pandas `DataFrame`s (in which case the result will have column names).

The `estimate_mi` method supports time lags of variables. When a positive lag Δ is supplied, the data points $y_{1+\Delta}, \ldots, y_n$ are compared against $x_1, \ldots, x_{n-\Delta}$, and symmetrically for negative lags. If multiple lags are specified, the set of y observations is kept fixed. This ensures that all estimates use the same subset of the variable of interest.

Both methods accept a mask parameter. The mask is applied to y values and the translated x values. This enables using a subset of data without manually adapting the mask to lags. Both methods also accept a conditioning variable, which may be multidimensional.

The nearest neighbor algorithm produces the most accurate results when the variables are roughly symmetrical and of same variance. If there are duplicate observations (e.g. due to low precision), some noise should be added. The package rescales variables and adds reproducible noise by default. However, the user needs to transform the marginal distributions to symmetric (e.g. by taking logarithms) if necessary.

Finally, there are additional parameters for returning correlation values (Eq. (2)), specifying that y is discrete, and tuning the algorithm. Full description and examples are included in ennemi documentation. For completeness, we have also included a method (`estimate_entropy`) for continuous entropy estimation.

2.2. Performance

The execution time is close to linear in sample size n, and increases with the number of conditioning variables and neighbors to search. Measurements of run time for different values of n and k are shown in Fig. 1.

The k-nearest neighbor algorithm consists of two main steps: find the L^∞ (maximum norm) distance to the k'th neighbor of a point, and count the points within that distance in marginal spaces. These steps use a multidimensional tree structure provided by the `scipy.spatial.cKDTree` class. The class is implemented in C++ and offers efficient vectorization.

Python enforces data consistency by executing only a single Python thread at a time. The `cKDTree` methods release the Global Interpreter Lock (GIL) while compiled code is executed, and therefore it is possible to use thread-level parallelism. Large estimation tasks parallelize nearly perfectly as the thread overhead and GIL contention are relatively small. We use simple heuristics to run very short tasks on a single thread.

The performance measurements in this section were done on a desktop computer with Intel i5-4670 (4 cores at 3.40 GHz) CPU, 16 GB of DDR3 RAM, Windows 10 (version 1910), 64-bit Python 3.8.5, NumPy 1.19.1, and SciPy 1.5.2. The code and other benchmarks are included in ennemi source repository.

2.3. Verification

We have implemented an extensive test suite consisting of unit and integration tests. A continuous integration (CI) system hosted on GitHub runs the unit tests on every pull request, using combinations of the supported operating systems and Python versions. The line coverage of unit tests is measured and required to be close to 100%. For testing the algorithms, we use analytical expressions of MI, most importantly those of Gaussian distribution and some described in [14].

The integration tests simulate more realistic workflows, using either real data or simulated datasets with numerical reference results. For example, the tests reproduce the results of [12,13].

Because Python is a dynamically-typed language, type errors occur only at run time. To detect these errors earlier, we have decorated all methods with type annotations defined in the standard library [15] for verification with the Mypy static analyzer. These annotations are also available to package users. The coverage is still partial, as NumPy and other dependencies are in progress of adding type information.

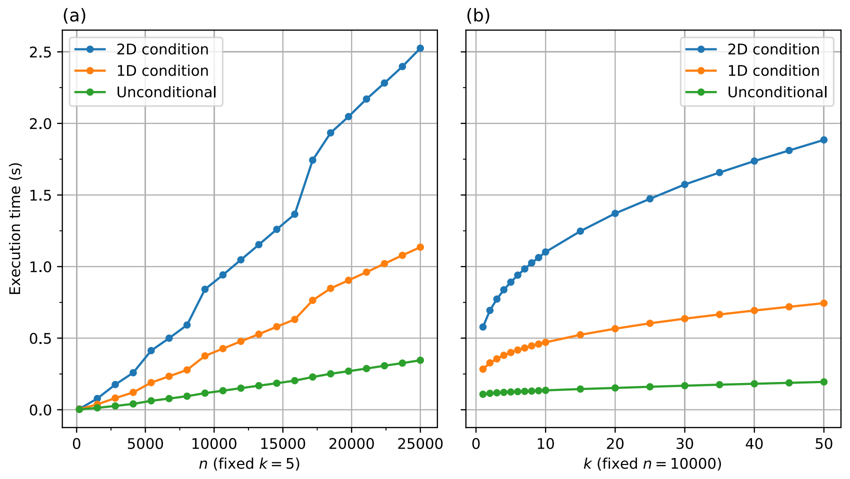

Fig. 1. Execution time (best of 10 repeats) of a single MI estimation with zero (unconditional), one (1D condition), and two (2D condition) conditioning variables, as a function of (a) number of data points n, and (b) the parameter k. All variables are independent standard Gaussians.

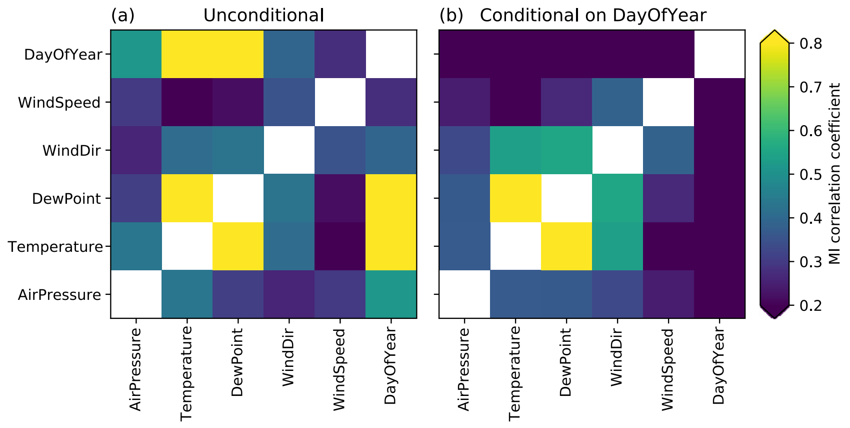

Fig. 2. The pairwise MI between each variable in the Helsinki meteorological data set at 3 PM local time. In (a) the unconditional correlation is shown, and in (b) the correlation is conditioned on the day of year.

We have also automated the package release process. When a release is created on GitHub, a script submits the package to Python Package Index (PyPI). The released code versions are archived on Zenodo, where the exact versions are citable [16].

3. Illustrative example

We demonstrate the basic usage with meteorological observations (wind speed and direction, dew point, temperature, and air pressure) from 2015 to 2019 at the Kaisaniemi weather station in Helsinki, Finland. An extended version of this example is included in ennemi documentation. The original data was obtained from the Finnish Meteorological Institute.[1]

```
from ennemi import estimate_mi, pairwise_mi
import numpy as np
import pandas as pd

data = pd.read_csv("kaisaniemi.csv",
    index_col=0, parse_dates=True)
```

Listing 1: Import of the package and data.

[1] https://en.ilmatieteenlaitos.fi/download-observations, accessed 13 August 2020.

To begin, we import ennemi, NumPy, and Pandas, and then load the data as a Pandas data frame (Listing 1). By visual inspection, the variables have roughly symmetric distributions, so there is no need to e.g. take logarithms. Therefore the automatic rescaling of variables suffices.

A good first step is to plot the pairwise MI between all variables. Because the algorithm assumes independent samples whereas meteorological data is highly autocorrelated, we have to select only one sample per day. For example, we can calculate the dependency between variables at 3 PM local time, both with and without conditioning on the day of year (Listing 2).

```
# The data is at hourly intervals and in UTC
mask = (data.index.hour == 13)

pairwise = pairwise_mi(data, mask=mask,
    normalize=True)
pairwise_doy = pairwise_mi(data, mask=mask,
    cond=data["DayOfYear"], normalize=True)
```

Listing 2: Calculation of pairwise MI between variables.

The resulting matrices can be plotted with e.g. Matplotlib, as is done in Fig. 2. The strong seasonal cycles of temperature and dew point are clearly visible in the unconditional plot. Conditioning removes this effect without requiring further model specification.

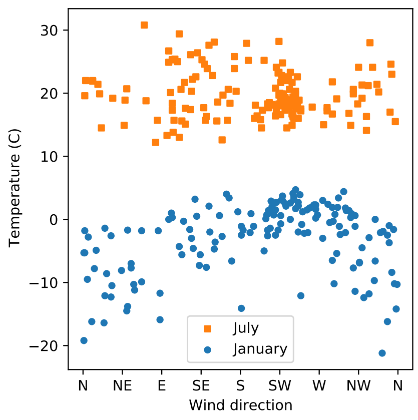

Fig. 3. The correlation between wind direction and temperature in Helsinki depends on the season, with Western winds associated with less extreme temperatures.

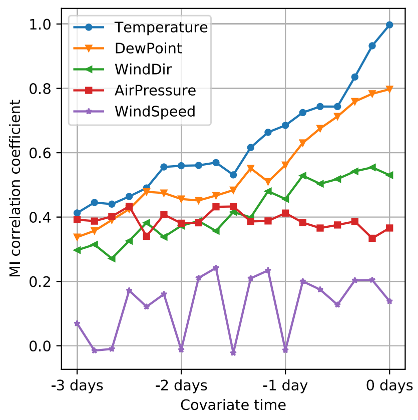

Fig. 4. The dependency between temperature at 3 PM and earlier observations of variables.

The correlation between temperature and wind direction increases with conditioning, correctly suggesting that the air mass source affects the temperature within a season. This dependency could be difficult to see with traditional methods because the sign of the correlation varies seasonally, as shown in Fig. 3.

```
# Lag up to three days with four hour spacing
lags = np.arange(0, 3*24 + 1, 4)

mi = estimate_mi(data["Temperature"], data,
     lags, cond=data["DayOfYear"],
     mask=afternoon_mask, normalize=True)
```

Listing 3: Estimation of time dependency after removing seasonal effect.

We can also estimate the time dependency between variables. For example, how long into the future can we predict the temperature? This can be calculated by passing covariate lags into `estimate_mi`. The seasonal effect is removed by conditioning on the day of year (Listing 3).

The results are displayed in Fig. 4. As can be expected, the correlations decrease as the time difference increases. The air pressure is an exception, suggesting that it is the slowest to change. The correlation between temperature and wind speed is very low regardless of the time delay.

There is significant noise in the lowest correlation values. This is because the normalization formula (Eq. (2)) is non-linear. Increasing the k parameter improves the accuracy of low values, but may introduce a small bias to high values. To increase the effective sample size from 1826 days without violating the independence assumption, we can average the results of separate, nearly identical estimations: for example, the temperature observations fixed at 2 PM, 3 PM, and 4 PM.

4. Impact

We have used ennemi to evaluate the applicability of MI for detecting non-linear correlations in large atmospheric datasets. We have used data from the University of Helsinki SMEAR stations network [17], where comprehensive long-term measurements of meteorological, atmospheric and ecological parameters are performed. These datasets allow detailed studies of various processes, interactions and feedbacks in the atmosphere–biosphere system.

In our past experience, MI has performed well in detecting known linear and non-linear correlations [4,18]. The primary issues are the necessity to work around autocorrelation and the inaccuracy of low MI values. There is also the need to symmetrize the marginal distributions of variables, although this process could feasibly be automated.

There are other similar measures of variable relationship such as maximal information coefficient [19] and transfer entropy [20]. The simplicity and the theoretical equivalence with Pearson correlation make MI attractive even when more advanced methods are available. Transfer entropy could be a future inclusion to ennemi.

Our new package is distinguished by being aimed specifically at data analysis. It requires little information-theoretic knowledge from the user and works with the common Python ecosystem for data manipulation. We have also worked extensively on testing and performance in order to provide a reliable building block for future applications.

5. Conclusions

We have presented a Python package for estimation of mutual information. The package is designed for the non-linear correlation detection as part of a modern data analysis pipeline. Therefore, it features integration with Pandas data types and supports masks, time lags, and normalization to correlation coefficient scale. Our implementation is also extensively tested, portable, and reasonably fast.

We believe that ennemi can help discover new relationships in datasets. The underlying measure has already been used both in specialized applications and general correlation analysis across multiple fields. By presenting a ready-to-use software package, we hope to encourage more widely applicable exploratory correlation analysis in different contexts.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by Academy of Finland PROFI3 funding (decision number 311932).

References

[1] Cover TM, Thomas JA. Elements of information theory. Wiley; 2006.
[2] Ihara S. Information theory for continuous systems. World Scientific; 1993.
[3] Lachmann A, Giorgi FM, Lopez G, Califano A. ARACNe-AP: gene network reverse engineering through adaptive partitioning inference of mutual information. Bioinformatics 2016;32(14):2233–5. http://dx.doi.org/10.1093/bioinformatics/btw216.
[4] Zaidan MA, Haapasilta V, Relan R, Paasonen P, Kerminen V-M, Junninen H, Kulmala M, Foster AS. Exploring non-linear associations between atmospheric new-particle formation and ambient variables: a mutual information approach. Atmos Chem Phys 2018;18(17):12699–714. http://dx.doi.org/10.5194/acp-18-12699-2018.
[5] Hernández CX, Pande VS. MDEntropy: Information-theoretic analyses for molecular dynamics. J Open Source Softw 2017;2(19):427. http://dx.doi.org/10.21105/joss.00427.
[6] Wollstadt P, Lizier J, Vicente R, Finn C, Martinez-Zarzuela M, Mediano P, Novelli L, Wibral M. IDTxl: The Information Dynamics Toolkit xl: a Python package for the efficient analysis of multivariate information dynamics in networks. J Open Source Softw 2019;4(34):1081. http://dx.doi.org/10.21105/joss.01081.
[7] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825–30.
[8] Candadai M, Izquierdo E. infotheory: A C++/Python package for multivariate information theoretic analysis. J Open Source Softw 2020;5(47):1609. http://dx.doi.org/10.21105/joss.01609.
[9] Yang A. pycit. URL https://pypi.org/project/pycit/.
[10] Moon Y-I, Rajagopalan B, Lall U. Estimation of mutual information using kernel density estimators. Phys Rev E 1995;52(3):2318–21. http://dx.doi.org/10.1103/PhysRevE.52.2318.
[11] Kraskov A, Stögbauer H, Grassberger P. Estimating mutual information. Phys Rev E 2004;69(6):066138. http://dx.doi.org/10.1103/PhysRevE.69.066138.
[12] Ross BC. Mutual information between discrete and continuous data sets. PLoS One 2014;9(2):e87357. http://dx.doi.org/10.1371/journal.pone.0087357.
[13] Frenzel S, Pompe B. Partial mutual information for coupling analysis of multivariate time series. Phys Rev Lett 2007;99(20):204101. http://dx.doi.org/10.1103/PhysRevLett.99.204101.
[14] Darbellay G, Vajda I. Entropy expressions for multivariate continuous distributions. IEEE Trans Inform Theory 2000;46(2):709–12. http://dx.doi.org/10.1109/18.825848.
[15] van Rossum G, Lehtosalo J, Langa L. PEP 484 – Type hints. Python Software Foundation; 2015, URL https://www.python.org/dev/peps/pep-0484/.
[16] Laarne P. polsys/ennemi, latest version. http://dx.doi.org/10.5281/zenodo.3834018.
[17] Hari P, Kulmala M. Station for measuring ecosystem–atmosphere relations (SMEAR II). Boreal Environ Res 2005;10(5):315–22, URL http://www.borenv.net/BER/archive/pdfs/ber10/ber10-315.pdf.
[18] Zaidan MA, Dada L, Alghamdi MA, Al-Jeelani H, Lihavainen H, Hyvärinen A, Hussein T. Mutual information input selector and probabilistic machine learning utilisation for air pollution proxies. Appl Sci 2019;9(20):4475. http://dx.doi.org/10.3390/app9204475.
[19] Reshef DN, Reshef YA, Finucane HK, Grossman SR, McVean G, Turnbaugh PJ, Lander ES, Mitzenmacher M, Sabeti PC. Detecting novel associations in large data sets. Science 2011;334(6062):1518–24. http://dx.doi.org/10.1126/science.1205438.
[20] Kaiser A, Schreiber T. Information transfer in continuous processes. Physica D 2002;166(1–2):43–62. http://dx.doi.org/10.1016/S0167-2789(02)00432-3.