Implemetation of an intrinsic dimension estimator for locally undersampled data

Tommaso Tarchi

June 28, 2024

University of Trieste

Original paper

Erba, V., Gherardi, M. Rotondo, P. Intrinsic dimension estimation for locally undersampled data. Sci Rep 9, 17133 (2019). https://doi.org/10.1038/s41598-019-53549-9

The algorithm

The problem

From the original paper by Erba, Gherardi and Rotondo:

"All the existing intrinsic dimension estimators are not reliable whenever the dataset is locally undersampled, and this is at the core of the so called **curse of dimensionality**".

The problem

From the original paper by Erba, Gherardi and Rotondo:

"All the existing intrinsic dimension estimators are not reliable whenever the dataset is locally undersampled, and this is at the core of the so called **curse of dimensionality**".

This is particularly relevant when dealing with high dimension or strongly curved manifolds (in which local sampling is required).

Proposed solution

We can project data on a unitary-radius hypershpere, where density can be easily estimated.

Full correlation integral ("FCI")

Empirical correlation integral:

$$\rho\left(r\right) = \frac{2}{N\left(N-1\right)} \sum_{1 \leq \mu < \nu \leq N} \theta\left(r - ||\mathbf{x}^{\mu} - \mathbf{x}^{\nu}||\right),$$

where r is the cutoff distance.

Full correlation integral ("FCI")

Empirical correlation integral:

$$\rho\left(r\right)=\frac{2}{N\left(N-1\right)}\sum_{1\leq\mu<\nu\leq N}\theta\left(r-||\mathbf{x}^{\mu}-\mathbf{x}^{\nu}||\right),$$

where r is the cutoff distance.

FCI estimator for a (uniformly sampled) (d-1)-dimensional sphere of radius 1:

$$\bar{\rho}_{r_s}\left(r\right)=\frac{1}{2}+\frac{\Omega_{d-1}}{2\Omega_d}\left(\bar{r}^2-2\right)F_{2,1}\left(\frac{1}{2},1-\frac{d}{2};\frac{3}{2};\left(\bar{r}^2-2\right)^2\right),$$

where Ω_d is the d-dimensional solid angle, $F_{2,1}$ is the (2,1)-hypergeometric function and $\bar{r}=\frac{r}{r_s}$.

Global ID estimator

Algorithm:

1. center and normalize data (i.e. project on hypersphere)
2. measure empirical correlation integral as function of the radius r
3. perform a non-linear regression on the computed empirical FCI using the estimator, with d and r_s as free parameters
4. add one to the estimated d, since the normalization step removes one degree of freedom

Multiscale ID estimator

Algorithm:

1. select a random datapoint x_0 and a *number of nearest neighbors* k, and fit FCI estimator using only x_0's k nearest neighbors
2. repeat step 1 for several values of k
3. select the optimal estimate of ID

Evaluation

Original datasets for global estimator

- $D_{d,D}$: uniform sampling of $\{0,1\}^d$, linearly embedded.
- $G_{d,D}$: sampling of $\mathbb{R}^d$ with the multivariate Gaussian distribution of covariance matrix $\mathbb{I}$ and null mean, linearly embedded.
- $H_{d,D}$: uniform sampling of $[0,1]^d$, linearly embedded.

Original datasets for multiscale estimator

- $C_{d,2d}$: uniform sampling of $[0,2]^d$, embedded with the map

$$\phi\left(x_1 ... x_d\right) = \left(x_2 \cos(x_1), x_2 \sin(x_1) ... x_1 \cos(x_d), x_1 \sin(x_d)\right).$$

- $B_{5n,81^2}$: dataset of high-contrast bitmap images (81×81 pixel) of n blobs.

Implementation

Used packages

- Vectorization: **NumPy**
- Random variables: **NumPy**, **Random**
- Non-linear fit: **SciPy** (scipy.optimize.curve_fit)
- Hypergeometric computation: **SciPy**, **mpmath**
- Nearest neighbors: **scikit-learn**
- TwoNN for comparison: **DADApy**
- Data visualization: **JSON**, **Matplotlib**

Numerical details - hypergeometric function

First issue: some hypergeometric function implementations do not work for particular sets of inputs.

Numerical details - hypergeometric function

First issue: some hypergeometric function implementations do not work for particular sets of inputs.

Solution: we use two methods:

1. scipy.special.hyp2f1
    - pros: compatible datatype with numpy
    - cons: cannot handle complex outputs and particular inputs
2. mpmath.hyp2f1
    - pros: high precision, can handle wide variety of inputs and complex outputs
    - cons: needs typecasting

We choose depending on the specific set of inputs.

Numerical details - hypergeometric function

Second issue: in any case, the function cannot be compute when the last argument is larger than 1.

Numerical details - hypergeometric function

Second issue: in any case, the function cannot be compute when the last argument is larger than 1.

Solution: we normalize the last argument (i.e. we divide by the largest radius considered).

Apparently, the algorithm effectiveness is not greatly invalidated. Also, the parameter r_s becomes useless (always $\simeq 1$ in fitting).

Numerical details - solid angles

Third issue: for high dimensions d (typically larger than 340) it is not possible to compute the solid angles ratio (division by zero).

Numerical details - solid angles

Third issue: for high dimensions d (typically larger than 340) it is not possible to compute the solid angles ratio (division by zero).

Solution: we use the *Stirling approximation* of the Γ function to approximate the angle ratio:

$$\frac{\Omega_{d-1}}{\Omega_d} = \frac{2\pi^{\frac{d-1}{2}}}{\Gamma\left(\frac{d-1}{2}\right)} \frac{\Gamma\left(\frac{d}{2}\right)}{2\pi^{\frac{d}{2}}} \simeq \dots = \frac{1}{2\pi e} \frac{(d-2)^{\frac{d-1}{2}}}{(d-3)^{\frac{d-2}{2}}},$$

where we used $\Gamma(x) \simeq \sqrt{2\pi(x-1)}\left(\frac{x-1}{e}\right)^{x-1}$.

Numerical details - solid angles

Third issue: for high dimensions d (typically larger than 340) it is not possible to compute the solid angles ratio (division by zero).

Solution: we use the *Stirling approximation* of the Γ function to approximate the angle ratio:

$$\frac{\Omega_{d-1}}{\Omega_d} = \frac{2\pi^{\frac{d-1}{2}}}{\Gamma\left(\frac{d-1}{2}\right)} \frac{\Gamma\left(\frac{d}{2}\right)}{2\pi^{\frac{d}{2}}} \simeq ... = \frac{1}{2\pi e} \frac{(d-2)^{\frac{d-1}{2}}}{(d-3)^{\frac{d-2}{2}}},$$

where we used $\Gamma(x) \simeq \sqrt{2\pi(x-1)} \left(\frac{x-1}{e}\right)^{x-1}$.

(**Actually**, there are still numerical issues for $d > 350$ (overflow), so we have to use a "log-exp" trick to compute the approximation)

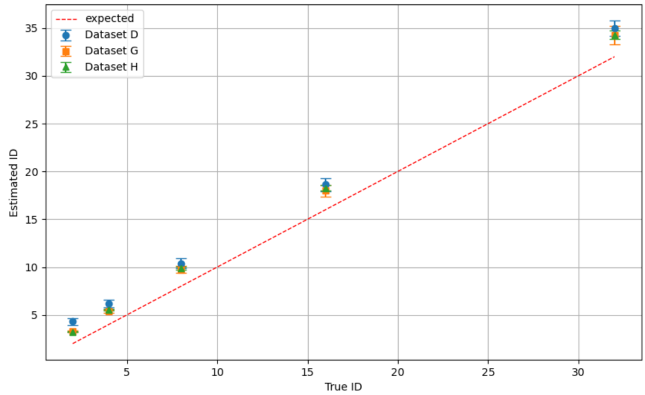
Linear embedding

For linear embedding of the datasets we used the following procedure:

1. add 0-padding to each datapoint until the embedding dimension is reached
2. choose a random hyperplane described by two random vectors in the embedding space (orthonormalized by Gram-Schmidt), and a random angle
3. rotate each datapoint of the extracted angle along the extracted hyperplane

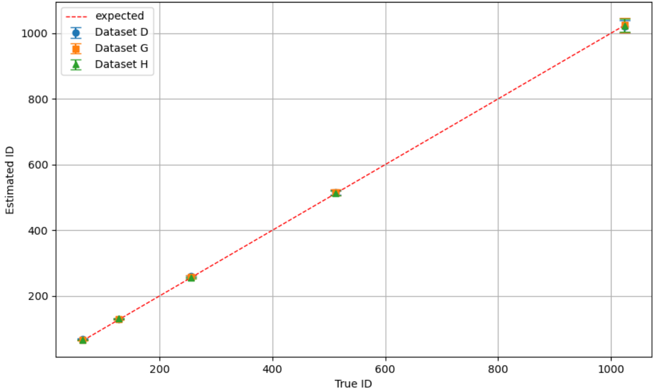
Results

Varying ID - small values



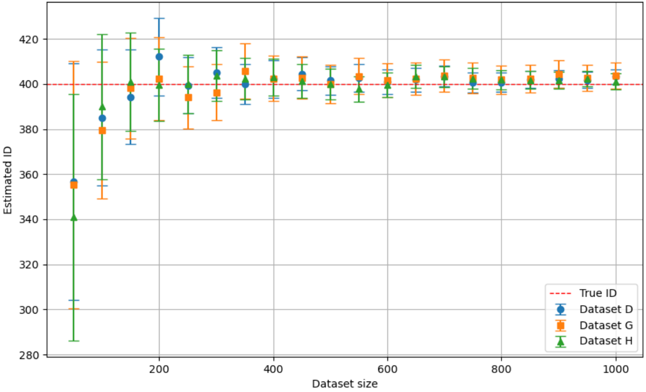
Increasing ID - embedding dimension = 1500, dataset size = 700

Varying ID - large values



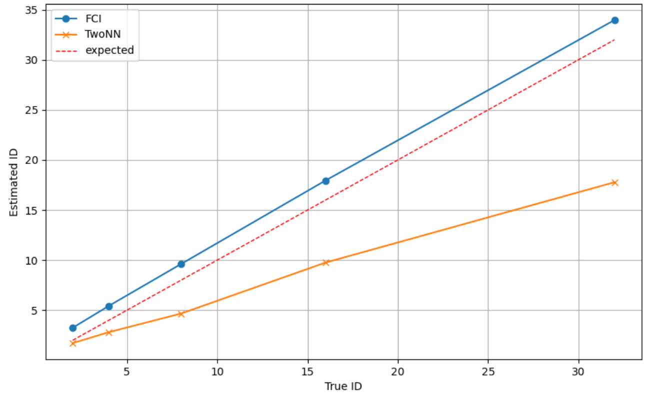
Increasing ID - embedding dimension = 1500, dataset size = 700

Varying dataset size



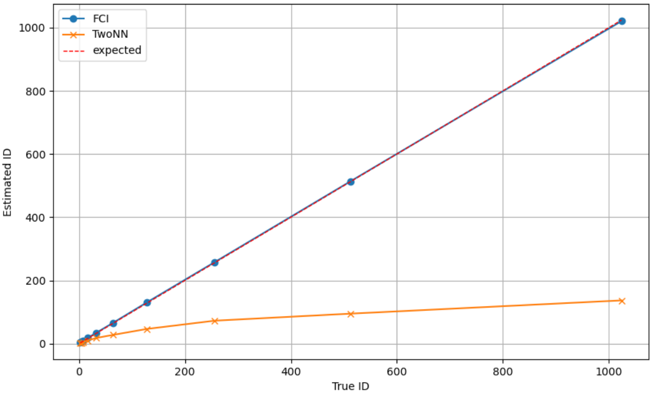
Increasing dataset size - ID = 400, embedding dimension = 1000

Comparison with TwoNN - small values



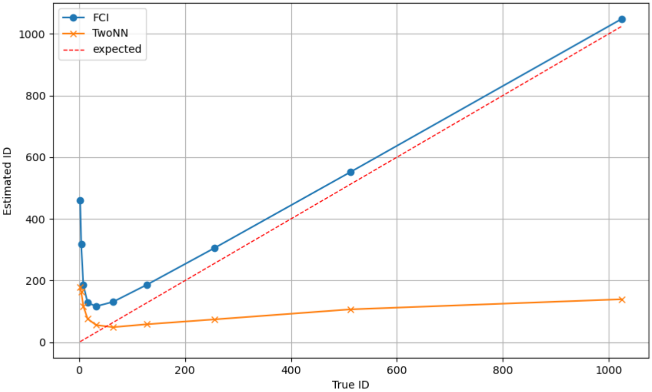
Increasing ID - dataset H with embedding dimension = 1500, dataset size = 700

Comparison with TwoNN - large values



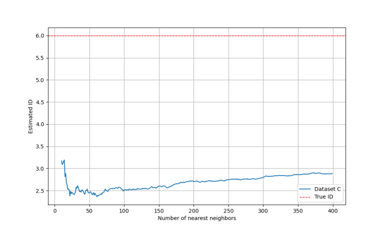
Increasing ID - dataset H with embedding dimension = 1500, dataset size = 700

Comparison with TwoNN - with noise

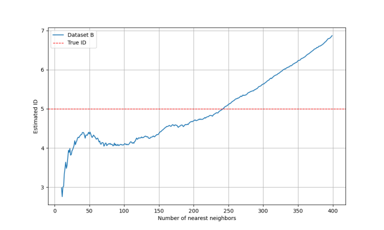


Increasing ID - dataset H with embedding dimension = 1500, dataset size = 700, with Gaussian noise ($\sigma = 0.08$).

Multiscale estimator



(a) Dataset C - increasing k, ID $= 6$, embedding dimension $= 12$, dataset size $= 10000$



(b) High-contrast images dataset - increasing k, ID $= 5$, embedding dimension $= 6561$, dataset size $= 10000$

Results - final considerations

- The estimator is not particularly effective for low dimensions ($\simeq$ constant overestimation), but extremely accurate for large ones.

Results - final considerations

- The estimator is not particularly effective for low dimensions ($\simeq$ constant overestimation), but extremely accurate for large ones.
- The estimator is pretty robust to Gaussian noise.

Results - final considerations

- The estimator is not particularly effective for low dimensions ($\simeq$ constant overestimation), but extremely accurate for large ones.
- The estimator is pretty robust to Gaussian noise.
- The multiscale version (at least my implementation!) does not seem to work in many cases.

Results - final considerations

- The estimator is not particularly effective for low dimensions ($\simeq$ constant overestimation), but extremely accurate for large ones.
- The estimator is pretty robust to Gaussian noise.
- The multiscale version (at least my implementation!) does not seem to work in many cases.
- No clear criterion to choose optimal estimation in multiscale version.

References

- Erba, V., Gherardi, M. Rotondo, P. Intrinsic dimension estimation for locally undersampled data. Sci Rep 9, 17133 (2019). https://doi.org/10.1038/s41598-019-53549-9
- https://github.com/TommasoTarchi/My_UL_library
- https://github.com/davidmickisch/torch-rot/blob/main/torch_rot/rotations.py

Thank you!