Microbiome Project Report

1. Data Base description

Initial data contains 14098 samples from 4 databases (AGP, ibd, ptb, t2d).
Every sample is containing information about the microbiome in a single patient on 3 different hierarchical taxonomic levels (O, F and G). Every patient is interpreted as a vector of frequencies of different microbiome taxas (sum over the components is 1, dimension is the amount of all presented taxonomies on a current level, for example, O). So, there are 3 databases of multidimensional data (one for each taxonomy level O, F, G). And the dimensions are higher for more detailed levels.

Database	O level	F level	G level
AGP	(9511; 168)	(9511; 258)	(9511; 535)
IBD	(86; 32)	(86; 64)	(86; 107)
PTB	(3457; 102)	(3457; 158)	(3457; 291)
T2D	(1044; 63)	(1044; 115)	(1044; 221)
Summary	(14098; 179)	(14098; 267)	(14098; 574)

First number is the number of items (‘patients’) in each database, the second number is the number of different taxonomy types (e.g. g__Acetobacter). Taxonomies f__[Weeksellaceae] and Weeksellaceae are considered as the same ones.

For more details on databases check Appendix 1.

2. Problem Statement.

Main Questions:

- Check are there any statistically significant clusters in the data? This could possibly be presented if microbiome for, e.g. old and young people are very different, and don’t have common cases (or have few). Other possible reasons – specific kind of microbiome for specific diseases, gender, behavior patterns could also lead to the clusters in the microbiome data. More difficult case is when clusters are presented with no evident explanation.
- Check if some lower-dimensional structure is presented, describe this structure if this is possible. Such a behavior is expected due to the common curse of conditionality. This is when the high-dimensional data is concentrated near low-dimensional nonlinear submanifold. It is another common possible case for the data dependencies.

In any of the described cases not all possible combinations of data are observed. In the first case it is expected that there are some gaps around the clusters – spaces with lower concentration of points, in addition for the clusters with high concentration. And there should be some reasons, for example when some kinds of bacteria tend to appear together and the others are not, causing concentrations and gaps. Submanifold case refers for examples of the cases when one some kinds of bacteria compete for food, and so, could not both be presented in a large number inside a single person, causing some kind of equational inter-dimensional dependencies like $x \cdot y=1$. When x is large, y is small and vice versa. This paragraph explanation is only to provide theoretical examples, explanations and common motivation, not real reasoning. All future speculations are left for biologists.

Finally, both submanifold and clustering are possibly presented, motivated by different reasons and interlaying together.

Intuitively, **the cluster** is the set of points with chosen number of subsets or “clouds” (clusters) such that the distance between every two points inside the cluster is “small” and the distance between the points from different clusters is “large”, or at least, “significantly larger”.

Such intuitive definition perfectly works in a full-dimensional case. But for real data we often observe **curse of dimensionality** situation: when the high-dimensional data is concentrated near low-dimensional submanifold. In this case, intuitive definition sometimes fails. Imagine some clusters on the line that becomes a spiral. Some clusters would not be “clouds” anymore. Some clusters would be on the different circles of spiral, but still be near in higher-dimensional space. That is why the clustering problem is strongly connected to the proper submanifold structure detection and should be investigated together.

One of the goals of Manifold Learning techniques is to find the low-dimensional one-chart coordinate system describing higher-dimensional data. Having such a representation could be used for finding clusters inside the representation and then the clusters in high-dimensional data would be “preimages” of these low-dimensional clusters.

The 4 different databases are analyzed to avoid incompleteness of a single source database and investigate if datasets gathered from different projects are separated, which is causing even more, source-specific clusters.

3. Workflow.

Low dimensional structure is found through 2-step process. First is the finding the lower-dimensional linear subspace containing most of the variation of data cloud. This means that the dataset is concentrated near this linear subspace, and the dataset projected on the chosen subspace would not significantly differ from the original dataset. PCA procedure is a standard algorithm that helps to find such a linear subspace. It also provides a specific orthonormal basis to represent the projected data. After PCA step all the data is rewritten in the coordinates associated with this orthonormal basis. The procedure requires only SVD decomposition which is just numerical linear algebra method. Algorithm is fast, stable, simple and computationally inexpensive.

The second step is one of the Manifold Learning algorithms that helps to find the proper chart to represent the projected by PCA data and to preserve some kind of the data local structure. These coordinates are non-linear, so all the Manifold Learning procedures are more advanced and computationally expensive than PCA. That is why so important to find low-dimensional PCA representation first. That helps to spend less computational resources and also to avoid direct computations in high-dimensional case in which most of the data is concentrated near lower-dimensional linear subspace and so there are many dimensions containing meaningless information (in which data is varying the least), causing Manifold Learning algorithms to spend more resources to find the desired low-dimensional structure and preserving meaningless local invariants coming from the complement (to chosen PCA) dimensions.

For the Manifold Learning techniques and computed nonlinear projection it also important to be able to return back to the PCA coordinate space. Such inverse map should also preserve the initial information containing in the data. Inverse PCA is a simple linear operation (just rewriting data projected to principal components back in the original space natural coordinates, ignoring all the principal components except the chosen ones). And its effectiveness could be estimated apriori, which would be described further.

The following workflow is proposed to find the answers on the stated question, taking into account the above motivation:

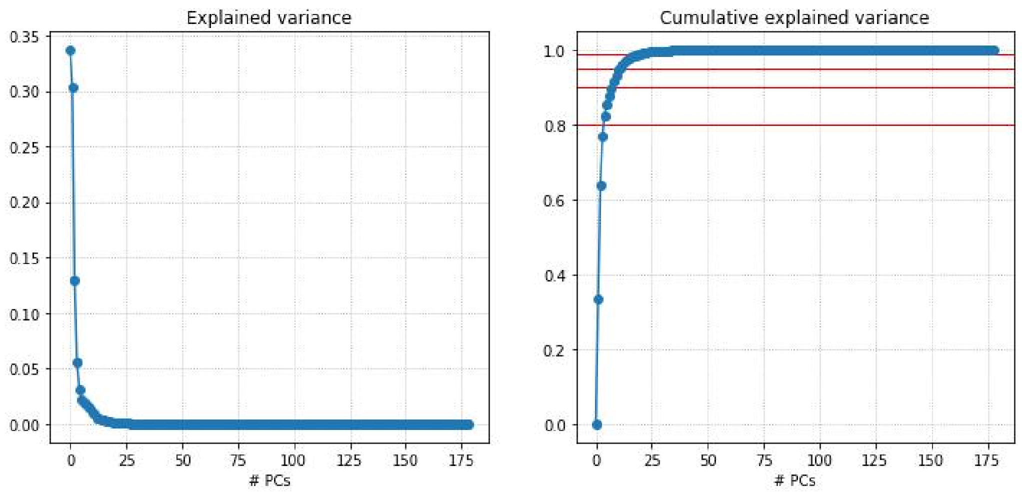
1. Apply standard **PCA** algorithm for the data. Find lower-dimensional (10s instead of 100s) affine subspace in the data containing most of the variance presented on one hand and as low dimensional as possible on another hand. (data → PCA projection)
   After this, apply intrinsic dimension estimation technique to forecast approximate dimension for low-dimensional representation.
2. Apply different kinds of manifold learning techniques to find low-dimensional (less than 10) representation for the data. **Isomap**, **LLE**, **modified LLE** were chosen. Find optimal parameters such as desired dimension and neighborhood size for the algorithms.
   (PCA projection → low-dim nonlinear representation)
3. Find the inverse transform from the low-dimensional representation via kernel regression algorithms. **KNN-regressor** was chosen (other variants such as **xgboost**, **random forest regressor**, **SVR** would be added potentially in future). Choose the optimal dimension parameter via comparison of the recovery errors (low-dim nonlinear representation → PCA projection). These errors would be described in details further.
4. Compare the clusters in the data, PCA projection and low-dimensional nonlinear representation. Every transformation could either merge different clusters or split them. We would use two indexes of quality – **Davies-Bouldin index** and **silhouette score** to measure the quality of clusters and then compare them in high-dimensional, PCA-projected and low-dimensional representations. Three methods – **Spectral Clustering**, **K-means** and **DBScan** algorithms are chosen to find the clusters. All the detailed information is provided further.
5. Compute visualizations of the low-dimensional representations. Construct 2d and 3d coordinate projections for Manifold Learning algorithms output and compute **t-SNE**, **UMAP** and **NCVis** algorithms for 2 and 3 dimensional visual representation for the same low-dimensional representation (original data or PCA-projected is too high-dimensional, so the computational cost is increasing significantly)

Currently complete workflow is computed for O taxonomy level, **Isomap** Manifold Learning method, **K-NN regressor** inverse reconstruction, 3 clustering methods are compared - **Spectral Clustering**, **K-means** and **DBScan** and all 3 visualizations (**t-SNE**, **UMAP**, **NCVis**). Relatively small reconstruction error is achieved and comparison of clustering methods with two different kinds of indexes seems to be enough for the empirical conclusions. The most computational cost is contained in the proper hyperparameters choices (dimension and neighborhood size for Manifold Learning algorithms, step or expected number of clusters chosen for clustering methods, many technical parameters for inverse mapping via K-NN regresion). Technical details on the hyperparameters grids, cross validation and other would be provided in appendixes. It is important to mention that adding a new Manifold learning algorithm would increase computational cost 2 times, 2 algorithms 3 times. Adding 2 or 3 other kernel regression methods would increase the cost 3 or 4 times. So even for O level taxonomy the full completeness of all additional algorithms would require 12 times more than the current version computation. And for 3 other levels F and G the whole process would be computed 36 times longer than it was computed now (current computations requires several days, some steps require about an hour, and there are many of them). So, probably it would be rational to keep 1 or 2 Manifold Learning methods and 1 or 2 kernel regression techniques to receive results for F and G levels faster (from 36 to 12 or even 3 coefficient). And only afterwards adding other algorithms to compare results, if necessary. Some processes, including proper grid choices could not be properly treated without human supervising (grid is like a window of parameters to optimize and choosing bad grid is like searching in the wrong window, computationally expensive, but inefficient).

4. PCA analyse

Database	O level	F level	G level
Summary before	(14098; 179)	(14098; 267)	(14098; 574)
Summary after	(14098; 20)	(14098; 42)	(14098; 68)

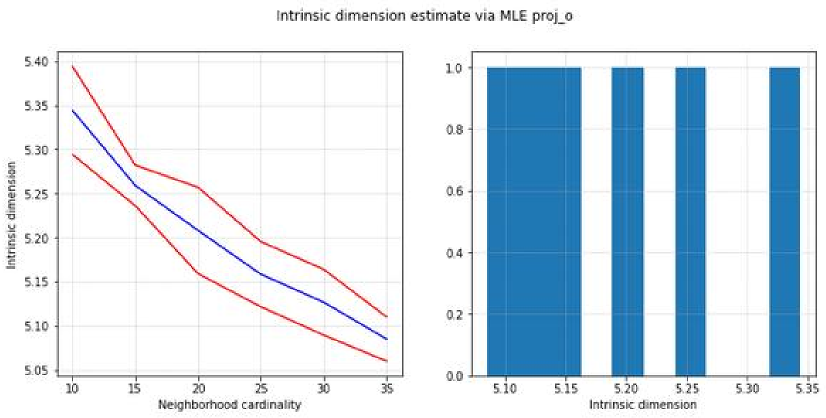
The table describes how the dimensions are changed after PCA projection with chosen 99% of cumulative explained variance. The graphics for the variances are presented below, all additional details, including the definition of CEV and explanations on the PCA method are in Appendix 2.



5. Nonlinear dimensional reduciton

Apriori dimension estimation 'Maximum Likelihood Estimation of Intrinsic Dimension' by Elizaveta Levina and Peter J. Bickel, 2004.

The graphic for the estimation is presented on the right, all the details on MLE method are contained in Appendix 3.



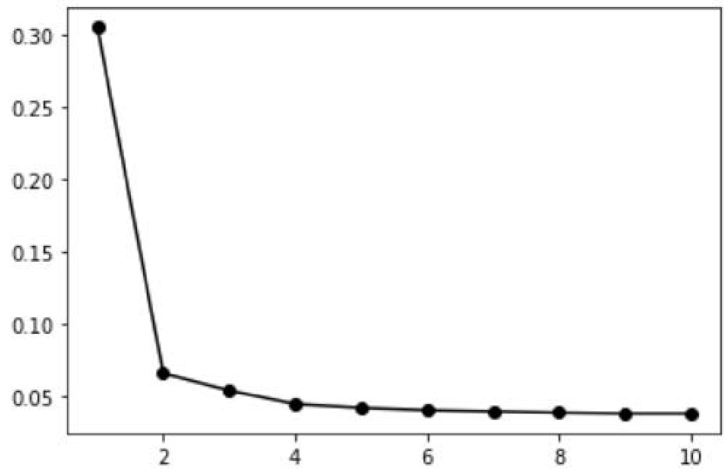
After the intrinsic dimension is estimated, Isomap algorithm is utilized to perform the nonlinear dimensionality reduction. More details on the algorithm are provided in Appendix 4.

6. Inverse mapping via kernel regression method

The goal of the low-dimensional nonlinear reduction is to achieve convenient representation for the data which is preserving most of the information in some sense. Alone nonlinear projections are not interesting in most of the cases by the reason that any data cloud could perfectly lie on some kind of a curve (1-dimensional submanifold) in many dimensional space. So it is important to construct also an inverse mapping from low-dimensional chart back to the original space. As it was already

discussed, inverse mapping from the PCA space back to the original is simple linear operation. So only the mapping from Isomap-projected data back to PCA space is desired.
One of the most simple methods to find the inverse mapping is regression. It is obvious that linear regression, the most simple one is not useful since the optimal linear transformation is PCA transform. So there is no chance to find the inverse mapping via linear regression (otherwise Isomap method would not be needed). More advanced regression techniques are nonlinear regression methods, for example kernel regression algorithms. In the current version K-Nearest Neighbors regression is utilized but several other variants are possible (Random Forest Regressor, Support Vector Regression, xgboost, etc.).

For any nonlinear regression method the most difficult part is the choice of proper hyperparameters. Classic cross validation grid search method is utilized in the current realization, HyperOpt (Bayesian hyperparameter optimization) is another possible solution. All the technical details and description of K-NN regressor are provided in Appendix 5.



The graph on the left is presenting the reconstruction error after the inverse mapping and inverse PCA transformation. Standard error $\frac{|x-\hat{x}|}{|x|}$ is computed. Here x is a point from the original dataset and $\hat{x}$ is estimated point (Isomap + K-NN regression + inverse PCA)
It is important to mention that all the components of all x vectors are non-negative, and the sum of components is 1. So x could not be a degenerate vector.

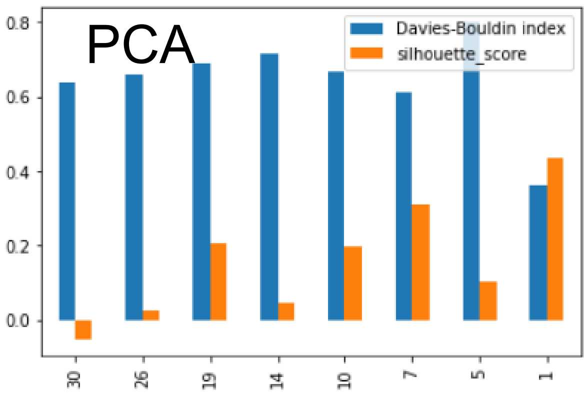
X coordinate of the graph is presenting the dimension parameter for Isomap algorithm. For each dimension a new cross validation grid search was performed. For dimension 5 (prognosed apriori via MLE) the error is 4.2%. For the next value, dimension 6, it is 4%. Afterwards the dimension falls down to 3.8%. For dimension 4, previous before 5 the error is 4.5%. So the dimension 5 is indeed optimal for the nonlinear dimension reduction, after this point the graph is practically stable, before the graph goes down significantly.

In the dimension 5 more than 95% of the lengths of the original database vectors are preserved. The chosen error shows how much of the length of the original vectors could be predicted using Isomap transformation. This intuitively tells how much information is preserved when the original data is replaced by the representation in the coordinate chart on low-dimensional manifold.

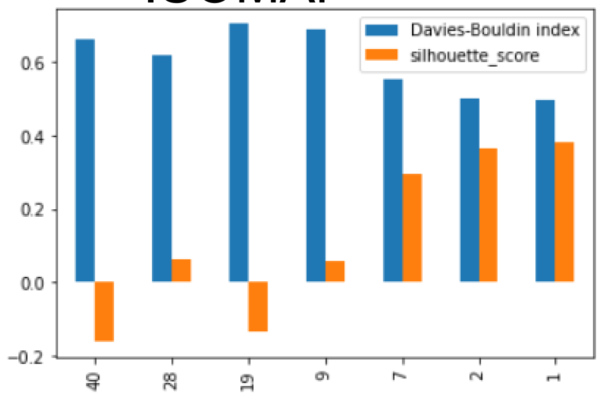
So the answer for the second stated question is positive. The desired low-dimensional representation exists, and could be efficiently found by the PCA projection + Isomap algorithm. Though such a representation is obviously not unique, so another Manifold Learning methods could be also utilized to receive another low-dimensional alternative representations (from the mathematical point of view, each diffeomorfism of the chart is another alternative representation, imagine chart written in Cartesian coordinates and cylindrical ones).

7. Clustering and indexes comparision.

Three different methods are utilized for clustering. First is DBScan, Density-based spatial clustering of applications with noise. This algorithm requires the only parameter, step, the maximum distance between two samples for one to be considered as in the neighborhood of the other (this is not a maximum bound on the distances of points within a cluster).

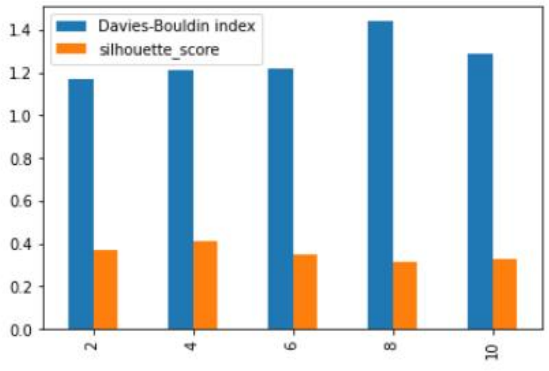


ISOMAP

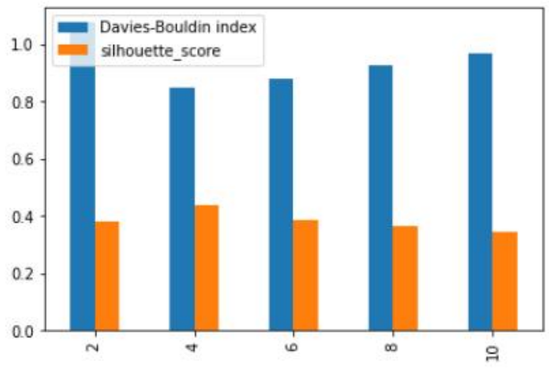


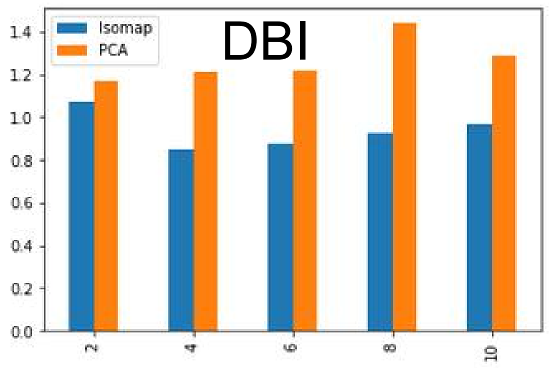
The pair of histograms is showing the DBScan results measured for a wide range of step parameters for PCA projected data and Isomap projected. Two kind of indexes are presented to measure the quality of clustering. Davies-Bouldin index should be minimized and silhouette score should be maximized. X axes shows the number of clusters for the chosen step. In both cases 1 cluster case has the best indexes. But for Isomap two clusters could also be possible (this could happen because of the nonlinear projection which sometimes cracks clusters – imagine the case with two point clouds merged into one on different pieces of spiral, which would be braked after the projection on the line). For PCA 2 cluster situation has not appeared on the chosen grid of steps. Possibly another, shorter grid interval could be chosen but this could lead to more cases of 1 cluster situations (we could not predict the number of clusters by the step). Negative silhouette cases are very bad scenarios. All the details about clustering methods and chosen indexes of quality are in the Appendix 6.

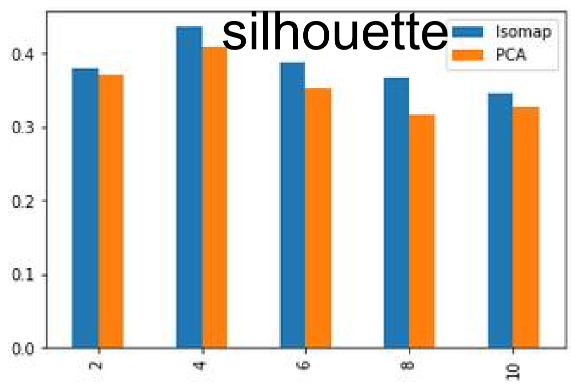
PCA



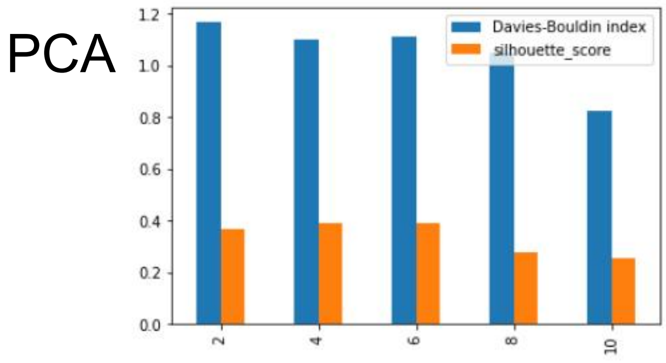
ISOMAP

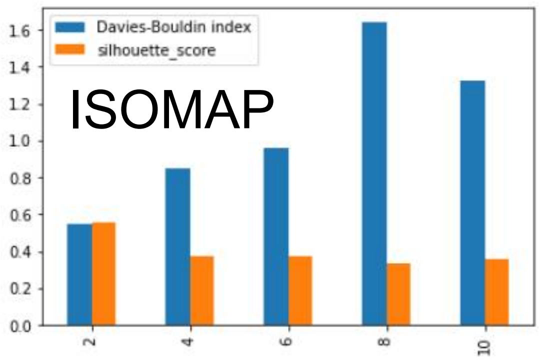


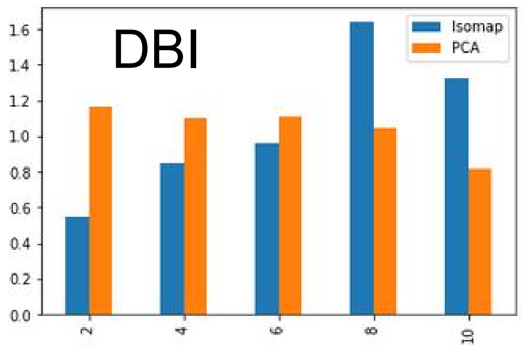


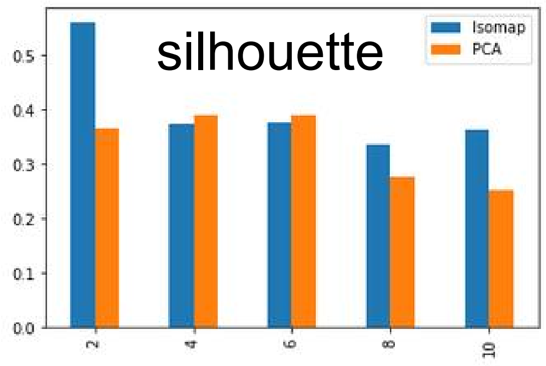


The last 4 histograms are for K-Means clustering method compared first by PCA/Isomap and two indexes and then compared by DBI/silhouette indexes and PCA or Isomap projections. The only parameter is the number of clusters which should be greater than one. For PCA 2 clusters seems to be the best case, but not significantly. For Isomap 4 clusters seems to be the best case. DBI is going down after Isomap which means that the quality of clusters is becoming higher after the projection. And the silhouette score is also rising, which also tells that the quality of clusters is better after Isomap. Second kind of comparision had no sense for DBScan method by the obvious reason that the number of clusters was different in Isomap and PCA cases. K-Means details are in Appendix 6.





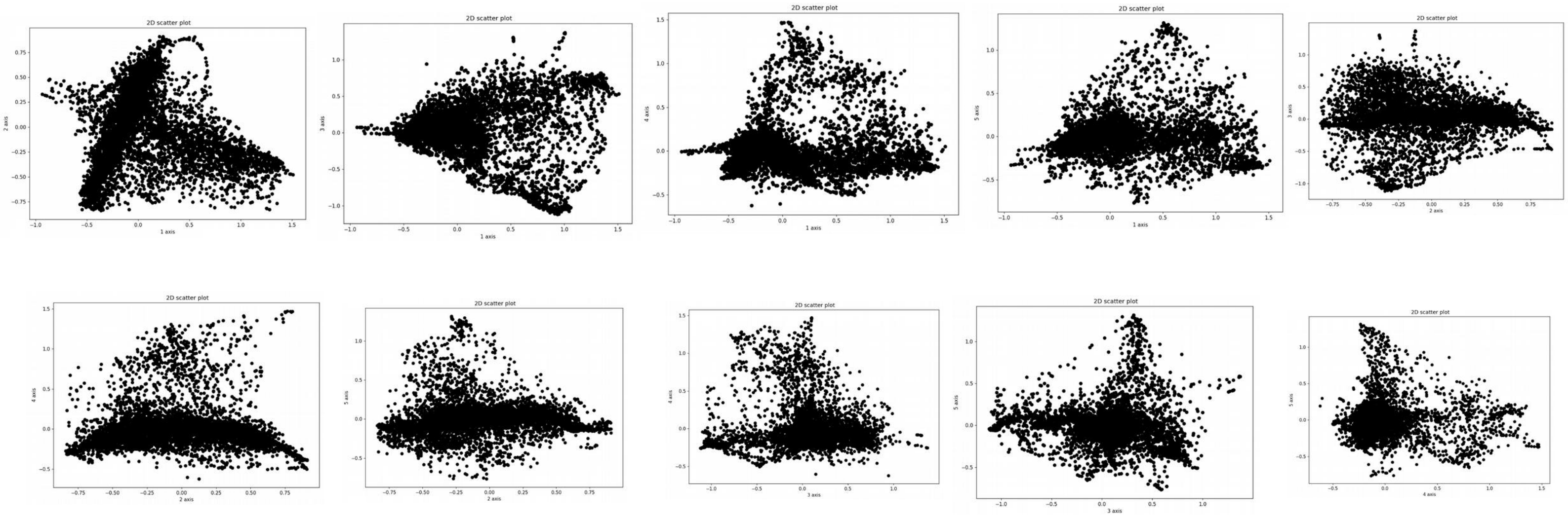




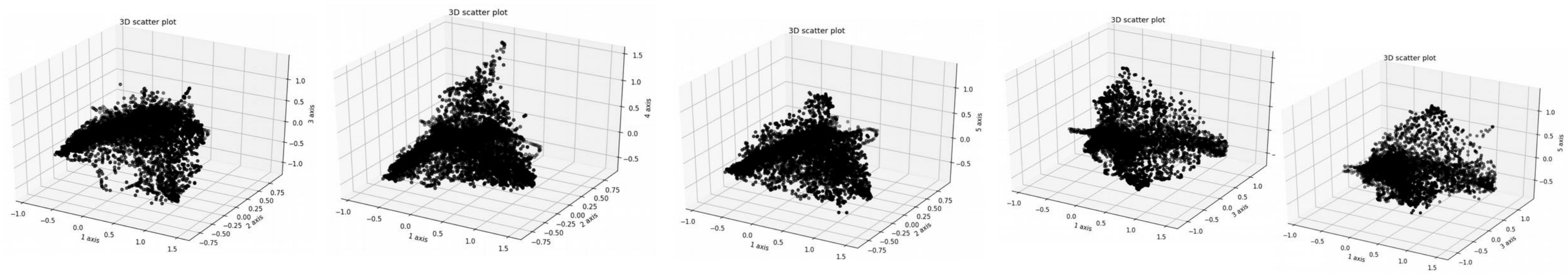
The last 4 histograms are for Spectral clustering method compared first by PCA/Isomap and two indexes and then compared by DBI/silhouette indexes and PCA or Isomap projections. The only parameter is the number of clusters which should be greater than one. For PCA 2-4-6 clusters seems to be the bests cases. For Isomap 2 clusters seems to be the best case. DBI is going down after Isomap in most of the cases which means that the quality of clusters is becoming higher (at least for hugher-quality clusters by the number parameter) after the projection. And the silhouette score is also rising in most of the cases, which also tells that the quality of clusters is better after Isomap (in general). The difference from the K-Means method is in not so stable behavior after the nonlinear dimensional reduction. Possible explanations and details on Spectral Clustering are in Appendix 6.

Empirically, it could be concluded that there are no statistically significant stable clusters in the data. But it seems that in the Isomap projection possibly 2 clusters exist. Also, in general the clustering quality is rising by both of the indexes after the Isomap projection, but Spectral Clustering case shows that it does not always happen. Most convincing explanation is that in the PC space there exists no clusters, but in chart the two pieces are presented because of the chart distortion (imagine spherical case with no clusters that is projected on the plane tangent to the North pole and instead of 1 cluster near the South pole there are up to 4 different clusters in the chart, similar example could also be provided for a cylinder or a torus).

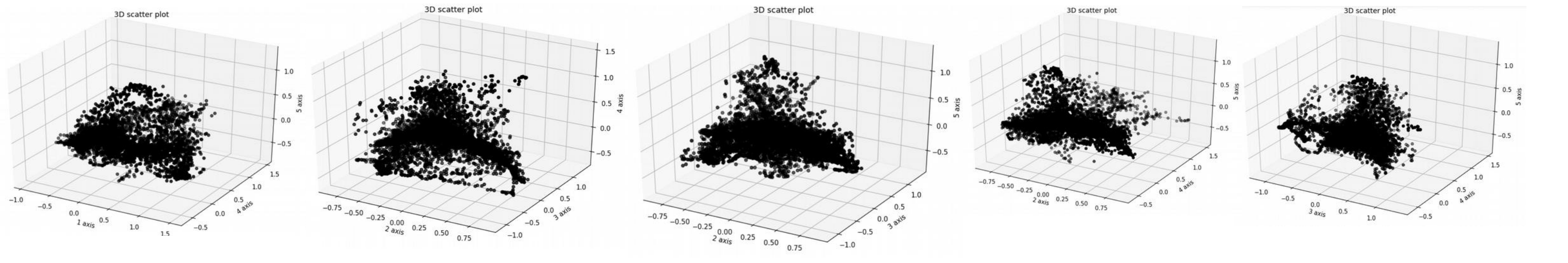
8. Visualization

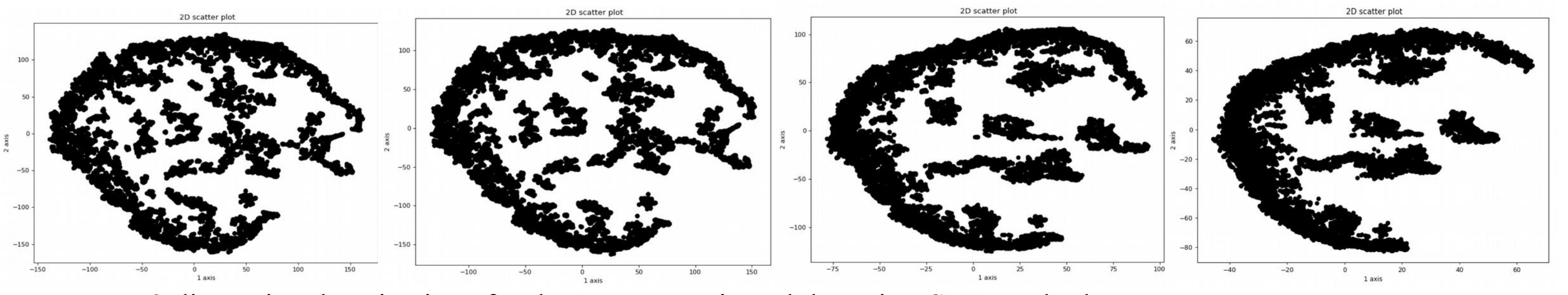


2 dimensional coordinate projections for the Isomap projected data

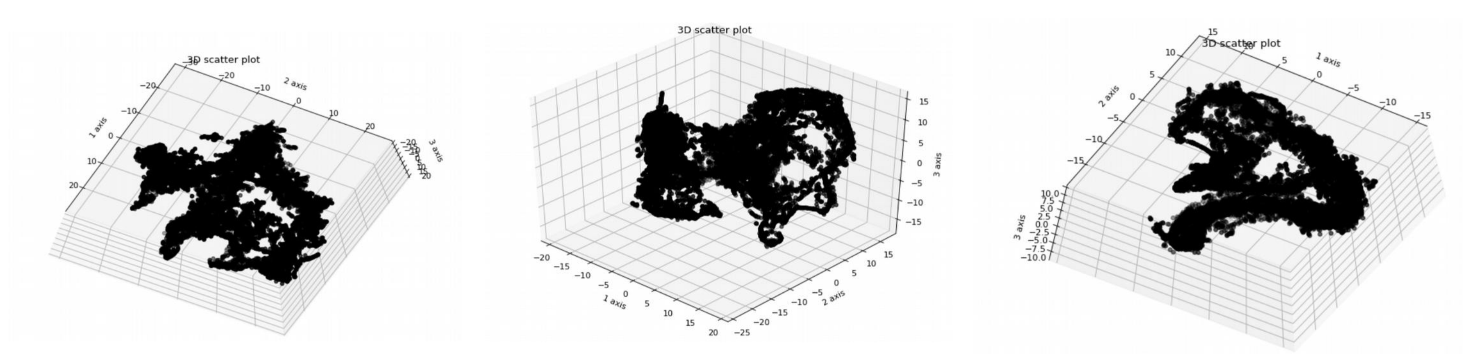


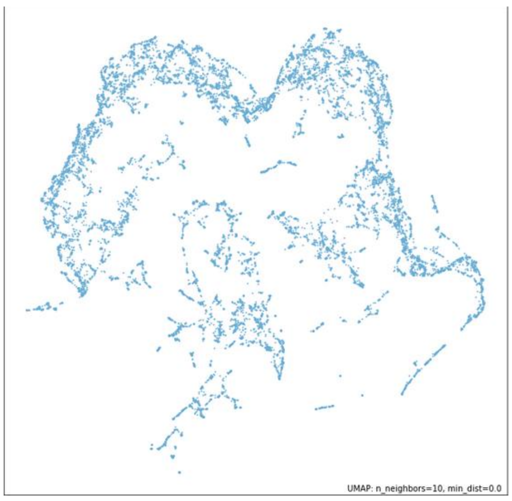
3 dimensional coordinate projections for the Isomap projected data

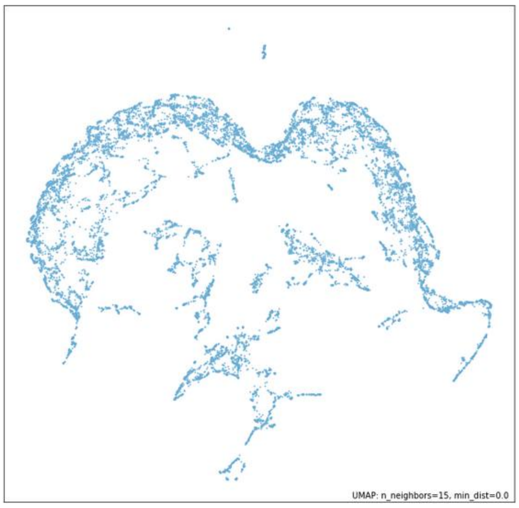


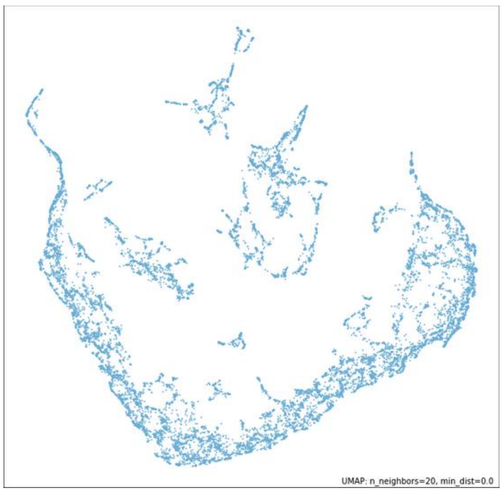


2 dimensional projections for the Isomap projected data via t-SNE method
(perplexity = 40, 50, 100 and 200).Below 3 dimensional projections for the same method are shown
(perplexity = 50, 120, 300)

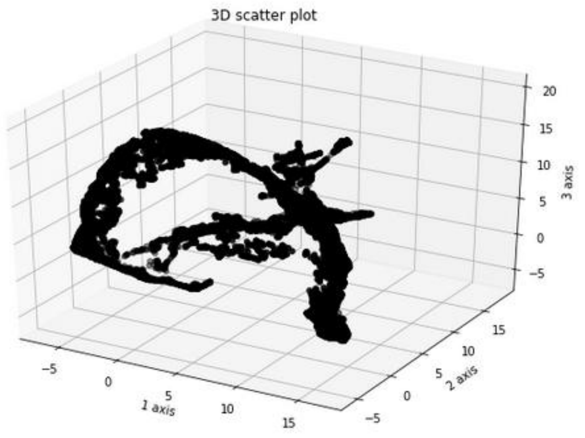






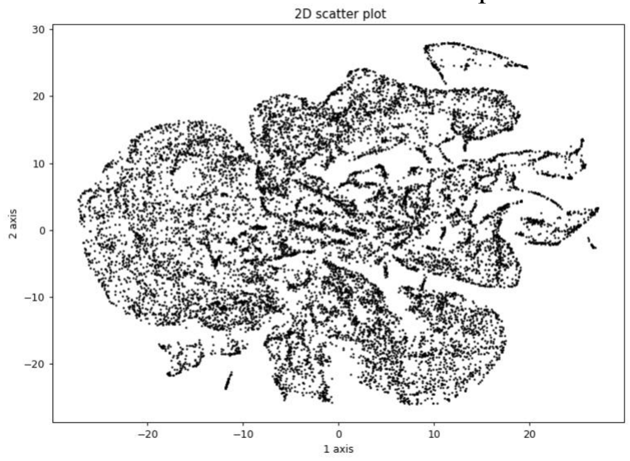


2d Umap projections of the Isomap data (n_neighbors = 10, 15, 20)



Umap 3d projection

2d NCVis method for Isomap



Simple coordinate projections do not demonstrate any visible clusters. 2D projections via t-SNE and Umap methods are providing some visual shapes that could be interpreted as clusters, but they do not appear in the 3d case or via NCVis method. So it seems that the visual clusters are the artifacts of the low dimensional algorithm. But anyway, existence and non-existence of clusters could not be precisely described by the visualization techniques. The most important pictures are 3 dimensional projections which could possibly confirm the existence of non-trivial cycles in the data which prevent the proper single chart representation, causing artificial clusters in the chart space. All the detailed information on t-SNE, Umap and NCVis methods would appear in Appendix

9. Conclusion

All the questions are answered with the proper reasoning during the analyses in the current workflow. So it is concluded that for the taxonomy level O the **proper submanifold dimension is 5**, for which **inverse mapping exists** and **provides fair reconstruction error** (95% in the sense of relative vector lengths). And **no significant stable clusters are found**, though it seems that 2 clusters possibly exist in the Isomap 5-dimensional chart projection, which is probably the artifact of the chart. In general case it seems that single chart representation could not describe any data cloud geometry (as for spheres or tori cases which are generally a family of submanifolds of any

dimension). But state of the art algorithms in Manifold Learning are all finding the single chart representations.
Also it could be interesting to investigate the topology of the dataset via TDA methods more deeply. There are several reasons for that.

- H_0 persistent homology provides the information about the connectivity in the data cloud at different scales, which is relative to the clustering problem
- H_1 persistent homology provides the information of 1-dimensional cycles in the data which could prevent a single chart representation (as the higher homology groups up to dimension 5 which was chosen for low-dimensional representation)
- Some kind of Morse filtration on the dataset could provide the reasonable partitioning which could give more information of the dataset geometry.

Another important question is whether it is possible to find different pieces of low-dimensional submanifold which could be possibly connected or even intersecting each other. And the reconstruction of such pieces should possibly replace the initial clustering investigation problem.

Appendix 1. More details on the Database

Current link containing original database, scripts and the pivot tables for the described workflow:
https://drive.google.com/drive/folders/1GeDPpmVXha88o9Ny3NVjOQfZoPzuWGT6?usp=sharing

HMP.zip contains the original database, containing .biom files. It includes 14102 items:
AGP database, 9511 items
IBD database, 86 items
PTB databse, 3462 items, including 5 empty folders (probably an error of data acquiring)
T2D database, 1043 items

AGP database is initially normalized to 1 (the entries could be interpreted as the frequencies), every other databse is not.

AGP database is also the only database that was corrected by the blooming procedure.
Citation:
"An important practical question is whether selfcollected microbiome samples can match those from better-controlled studies. Most AGP samples are stools collected on dry swabs and shipped without preservative to minimize costs and avoid exposure to toxic preservatives. Escherichia coli and a few other taxa grow in transit, so based on data from controlled-storage studies as previously described (15), we removed sub-operational taxonomic units (sOTUs) (16) (median of 7.9% of sequences removed per sample) shown to bloom."
for more details follow the articles from Bloom Analyses Details directory.

HMP_1.zip contains the same database, but every directory has also a table-format .txt file converted from biom format. It contains the whole taxonomic information and OTU IDs (for all databases), and also nucleotide chains (for AGP database). Every directory contains bash scripts for converting biom data to table format.

HMP_2.zip also contains 3 tables - pivot for the taxonomy levels O, F and G and scripts for the splitting the tabular data. As a result of conversion 5 empty folders in ptb database were found and also cut tails in the same database. In the other 3 databases if any taxonomy level was not presented, there was a s__ empty marker (or other letter indicating taxonomy level instead of s). And in the ptb database for some reason to be investigated, in some positions in some files there were just empty fields. The script is working in such a manner that it replaces empty fields with the s__ (or other letter indicating taxonomy level instead of s) marker. Also the script checks the sum of the whole file bacteria quantity before and after the splitting.

K, P, C, O, F, G, S levels of taxonomy were initially presented in every database.
And only O, F and G levels were chosen for the analyses. S level had too many nondescript fields and the first 3 levels were too generic. All the **hierarchical dependencies** between the levels **were ignored** in the current analyses. This is a crucial point and we cannot match any data on the different levels because of the reason that PCA coordinates taken as the first step are mixing the different taxonomic entries. For example, imagine that O_somebacteria is containing F_firstbacteria and F_secondbacteria subfamilies. During the first PCA step some weighted coordinates are chosen, and these coordinates could be like w1*F_firstbacteria + w2 * F_second_bacteria + w3*F_thirdbacteria + …, where wi are the weights and F_thirdbacteria is not contained in the O_somebacteria family. **So we could not match the hierarchical structure on different taxonomic levels with each other** even for the visualization purposes in the current workflow.

Appendix 2. PCA explanations and CEV details

PCA method is quite simple. Realization in scikit-learn library utilize SVD algorithm – Singular Value Decomposition to perform the correct projection. Let D be the data matrix (rows are vectors, each dimension is just a bacteria type, each vector is a patient). Size of D is N times d. N is the number of patients and d is the number of different bacteria in current taxonomic level. Let M be the data mean. M has the size 1 times N, this is a row-vector. Let h be N times 1 column-vector.

$$M_i = \frac{1}{n} \sum_{i=1}^{n} D_{ij}$$

D - h·M = $\underline{D}$ = U·S·V^T – SVD decomposition, U is orthogonal N times N matrix, V is orthogonal d times d matrix, S is rectangular N times d matrix with non-negative values on the diagonal sorted by the absolute value (each lower element has less absolute value). $\underline{D}$ is centered data matrix. V rows are called principal components. Diagonals elements of S are singular values corresponding to the principal components vectors.

Then PCA projection is computed as P = $U_L \cdot S_L$, where U_L is N times L matrix containing first L columns of U, S is L times L matrix containing first L rows and columns of S singular values matrix. The proper choice of the number of components L is reasoned by CEV estimation – cumulative explained variance. P has size N times L.

Let EV, explained variance be a vector collecting normalized eigenvalues (singular values).

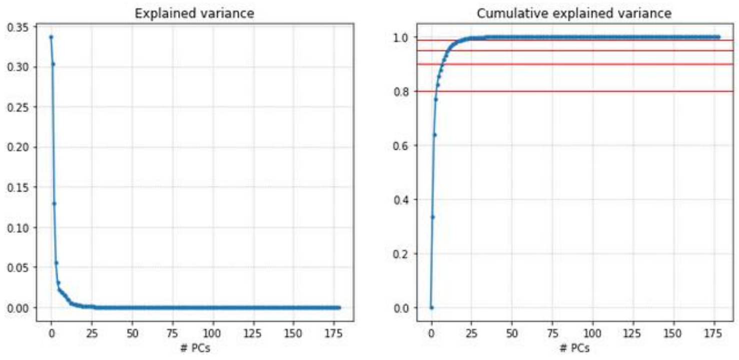
$$EV_i = \frac{S_i}{S} \quad \text{where} \quad S = \sum_{i=1}^{n} S_i$$

Then CEV, cumulative explained variance is just vector

$$CEV_i = \frac{\sum_{j=1}^{i} S_j}{S}$$

graphics for EV and CEV for the O taxonomic level are presented on the right.

For CEV horizontal lines showing 0.8, 0.9, 0.95 and 0.99 levels are shown. We could see that CEV is practically stable after 0.99 level already.



From the geometric point of view, PCA projection is very similar with the process of finding the principal axes of the quadratic form or diagonalizing inertia tensor. But instead of inertia deviation in the following direction is optimized. First components of PCA are containing the most varying directions, all orthogonal to each other and the length of eigenvectors are all ones to preserve the scale (if the basis vector is not normalized to one, then the projected data variance would be normalized to the length of the vector).

Formulas for inverse PCA transform are also quite simple.

$\hat{D} = P \cdot V_L$ where V_L is L times d matrix containing L first rows of V matrix form SVD.

So $\hat{D}$ has size N times d, exactly as the initial data matrix D.

Inverse transform error calculation shows that for 3 worst cases the reconstruction error is 1.19%, 1.09% and 1.01% respectively. For the other (14095) cases error is less than 1% as was expected.

The error is calculated as $\frac{|x-\hat{x}|}{|x|}$ where x is the original vector, and $\hat{x}$ is estimated vector.

Appendix 3. Maximum likelihood estimation of intrinsic dimension

Maximum likelihood estimation of intrinsic dimension was proposed by Elizaveta Levina and Peter J. Bickel in 2004. https://dl.acm.org/doi/10.5555/2976040.2976138

The idea of the nearest-neighbor intrinsic dimension estimation is based on fact of the that the number of sample points in $\mathbf{X} = \mathbb{R}^n$ falling into a ball around $\mathbf{x}$ is proportional to the radius of of the ball R, the dimension of the ball d and the density $f(x)$ for which the well-known estimator exists

$$f(x) = \frac{k / n}{R^d V(d)},$$

where $V(d) = \pi^{d/2}[\Gamma(d/2+1)]^{-1}$ is the volume of unit sphere in $\mathbb{R}^d$ and $\Gamma(d) = (d-1)!$ is the Gamma function.

Then the intrinsic dimension d of the sample can be regressed out or estimated by the maximum likelihood appoarch.

Consider a dataset $\mathbf{X} \in \mathbb{R}^n$ of intrinsic dimension d. Fix a sample point $\mathbf{x} \in \mathbf{X}$. Define $B(\mathbf{x}, t)$ as a ball centered at $\mathbf{x}$ with radius t, such that $0 \leq t \leq R$, where R is arbitrary small radius. Consider a random process, which counts the number of observations within a distance t to $\mathbf{x}$

$$N(\mathbf{x}, t) = \sum_n \mathbb{I}\{ \| \mathbf{x}_i - \mathbf{x} \| \leq t \} = \# \{ \mathbf{x}_i \in B(\mathbf{x}, t) \}.$$

Approximating this (binomial, w/ fixed n) random process by a Poisson process and suppressing the dependence on $\mathbf{x}$ for now, we can write the rate $\lambda(t)$ of the Poisson process $N(t)$ as

$$\lambda(t) = f(\mathbf{x}) \cdot V(d) \cdot d \cdot t^{d-1}.$$

Let $\theta = \log f(x)$ then log-likelihood of the observed process is:

$$L(d, \theta) = \int_0^R \log \lambda(t) dN(\mathbf{x}, t) - \int_0^R \lambda(t) dt$$

which should be maximized over parameters d and θ.

Maximum likelihood estimator for dimension d at point $\mathbf{x}$ is:

$$\hat{d}_R(\mathbf{x}) = \left[\frac{1}{N(\mathbf{x}, R)} \sum_{j=1}^{N(\mathbf{x}, R)} \log \frac{R}{r_j(\mathbf{x})} \right]^{-1}$$

where $r_j = \| \mathbf{x}_j - \mathbf{x} \|$, $j = 1, 2, \ldots$ is the distance to the j-th nearest neighbor.

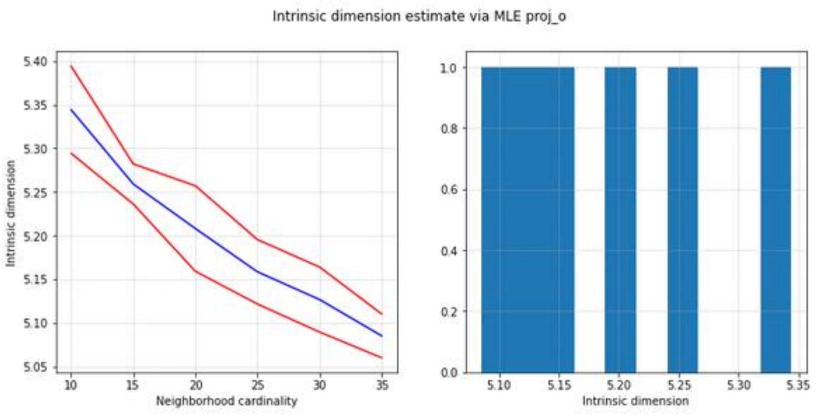
In practice, it may be more convenient to grow the number of neighbors k rather than the radius of the sphere R. Let we use k nearest neighbors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of a point $\mathbf{x}$ and $R = \mathbf{x}_k$ then the MLE estimator is:

$$\hat{d}_k(\mathbf{x}) = \left[\frac{1}{k-1} \sum_{j=1}^{k-1} \log \frac{r_k(\mathbf{x})}{r_j(\mathbf{x})} \right]^{-1}$$

Averaging over $\hat{d}_k(\mathbf{x})$ gives the intrinsic dimension of the dataset $\mathbf{X}$.

The goal is to estimate intrinsic dimensionality of data, the estimation of dimensionality is scale dependent (depending on how much you zoom into the data distribution you can find different dimesionality), so it is proposed to average it over different scales, the interval of the scales [k1, k2] are the only parameters of the algorithm.

Applying this method to PCA projected taxonomy O gives us dimension 5 as it is shown on the graphs below. Neighborhood cardinality (size of ball) is chosen between 10 and 35 points.

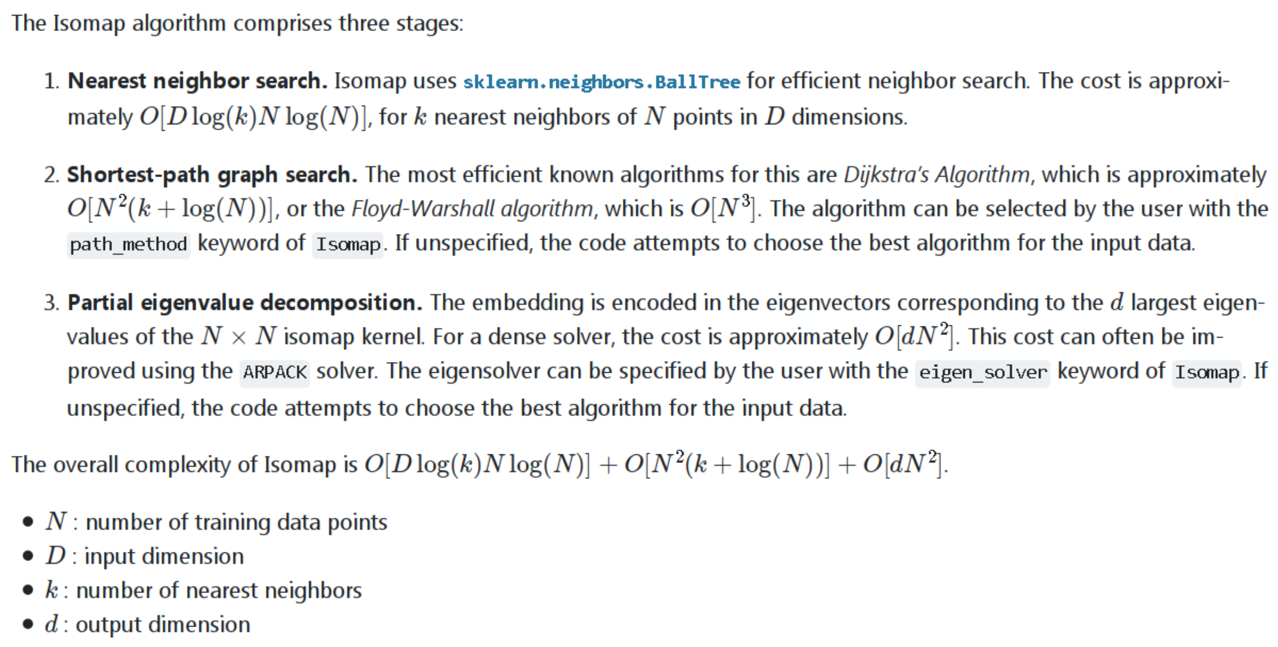


Appendix 4. Isomap Manifold Learning method

Original Isomap method was proposed by J.B. Tenenbaum, V. De Silva and J.C. Langford in the article “A global geometric framework for nonlinear dimensionality reduction” in Science vol. 290 (issue 5500), December 2000.

Original article could be found here: http://web.mit.edu/cocosci/Papers/sci_reprint.pdf

One of the earliest approaches to manifold learning is the Isomap algorithm, short for Isometric Mapping. Isomap can be viewed as an extension of Multi-dimensional Scaling (MDS) or Kernel PCA. Isomap seeks a lower-dimensional embedding which maintains geodesic distances between all points. Isomap from scikit-learn library is taken for current analysis.



The Isomap algorithm comprises three stages:

1. **Nearest neighbor search.** Isomap uses `sklearn.neighbors.BallTree` for efficient neighbor search. The cost is approximately $O[D\log(k)N\log(N)]$, for k nearest neighbors of N points in D dimensions.
2. **Shortest-path graph search.** The most efficient known algorithms for this are *Dijkstra's Algorithm*, which is approximately $O[N^2(k+\log(N))]$, or the *Floyd-Warshall algorithm*, which is $O[N^3]$. The algorithm can be selected by the user with the `path_method` keyword of `Isomap`. If unspecified, the code attempts to choose the best algorithm for the input data.
3. **Partial eigenvalue decomposition.** The embedding is encoded in the eigenvectors corresponding to the d largest eigenvalues of the $N \times N$ isomap kernel. For a dense solver, the cost is approximately $O[dN^2]$. This cost can often be improved using the `ARPACK` solver. The eigensolver can be specified by the user with the `eigen_solver` keyword of `Isomap`. If unspecified, the code attempts to choose the best algorithm for the input data.

The overall complexity of Isomap is $O[D\log(k)N\log(N)] + O[N^2(k+\log(N))] + O[dN^2]$.

- N : number of training data points
- D : input dimension
- k : number of nearest neighbors
- d : output dimension

Scikit version depends only on the parameter d – output dimension. For O taxonomy level this parameter is estimated as 5 via MLE method, and later it is checked that 5 is indeed the proper dimension by the reconstruction error after inverse K-NN regression method.

Appendix 5. K-NN regressor and cross validation grid search

In *k*-NN regression, the *k*-NN algorithm is used for estimating continuous variables. One such algorithm uses a weighted average of the *k* nearest neighbors, weighted by the inverse of their distance. This algorithm works as follows:

1. Compute the Euclidean or Mahalanobis distance from the query example to the labeled examples.
2. Order the labeled examples by increasing distance.
3. Find a heuristically optimal number *k* of nearest neighbors, based on RMSE (root-mean-square deviation). This is done using cross validation.
4. Calculate an inverse distance weighted average with the *k*-nearest multivariate neighbors.

Neighbors-based regression can be used in cases where the data labels are continuous rather than discrete variables. The label assigned to a query point is computed based on the mean of the labels of its nearest neighbors.

Scikit-learn library contains implementation of the neighbors regressor - KNeighborsRegressor which implements learning based on the k nearest neighbors of each query point, where k is an integer value specified by the user.

The basic nearest neighbors regression uses uniform weights: that is, each point in the local neighborhood contributes uniformly to the classification of a query point. Under some circumstances, it can be advantageous to weight points such that nearby points contribute more to the regression than faraway points. This can be accomplished through the `weights` keyword. The default value, `weights = 'uniform'`, assigns equal weights to all points. `weights = 'distance'` assigns weights proportional to the inverse of the distance from the query point. Alternatively, a user-defined function of the distance can be supplied, which will be used to compute the weights.

K-NN – Regressor algorithm has been chosen as one of kernel regression methods to find the inverse mapping from the coordinate low-dimensional space (on non-linear submanifold) to the principal coordinates space. Hyperparameters are chosen through grid search and cross validation (standard 20% to 80%).

List of hyperparameters for the grid search:
'n_neighbors' : [3, 4, 5, 6, 7, 8, 10, 15] - number of neighbors (K from K-NN)

'weights' : ('uniform', 'distance') -

- ‘uniform’ : uniform weights. All points in each neighborhood are weighted equally.
- ‘distance’ : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

'leaf_size' : [30, 40, 50, 60, 100, 10, 20] - Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

'p' : [2, 3, 1] - Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

Appendix 6. Clustering and quality indexes

If the ground truth labels are not known, evaluation for the cluster quality must be performed using the model itself. The **Silhouette Score** is an example of such an evaluation, where a higher Silhouette Score score relates to a model with better defined clusters. The Silhouette Score is defined for each sample and is composed of two scores:

- **a**: The mean distance between a sample and all other points in the same class.
- **b**: The mean distance between a sample and all other points in the *next nearest cluster*.

The Silhouette Coefficient *s* for a single sample is then given as:

The Silhouette Coefficient for a sample is `(b - a) / max(a, b)`. To clarify, `b` is the distance between a sample and the nearest cluster that the sample is not a part of.

Advantages:

- The score is bounded between -1 for incorrect clustering and +1 for highly dense clustering. Scores around zero indicate overlapping clusters.
- The score is higher when clusters are dense and well separated, which relates to a standard concept of a cluster.

Drawbacks:

- The Silhouette Coefficient is generally higher for convex clusters than other concepts of clusters, such as density based clusters like those obtained through DBSCAN.

References:

- Peter J. Rousseeuw (1987). "Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis". Computational and Applied Mathematics 20: 53–65. doi:10.1016/0377-0427(87)90125-7.

Example:

https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html
this link is an example of how silhouette index could be useful for the number of clusters estimation in K-Means method.

If the ground truth labels are not known, the **Davies-Bouldin index** can be also used to evaluate the model, where a lower Davies-Bouldin index relates to a model with better separation between the clusters.

This index signifies the average 'similarity' between clusters, where the similarity is a measure that compares the distance between clusters with the size of the clusters themselves.

Zero is the lowest possible score. Values closer to zero indicate a better partition.

Advantages:

- The computation of Davies-Bouldin is simpler than that of Silhouette scores.

- The index is computed only quantities and features inherent to the dataset.

Drawbacks:

- The Davies-Boulding index is generally higher for convex clusters than other concepts of clusters, such as density based clusters like those obtained from DBSCAN.
- The usage of centroid distance limits the distance metric to Euclidean space.

Mathematical formulation:

The index is defined as the average similarity between each cluster C_i for $i = 1, \ldots, k$ and its most similar one C_j. In the context of this index, similarity is defined as a measure R_{ij} that trades off:

- s_i, the average distance between each point of cluster i and the centroid of that cluster – also know as cluster diameter.
- d_{ij}, the distance between cluster centroids i and j.

A simple choice to construct R_ij so that it is nonnegative and symmetric is:

$$R_{ij} = \frac{s_i + s_j}{d_{ij}}$$

Then the Davies-Bouldin index is defined as:

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} R_{ij}$$

References:

- Davies, David L.; Bouldin, Donald W. (1979). "A Cluster Separation Measure" IEEE Transactions on Pattern Analysis and Machine Intelligence. PAMI-1 (2): 224-227. doi:10.1109/TPAMI.1979.4766909.
- Halkidi, Maria; Batistakis, Yannis; Vazirgiannis, Michalis (2001). "On Clustering Validation Techniques" Journal of Intelligent Information Systems, 17(2-3), 107-145. doi:10.1023/A:1012801612483.
- Wikipedia entry for Davies-Bouldin index.

The `DBSCAN` algorithm views clusters as areas of high density separated by areas of low density. Due to this rather generic view, clusters found by DBSCAN **can be any shape**, as **opposed to k-means** which **assumes that clusters are convex shaped**. The central component to the DBSCAN is the concept of *core samples*, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample (but are not themselves core samples). There are two parameters to the algorithm, `min_samples` and `eps`, which define formally what we mean when we say *dense*. Higher `min_samples` or lower `eps` indicate higher density necessary to form a cluster.

More formally, we define a core sample as being a sample in the dataset such that there exist `min_samples` other samples within a distance of `eps`, which are defined as *neighbors* of the core sample. This tells us that the core sample is in a dense area of the vector space. A cluster is a set of core samples that can be built by recursively taking a core sample, finding all of its neighbors that are core samples, finding all of *their* neighbors that are core samples, and so on. A cluster also has a set of non-core samples, which are samples that are neighbors of a core sample in the cluster but are not themselves core samples. Intuitively, these samples are on the fringes of a cluster.

Any core sample is part of a cluster, by definition. Any sample that is not a core sample, and is at least `eps` in distance from any core sample, is considered an outlier by the algorithm.

While the parameter `min_samples` primarily controls how tolerant the algorithm is towards noise (on noisy and large data sets it may be desirable to increase this parameter), the parameter `eps` is *crucial to choose appropriately* for the data set and distance function and usually cannot be left at the default value. It controls the local neighborhood of the points. When chosen too small, most data will not be clustered at all (and labeled as `-1` for "noise"). When chosen too large, it causes close clusters to be merged into one cluster, and eventually the entire data set to be returned as a single cluster. Some heuristics for choosing this parameter have been discussed in the literature, for example based on a knee in the nearest neighbor distances plot (as discussed in the references below).

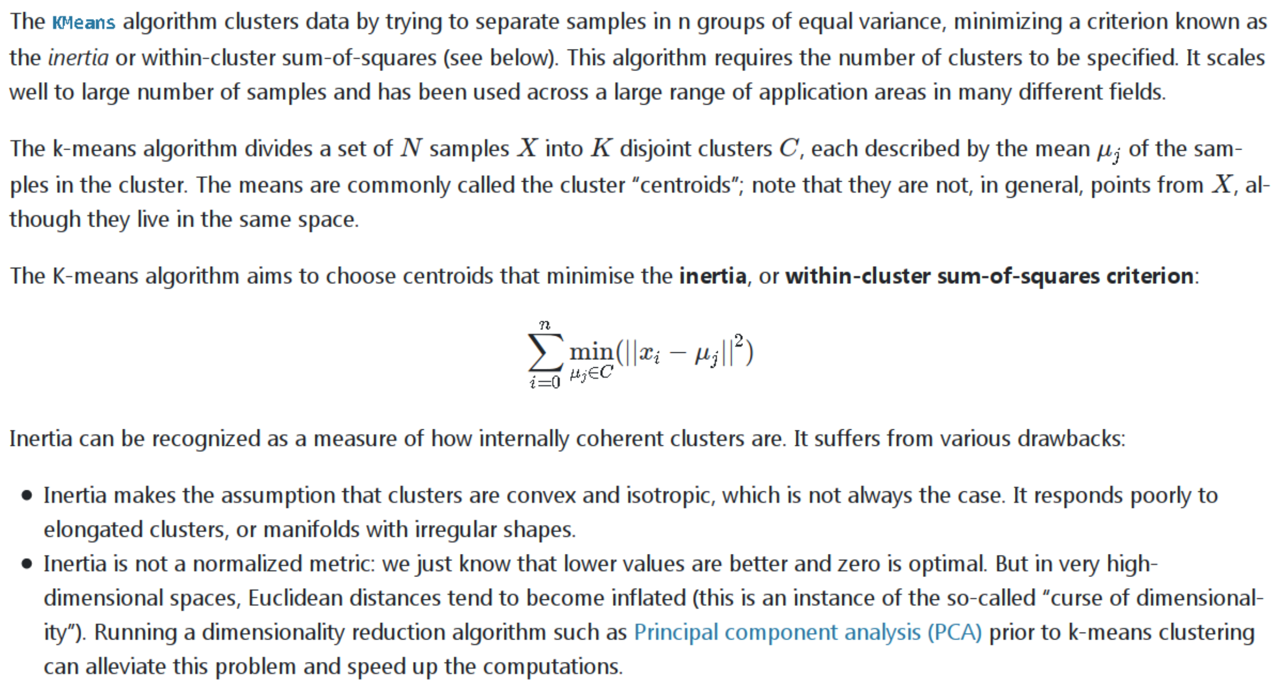
Implementation:

The DBSCAN algorithm is deterministic, always generating the same clusters when given the same data in the same order. However, the results can differ when data is provided in a different order. First, even though the core samples will always be assigned to the same clusters, the labels of those clusters will depend on the order in which those samples are encountered in the data. Second and more importantly, the clusters to which non-core samples are assigned can differ depending on the data order. This would happen when a non-core sample has a distance lower than `eps` to two core samples in different clusters. By the triangular inequality, those two core samples must be more distant than `eps` from each other, or they would be in the same cluster. The non-core sample is assigned to whichever cluster is generated first in a pass through the data, and so the results will depend on the data ordering.

The current implementation uses ball trees and kd-trees to determine the neighborhood of points, which avoids calculating the full distance matrix (as was done in scikit-learn versions before 0.14). The possibility to use custom metrics is retained; for details, see `NearestNeighbors`.

References:

- "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise" Ester, M., H. P. Kriegel, J. Sander, and X. Xu, In Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, pp. 226–231. 1996
- "DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. Schubert, E., Sander, J., Ester, M., Kriegel, H. P., & Xu, X. (2017). In ACM Transactions on Database Systems (TODS), 42(3), 19.



The `KMeans` algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the *inertia* or within-cluster sum-of-squares (see below). This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields.

The k-means algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean μ_j of the samples in the cluster. The means are commonly called the cluster "centroids"; note that they are not, in general, points from X, although they live in the same space.

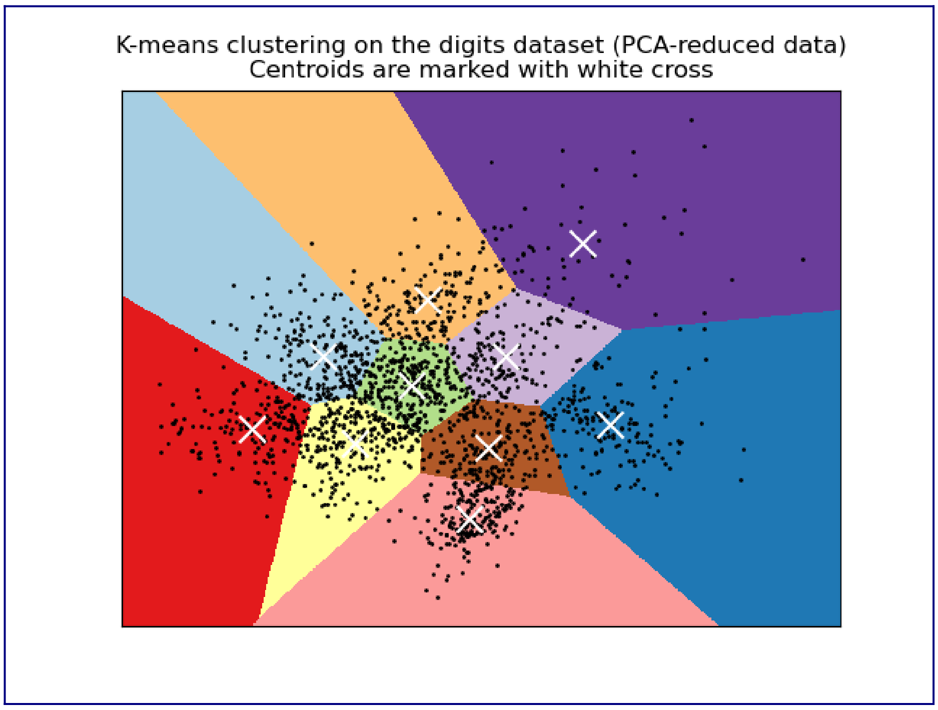
The K-means algorithm aims to choose centroids that minimise the **inertia**, or **within-cluster sum-of-squares criterion**:

$$\sum_{i=0}^{n} \min_{\mu_j \in C}(||x_i - \mu_j||^2)$$

Inertia can be recognized as a measure of how internally coherent clusters are. It suffers from various drawbacks:

- Inertia makes the assumption that clusters are convex and isotropic, which is not always the case. It responds poorly to elongated clusters, or manifolds with irregular shapes.
- Inertia is not a normalized metric: we just know that lower values are better and zero is optimal. But in very high-dimensional spaces, Euclidean distances tend to become inflated (this is an instance of the so-called "curse of dimensionality"). Running a dimensionality reduction algorithm such as Principal component analysis (PCA) prior to k-means clustering can alleviate this problem and speed up the computations.

K-means is often referred to as Lloyd's algorithm. In basic terms, the algorithm has three steps. The first step chooses the initial centroids, with the most basic method being to choose k samples from the dataset X. After initialization, K-means consists of looping between the two other steps. The first step assigns each sample to its nearest centroid. The second step creates new centroids by taking the mean value of all of the samples assigned to each previous centroid. The difference between the old and the new centroids are computed and the algorithm repeats these last two steps until this value is less than a threshold. In other words, it repeats until the centroids do not move significantly.



K-means is equivalent to the expectation-maximization algorithm with a small, all-equal, diagonal covariance matrix.

The algorithm can also be understood through the concept of Voronoi diagrams. First the Voronoi diagram of the points is calculated using the current centroids. Each segment in the Voronoi diagram becomes a separate cluster. Secondly, the centroids are updated to the mean of each segment. The algorithm then repeats this until a stopping criterion is fulfilled. Usually, the algorithm stops when the relative decrease in the objective function between iterations is less than the given tolerance value. This is not the case in this implementation: iteration stops when centroids move less than the tolerance.

Given enough time, K-means will always converge, however this may be to a local minimum. This is highly dependent on the initialization of the centroids. As a result, the computation is often done several times, with different initializations of the centroids. One method to help address this issue is the k-means++ initialization scheme, which has been implemented in scikit-learn (use the `init='k-means++'` parameter). This initializes the centroids to be (generally) distant from each other, leading to provably better results than random initialization, as shown in the reference.

The algorithm supports sample weights, which can be given by a parameter `sample_weight`. This allows to assign more weight to some samples when computing cluster centers and values of inertia. For example, assigning a weight of 2 to a sample is equivalent to adding a duplicate of that sample to the dataset.

References:

- “k-means++: The advantages of careful seeding” Arthur, David, and Sergei Vassilvitskii, *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics (2007)

`SpectralClustering` performs a low-dimension embedding of the affinity matrix between samples, followed by clustering, e.g., by KMeans, of the components of the eigenvectors in the low dimensional space. It is especially computationally efficient if the affinity matrix is sparse and the `amg` solver is used for the eigenvalue problem.

The present version of SpectralClustering requires the number of clusters to be specified in advance. It works well for a small number of clusters, but is not advised for many clusters.

For two clusters, SpectralClustering solves a convex relaxation of the normalised cuts problem on the similarity graph: cutting the graph in two so that the weight of the edges cut is small compared to the weights of the edges inside each cluster. This criteria is especially interesting when working on images, where graph vertices are pixels, and weights of the edges of the similarity graph are computed using a function of a gradient of the image.

References:

- “A Tutorial on Spectral Clustering” Ulrike von Luxburg, 2007
- “Normalized cuts and image segmentation” Jianbo Shi, Jitendra Malik, 2000
- “A Random Walks View of Spectral Segmentation” Marina Meila, Jianbo Shi, 2001
- “On Spectral Clustering: Analysis and an algorithm” Andrew Y. Ng, Michael I. Jordan, Yair Weiss, 2001
- “Preconditioned Spectral Clustering for Stochastic Block Partition Streaming Graph Challenge” David Zhuzhunashvili, Andrew Knyazev

Appendix 7. t-SNE, UMAP and NCVis methods

t-distributed Stochastic Neighbor Embedding (**t-SNE**) converts affinities of data points to probabilities. The affinities in the original space are represented by Gaussian joint probabilities and the affinities in the embedded space are represented by Student's t-distributions. This allows t-SNE to be particularly sensitive to local structure and has a few other advantages over existing techniques:

- Revealing the structure at many scales on a single map
- Revealing data that lie in multiple, different, manifolds or clusters
- Reducing the tendency to crowd points together at the center

While Isomap, LLE and variants are best suited to unfold a single continuous low dimensional manifold, t-SNE will focus on the local structure of the data and will tend to extract clustered local groups of samples as highlighted on the S-curve example. This ability to group samples based on the local structure might be beneficial to visually disentangle a dataset that comprises several manifolds at once as is the case in the digits dataset.

The Kullback-Leibler (KL) divergence of the joint probabilities in the original space and the embedded space will be minimized by gradient descent. Note that the KL divergence is not convex, i.e. multiple restarts with different initializations will end up in local minima of the KL divergence. Hence, it is sometimes useful to try different seeds and select the embedding with the lowest KL divergence.

The disadvantages to using t-SNE are roughly:

- t-SNE is computationally expensive, and can take several hours on million-sample datasets where PCA will finish in seconds or minutes
- The Barnes-Hut t-SNE method is limited to two or three dimensional embeddings.
- The algorithm is stochastic and multiple restarts with different seeds can yield different embeddings. However, it is perfectly legitimate to pick the embedding with the least error.
- Global structure is not explicitly preserved. This problem is mitigated by initializing points with PCA (using `init='pca'`).

The main purpose of t-SNE is visualization of high-dimensional data. Hence, it works best when the data will be embedded on two or three dimensions.

Optimizing the KL divergence can be a little bit tricky sometimes. There are five parameters that control the optimization of t-SNE and therefore possibly the quality of the resulting embedding:

- perplexity
- early exaggeration factor
- learning rate
- maximum number of iterations
- angle (not used in the exact method)

The perplexity is defined as $k = 2^S$ where S is the Shannon entropy of the conditional probability distribution. The perplexity of a k-sided die is k, so that k is effectively the number of nearest neighbors t-SNE considers when generating the conditional probabilities. Larger perplexities lead to more nearest neighbors and less sensitive to small structure. Conversely a lower perplexity considers a smaller number of neighbors, and thus ignores more global information in favour of the local neighborhood. As dataset sizes get larger more points will be required to get a reasonable sample of the local neighborhood, and hence larger perplexities may be required. Similarly noisier datasets will require larger perplexity values to encompass enough local neighbors to see beyond the background noise.

The maximum number of iterations is usually high enough and does not need any tuning. The optimization consists of two phases: the early exaggeration phase and the final optimization. During early exaggeration the joint probabilities in the original space will be artificially increased by multiplication with a given factor. Larger factors result in larger gaps between natural clusters in the data. If the factor is too high, the KL divergence could increase during this phase. Usually it does not have to be tuned. A critical parameter is the learning rate. If it is too low gradient descent will get stuck in a bad local minimum. If it is too high the KL divergence will increase during optimization. More tips can be found in Laurens van der Maaten's FAQ ("t-Distributed Stochastic Neighbor Embedding"). The last parameter, angle, is a tradeoff between performance and accuracy. Larger angles imply that we can approximate larger regions by a single point, leading to better speed but less accurate results.

"How to Use t-SNE Effectively" provides a good discussion of the effects of the various parameters, as well as interactive plots to explore the effects of different parameters.

Uniform Manifold Approximation and Projection (**UMAP**) is a dimension reduction technique that can be used for visualisation similarly to t-SNE, but also for general non-linear dimension reduction. The algorithm is founded on three assumptions about the data

1. The data is uniformly distributed on Riemannian manifold;
2. The Riemannian metric is locally constant (or can be approximated as such);
3. The manifold is locally connected.

From these assumptions it is possible to model the manifold with a fuzzy topological structure. The embedding is found by searching for a low dimensional projection of the data that has the closest possible equivalent fuzzy topological structure.

The details for the underlying mathematics can be found in paper on ArXiv:

McInnes, L, Healy, J, *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*, ArXiv e-prints 1802.03426, 2018

Noise Contrastive Approach for Scalable Visualization (**NCVis**) is an efficient solution for data visualization and dimensionality reduction. It uses HNSW to quickly construct the nearest neighbors graph and a parallel (batched) approach to build its embedding. Efficient random sampling is achieved via PCGRandom. Detailed application examples can be found here.

More details: https://arxiv.org/abs/2001.11411 (A. Artemenkov, M. Panov, 2020)

Visualization through various methods is another heuristic try to show that there are no real clusters, and that the data is well connected. 3-dimensional projections via UMAP and t-SNE methods are providing us this kind of evidence. Though these kind of methods sometimes show the clusters in the data where no real clusters presented and also shows that no clusters are observed in the cases when the clusters are appearing in the data. Many different parameters for t-SNE and Umap algorithm are presenting the dynamics of different data representation, though even such kind of heuristic analyses could never provide a complete evidence. (for more details refer to https://distill.pub/2016/misread-tsne/ and https://jlmelville.github.io/uwot/abparams.html)