

Learning the Manifolds!

Comparative Analysis of Different Dimensionality Reduction Methods

Araz Abedini, Ramin Tavakoli

Directed by Mahdi Lotfi


Dimensionality reduction simplifies high-dimensional data for easier visualization and analysis. In this project, we will implement and apply linear and non-linear dimensionality reduction methods to real-world datasets to reveal and visualize the hidden structures.


1. Implementation of PCA


Let's warm up by implementing the simplest and most straightforward method for dimensionality reduction: **Principal Component Analysis**!

1.1. Implement the algorithm

Open the `pca.py` file and begin by completing the `PCA` class.

 **Implementation note:** You need to vectorize your code using concepts from the labs. Avoid using loops for vector calculations; only use loops for the outermost iteration of the algorithms, alternating between computing the centroid and assigning points to clusters. Violating this incurs a **50% deduction from the related task score**. Also keep in mind that you may only use **basic NumPy features** to implement your algorithms. Using `numpy.linalg` or external libraries like `scikit-learn` is strictly prohibited unless explicitly allowed.

 **Coding practices:** Focus on writing clean, readable code by applying best practices like **Object-Oriented Programming (OOP)**. The cleaner and more well-structured your code is, the higher your chances of earning bonus points.

 **Option:** Can you prove that the projection directions chosen by PCA are those that maximize the variance of the projected data points? Provide your argument using LaTeX.

1.2. Test on linear data

Complete the `generate_plane` function in the `dataset.py` file to create a linear hyperplane in a d-dimensional space. Add Gaussian noise to the generated points and apply a labeling strategy for improved visualization. For example, if you are working with four classes, you could partition the hyperplane into a 4-block grid and assign a different label to each partition.

Visualize a 2-dimensional hyperplane within a 3-dimensional space, ensuring that the four classes are clearly represented.

Next, apply the PCA algorithm to the hyperplane to project the data into a 2-dimensional space. After the projection, visualize the results and analyze the outcomes.

1.3. Test on swiss roll

Load the Swiss Roll dataset using the `dataset.py` file. Visualize it to analyze its structure and key properties.

Next, open the `pca.py` file and complete the main block to apply PCA to the Swiss Roll dataset. Reduce the dimension with 2 components and visualize the transformed data to observe dimensionality reduction.

Finally, reconstruct the data from its lower-dimensional representation and compare it with the original dataset to evaluate the reconstruction error.

2. Implementation of Isomap

A more general method for dimensionality reduction is **Isometric Mapping**, which is a type of manifold learning algorithm. [Read the Isomap paper first].

We will implement Isomap step-by-step to illustrate its effectiveness in preserving the geometric properties of data while reducing its dimensionality.

2.1. Calculate pairwise distances

Isomap operates by assessing the similarities between data points.

Before proceeding, let's ensure that we have defined the necessary metrics for measuring similarity. In the `geo.py` file, we will implement two types of neighborhoods:

- **KNearestNeighbors**: An adjacency matrix is constructed based on the k nearest neighbors. Specifically, by sorting the points in ascending order of their distances, the first k points are identified as neighbors of the current point, while all other points are excluded.
- **EpsNeighborhood**: Defines a neighborhood around a given point based on a specified radius, denoted as ϵ . Points that fall within this radius are considered neighbors of the current point, while those outside this distance are excluded from the neighborhood.

2.2. Compute geodesic graph

After constructing the adjacency matrix, Isomap requires the computation of geodesic distances, which represent the meaningful distances between each pair of data points on the graph. These distances capture the shortest paths through the graph, reflecting the true geometric relationships among the points.

Open the `isomap.py` file and complete the `Isomap._compute_geodesic_distances` method to build the geodesic graph. Utilize the `adj_calculator` to obtain the adjacency matrix, and apply an algorithm such as Dijkstra's to compute the geodesic distances between each pair of points.

Note that the result will be a matrix where the geodesic distance between points P_i and P_j is located at the (i, j) component of the matrix.

2.3. Apply to the swiss roll

3. Implementation of Locally Linear Embedding

3.1. Find local transformations

3.2. Test on swiss roll

4. Learning the Manifold of Images

4.1. Download and prepare the dataset

4.2. Implement the new distance method

4.3. Test and visualize the algorithms

5. Optional: Implementation of MDS!

5.1. Calculate gradients

5.2. Implement gradient descent

5.3. Test your implementation

5.4. Adopt Isomap to use MDS

References & Resources

- https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- <https://www.science.org/doi/10.1126/science.290.5500.2323>
- <https://www.science.org/doi/10.1126/science.290.5500.2319>
- https://en.wikipedia.org/wiki/Gram_matrix
- https://en.wikipedia.org/wiki/Covariance_matrix
- <https://youtu.be/EgakZw6K1QQ?si=NGKmuoSXslyQtJCd>
- Pattern Recognition and Machine Learning by Christopher M. Bishop