Homework 2 report

Machine Learning for Signal Processing

Institute of Computer Science and Engineering

04/02/2024

Supervised by :
Do Huu Phu
dohuuphu25.ee11@nycu.edu.tw

Produced by :
PAULY Alexandre
alexandre.pauly@cy-tech.fr

Contents

1 Introduction

Context:

As part of the Machine Learning for Signal Processing module, an assignment focusing on non-linear dimensional reduction techniques has been initiated. This assignment is designed to reinforce the theoretical concepts covered within the module by implementing and exploring Isometric Mapping (ISOMAP) and t-distributed Stochastic Neighbor Embedding (t-SNE) algorithms in Python. The primary goal is to implementing these techniques without relying on pre-existing libraries, thereby deepening understanding and mastery of the underlying principles of non-linear dimensionality reduction.

Objective:

In this report, we focus on understanding and implementing ISOMAP and t-SNE from scratch. The main objective is to explain in detail each step of the implementation of these two techniques, including the underlying mathematical principles, computational procedures, and visualization techniques. Moreover, we will demonstrate the application of ISOMAP and t-SNE on the MNIST dataset, a collection of handwritten digits, by plotting the original data distribution and the resulting reduced-dimensional representations. Through this comparative analysis, we seek to elucidate the distinctive characteristics, advantages, and limitations of ISOMAP and t-SNE in non-linear dimensionality reduction tasks.

2 Implementation

2.1 Isometric Mapping (ISOMAP)

Isometric Mapping (ISOMAP) is a non-linear dimensionality reduction technique used to preserve the intrinsic geometric structure of high-dimensional data in a lower-dimensional space. It leverages the notion of geodesic distances, which measure the shortest path between two points along the manifold, to accurately represent the pairwise distances between data points.

To implement this method, we start by loading and pre-processing the data. Next, we construct a neighborhood graph based on the pairwise distances between data points with the Euclidean distances to approximate the structure of the dataset. Then, for each point, we identify its k-nearest neighbors based on these distances and construct an adjacency matrix representing the connectivity of the neighborhood graph.

After constructing the neighborhood graph, we compute the shortest path distances between all pairs of data points using techniques such as Dijkstra's algorithm. These distances capture the structure of the data, accounting for relationships between points.

Once the shortest path distances are computed, we apply a dimensional reductional to embed the data into a lower-dimensional space while preserving the pairwise distances as much as possible. This embedding reveals the underlying structure of the data in a more interpretable form, making it easier to visualize and analyze.

```python
def dimensional_reduction(N,D):
  C = np.eye(N) - np.ones((N, N)) / N
  B = -0.5 * np.dot(np.dot(C, D**2), C)
  A, V = np.linalg.eigh(B)
  idx = np.argsort(A)[::-1]
  X_hat = np.dot(V[:,idx[:2]],np.diag(A[idx[:2]]))

  X_hat *= -1

  return X_hat

X_hat = dimensional_reduction(N,D)
```

Code 1 : *Dimensional reduction*

Finally, we visualize the results by plotting the embedded data points in the lower-dimensional space. This allows us to assess the effectiveness of ISOMAP in preserving the intrinsic geometry of the data and uncovering meaningful relationships between data points.

```python
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

def categorical_scatter_2d(X2D, class_idxs, ms=3, ax=None, alpha=0.1,
                           legend=True, figsize=None, show=False,
                           savename=None):
    ## Plot a 2D matrix with corresponding class labels: each class diff colour
    if ax is None:
        fig, ax = plt.subplots(figsize=figsize)

    #ax.margins(0.05) # Optional, just adds 5% padding to the autoscaling
    classes = list(np.unique(class_idxs))
    markers = ['o', 's', 'o', 's', 'o', 's', 'o', 's', 'o', 's']
    colors = plt.cm.tab10(np.linspace(0, 1, len(classes)))

    # Plot the image to plot
    # Hint: Create the loop of "classes" and use ax.plot()
    for i, class_idx in enumerate(classes):
        class_X = X2D[class_idxs == class_idx]
        ax.plot(class_X[:, 0], class_X[:, 1], marker=markers[i], linestyle='', color=colors[i], label=str(class_idx), alpha=alpha, ms=ms, markeredgecolor='black', markeredgewidth=0.5)

    if legend:
        ax.legend()

    if savename is not None:
        plt.tight_layout()
        plt.savefig(savename)

    if show:
        plt.show()

    return ax
```

Code 2 : *Plot function*

Advantages:

- ISOMAP is highly effective at capturing non-linear relationships, making it a powerful tool for complex data.
- It focuses on preserving the global data structure, which is vital for datasets with manifold-like structures.
- This method often leads to more interpretable results in lower-dimensional space, making it useful for visualization.

Disadvantages:

- It can be computationally expensive, especially for large datasets, due to the construction of the neighborhood graph and geodesic distance calculations.
- The choice of the number of neighbors in the neighborhood graph can significantly impact the results.
- Like many non-linear techniques, Isomap may overfit noisy data.

2.2 t-distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE is also a non-linear dimensionality reduction technique commonly used for visualizing high-dimensional data in a lower-dimensional space while preserving structure. It operates by modeling the pairwise similarities between data points in both the high-dimensional and low-dimensional spaces, with a focus on preserving local neighborhood relationships.

To implement t-SNE, we start by computing the pairwise similarities between data points in the high-dimensional space. This involves calculating conditional probabilities that measure the likelihood of observing point j given point i, using a Gaussian kernel function based on the Euclidean distances between points.

```python
def neg_squared_euc_dists(X):
    """Compute matrix containing negative squared euclidean
    distance for all pairs of points in input matrix X

    # Arguments:
        X: matrix of size NxD
    # Returns:
        NxN matrix D, with entry D_ij = negative squared
        euclidean distance between rows X_i and X_j
    """

    sum_X = np.sum(np.square(X), 1)
    D = np.add(np.add(-2 * np.dot(X, X.T), sum_X).T, sum_X)
    return -D

def softmax(X, diag_zero=True):
    """Take softmax of each row of matrix X."""
    if diag_zero:
        np.fill_diagonal(X, 0)
    e_X = np.exp(X)
    return e_X / np.sum(e_X, axis=1)

def calc_prob_matrix(distances, sigmas=None):
    """Convert a distances matrix to a matrix of
       probabilities."""
    if sigmas is not None:
        two_sig_sq = 2. * np.square(sigmas.reshape((-1, 1)))
        return softmax(distances / two_sig_sq)
    else:
        return softmax(distances)
```

Code 3 : *Euclidean distances matrix*

Next, we compute the joint probabilities that measure the similarity between all pairs of points in the high-dimensional space. These joint probabilities are derived from the conditional probabilities and are symmetrically normalized to ensure that the sum of probabilities over all pairs equals one.

```python
def q_joint(Y):
    """Given low-dimensional representations Y, compute
    matrix of joint probabilities with entries q_ij."""
    # Get the distances from every point to every other
    distances = neg_squared_euc_dists(Y)
    # Take the elementwise exponent
    exp_distances = np.exp(-distances)
    # Fill diagonal with zeroes so q_ii = 0
    np.fill_diagonal(exp_distances, 0.)
    # Divide by the sum of the entire exponentiated matrix
    return exp_distances / np.sum(exp_distances), None

def q_tsne(Y):
    """t-SNE: Given low-dimensional representations Y,
       compute
    matrix of joint probabilities with entries q_ij."""

    distances = neg_squared_euc_dists(Y)
    inv_distances = 1 / ((1 + distances) + 1.e-6)
    np.fill_diagonal(inv_distances, 0.)

    return inv_distances / np.sum(inv_distances),
       inv_distances

def tsne_grad(P, Q, Y, inv_distances):
    """Estimate the gradient of t-SNE cost with respect to Y.
       """
    '''Write your code here'''
    pq_diff = P - Q
    pq_expanded = np.expand_dims(pq_diff, 2)
    y_diffs = np.expand_dims(Y, 1) - np.expand_dims(Y, 0)

    # Expand our inv_distances matrix so can multiply by
       y_diffs
    distances_expanded = np.expand_dims(inv_distances, 2)

    # Multiply this by inverse distances matrix
    y_diffs_wt = y_diffs * distances_expanded

    # Multiply then sum over j's
    grad = 4. * (pq_expanded * y_diffs_wt).sum(1)

    return grad
```

Code 4 : *Joint probabilities and gradient of t-SNE*

Once the pairwise similarities are computed, we apply gradient descent to minimize the divergence between the joint probabilities in the high-dimensional space and the joint probabilities in the low-dimensional space. This optimization process adjusts the positions of data points in the lower-dimensional space to better reflect their similarities in the high-dimensional space.

Finally, we visualize the results by plotting the data points in the lower-dimensional space obtained through t-SNE. This allows us to explore the structure of the data and reveal any underlying patterns or clusters that may exist.

```
def estimate_sne(X, y, P, rng, num_iters, q_fn, grad_fn,
   learning_rate,
                 momentum, plot):
    """Estimates a SNE model.

    # Arguments
        X: Input data matrix.
        y: Class labels for that matrix.
        P: Matrix of joint probabilities.
        rng: np.random.RandomState().
        num_iters: Iterations to train for.
        q_fn: Function that takes Y and gives Q prob matrix.
        plot: How many times to plot during training.
    # Returns:
        Y: Matrix, low-dimensional representation of X.
    """

    # Initialise our 2D representation
    Y = rng.normal(0., 0.0001, [X.shape[0], 2])

    # Initialise past values (used for momentum)
    if momentum:
        Y_m2 = Y.copy()
        Y_m1 = Y.copy()

    # Start gradient descent loop
    for i in range(num_iters):

        # Get Q and distances (distances only used for t-SNE)
        Q, distances = q_fn(Y)
        # Estimate gradients with respect to Y
        grads = grad_fn(P, Q, Y, distances)

        # Update Y
        Y -= learning_rate * grads
        if momentum:  # Add momentum
            Y += momentum * (Y_m1 - Y_m2)
            # Update previous Y's for momentum
            Y_m2 = Y_m1.copy()
            Y_m1 = Y.copy()

        # Plot sometimes
        if plot and i % (num_iters / plot) == 0:
            categorical_scatter_2d(Y, y, alpha=1.0, ms=6,
                                   show=True, figsize=(9, 6))

    return Y

# Set global parameters (You can change)
PERPLEXITY = 20
SEED = 1                 # Random seed
MOMENTUM = 0.9
LEARNING_RATE = 100.
```

```
NUM_ITERS = 500        # Num iterations to train for
TSNE = True            # If False, Symmetric SNE
NUM_PLOTS = 5          # Num. times to plot in training


# numpy RandomState for reproducibility
rng = np.random.RandomState(SEED)

# Obtain matrix of joint probabilities p_ij
P = p_joint(X, PERPLEXITY)

# Fit SNE or t-SNE
Y = estimate_sne(X, y, P, rng,
          num_iters=NUM_ITERS,
          q_fn=q_tsne if TSNE else q_joint,
          grad_fn=tsne_grad if TSNE else symmetric_sne_grad,
          learning_rate=LEARNING_RATE,
          momentum=MOMENTUM,
          plot=NUM_PLOTS)
```

Code 5 : *Plot function*

Advantages:

- t-SNE can help visualize high-dimensional data that has non-linear relationships as well as outliers
- This method is often used for clustering and can help identify groups of similar data points within the data.

Disadvantages:

- It involves complex calculations as it calculates the pairwise conditional probability for each point. Due to this, it takes more time as the number of data points increases.
- Due to the randomness in the algorithm, even though code and data points are the same in each iteration, we may get different results across runs (here the random seed id fixed).

3 Results

The data used in this study is derived from the MNIST dataset, which is extensively employed for handwritten character recognition and image classification tasks. This dataset comprises images of handwritten digits from zero to nine, each represented as a matrix of grayscale pixels. These images serve as digital representations of handwritten digits captured under various conditions, including diverse writing styles, different sizes, and varying levels of noise.

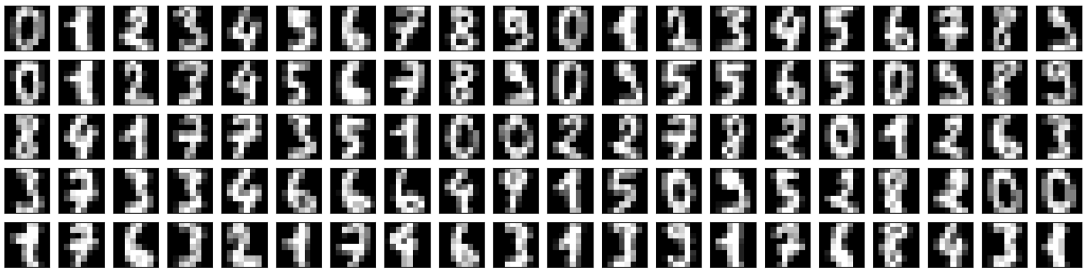


Figure 1: Examples of data from the MNIST dataset

Now that the methods and data have been presented, we can analyse the results.

3.1 Isometric Mapping (ISOMAP)

Classification quality :

Overall, the classification is well represented by the different classes. Some stand out and are quite distinct, such as class 4 (visible in purple circles) regardless of the number of neighbors. Similarly, for class 0 as long as k $<$ 10, beyond which it merges with class 6.

For the others, they are much closer together; however, class 7 manages to dissociate from the main cluster, even though it is close to class 1 due to their geometric resemblance. But their respective point clouds seem more tightly grouped and only let a few data points escape. Given that this method is more effective at capturing global structures or large-scale relationships between data, it's not surprising that 1 and 7 are close.

On the other hand, classes 3, 8, and 9 are much more scattered and easily blend together, which is quite evident as they are geometrically close, regardless of the value of k. However, as k increases, certain classes like 5 and 2 start to merge.

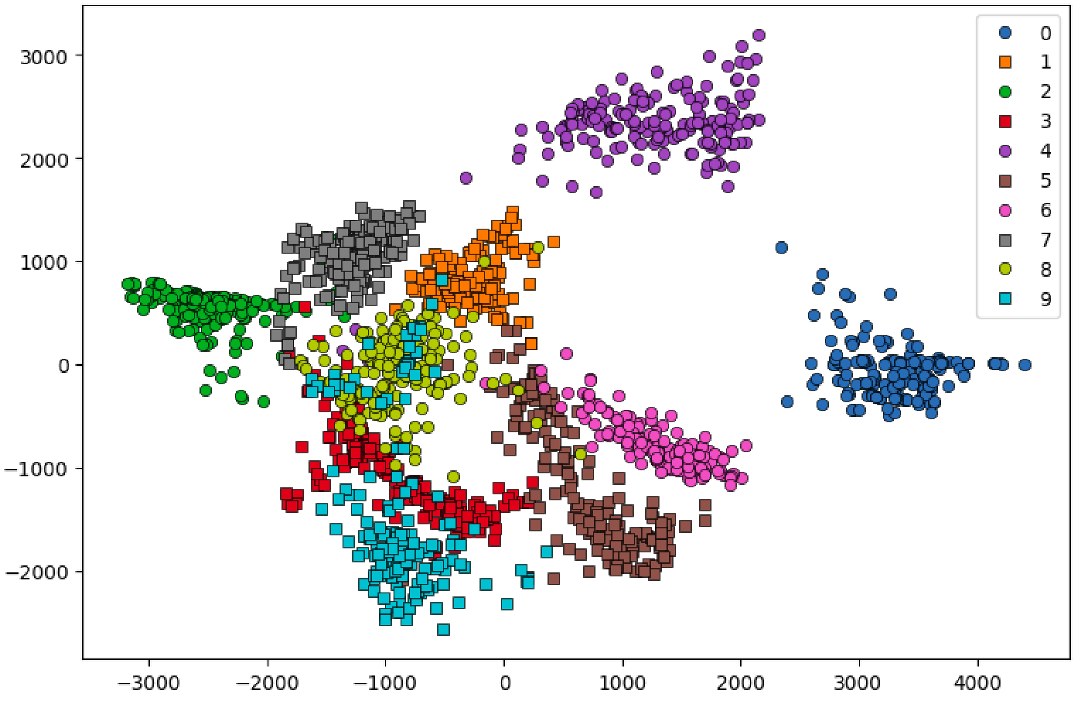


Figure 2: Results of ISOMAP with 5 neighbors

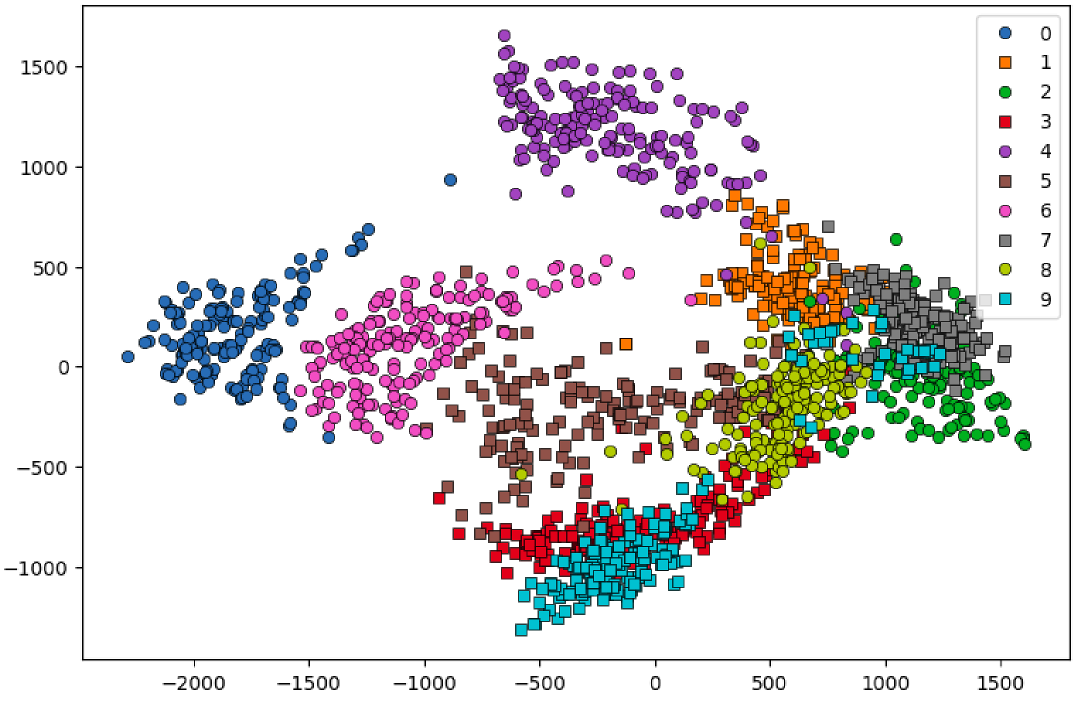


Figure 3: Results of ISOMAP with 7 neighbors

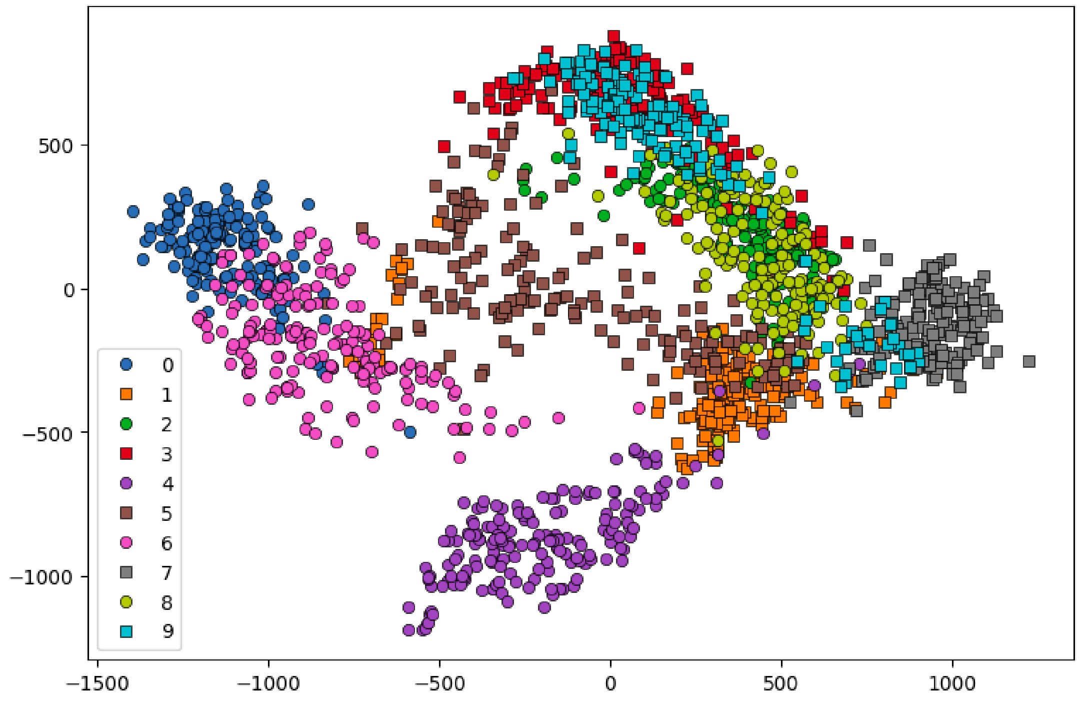


Figure 4: Results of ISOMAP with 10 neighbors

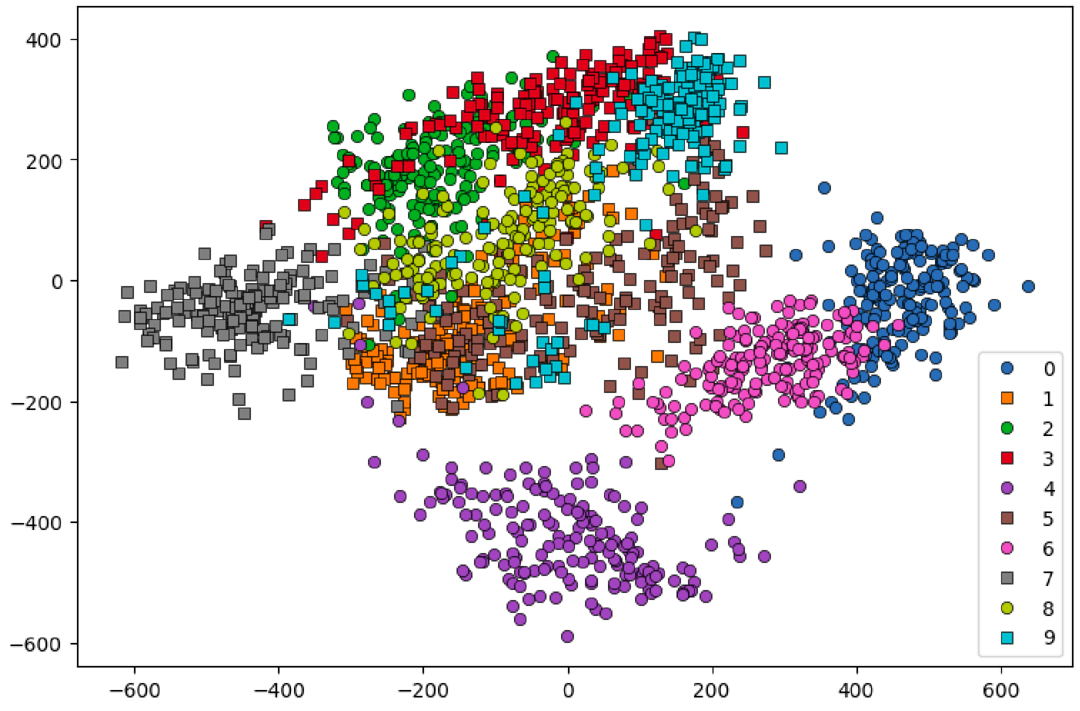


Figure 5: Results of ISOMAP with 30 neighbors

As mentioned earlier, we can clearly observe the difference between each classification. The number of neighbors is very important. Even though the classification is not efficient.

If we choose the example where k=3, we can see above this value is not sufficient, as shown by the distance matrix lots of 0. So, for this reason, is important to choose the good ones value.

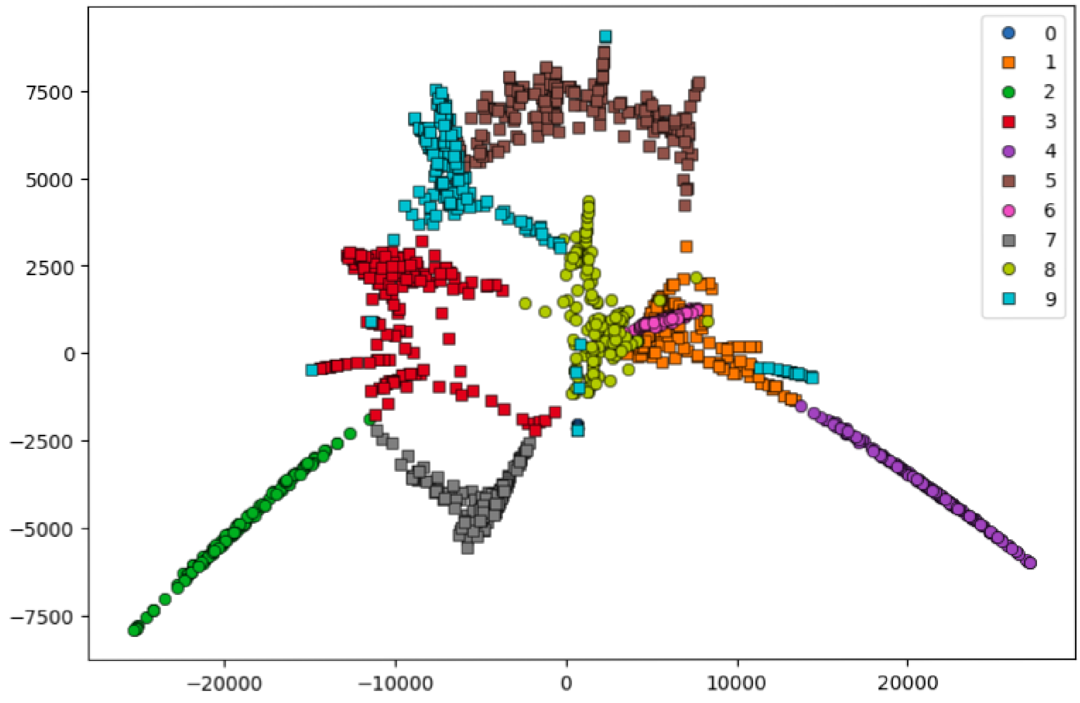
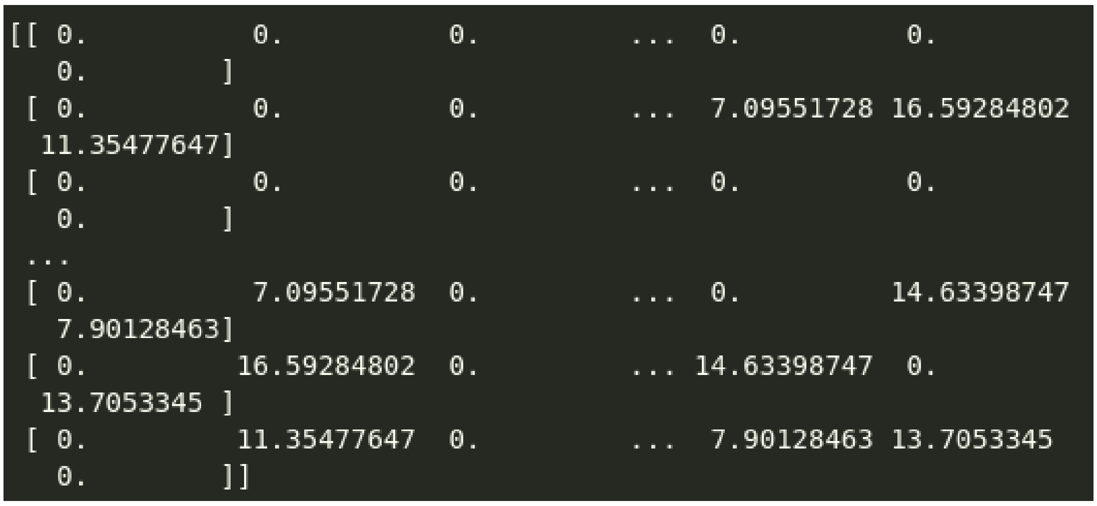


Figure 6: Results of ISOMAP with 3 neighbors



```
[[ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  7.09551728 16.59284802
  11.35477647]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 ...
 [ 0.          7.09551728  0.         ...  0.         14.63398747
   7.90128463]
 [ 0.         16.59284802  0.         ... 14.63398747  0.
  13.7053345 ]
 [ 0.         11.35477647  0.         ...  7.90128463 13.7053345
   0.        ]]
```

Figure 7: Distance matrix with 3 neighbors

Training time :

Regarding training time, ISOMAP requires a high investment. Obviously, the required time depends of the number of data and the number of neighbors, but we can have a result in approximately 30 seconds. It's very fast if we need to compare with the next method. But the mnist dataset contains 60,000 images (or 70,000 for the full dataset), which would take much longer. And in our case, the size of each image is 8 x 8, not 28 x 28 like the full dataset. In fact, this is one of its drawbacks, due to the search for neighbours and the shortest paths.

3.2 t-distributed Stochastic Neighbor Embedding (t-SNE)

Classification quality :

First iteration of t-SNE (**Figure 8**): The results remain similar despite variations in the parameters, except for a slight variation due to randomness, although the random seed is fixed.

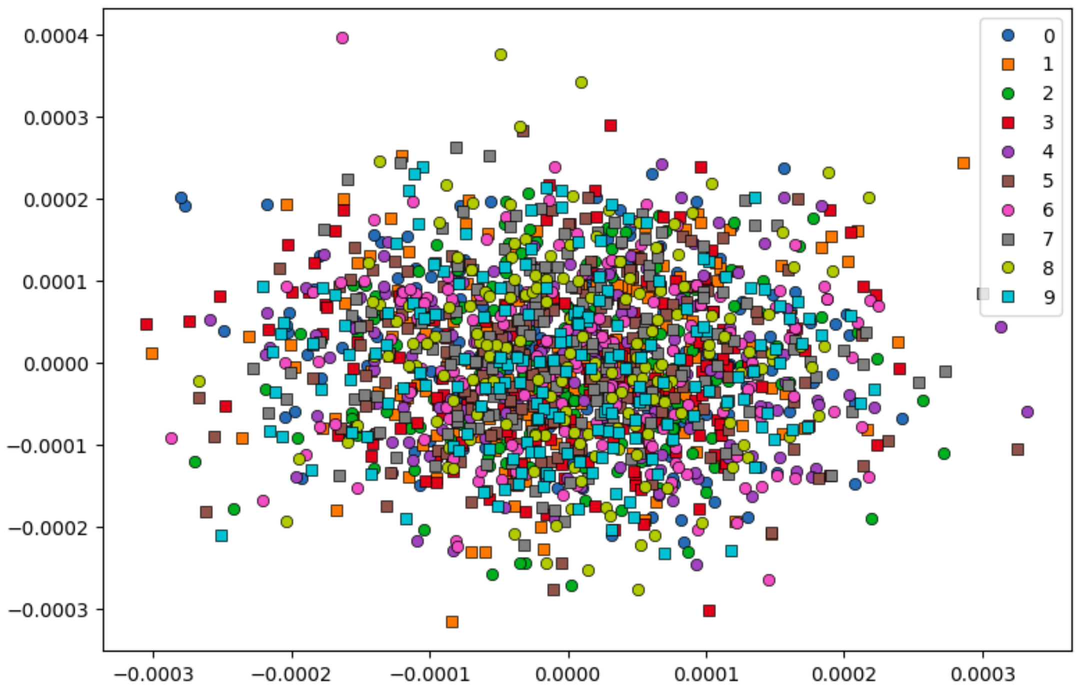
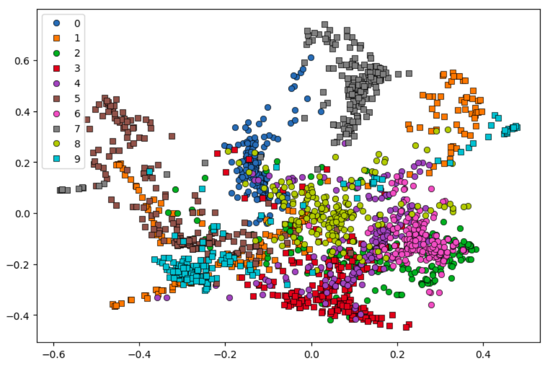
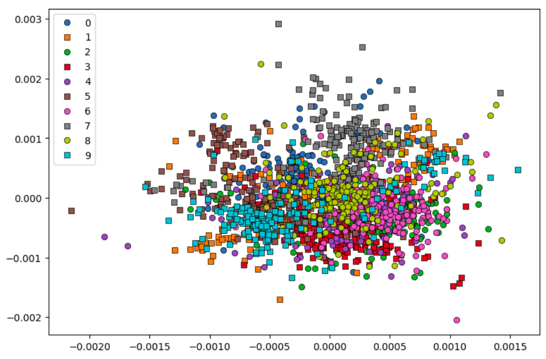


Figure 8: t-SNE 1st iteration

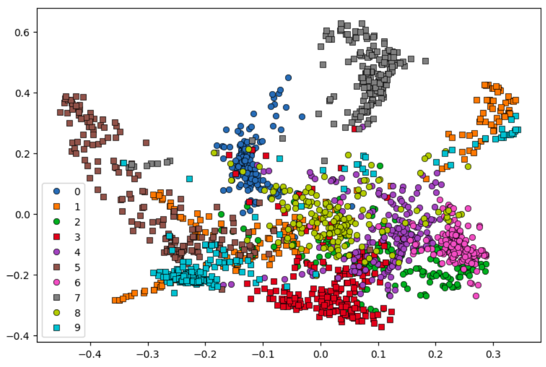
Overall, even with hyperparameter variations, the method requires 100 iterations to begin discerning clusters (**Figure 9**), except when the learning rate increases (*example : learning_rate = 1000*). Additionally, when momentum decreases, clusters take more time to form.



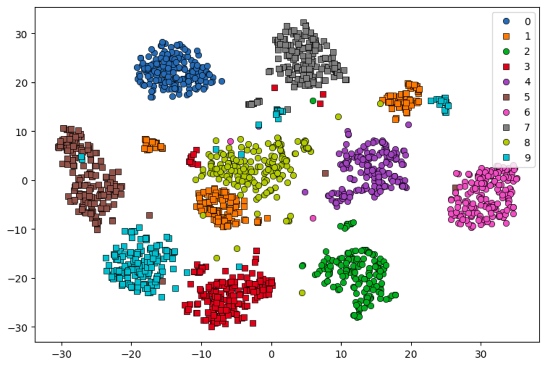
(a) t-SNE withperplexe=10, momentum=0.9, learning_rate=10



(b) t-SNE with perplexe=20, momentum=0.5, learning_rate=10



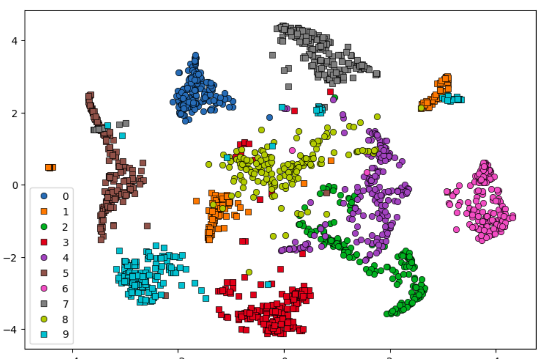
(c) t-SNE with perplexe=20, momentum=0.9, learning_rate=10



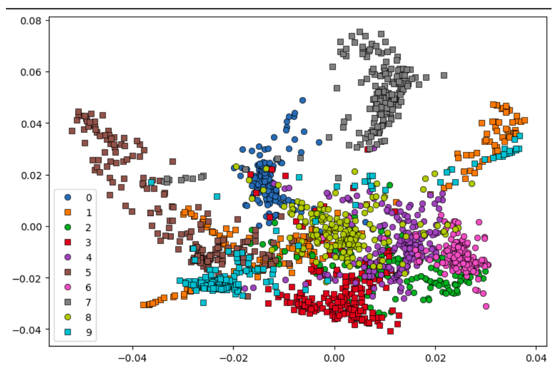
(d) t-SNE with perplexe=20, momentum=0.9, learning_rate=1000

Figure 9: Comparison of t-SNE with different configurations at the 100th iteration

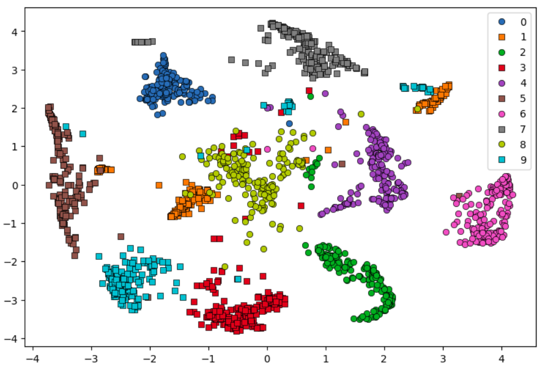
Some configurations already have their definitive cluster, such as configuration **(d)** in **Figure 10**, which hardly changes over iterations, while others approach a coherent result after 200 iterations. Additionally, configurations **(a)** and **(c)** show almost similar results despite having twice the value for the perplexity parameter.



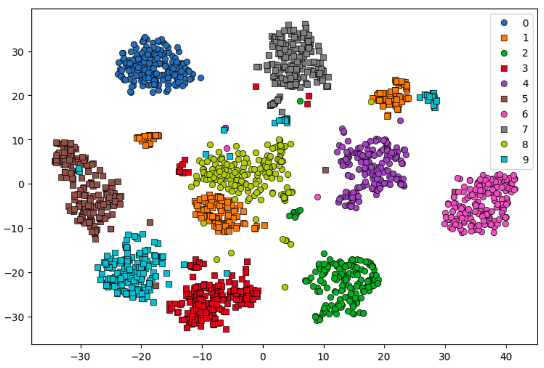
(a) t-SNE withperplexe=10, momentum=0.9, learning_rate=10



(b) t-SNE with perplexe=20, momentum=0.5, learning_rate=10



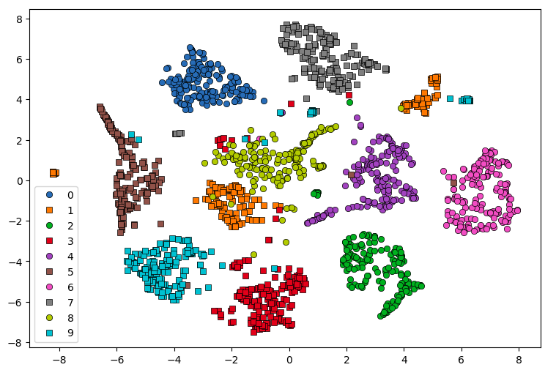
(c) t-SNE with perplexe=20, momentum=0.9, learning_rate=10



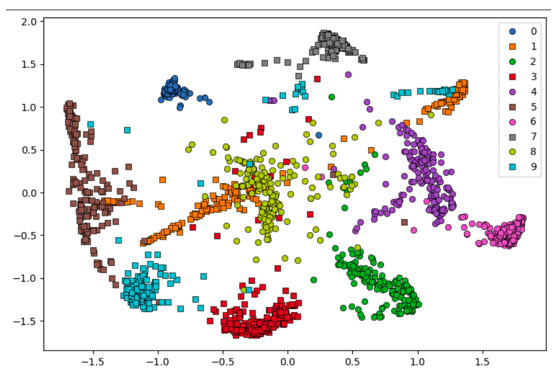
(d) t-SNE with perplexe=20, momentum=0.9, learning_rate=1000

Figure 10: Comparison of t-SNE with different configurations at the 200th iteration

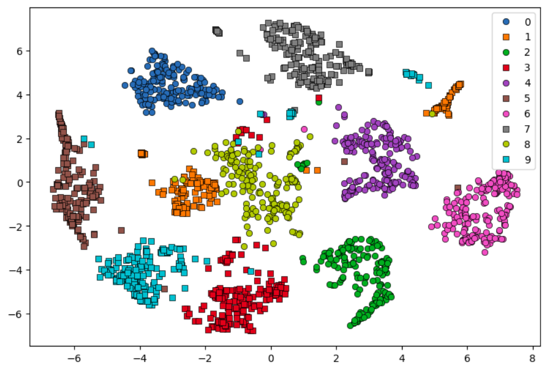
Overall, different clusters are distinguishable with fairly similar shapes, although configuration **(b)** in **Figure 11** appears more "stretched". Among these results, **(d)** demonstrates a much better result due to the precision of its clusters.



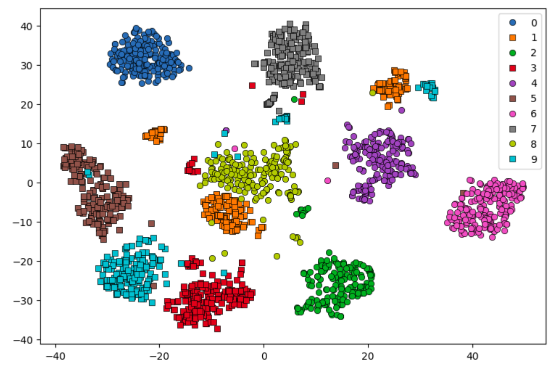
(a) t-SNE withperplexe=10, momentum=0.9, learning_rate=10



(b) t-SNE with perplexe=20, momentum=0.5, learning_rate=10



(c) t-SNE with perplexe=20, momentum=0.9, learning_rate=10



(d) t-SNE with perplexe=20, momentum=0.9, learning_rate=1000

Figure 11: Comparison of t-SNE with different configurations at the 400$^{\text{th}}$ iteration

For the continuation of the analysis, we will base ourselves on the classification below (**Figure 12**) (which corresponds to configuration **(d)**). We easily notice that classes physically close, such as 5, 9, and 3, are positioned side by side because they share similarities in their shapes.

However, each image in the dataset is of size 8 x 8, while the MNIST dataset is of size 28 x 28. This difference likely results in information loss upon loading the data, making it more difficult for this method to capture fine relationships between the data. It is therefore much more influenced by writing style and data pixelation (see dataset). This is why classes 1 and 8 are found in the center; they share common characteristics with other clusters without being dissociated from the rest.

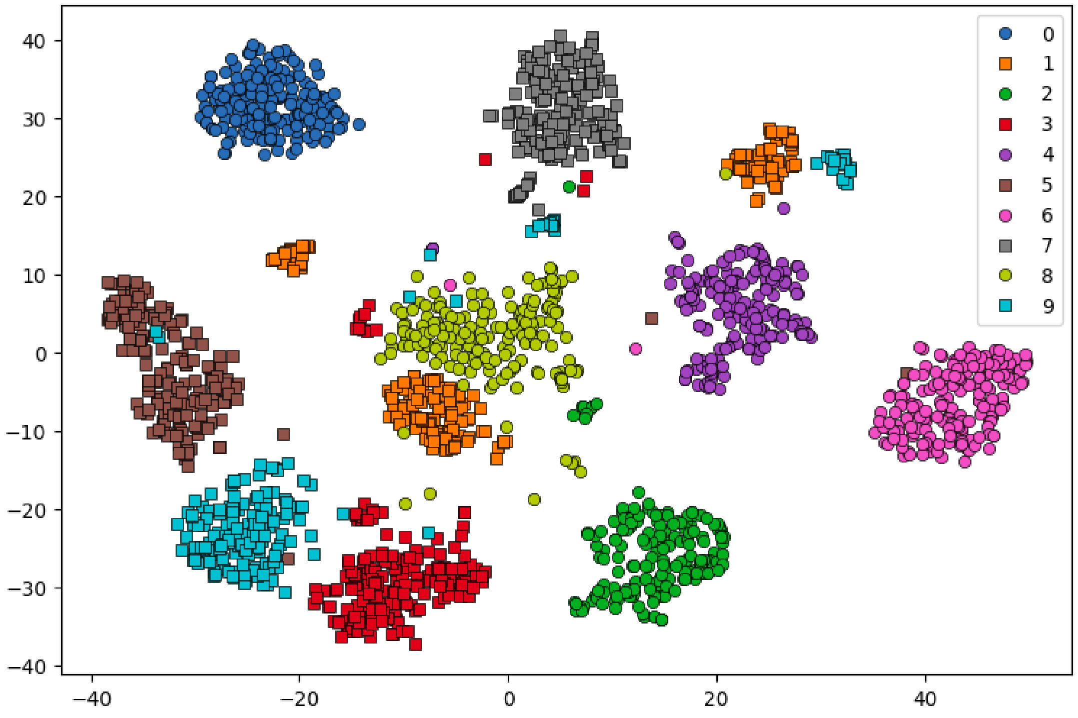


Figure 12: t-SNE with perplexe=20, momentum=0.9, learning_rate=1000

Training time:

In terms of efficiency, this method proves to be less effective than the previous one. The training time for classification take from 1:30 to 2:00 minutes on 500 iterations is relatively long, especially considering the low number of images in the dataset (1797 images). In fact, the number of iterations will considerably influence the efficiency of the programme, as will the number of iterations.

4 Conclusion

The implementation and comparative analysis of ISOMAP and t-SNE in the context of non-linear classification revealed significantly different results and interesting observations.

Initially, we delved into the mechanics of each method, elucidating their underlying principles, architectures, and training methodologies. ISOMAP was presented as a manifold learning technique focused on preserving the intrinsic geometric structure of high-dimensional data, while t-SNE was described as a dimensionality reduction method emphasizing the preservation of local similarities in the data.

Subsequently, we applied these methods to a dataset comprising handwritten digits from the MNIST dataset. Our practical implementation demonstrated that while ISOMAP provides a representation of the data based on geometric distances, t-SNE offers a visualization that emphasizes local similarities between data points.

The analysis of the results unveiled that t-SNE tends to organize clusters in a more visually distinct manner compared to the other method, which typically forms clusters based on geometric proximity. Additionally, t-SNE often produces circular arrangements of clusters, with certain clusters positioned at the center, possibly reflecting shared characteristics among those clusters.

In conclusion, our exploration of these methods has shed light on their respective strengths and limitations in capturing non-linear relationships within data. While ISOMAP offers a geometrically informed representation, t-SNE excels in revealing local similarities and organizing data into visually interpretable clusters. The choice between both ultimately depends on the specific goals and characteristics of the dataset, such as the desired emphasis on geometric structure versus local similarity preservation. This study contributes to a deeper understanding of manifold learning techniques and their applicability in diverse contexts of data analysis and visualization.

To further validate and extend our findings, it would be beneficial to conduct experiments on the MNIST dataset using the entire dataset with images represented in their original size of 28x28 pixels. This comprehensive analysis would provide deeper insights into the performance of two methods across a broader range of data dimensions and complexities. Such experimentation could offer valuable guidance for practitioners seeking to leverage manifold learning techniques for high-dimensional data analysis and visualization.

References

[1] Van der Maaten, Laurens, and Geoffrey Hinton. "Visualizing data using t-SNE." Journal of machine learning research 9.11 (2008), https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.

[2] Isomap,s