TDT4113 (PLAB) Project 6: Dimensionality Reduction for Visualization

— Assignment Sheet —

Purpose of this project:

- Gain a basic appreciation of data science programming and data visualization
- Implement a data science algorithm with given pseudocode
- Be aware of efficiency of the implemented code

Practical Information:

- This project will be done in groups of size four.
- You must upload your code for this project to BLACKBOARD by **08:00 (in the morning) on Wednesday, 7 April, 2021**.
- Your program can use only the allowed packages; cheating will fail your project
- You must demonstrate and get approval by **20:00 on Wednesday, 7 April, 2021**; early demos are possible.
- Your code needs to have a Pylint level of at least 8.0

1 Introduction

Nowadays, the rapid growth of the amount of data demands more efficient methods for information digesting and communicating. Data visualization facilitates viewers to quickly digest information in massive data. It is therefore increasingly applied as a critical component in various domains.

In digital world, a data item is often described with multiple numerical features. If each feature is treated as a dimension, the item is a data point in a multidimensional space. It is quite often that the dimensionality becomes hundreds or more. However, human are three-dimensional beings. It is difficult for us to see the data distribution in four or higher dimensional spaces.

Dimensionality reduction (DR) is the transformation of data from a high-dimensional space into a low-dimensional space so that the transformed representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension. If the descriptions of the data items are mapped to 2D or 3D space, DR becomes a useful tool for visualization as human users can see the mapped data points.

There are many different dimensionality reduction methods in the literature, of which some examples are given in Wikipedia. In the project you will practice a linear method and two nonlinear methods. The detailed tasks are given in Sections 4, 5 and 6, respectively. Before the task descriptions, we will briefly review the related mathematics in Section 2 and introduce the data sets in Section 3.

2 Related Mathematics

Here we summarize a few mathematical facts which are needed in the project. You may recap your university mathematics if you want more details.

- A *matrix* here is an array of real-valued numbers arranged in rows and columns. For example, the dimension of the matrix below is 2×3 (read "two by three"), because there are two rows and three columns:

$$\begin{pmatrix} 3.0 & -5.3 & 2.1 \\ 12.4 & -1.9 & -7.6 \end{pmatrix} \tag{1}$$

- The *matrix product* of an $m \times r$ matrix A and a $r \times n$ matrix B, denoted by C, is an $m \times n$ matrix, where: $C_{ij} = \sum_{k=1}^{r} A_{ik} B_{kj}$.

- A pair of *eigenvalue* λ and *eigenvector* u for a matrix A fulfills $Au = \lambda u$.

- The *covariance matrix* of a multidimensional random variable x is $\Sigma = \mathbb{E}\left[(x-\mu)(x-\mu)^T\right]$, where $\mu = \mathbb{E}[x]$. In practice the mathematical expectation $\mathbb{E}$ is often replaced by sample average.

- Covariance matrices are symmetric and positive-semidefinite. That is, all eigenvalues of a covariance matrix are nonnegative.

3 Data Sets

In this project you will use a synthetic data sets `SWISS-ROLL` and a real-world data set `OPTDIGITS`.

3.1 SWISS-ROLL

This data set contains 2000 points in 3D space, where the points distribute in a subspace like a swill roll (see Figure 1).

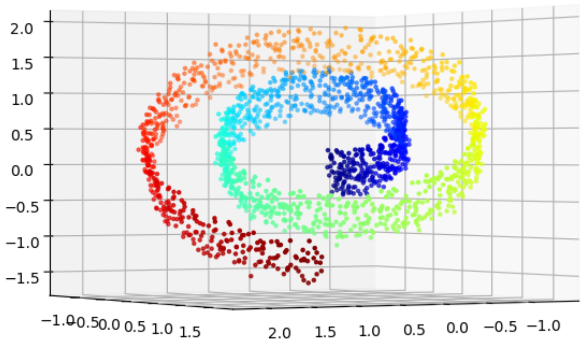

Figure 1: Scatter plot of the `SWISS-ROLL` data set in 3D space. Here each dot represents a data point. We color the points from blue to red according to their order from inner to outer.

The intrinsic dimensionality for the roll is two. Your task is to unfold the roll to a 2D space and display the mapped points on a computer screen. The expected output is roughly a rectangle, where the color order (from blue to red) should be well preserved from one side to the other side of the rectangle.

3.2 OPTDIGITS

The data set comes from the Optical Recognition of Handwritten Digits Data Set in the UCI repository[1]. It contains grayscale images of handwritten digits, where the image resolution is 8×8. See Figure 2 for example images. We have vectorized each image by concatenating the rows. In this way the data set contains 5620 data points in 64 dimensional space.

We provide the data set in two files:

[1]https://archive.ics.uci.edu/ml/datasets/optical+recognition+of+handwritten+digits

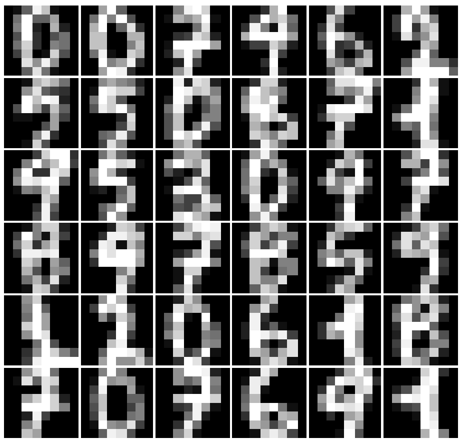
Figure 2: Example images in the `OPTDIGITS` data set.

1. `digits.csv` which contains the data matrix in comma separated value (CSV) format, with each row for a data point (i.e. an image)

2. `digits_label.csv` which contains the labels (i.e. the digits in the corresponding images)

Notice that the labels should not be used in dimensionality reduction. They are only used to verify whether the distribution of mapped points are meaningful. For example, after DR, you can color the mapped points according to their labels in the visualization.

The task is to map the data points to 2D space and show them on a computer screen. The visualization should basically show the ten digit clusters. That is, data points belonging to the same digit should distribute nearby while those of different digits should be well separated.

4 Principal Component Analysis (PCA)

First we will practice a classical linear DR method PCA. The method finds a low-dimensional subspace where the average squared distance from the points to the subspace is minimized. You can consult for example the Wikipedia to see theoretical derivation of PCA. Here we only summarize the application steps of PCA.

The PCA procedure comprises two phases:

1. *fit*: to fit a PCA model for the given data points $\{x_i\}_{i=1}^N$. Detailed steps are:

   (a) Center the data: $x_i \leftarrow x_i - \mu$, where $\mu = \frac{1}{N}\sum_i x_i$

   (b) Compute the covariance matrix Σ of the centered data; you can use `numpy.cov`

   (c) Perform truncated eigendecomposition of Σ

      - If $D - 1 > d$, you can use `scipy.sparse.linalg.eigs`
      - If $D - 1 = d$, first use `numpy.linalg.eigh` to find all eigenvalues and eigenvectors and sort the eigenvalues

   Keep the eigenvectors which correspond to the d largest eigenvalues; the eigenvectors as column vectors form the transformation matrix F;

2. *transform*: to use the PCA model to transform a data point x to $y \in \mathbb{R}^2$. It comprises two steps:

   (a) center the data point $x \leftarrow x - \mu$

   (b) project the data: $y \leftarrow F^T x$

For each data point x_i in `digits.csv`, the resulting y_i is two-dimensional. You can then visualize the y_i's using `matplotlib.pyplot.scatter`.

Implementation – Part 1:

- Implement the PCA class which contains at least two functions `fit` and `transform`. Or you can combine the two functions into a single function `fit_transform` which needs not center the data twice.
- Apply your PCA method to the two data sets. Visualize the mapped points using scatter plots.
- You cannot import any existing packages or code that contains PCA. You can use only `numpy`, `scipy` and `matplotlib`.

5 Isometric Mapping (Isomap)

As you will see, PCA as a linear dimensionality reduction method does not perform well when the subspace is a curved. In mathematics such a subspace is called a curved manifold where Euclidean distances are reliable only in small neighborhoods.

Therefore we need nonlinear methods to overcome this. In this section we discuss a nonlinear method called Isometric Mapping or Isomap [Tenenbaum00]. The method comprises two phases:

1. *compute geodesics*: it takes the high-dimensional data points $\{x_i\}_{i=1}^N$ and a user specified parameter k as input, and returns a distance matrix D, where D_{ij} is the shortest-path distance between x_i and x_j along the manifold (called geodesic distance)

   In theory we need the manifold metric to calculate the geodesics in Phase 1. This is often not available in data science, but you can use program to approximate the calculation. Here the Euclidean distances in small neighborhoods are still reliable. So you can approximate path along the manifold by connecting the local Euclidean distances. Detailed steps are as follows:

   (a) Compute all pairwise Euclidean distances. It is wise to code this in a utility function; try to avoid nested loops in Python over (i, j)'s because they are slow. Instead, write out the square of difference you will get three terms and each term can be coded with matrix operations.

   (b) For each point, keep only the distances of the k-nearest neighbors (kNN, k =10 to 50), while setting the others to 0. This gives a distance matrix $D^{k\text{NN}}$ of the kNN graph;

   (c) Perform a shortest-path graph search using $D^{k\text{NN}}$. You can use `sklearn.utils.graph_shortest_path.graph_shortest_path`. Return the approximated geodesic distance matrix D^{geodesic}

2. *Multidimensional Scaling (MDS)*: it applies the classical MDS over the geodesic distance matrix D^{geodesic}, and returns the coordinates of the low-dimensional mapped points $Y = \{y_i\}_{i=1}^N$. You can find the the steps of classical MDS in Wikipedia as follows.

   (a) Set up the squared proximity matrix $D^{(2)}$, where $D_{ij}^{(2)}$ is the square of the (i, j)-th entry in D^{geodesic}.

   (b) Apply double centering: $B = -\frac{1}{2}JD^{(2)}J$ using the centering matrix $J = I - \frac{1}{N}\mathbf{1}\mathbf{1}^T$, where N is the number of data points, I is the identity matrix, and $\mathbf{1}$ is a vector filled with ones.

   (c) Determine the m largest eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ and corresponding eigenvectors $e_1, e_2, \ldots, e_m$ of B (where $m = 2$ for this project).

   (d) $Y \leftarrow E_m\Lambda_m^{1/2}$, where $E_m = [e_1, \ldots, e_m]$ and Λ_m is a diagonal matrix with $\lambda_1, \ldots, \lambda_m$ in the diagonal. The resulting matrix $Y \in \mathbb{R}^{N\times 2}$, where each row gives the coordinates of the mapped points.

Implementation – Part 2:

- Implement the Isomap method for dimensionality reduction.
- Apply your Isomap method to the two data sets. Visualize the mapped points using scatter plots.
- You can use only `numpy`, `scipy`, `matplotlib` and `graph_shortest_path` from `sklearn.utils.graph_shortest_path`. You cannot import other packages or functionalities in Scikit-Learn (sklearn). You cannot use other packages or code that contains Isomap.

6 Student t-Distributed Stochastic Neighbor Embedding (t-SNE)

As you will see, Isomap works reasonably well for the `SWISS-ROLL` data set but it still fails to reveal clusters in the `OPTDIGITS` data set. In this section we will discuss another method called t-SNE [vanderMaaten08] to address this.

6.1 t-SNE learning objective

Again, let $X = \{x_i\}_{i=1}^N$ be the input high-dimensional data points and $Y = \{y_i\}_{i=1}^N$ the mapped data points in 2D space. The t-SNE method also comprises the following phases.

1. Compute pairwise similarities between the input data points; originally t-SNE employs the entropic affinities. Here for simplicity you will use the k-NN graph (denoted as p, k =10 to 50):

$$p_{ij} = \begin{cases} 1 & \text{if } i \neq j \text{ and } (x_i \text{ is among the k-nearest neighbor of } x_j \text{ or vice versa}) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

2. Seek the mapped data points $Y = \{y_i\}_{i=1}^N$ such that $p \approx q$, where q is similarity matrix in the 2D space and defined as

$$q_{ij} = \frac{1}{1 + \|y_i - y_j\|^2} \tag{3}$$

with $\|y_i - y_j\|^2$ being the squared Euclidean distance between y_i and y_j. Zero the diagonal $q_{ii} = 0$ for $i = 1, \ldots, N$.

The approximation is achieved by minimizing the following divergence (called Kullback-Leibler divergence) between normalized version of p and q:

$$D(P, Q) = \sum_{i=1}^{N} \sum_{j=1}^{N} P_{ij} \ln \frac{P_{ij}}{Q_{ij}} \tag{4}$$

where

$$P_{ij} = \frac{p_{ij}}{\sum_{i=1}^{N} \sum_{j=1}^{N} p_{ij}} \quad \text{and} \quad Q_{ij} = \frac{q_{ij}}{\sum_{i=1}^{N} \sum_{j=1}^{N} q_{ij}}. \tag{5}$$

with the convention $0 \ln 0 = 0$.

6.2 t-SNE optimization

The above minimization procedure forms the t-SNE algorithm, which is an unconstrained optimization of a smooth function:

$$Y_{\text{t-SNE}} = \arg\min_{Y} \ D(P, Q). \tag{6}$$

Note that Q is a function of q, which is in turn a function of Y. Such optimization can be implemented by gradient descent with momentum:

1. For $i = 1, \ldots, N$
   - sample y_{id} from Normal distribution $\mathcal{N}(0, 10^{-4})$ for $d = 1, 2$
   - initialize the gain $g_i = 1$ and the change $\Delta_i = 0$
2. For iteration=1 to maximum_iteration
   (a) calculate the gradient over each y_i (denote by ∇_i):

   $$\nabla_i \stackrel{\text{def}}{=} \frac{\partial D(P, Q)}{\partial y_i} = 4 \sum_j (P_{ij} - Q_{ij})\, q_{ij}\, (y_i - y_j) \tag{7}$$

   (b) update the gain g_i: (for $d = 1, 2$)

   $$g_{id} \leftarrow \begin{cases} g_{id} + 0.2 & \text{if } \operatorname{sign}(\nabla_{id}) \neq \operatorname{sign}(\Delta_{id}) \\ g_{id} \times 0.8 & \text{if } \operatorname{sign}(\nabla_{id}) = \operatorname{sign}(\Delta_{id}) \end{cases} \tag{8}$$

   If $g_{id} < 0.01$, set $g_{id} = 0.01$.

(c) update the change $\Delta_i \leftarrow \alpha\Delta_i - \epsilon g_i \nabla_i$

(d) update $y_i \leftarrow y_i + \Delta_i$ where $\epsilon > 0$ is the learning step size

In the loop, α is the momentum amount, and ϵ is the learning step size. In this project, you can set maximum iteration to 1000, $\alpha = 0.8$ and $\epsilon = 500$.

6.3 Implementation details

There are a few tricks to improve the efficiency and convergence:

- *"lying" P at beginning*: the t-SNE authors suggested use $4P$ in place of P in the first 100 iterations; this will create more empty space between clusters and improve the visualization at the end; since the 101st iteration change back to normal P
- *smaller momentum at beginning*: another initialization trick is to to set $\alpha = 0.5$ in the first 250 iterations; this gives more freedom for the algorithm to explore the embedding structure;
- *use matrix operations*: avoid loops over the data points, which is rather slow in Python; instead, try to organize the computation in matrix products or matrix-wise commands in Python.
- *fix the random seed*: use `numpy.random.seed` to set a constant seed before sampling $\{y_i\}_{i=1}^N$; this can help you repeat the results and debug your program.

Implementation – Part 3:

- Implement the `tSNE` algorithm for dimensionality reduction.
- Apply your t-SNE method to the `OPTDIGITS` data set. Visualize the mapped points using scatter plot. Visualizing `SWISS-ROLL` is optional.
- You can use only `numpy` and `matplotlib`. You cannot use other packages or code that contains t-SNE.

7 Demonstration

During the demonstration session for this project, you will be asked to run your program for the following tasks:

- Run PCA to visualize the `SWISS-ROLL` data set
- Run PCA to visualize the `OPTDIGITS` data set
- Run Isomap to visualize the `SWISS-ROLL` data set
- Run Isomap to visualize the `OPTDIGITS` data set
- Run t-SNE to visualize the `OPTDIGITS` data set

Your visualizations will be compared with the answers. You will get pass if all of them are close to the answers.

8 Extra challenges

This is an optional part of the assignment. You do not have to do it to get full credit for the project.

- Try to speed up the kNN graph construction. You may try `sklearn.neighbors.NearestNeighbors`. Then compare its running time with your own implementation.
- In t-SNE, you may record Y after each iteration. Then you can visualize the sequence of mapped points as an video, with each scatter plot as a frame.
- Consider an even larger data set, for example the MNIST data set. You may find the original data set in `http://yann.lecun.com/exdb/mnist/`. Or you can download the .csv files from `https://www.python-course.eu/neural_network_mnist.php`.

  For such a large data set, you cannot store the $N \times N$ dense matrices in memory. Think about how to overcome this scalability problem.

— o —

References

[Tenenbaum00] TENENBAUM, J.; DE SILVA, V.; LANGFORD, J. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[vanderMaaten08] VAN DER MAATEN, L.; HINTON, G. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.