Final exam. Due Sunday, December 15, 2024, 11:59 PM.

Resources: *You can any course materials and textbooks. You are not allowed to collaborate with your classmates or use anybody's help. You are not allowed to use AI.*

Programming: *You can use any suitable language. Matlab or Python are preferable.*

Typing: *Use latex or any other suitable text editor. I will subtract 10% of the maximal score if the file is hand-written.*

Submission: *You should upload on ELMS a single pdf file with your codes linked to it.*

1 Problem 1. Linear Algebra.

Linear discriminant analysis (LDA) (also known as *Multiple Discriminant Analysis* (MDA)) [1] is a linear dimensional reduction method aiming at projecting data from different categories to a low-dimensional space so that the images of data from different categories are separated as much as possible.

Let X be a $n \times d$ matrix whose rows are d-dimensional data points. We assume that the dataset consists of $c > 1$ categories. Let $\mathcal{I}_i$ denote the set of indices of data from category i, $i = 1, \ldots, c$, and

$$|\mathcal{I}_i| = n_i, \quad \mathcal{I}_i \cap \mathcal{I}_j = \varnothing, \quad \mathcal{I}_1 \cup \ldots \cup \mathcal{I}_c = \{1, \ldots, n\}.$$

We seek a $d \times d_1$ matrix W, $d_1 \le c - 1$, that maps the data onto a d_1-dimensional space:

$$Y = XW, \quad \text{or,} \quad y_k = W^\top x_k, \quad k = 1, \ldots n, \tag{1}$$

such that images of the data from different categories under this mapping are separated as much as possible. In (1), $x_i \in \mathbb{R}^d$ is the kth row of X written as a column vector. Likewise is y_k.

To pose this problem mathematically, we define the mean for each category in spaces $\mathbb{R}^d$ and $\mathbb{R}^{d_1}$,

$$m_i = \frac{1}{n_i} \sum_{k \in \mathcal{I}_i} x_k, \quad \tilde{m}_i = \frac{1}{n_i} \sum_{k \in \mathcal{I}_i} y_k = W^\top m_i, \quad i = 1, \ldots, c. \tag{2}$$

To describe the data variation within and between the categories, we define the *within-class* and *between-class scatter matrices* S_w and S_b, respectively. These matrices are of size $d \times d$. The within-class scatter matrix is defined as

$$S_w := \sum_{i=1}^{c} S_i, \tag{3}$$

where S_i is the scatter matrix of category i defined as

$$S_i := \sum_{k \in \mathcal{I}_i} (x_k - m_i)(x_k - m_i)^\top \equiv \left(X_{\mathcal{I}_i,:} - \mathbf{1}_{n_i \times 1} m_i^\top\right)^\top \left(X_{\mathcal{I}_i,:} - \mathbf{1}_{n_i \times 1} m_i^\top\right). \tag{4}$$

The between-class scatter matrix is defined as

$$S_b := \sum_{i=1}^{c} n_i (m_i - m)(m_i - m)^\top, \quad \text{where} \quad m := \frac{1}{n} \sum_{i=1}^{c} n_i m_i \equiv \frac{1}{n} \sum_{k=1}^{n} x_k \tag{5}$$

is the overall mean. Note that the rank of $S - b$ is at most $c-1$ as it is a sum of c rank-1 matrices, and these matrices are not independent (see below).

We use a similar notation with tilde on top for the mapped data. The within- and between-class scatter matrices for the mapped data are:

$$\tilde{S}_w = W^\top S_w W, \quad \tilde{S}_b = W^\top S_b W. \tag{6}$$

Our goal is to find a $d \times d_1$ matrix W, mapping the data within each category into clusters and mapping the clusters corresponding to different categories as far as possible from each other. Therefore, we define the objective function as

$$J(w) = \frac{w^\top S_b w}{w^\top S_w w}. \tag{7}$$

The function $J(w)$ is minimized by solving the *generalized eigenvalue problem* (see below). It has at most $c-1$ nonzero eigenvalues. We compose the matrix W out of the eigenvectors corresponding to the top $d_1 \le \mathsf{rank}(S_b)$ eigenvalues.

1. Prove that the rank of S_b is at most $c-1$. *Hint: Show that it can be represented as a sum of $c-1$ rank-1 matrices.*

2. Assume that the within-class scatter matrix S_w is nonsingular. Calculate the gradient of $J(w)$ with respect to w. Look at it attentively and show that the stationary points of $J(w)$ are those where

   $$S_b w = \lambda S_w w. \tag{8}$$

   What are these values of λ? The problem of finding pairs (λ, w) satisfying (8) is called the *generalized eigenvalue problem.*

3. Then use the Cholesky decomposition of S_w, $S_w = LL^\top$, to reduce the generalized eigenvalue problem to a symmetric eigenvalue problem of the form $Ay = \lambda y$.

4. Apply LDA to the MNIST set of training images of 3, 8, and 9. Thus, you have a dataset consisting of three categories. Hence, S_b has most rank 2. Solve the arising generalized eigenvalue problem using the standard functions in Matlab or Python. Display the mapped training data of 3, 8, and 9 in 2D colored in three different colors, respectively. Include legend. You should see that the result is quite good☺.

5. For comparison, map the MNIST set of training images of 3, 8, and 9 onto the first two PCAs (the top two right singular vectors of the centered data matrix $X - \mathbf{1}_{n\times 1} m^\top$). You will see that this mapping mixes the images of 3, 8, and 9 much more.

2 Problem 2. Optimization

In this problem, you should show your knowledge of optimization algorithms and the ability to apply them. I provide the function and its gradient, a plotting function, and a few auxiliary functions.

Suppose you have a spring system consisting of 21 nodes and 24 springs, see Fig. 1. Imagine that you need to attach the red nodes to a hoop so that the total spring energy at equilibrium is minimal. The radius of the hoop is $R = 3$. The equilibrium length of each spring is $r_0 = 1$, and the spring constant of each spring is $\kappa = 1$. The energy of the spring system is

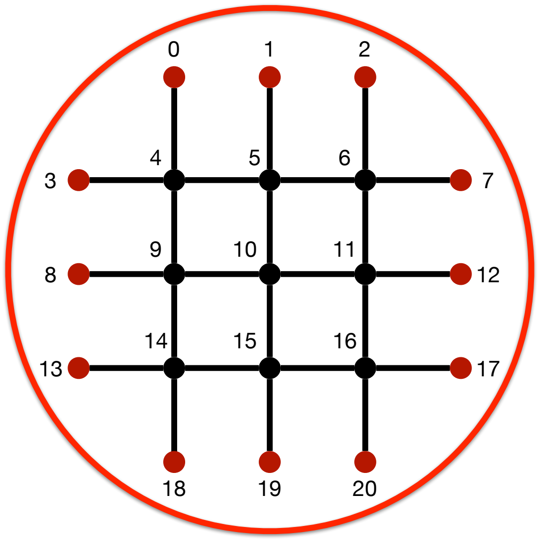

Figure 1: The spring system in Problem 2.

$$E = \frac{\kappa}{2} \sum_{\text{spring}} \left(\|\mathbf{r}_{\text{spring}[0]} - \mathbf{r}_{\text{spring}[1]}\| - r_0 \right)^2 \tag{9}$$

where the sum is taken over all springs, and $\mathbf{r}_{\text{spring}[0]}$ and $\mathbf{r}_{\text{spring}[1]}$ denote the (x, y)-coordinates of the endpoints of the spring.

Since the positions of the 12 red nodes must be on the hoop, they are defined by their angles on the hoop. The 9 black nodes are free. Their positions are described by their (x, y) coordinates. Thus, we need to optimize the energy with respect to $12 + 9 + 9 = 30$ parameters, the first 12 of which are the angles of the red nodes, the next 9 are the x-coordinates of the black nodes, and the last 9 are the y-coordinates of the black nodes.

The derivative of the energy with respect to the angle θ_i of a red node i is

$$\frac{\partial E}{\partial \theta_i} = \kappa R \left(\|\mathbf{r}_i - \mathbf{r}_j\| - r_0 \right) \frac{-x_i \sin\theta_i + y_i \cos\theta_i}{\|\mathbf{r}_i - \mathbf{r}_j\|}, \tag{10}$$

where j is the unique node connected to the red node i by a spring. The derivatives of the energy with respect to x_i and y_i, the x- and y-coordinates of a black node node i, are, respectively

$$\frac{\partial E}{\partial x_i} = \kappa \sum_{j \sim i} \left(\|\mathbf{r}_i - \mathbf{r}_j\| - r_0 \right) \frac{x_i}{\|\mathbf{r}_i - \mathbf{r}_j\|}, \tag{11}$$

$$\frac{\partial E}{\partial y_i} = \kappa \sum_{j \sim i} \left(\|\mathbf{r}_i - \mathbf{r}_j\| - r_0 \right) \frac{y_i}{\|\mathbf{r}_i - \mathbf{r}_j\|}, \tag{12}$$

where the summation is over all nodes j connected to the black node i by springs.

Task:

- Implement any **two different** optimization methods to find the minimal value of the spring energy.
- Plot the spring energy and the norm of its gradient versus the iteration number for each method.
- Plot the resulting view of the spring system stretched on the hoop for each method.
- Print the positions of the nodes, the resulting energy, and the norm of the gradient for each method.

Some useful arrays and functions.

- The enumeration of the nodes in Python is shown in Fig. 1. The enumeration in Matlab is shifted by 1. Below I refer to the indexing and functions in Python. The functions in Matlab are similar though slightly adjusted for Matlab specifics.
- `Asymm` is the adjacency matrix of the spring system. `A` is its superdiagonal part.
- `ind_hoop` is the list of indices of the hoop nodes (the red nodes).
- `ind_hoop` is the list of indices of the free nodes (the black nodes).
- `springs` is a 2×24 array whose columns are the indices of the spring end nodes.
- Function `draw_spring_system(pos,springs,R,ind_hoop,ind_free)` plots the spring system stretched on the hoop.
- Function `compute_gradient(theta,pos,Asymm,r0,kappa,R,ind_hoop,ind_free)` computes the gradient of the energy function.
- Function `Energy(theta,pos,springs,r0,kappa)` computes the energy function.
- Function `vec_to_pos(vec)` converts the vector of parameters,

$$\begin{aligned}\texttt{vec} = [&\theta_0,\theta_3,\theta_8,\theta_{13},\theta_{18},\theta_{19},\theta_{20},\theta_{17},\theta_{12},\theta_7,\theta_2,\theta_1,\\ &x_4,x_5,x_6,x_9,x_{10},x_{11},x_{14},x_{15},x_{16},y_4,y_5,y_6,y_9,y_{10},y_{11},y_{14},y_{15},y_{16}],\end{aligned}$$

  to the vector of angles `theta` of the red nodes and 21×2 array `pos` of (x, y) coordinates of all nodes.
- Function `gradient(vec)` evaluates the energy gradient given the vector of parameters as input. It is created for use in the optimization routine. It calls `vec_to_pos(vec)` and `compute_gradient(theta,pos,Asymm,r0,kappa,R,ind_hoop,ind_free)`.
- Function `func(vec)` evaluates the energy function given the vector of parameters as input. It is created for use in the optimization routine. It calls `vec_to_pos(vec)` and `Energy(theta,pos,springs,r0,kappa)`.

3 Problem 3. Monte Carlo.

The unit cube in $\mathbb{R}^d$ centered at the origin is the set

$$C^d = \left\{ \mathbf{x} \in \mathbb{R}^d \mid \max_{1 \le i \le d} |x_i| \le \tfrac{1}{2} \right\}, \tag{13}$$

while the unit ball in $\mathbb{R}^d$ centered at the origin is the set

$$B^d = \left\{ \mathbf{x} \in \mathbb{R}^d \mid \sum_{i=1}^{d} x_i^2 \le 1 \right\}. \tag{14}$$

Obviously, all centers of the $(d-1)$-dimensional faces of C^d, i.e., the points with one coordinate $\pm\frac{1}{2}$ and the rest zeros, lie inside B^d. The most remote points of C^d from the origin are the corners with all coordinates $\pm\frac{1}{2}$. The distance of the corner of C^d from the origin is $\sqrt{d}/2$. For $d \ge 5$, the corners of C^d and some of their neighborhoods lie outside B^d. The d-dimensional volume of C^d is one, while the volume of B^d is

$$\mathrm{Vol}(B^d) = \frac{\pi^{d/2}}{\frac{d}{2}\Gamma\left(\frac{d}{2}\right)}. \tag{15}$$

$\mathrm{Vol}(B^d)$ tends to zero as $d \to \infty$. Hence, the fraction of the unit cube C^d lying inside B^d also tends to zero as $d \to \infty$. You can read about this in more detail and see some illustrations e.g. in [2] (read at least up to Section 1.2.2 inclusively).

Calculate

$$\mathrm{Vol}(B^d \cap C^d) \quad \text{in} \quad d = 5, 10, 15, 20 \tag{16}$$

using Monte Carlo integration in two ways.

1. Way one uses a sequence of independent uniformly distributed random variables in the unit cube C^d.
2. Way two uses a sequence of independent uniformly distributed random variables in the unit ball B^d. (You need to think of a way to generate such a random variable.)

References

[1] R. O. Duda, P. E. Hart, D. G. Stork, Pattern Classification, 2nd Edition, John Wiley & Sons, Inc. 2001

[2] High-Dimensional Space, notes by Venkatesan Guruswami (Professor, Computer Science Dept, Carnegie Mellon University)