Overview

This MATLAB code uses the Robust Optimal Scoring Discriminant Analysis (OSDA) algorithm as presented by Huang & Zhang (2020).

1 Prerequisites

Any version of MATLAB or Python 3 will work for this implementation.

2 Input Data

The input data for the ROSDA algorithm consists of two matrices: X and Y. The X matrix is a feature matrix with dimensions $n \times p$, where n represents the number of samples and p denotes the number of features; in the example, a synthetic 100×40 matrix is created using the command `X = randi([0 10], 100, 40)` and then normalized with `X = normalize(X)`. This matrix can be replaced with an image matrix. The Y matrix is a target matrix of size $n \times K$, where K is the number of classes; the example creates random binary labels (0 or 1) for 100 samples, which is substituted with the classes for the image matrix.

3 Instructions

3.1 Set Parameters

The ROSDA algorithm begins with initializing key parameters. A convergence threshold for the residual sum of squares (RSS) change is established with `tolerance = 1e-14`, ensuring the algorithm stops when updates become sufficiently small. To ensure a particular number of loops are achieved, `max_iteration = 10000` defines the maximum number of iterations needed before the cycle is complete. Parameter `K = 2` specifies the number of classes and needs to be adjusted based on the classes needed. Initial values for the residual sum of squares are set as `RSSold = 2` and `RSS = 10`. Finally, an iteration counter is initialized with `iter = 0`.

3.2 Prepare Data

In the ROSDA algorithm, we next initialize the matrices X and Y using the previously defined conditions. Again, X can be replaced with the image matrix and Y replaced with the classes.

3.3 Initialize Variables

Next, we will initialize some variables needed for the ROSDA loop. The variable `n = size(X,1)` determines the number of rows within the image. The value `Q = K-1` is established, where K is the number of classes. The initial feature coefficients are defined as `betaj = ones(size(X,2), Q)`, creating a matrix of ones with dimensions $p \times Q$. Similarly, initialized is `Thetaj = ones(K, Q)`, a matrix of ones with dimensions $K \times Q$. Finally, the variable D is computed as $D = \left(\frac{1}{n}\right) \times \text{transpose}(Y) \times Y$.

3.4 Run ROSDA Algorithm

The code uses the following Algorithm presented by Huang & Zhang (2020) [1].

Algorithm 1 Robust OSDA (ROSDA)

1: Input: Y is a $n \times K$ matrix, X is a $n \times p$ matrix, and $Q = K - 1$.
2: Output: $(B^{(j)}_{p\times Q}, \Theta^{(j)}_{K\times Q})$.
3: Initialize $B^{(0)}$ and $\Theta^{(0)}$ as matrices of 1's.
4: Standardize X.
5: Compute $D = \frac{1}{n} Y^\top Y$.
6: Run a loop until the criterion, $|\text{old RSS} - \text{RSS}|/\text{RSS} \leq$ Tolerance and the number of iterations reach a maximum number, is achieved.
7: Compute $z_i^{(j-1)} = \|Y_i\Theta^{(j-1)} - X_i B^{(j-1)}\|_2^2$.
8: In jth iteration, compute the weight matrix $W = \text{diag}(W_1, \cdots, W_n) \in \mathbb{R}^{n\times n}$ with $W_i = \Psi'(z_i^{(j-1)})$.
9: Assign RSS as the old RSS
10: Compute $B^{(j)} = (X^\top WX)^{-1} X^\top WY\Theta^{(j-1)}$ by backward substitution.
11: Compute P_Y as the matrix of the left singular vectors of Y.
12: Compute Θ_0 as the Q smallest right singular vectors of $(I_{n\times n} - P_{WX})WP_Y$.
13: Compute $\Theta = \frac{1}{\sqrt{n}} D^{-1} Y^\top P_Y \Theta_0$.
14: Update RSS as $\frac{1}{n}\sum_{i=1}^{n} \Psi(z_i)$, where $z_i = \|Y_i\Theta^{(j)} - X_i B^{(j)}\|_2^2$.
15: Repeat the above procedure 5-13 until the stopping criterion is satisfied.
16: For each data X_i, predict classes by finding which Θ_k is closest to $X_i B$ for $1 \leq k \leq K$.

Within the function $\Psi(z_i)$ and the function $\Psi'(z_i)$, the tuning parameter ζ needs adjusted to be optimal.

3.5 Predict Classes

To predict the classes, the code calculates XB and finds the closest Θ_k for $X_i B$.

3.6 Calculate Accuracy

The `predicted_classes` matrix is converted back to a vector of labels (0 or 1) to match the original format of `Y_labels`. The accuracy is then computed as `accuracy = sum(predicted_labels == Y_labels) / n * 100`, where the number of correct predictions (determined by comparing `predicted_labels` with `Y_labels`) is divided by the total number of samples (n) and multiplied by 100 to yield a percentage. The results are then displayed.

4 Notes

The same principles apply to the Python code as stated in this documentation using MATLAB code.

References

[1] Huang, H. H., & Zhang, T. (2020). Robust discriminant analysis using multi-directional projection pursuit. *Pattern Recognition Letters*, 138, 651-656.