Multilinear discriminant analysis

Milica Miskovic

May 16, 2020

Multilinear Discriminant Analysis (MDA) is an extension of the traditional method LDA which improves the generalization by dividing the discriminant analysis into iterative steps. The biggest advantage of MDA is that input data is unchanged and keeps the tensor form, therefore the structure of the data is preserved, which provides insights in more complex, nonlinear correlations. The results of conducted experiments vary which imply that performance of the algorithm greatly depends on the structure of the data.

1 INTRODUCTION

Since the graphical data tends to be large, the development in computer vision (beyond object detection, classification and face recognition) in recent years caused it to be more complex and has reveled more complex connections. This largely motivated the accelerated development of new and more intricate techniques for dimensionality reduction. The most widely used methods (some of which we have also covered in class, LDA and PCA) tend to represent the complex input data as vectors which can cause a plethora of problems. First, *curse of dimensionality* may occur since the 'flattened' data is represented by a vector in much higher dimensional space than necessary which can cause that number of dimensions exceeds the number of observations even in larger data sets. Secondly, since LDA is *limited by number of classes in the data* more feature dimensions can be gained by using alternative methods, which allows exploration of more complex relationships between observational points. By reducing the number of dimensions, of the input data *computational complexity* of the algorithm can be reduced.

2 MDA

MDA (Multidimensional Discriminant analysis, also known as DATER - Discriminant Analysis for Tensor Representation) is a supervised dimensionality reduction framework, basically an extension of LDA, meaning that LDA is a special case of DATER where input data is reduced to first order tensors [3]. MDA provides more discriminate power by handling data as higher order tensors and has similar objective as LDA, to find projection objects that will optimally separate observations form different classes, just those objects are mode-specific projections (matrices) and observations and in form of tensors. Tensor is a higher order algebraic form, the generalization on vectors and matrices, or a multidimensional array.

2.1 Notation and tensor algebra

To avoid confusion, it is worthwhile to note that **vectors** are represented by italic lowercase symbols (*v, x, y*), **matrices** as upper case (S, D, Λ) and **Tensors** are denoted by bold uppercase symbols (such as **A, B, C**). A tensor that is n^{th} order (has n dimensions) is denoted as $\boldsymbol{X} \in \mathbb{R}^{m_1,m_2,\ldots,m_n}$. Also some tensor operations that MDA relies on are:

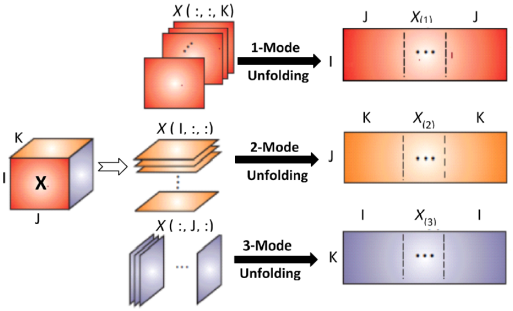


Figure 2.1: K-mode unfolding of a tensor. [2]

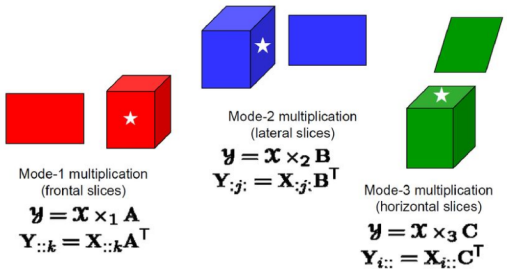


Figure 2.2: K-mode matrix-tensor multiplication. [8]

- **k- mode unfolding** a tensor is a way of decomposition of tensors into matrices (slices) from the k^{th} direction.

$$X_k \in \mathbb{R}^{m_k \times \prod_{i \neq k} m_i} \Leftarrow_k \mathbf{X} \in \mathbb{R}^{m_1 \times m_2 \times \ldots \times m_n}$$
$$F^k_{i_k,j} = \mathbf{X}_{i_1,\ldots,i_n}, j = 1 + \sum_{l=1,l\neq k}^{n} (i_l - 1) \prod_{o=l+1,o\neq k}^{n} m_o \tag{2.1}$$

- **k- mode product** of a tensor with matrix

$$\mathbf{B} = \mathbf{A} \times_k U \tag{2.2}$$
$$\mathbf{B}_{i_1,\ldots,i_k,j,i_{k+1},\ldots,i_n} = \sum_{i=1,l\neq k}^{m_k} \mathbf{A}_{i_1,\ldots,i_k,i,i_{k+1},\ldots,i_n}) * U_{i,j}, j = 1,\ldots,m'_k \tag{2.3}$$

2.2 DTC AND ITERATIVE OPTIMIZATION METHODOLOGY

With the same objective as LDA, MDA optimizes the **Discriminant Tensor Criterion**, which designed to find the n sub-spaces interrelated from all n directions(with n being the order of the data tensor) that will maximize the inter-class to intra-class variance ratio of the data. The inter-class variance is measured by the sum of distances between average tensors by class and tensor of complete sample average, weighted by the number of observations in class. Analogously to LDA, the variance within the class is proportional to summed distances from each observation tensor to to tensor of the corresponding class average. Discriminant tensor criterion:

$$U^*_{k=1,\dots,n} = arg\,max \frac{\sum_c n_c \|\bar{\boldsymbol{X}}_c \times_1 U_1 \dots \times_n U_n - \bar{\boldsymbol{X}} \times_1 U_1 \dots \times_n U_n\|^2}{\sum_i \|\mathbf{X}_i \times_1 U_1 \dots \times_n U_n - \bar{\boldsymbol{X}}_{c_i} \times_1 U_1 \dots \times_n U_n\|^2} \tag{2.4}$$

There is no closed form solution for the DTC and it poses a nonlinear optimization problem that can be solved only with an iterative method.

K-mode optimization is an algorithm used to optimize DTC objective function from one direction at a time. In order to solve 2.4 optimization problem, we have to note that form any direction we have:

$$U^*_k = arg\,max \frac{\sum_c n_c \|\bar{\boldsymbol{X}}_c \times_k U_k - \bar{\boldsymbol{X}} \times_k U_k\|^2}{\sum_i \|\mathbf{X}_i \times_k U_k - \bar{\boldsymbol{X}}_{c_i} \times_k U_k\|^2} \tag{2.5}$$

and:

$$\begin{aligned}
\sum_i \|\mathbf{X}_i \times_k U_k - \bar{\boldsymbol{X}}_{c_i} \times_k U_k\|^2 = \\
\sum_i \|{X_i}^{k^T} U_k - \bar{X}^{k^T}_{c_i} U_k\|^2_F = \\
\sum_i Tr[{U_k}^T (X_i{}^k - \bar{X}^k_{c_i})(X_i{}^k - \bar{X}^k_{c_i})^T U_k] = \\
Tr[{U_k}^T (\sum_i (X_i{}^k - \bar{X}^k_{c_i})(X_i{}^k - \bar{X}^k_{c_i})^T) U_k] = \\
Tr\left[{U_k}^T (\sum_{j=1}^{\prod_{o\neq k} m_o} \sum_i (X_i{}^{k,j} - \bar{X}^{k,j}_{c_i})(X_i{}^{k,j} - \bar{X}^{k,j}_{c_i})^T) U_k\right] = \\
Tr({U_k}^T S_W U_K)
\end{aligned} \tag{2.6}$$

where ${X_i}^{k,j}$ is the j^{th} column of k-mode unfolded matrix of tensor $\boldsymbol{X_i}$ (tensor of observation i) , and analogously $\bar{X}^{k,j}_{c_i}$ and $\bar{X}^{k,j}$ are columns of unfolded tensors of average for the class c and tensor of global average. The derivation 2.6 reduces 2.4 to discriminant analysis problem, solved similarly to traditional LDA method.

As mentioned, k-mode discriminant analysis learn discriminant subspaces by unfolding the tensors and consider column vectors of those matrices as new objects that allow clustering of observations according to those objects. Therefore, the effective sample size is augmented (from the number of observations to number of column vectors of unfolded tensors of observations) which allows increase in efficiency with the small data-sets compared to LDA and avoiding a *small sample problem* [3].

2.3 ALGORITHM

Figure 2.3: The methodology of MDA. [1]

Multilinear Discriminant Analysis:

Given the sample set $\tilde{\mathbf{X}} \in \mathbb{R}^{m_1 \times m_2 \times \ldots \times m_n \times N}$, their class labels $c_i \in \{1, 2, ..., N_c\}$, and the final lower dimensions $m_1' \times m_2' \times \ldots \times m_n'$.

1. Initialize $U_1^0 = I_{m_1}, U_2^0 = I_{m_2}, ..., U_n^0 = I_{m_n}$;
2. For $t = 1, 2, \ldots, T_{max}$ do

a) For $k = 1, 2, \ldots, n$ do

$$\mathbf{Y}_i = \mathbf{X}_i \times_1 U_1^t \ldots \times_{k-1} U_{k-1}^t \times_{k+1} U_{k+1}^{t-1} \ldots \times_n U_n^{t-1}$$

$$Y_i^k \Leftarrow_k \mathbf{Y}_i$$

$$S_B = \sum_{j=1}^{\prod_{o \neq k} m_o} S_B^j, S_B^j = \sum_{c=1}^{N_c} n_c (\bar{Y}_c^{k,j} - \bar{Y}^{k,j})(\bar{Y}_c^{k,j} - \bar{Y}^{k,j})^{\mathrm{T}}$$

$$S_W = \sum_{j=1}^{\prod_{o \neq k} m_o} S_W^j, S_W^j = \sum_{i=1}^{N} (Y_i^{k,j} - \bar{Y}_{c_i}^{k,j})(Y_i^{k,j} - \bar{Y}_{c_i}^{k,j})^{\mathrm{T}}$$

$$S_B U_k^t = S_W U_k^t \Lambda_k, \quad U_k^t \in \mathbb{R}^{m_k \times m_k'}$$

b) If $t > 2$ and $\| U_k^t - U_k^{t-1} \| < m_k' m_k \varepsilon, k = 1, .., n$, break;

3. Output the projections $U_k = U_k^t \in \mathbb{R}^{m_k \times m_k'}, k = 1, ..., n$.

Algorithm given in Figure 2.12 [1] shows us the methodology and guidelines for the software implementation. The number of iterations can be set arbitrarily (tmax = 1000, 100000 ...)or the optimization process may be stopped when projection matrices from two consecutive steps are similar enough (the algorithm converges). After executing on given data set, we will end up with the set of projection matrices, and with them and reduced to a new lower dimensions $m_1', m_2' ... m_n'$ so that $U_k = U_k^t \in \mathbb{R}^{m_k \times m_k'}$,the projections of data onto the lower dimensional subspace are computed by

$$\boldsymbol{Z_i} = \boldsymbol{X_i} \times_1 U_1 \ldots \times_n U_n \tag{2.7}$$

and tensor Z_i is unfolded by k-mode methodology to a scalar, or vector of appropriate size for graphical representation.

2.4 EXPERIMENTS

The algorithm of MDA given in Figure 2.12 was implemented in the programming language Python with helper functions for k-mean product and unfolding from the library TensorLy [6].

2.5 DATA

The data-set consists out of 566 images of Electrical Engineering students generation 2005, University of Belgrade. Not all images in data-set are in color, and since those images would

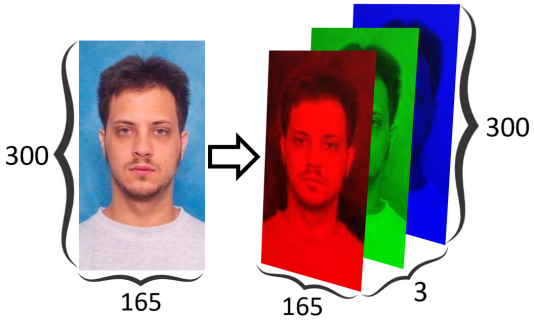


Figure 2.4: Decomposition of the color image to a tensor.

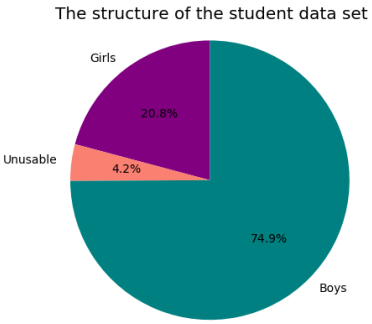


Figure 2.5: The structure of the data.

not be useful for the illustration of the MDA methodology, the subset of the sample that consist only color images is extracted. Also, in order to further reduce dimensions of the data, images were cropped lengthwise. After extraction of unfit images we are left with 542 images out of which 118 are girls, 424 boys and length of the images is shortened from 300 pixels to 230 pixels. Further, in order to balance the classes and avoid long processing times and miss-classification the data is further subset to contain same number of observations in both classes, 118 girls and 118 boys. The subset of boys was picked at random out of initial 424 images. Each image is decomposed into 3 color channels RGB, therefore, each image is displayed as a tensor and the whole data set becomes 4th order tensor with dimensions (542, 230, 165, 3).

Figure 2.6: The average girl.

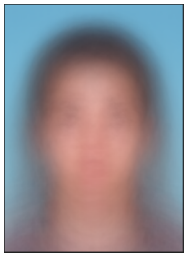


Figure 2.7: The average.

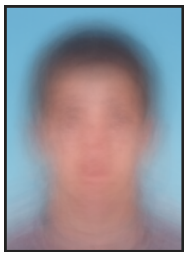


Figure 2.8: The average boy.

The average of both classes and average of the total data-set was computed and MDA analysis was conducted. The dimensionality is reduced to 1 so that it is comparable to the LDA method which is restricted by number of classes. The MDA algorithm implemented as shown in Appendix proved to be computationally demanding with much longer times of execution than SkLearn LDA implementation(over 10 times longer). The visual separation of the data is not impressively better (Figures 2.9 and 2.10). The data reduced by both

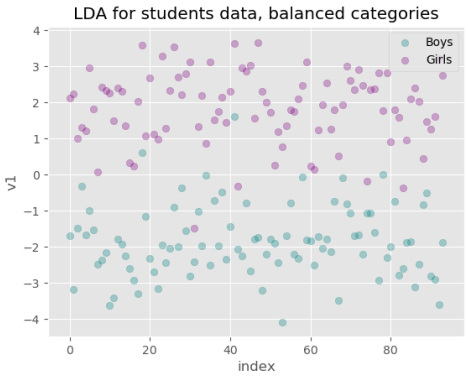


Figure 2.9: LDA.

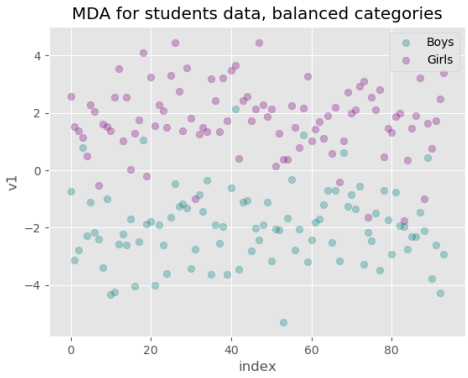
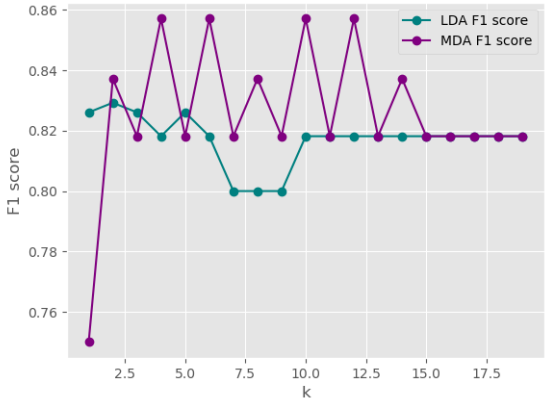


Figure 2.10: MDA.

methodologies was subjected to KNN classifying algorithm for multiple different values of hyper-parameter k. It is worth mentioning that the reduced data by either method seem to be linearly separable and it can be assumed that linear classifiers (perceptron, SVM...) would perform reasonably well. For the both methods the data was divided into train and test sets after dimensionality reduction, at random and KNN classifiers for both reduction methods proved to have very similar F1 [7] scores (Figure 2.11).

Figure 2.11: F1 scores for k= 1, ..., 20.



2.6 MNIST DATA

Another experiment was conducted with MNIST dataset inspired by the experiment from the original paper on the ORL faces data-set [1]. The 30 images for each class digits 0-3 were picked at random and concatenated to form tensors with the dimensions (28, 28, 3). The purpose of this experiment is to explore efficiency of MDA method with small data-sets compared to LDA. The total data-set is represented by a tensor (40, 28, 28, 3). The main

Figure 2.12: The ORL data example. [1]

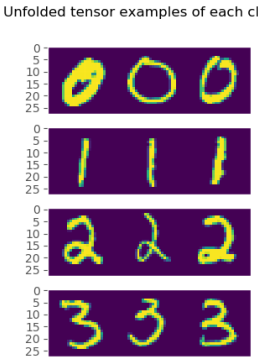


Figure 2.13: The example of unfolded tensor for MNIST data.

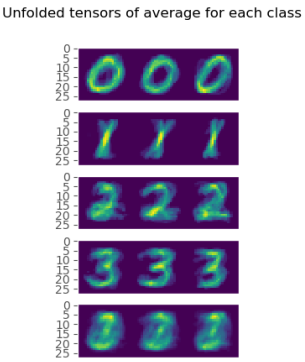


Figure 2.14: The unfolded tensor averages.

idea behind this choice of sample is to provide similar data to the ORL set, but smaller in dimension. LDA provided reasonable separation in both two and three directions (Figure 2.12 and 2.13).

The MDA analysis was conducted by utilizing the same Python implementation of the Figure 2.3 methodology. However, the intra-class scatter matrix proved to be singular for many different randomly selected samples, therefore no eigenvectors could be computed and iterative method was not feasible. Theoretically [3] the singularity problem should be far less likely with MDA compared to LDA since the intra-class scatter matrices computed step-wise are much smaller in dimension than one produced by LDA.

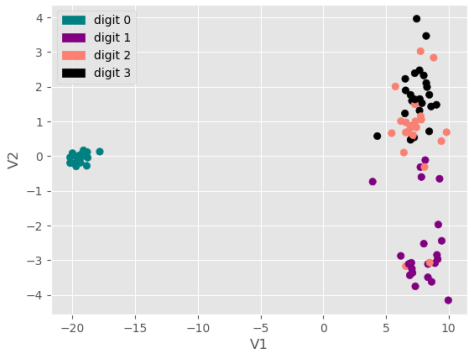


Figure 2.15: 2D LDA for MNIST data subset.

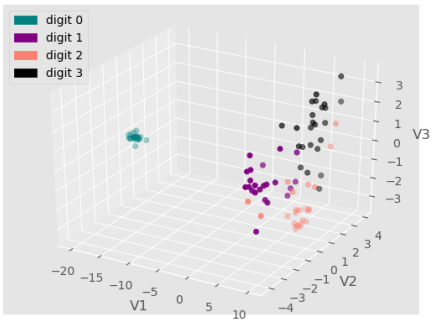


Figure 2.16: 3D LDA for MNIST subset.

2.7 Discussion of the results and conclusion

The more general MDA as the dimensionality reduction technique theoretically should provide the wast array of improvements compared to its special case LDA method.
The substandard performance of the methodology in two experiments conducted in this project might be the consequence due to:

- Data in the Student data-set was easily linearly separable, therefore MDA 'picked up' the same directions as LDA.
- There was not enough variability in the color channels in the Student data-set. The differences between channels were not pronounced. The assumption is that this method would perform much better with imagery data-sets in which the color is distinctive feature of the classes, like geospatial imagery, MRI data, crop imagery for disease detection etc.
- The coding. LDA function is a part of the SkLearn library and MDA algorithm was implemented by a student which is the main cause of the computational complexity and long execution times.
- The complexity of the data-sets for the experiments in this project is inherently restricted by computational power of the personal computer.

It would be interesting to implement more optimized MDA algorithm and to assess the performance on the more complicated data-sets where the advantages of the methodology could be undoubtedly confirmed and explored.

REFERENCES

[1] S. Yan,D. Xu, Q. Yang,L. Zhang, X. Tang,H. Zhang *Multilinear Discriminant Analysis for Face Recognition*. IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 16, NO. 1, JANUARY 2007.

[2] M. Bessaoudi, M. Belahcene, A. Ouamane, A. Chouchane, S. Bourennane *Multilinear Enhanced Fisher Discriminant Analysis for robust multimodal 2D and 3D face verification*. Article in Applied Intelligence · April 2019 DOI: 10.1007/s10489-018-1318-8

[3] S. Yan, D. Xu, Q. Yang, L. Zhang, X.Tang, H. Zhang *Discriminant Analysis with Tensor Representation*. Conference Paper · July 2005, DOI: 10.1109/CVPR.2005.131 · Source: IEEE Xplore.

[4] J. Huang, K. Su, J. El-Den,T. Hu, and J. Li *An MPCA/LDA Based Dimensionality Reduction Algorithm for Face Recognition*. Mathematical Problems in Engineering Volume 2014, Article ID 393265

[5] M.Deypir, R.Boostani, T. Zoughi *Ensemble based multi-linear discriminant analysis with boosting*. Scientia Iranica D (2012) 19(3), 654-661

[6] TensorLy, `http://tensorly.org/stable/user_guide/tensor_basics.html`

[7] SkLearn, F1 score, `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html`

[8] K-mode product ilustration, `https://slideplayer.com/slide/14566967/90/images/8/Pictorial+view+of+Tensor+Multiplication.jpg`

Appendix

Implementation of the MDA algorithm Figure 2.3.

```
for _ in range(1000):
    for k in range(1, 4):
        y_bar_b_k = tl.unfold(average_boy, mode=k-1)
        # print(y_bar_b_k.shape)
        y_bar_g_k = tl.unfold(average_boy, mode=k-1)
        # print(y_bar_g_k.shape)
        y_bar_k = tl.unfold(average, mode=k-1)
        # print(y_bar_k.shape)
        # print(int((230*165*3)/data.shape[k]))
        s_b, s_w = np.zeros((data.shape[k], data.shape[k])),
        np.zeros((data.shape[k], data.shape[k]))
        for j in range(int((230*165*3)/data.shape[k])):
            # print(data.shape[k])
            b = y_bar_b_k[:, j] - y_bar_k[:, j]
            b.shape = (len(b), 1)
            g = y_bar_g_k[:, j] - y_bar_k[:, j]
            g.shape = (len(g), 1)
            s_b_j = n_b*np.dot(b, b.T) + n_g*np.dot(g, g.T)
            s_b += s_b_j
            for i in range(0, len(labels)):
                x_i = data[i, :, :, :].reshape((230, 165, 3))
                y_i = tl.tenalg.mode_dot(tl.tenalg.mode_dot
                (tl.tenalg.mode_dot
                (x_i, u_k[0], mode=0),
                 u_k[1], mode=1), u_k[2], mode=2)
                y_k = tl.unfold(y_i, mode=k-1)
                if labels[i] == 1:
                    b = y_k[:, j] - y_bar_b_k[:, j]
                    b.shape = (len(b), 1)
                    s_w_j = np.dot(b, b.T)
                else:
                    g = y_k[:, j] - y_bar_g_k[:, j]
                    g.shape = (len(g), 1)
                    s_w_j = np.dot(g, g.T)
                s_w += s_w_j
        eigenval, eigenvec = scipy.linalg.eigh(s_b, s_w)
        u_k[k] = eigenvec
filename = str('output_%d.txt' %k)
pd.DataFrame(u_k[k]).to_csv(filename)
```

The whole experiment 1

```
from PIL import Image
import matplotlib.pyplot as plt
import tensorly as tl
from tensorly import tenalg
import numpy as np
import glob
import time
import os
import scipy.linalg
import pandas as pd

# image = np.array(Image.open(r"C:\Users...\crop_465.jpg"))
# b, g, r = image[:, :, 0], image[:, :, 1], image[:, :, 2]
# plt.imshow(g)
# plt.show()

data_dir = os.path.join(os.getcwd(), "data")
# im_path = os.path.join(data_dir, "crop_465.jpg")
# im = np.array(Image.open(im_path))
# print(im.shape) # shape (230, 165, 3), type np.array - OK

def unfold(image):
#     mode_1_unfolding = np.concatenate(image[:, :, 0], image[:, :, 1], image[:, :, 2
#     mode_2_unfolding = image[:, 0, :]
#     for i in range(1, image.shape[1]):
#         mode_2_unfolding = np.concatenate(mode_2_unfolding, image[:, i, :])
#     mode_3_unfolding = image[0, :, :]
#     for i in range(1, image.shape[0]):
#         mode_3_unfolding = np.concatenate(mode_3_unfolding, image[i, :, :])
#     return mode_1_unfolding, mode_2_unfolding, mode_3_unfolding


# im_R = im.copy()
# im_R[:, :, (1, 2)] = 0
# im_G = im.copy()
# im_G[:, :, (0, 2)] = 0
# im_B = im.copy()
# im_B[:, :, (0, 1)] = 0

filelist = sorted(glob.glob(os.path.join(data_dir, "*.jpg")))
# print(filelist)
grayscale_sephia = [60, 88, 125, 129, 135, 169, 192, 242, 245, 249, 265, 277,
                    286, 361, 354, 366, 426, 442, 459, 469, 470, 486, 557, 559]
for index in sorted(grayscale_sephia, reverse=True):
```

```
    del filelist[index]

data = np.array([np.array(Image.open(fname)) for fname in filelist])  # shape (542, 2
# print(type(data[0]))
# print(data.shape)
labels = list(np.genfromtxt(os.path.join(data_dir, 'labels.txt'), delimiter=','))
for index in sorted(grayscale_sephia, reverse=True):
    del labels[index]

mask_girls = [x == 1 for x in labels]
# print(mask_girls)
mask_boys = [not i for i in mask_girls]
# print(mask_boys)
girls = data[mask_girls, :, :, :]  # shape (118, 230, 165, 3)
# print(girls.shape)
boys = data[mask_boys, :, :, :]  # shape (424, 230, 165, 3)
# print(boys.shape)


def average_image(a_lot_of_images):
    avg = np.zeros((a_lot_of_images.shape[1], a_lot_of_images.shape[2], a_lot_of_imag
    for i in range(a_lot_of_images.shape[0]):
        avg[:, :, 0] += a_lot_of_images[i, :, :, 0]
        avg[:, :, 1] += a_lot_of_images[i, :, :, 1]
        avg[:, :, 2] += a_lot_of_images[i, :, :, 2]
    for i in [0,1,2]:
        avg[:, :, i] = avg[:, :, i]/a_lot_of_images.shape[0]
    avg = np.array(np.round(avg), dtype=np.uint8)
    return avg


average_girl = average_image(girls)
# average_girl = Image.fromarray(average_girl, 'RGB')
# average_girl.show()
average_boy = average_image(boys)
# average_boy = Image.fromarray(average_boy, 'RGB')
# average_boy.show()
average = average_image(data)
# average = Image.fromarray(average, 'RGB')
# average.show()

#
# # MDA algorithm
# x_b = boys
```

```
# x_g = girls
# n_b = 424
# n_g = 118
u_k = [np.identity(data.shape[1]), np.identity(data.shape[2]), np.identity(data.shape
# # pick a  number of iteration t_max
#
for _ in range(1000):
    for k in range(1, 4):
        y_bar_b_k = tl.unfold(average_boy, mode=k-1)
        # print(y_bar_b_k.shape)
        y_bar_g_k = tl.unfold(average_boy, mode=k-1)
        # print(y_bar_g_k.shape)
        y_bar_k = tl.unfold(average, mode=k-1)
        # print(y_bar_k.shape)
        # print(int((230*165*3)/data.shape[k]))
        s_b, s_w = np.zeros((data.shape[k], data.shape[k])), np.zeros((data.shape[k],
        for j in range(int((230*165*3)/data.shape[k])):
            # print(data.shape[k])
            b = y_bar_b_k[:, j] - y_bar_k[:, j]
            b.shape = (len(b), 1)
            g = y_bar_g_k[:, j] - y_bar_k[:, j]
            g.shape = (len(g), 1)
            s_b_j = n_b*np.dot(b, b.T) + n_g*np.dot(g, g.T)
            s_b += s_b_j
            for i in range(0, len(labels)):
                x_i = data[i, :, :, :].reshape((230, 165, 3))
                y_i = tl.tenalg.mode_dot(tl.tenalg.mode_dot(tl.tenalg.mode_dot(x_i, u
                y_k = tl.unfold(y_i, mode=k-1)
                if labels[i] == 1:
                    b = y_k[:, j] - y_bar_b_k[:, j]
                    b.shape = (len(b), 1)
                    s_w_j = np.dot(b, b.T)
                else:
                    g = y_k[:, j] - y_bar_g_k[:, j]
                    g.shape = (len(g), 1)
                    s_w_j = np.dot(g, g.T)
                s_w += s_w_j
        eigenval, eigenvec = scipy.linalg.eigh(s_b, s_w)
        u_k[k] = eigenvec
filename = str('output_%d.txt' %k)
pd.DataFrame(u_k[k]).to_csv(filename)
```

lda for experiment 1

```
from PIL import Image
import random
import numpy as np
import pandas as pd
import glob
import os
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt
plt.style.use('ggplot')

data_dir = os.path.join(os.getcwd(), "girls")
filelist_g = sorted(glob.glob(os.path.join(data_dir, "*.jpg")))
# grayscale_sephia = [60, 88, 125, 129, 135, 169, 192, 242, 245, 249, 265, 277,
                    # 286, 361, 354, 366, 426, 442, 459, 469, 470, 486, 557, 559]
# for index in sorted(grayscale_sephia, reverse=True):
#     del filelist[index]
# labels = list(np.genfromtxt(os.path.join(data_dir, 'labels.txt'), delimiter=','))
# for index in sorted(grayscale_sephia, reverse=True):
#     del labels[index]
data_dir = os.path.join(os.getcwd(), "boys")
filelist = sorted(glob.glob(os.path.join(data_dir, "*.jpg")))
filelist_b = random.sample(filelist, 118)

data = []
for fname in filelist_g:
    im = np.array(Image.open(fname))
    # print(im.shape)
    im_concat = np.concatenate((im[:,:,[0]], im[:,:,[1]], im[:,:,[2]]), 1).flatten()
    # print(im_concat.shape)
    data.append(im_concat)
# print(type(x), x.shape)
for fname in filelist_b:
    im = np.array(Image.open(fname))
    # print(im.shape)
    im_concat = np.concatenate((im[:,:,[0]], im[:,:,[1]], im[:,:,[2]]), 1).flatten()
    # print(im_concat.shape)
    data.append(im_concat)
x = np.array(data)
```

```
labels = ['1']*len(filelist_g) + ['0']*len(filelist_b)

# print(len(data), len(labels))


xTrain, xTest, yTrain, yTest = train_test_split(x, labels, test_size=0.2)
# print(len(xTrain), len(yTrain), yTrain)

lda = LinearDiscriminantAnalysis(n_components=1)
lda.fit(xTrain, yTrain)
Y_lda = lda.transform(xTrain)
xTest = lda.transform(xTest)

Y_mda = list(pd.read_csv('mda_ee_students.csv'))
# print(Y_mda)
print(type(Y_lda), len(Y_lda), len(yTrain))
print(type(Y_mda), len(Y_mda), len(yTrain))

g, b = [], []
for i in range(0, len(yTrain)):
    if yTrain[i] == '1':
        g.append(np.asarray(Y_lda[i]))
    elif yTrain[i] == '0':
        b.append(np.asarray(Y_lda[i]))
n_g, n_b = len(g), len(b)
# print(n_g, n_b, g,b)

# plt.scatter(range(n_b), b, c='teal', label='Boys', alpha=0.3)
# plt.scatter(range(n_g), g, c='purple', label='Girls', alpha=0.3)
# plt.legend(loc='upper right')
# plt.title('LDA for students data, balanced categories')
# plt.xlabel('index')
# plt.ylabel('v1')
# plt.show()
#
# g, b = [], []
# for i in range(0, len(yTrain)):
#     if yTrain[i] == '1':
#         g.append(np.asarray(Y_mda[i]))
#     elif yTrain[i] == '0':
#         b.append(np.asarray(Y_mda[i]))
# n_g, n_b = len(g), len(b)
# # print(n_g, n_b, g ,b)
#
```

```
# plt.scatter(range(n_b), b, c='teal', label='Boys', alpha=0.3)
# plt.scatter(range(n_g), g, c='purple', label='Girls', alpha=0.3)
# plt.legend(loc='upper right')
# plt.title('MDA for students data, balanced categories')
# plt.xlabel('index')
# plt.ylabel('v1')
# plt.show()


################ knn
f1_lda, f1_mda = [], []
for k in range(1, 20):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(Y_lda, yTrain)
    predicted = model.predict(xTest)
    f1_lda.append(f1_score(list(yTest), predicted, pos_label='1'))
# print(f1_lda)

for k in range(1, 20):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(Y_mda, yTrain)
    predicted = model.predict(xTest)
    f1_mda.append(f1_score(list(yTest), predicted, pos_label='1'))


plt.plot(range(1,20), f1_lda,  marker='o', color='teal', label='LDA F1 score')
plt.plot(range(1,20), f1_mda,   marker='o', color='purple', label='MDA F1 score')
plt.legend()
plt.xlabel('k')
plt.ylabel('F1 score')
plt.show()

# labels = 'Girls', 'Unusable', 'Boys'
# sizes = [118, 24, 424]
# colors = 'purple', 'salmon', 'teal'
# fig1, ax1 = plt.subplots()
# ax1.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)
# ax1.axis('equal')
# plt.title('The structure of the student data set')
# plt.show()
```

The MNIST

```
from mnist import MNIST
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import os
import tensorly as tl
import scipy.linalg
from tensorly import tenalg
import random
plt.style.use('ggplot')


mndata = MNIST(os.getcwd())
images, labels = mndata.load_training()
# print(len(images[0]))
# print(len(labels))
images_test, labels_test = mndata.load_testing()

# print(len(images))
for i in images_test:
    images.append(i)
# print(len(images))

# print(len(labels))
for i in labels_test:
    labels.append(i)

a_0, a_1, a_2, a_3 = ([] for i in range(4))

for i in range(0, len(images)):
    if labels[i] == 0:
        a_0.append(np.asarray(images[i]).reshape(28, 28))
    elif labels[i] == 1:
        a_1.append(np.asarray(images[i]).reshape(28, 28))
    elif labels[i] == 2:
        a_2.append(np.asarray(images[i]).reshape(28, 28))
    elif labels[i] == 3:
        a_3.append(np.asarray(images[i]).reshape(28, 28))

############### make tensor out of digits 0, 1, 2, 3
def chunks(lst, n):
    for i in range(0, len(lst), n):
        yield lst[i:i + n]


def tensorfy(lst, n):
```

```
    b = list(chunks(lst, n))
    x = []
    for i in range(0, len(b)):
        x.append(np.stack(b[i], axis=2))
    x = np.stack(x)
    return x


################## pick digit images at random
sample_0 = random.sample(range(0, len(a_0)), 30)
sample_1 = random.sample(range(0, len(a_1)), 30)
sample_2 = random.sample(range(0, len(a_2)), 30)
sample_3 = random.sample(range(0, len(a_3)), 30)

zero, one, two, three = [], [], [], []
for i in sample_0:
    zero.append(a_0[i])
for i in sample_1:
    one.append(a_1[i])
for i in sample_2:
    two.append(a_2[i])
for i in sample_3:
    three.append(a_3[i])

############################## plot examples
# a_0_10, a_1_10, a_2_10, a_3_10 = zero[0].reshape(28, 28), one[0].reshape(28, 28), t
# for i in range(1, 3):
#     a_0_10 = np.concatenate((a_0_10, zero[i].reshape(28, 28)), axis=1)
#     a_1_10 = np.concatenate((a_1_10, one[i].reshape(28, 28)), axis=1)
#     a_2_10 = np.concatenate((a_2_10, two[i].reshape(28, 28)), axis=1)
#     a_3_10 = np.concatenate((a_3_10, three[i].reshape(28, 28)), axis=1)
#
# fig, axs = plt.subplots(4)
# fig.suptitle('Unfolded tensor examples of each class')
# axs[0].matshow(a_0_10)
# axs[1].matshow(a_1_10)
# axs[2].matshow(a_2_10)
# axs[3].matshow(a_3_10)
# for ax in axs:
#     ax.set_xticks([])
#     ax.grid(False)
# plt.show()
```

```
tensor_0 = tensorfy(zero, 3)
tensor_1 = tensorfy(one, 3)
tensor_2 = tensorfy(two, 3)
tensor_3 = tensorfy(three, 3)
# print(tensor_0.shape, tensor_1.shape, tensor_2.shape, tensor_3.shape)

data = np.concatenate((tensor_0, tensor_1, tensor_2, tensor_3), axis=0)
# print(data.shape)
labels = [0]*10 + [1]*10 + [2]*10 + [3]*10
# print(tensor_0.shape, data.shape)


def average_image(a_lot_of_images):
    avg = np.zeros((a_lot_of_images.shape[1], a_lot_of_images.shape[2], a_lot_of_imag
    for i in range(a_lot_of_images.shape[0]):
        avg[:, :, 0] += a_lot_of_images[i, :, :, 0]
        avg[:, :, 1] += a_lot_of_images[i, :, :, 1]
        avg[:, :, 2] += a_lot_of_images[i, :, :, 2]
    for i in [0,1,2]:
        avg[:, :, i] = avg[:, :, i]/a_lot_of_images.shape[0]
    avg = np.array(np.round(avg), dtype=np.uint8)
    return avg


average_0 = average_image(tensor_0)
average_1 = average_image(tensor_1)
average_2 = average_image(tensor_2)
average_3 = average_image(tensor_3)
average = average_image(data)
################ plot averages
# plot_0 = np.concatenate((average_0[:,:,0], average_0[:,:,1], average_0[:,:,2]), axi
# plot_1 = np.concatenate((average_1[:,:,0], average_1[:,:,1], average_1[:,:,2]), axi
# plot_2 = np.concatenate((average_2[:,:,0], average_2[:,:,1], average_2[:,:,2]), axi
# plot_3 = np.concatenate((average_3[:,:,0], average_3[:,:,1], average_3[:,:,2]), axi
# plot_all = np.concatenate((average[:,:,0], average[:,:,1], average[:,:,2]), axis=1)
#
# # fig, axs = plt.subplots(5)
# # fig.suptitle('Unfolded tensors of average for each class')
# # axs[0].matshow(plot_0)
# # axs[1].matshow(plot_1)
# # axs[2].matshow(plot_2)
# # axs[3].matshow(plot_3)
# # axs[4].matshow(plot_all)
# # for ax in axs:
```

```
# #     ax.set_xticks([])
# #     ax.grid(False)
# # plt.show()

u_k = [np.identity(data.shape[1]), np.identity(data.shape[2]), np.identity(data.shape

for _ in range(1000):
    for k in range(1, 4):
        y_bar_0_k = tl.unfold(average_0, mode=k-1)
        y_bar_1_k = tl.unfold(average_1, mode=k-1)
        y_bar_2_k = tl.unfold(average_2, mode=k-1)
        y_bar_3_k = tl.unfold(average_3, mode=k-1)
        y_bar_k = tl.unfold(average, mode=k-1)
        # print(int((230*165*3)/data.shape[k]))
        s_b, s_w = np.zeros((data.shape[k], data.shape[k])), np.zeros((data.shape[k],
        for j in range(int((28*28*3)/data.shape[k])):
            # print(data.shape[k])
            yj0 = y_bar_0_k[:, j] - y_bar_k[:, j]
            yj0.shape = (len(yj0), 1)
            yj1 = y_bar_1_k[:, j] - y_bar_k[:, j]
            yj1.shape = (len(yj1), 1)
            yj2 = y_bar_2_k[:, j] - y_bar_k[:, j]
            yj2.shape = (len(yj2), 1)
            yj3 = y_bar_3_k[:, j] - y_bar_k[:, j]
            yj3.shape = (len(yj3), 1)
            s_b_j = 10*np.dot(yj0, yj0.T) + 10*np.dot(yj1, yj1.T) + 10*np.dot(yj2, yj
            s_b += s_b_j
            for i in range(0, len(labels)):
                x_i = data[i, :, :, :].reshape((28, 28, 3))
                y_i = tl.tenalg.mode_dot(tl.tenalg.mode_dot(tl.tenalg.mode_dot(x_i, u
                y_k = tl.unfold(y_i, mode=k-1)
                if labels[i] == 0:
                    b = y_k[:, j] - y_bar_0_k[:, j]
                    b.shape = (len(b), 1)
                    s_w_j = np.dot(b, b.T)
                elif labels[i] == 1:
                    b = y_k[:, j] - y_bar_1_k[:, j]
                    b.shape = (len(b), 1)
                    s_w_j = np.dot(b, b.T)
                elif labels[i] == 2:
                    b = y_k[:, j] - y_bar_2_k[:, j]
                    b.shape = (len(b), 1)
                    s_w_j = np.dot(b, b.T)
                else:
```

```
                    b = y_k[:, j] - y_bar_3_k[:, j]
                    b.shape = (len(b), 1)
                    s_w_j = np.dot(b, b.T)
                s_w += s_w_j
        eigenval, eigenvec = scipy.linalg.eigh(s_b, s_w)
        u_k[k] = eigenvec
```

The MNIST LDA

```
from mnist import MNIST
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import os
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from mpl_toolkits.mplot3d import Axes3D
import random
plt.style.use('ggplot')


mndata = MNIST(os.getcwd())
images, labels = mndata.load_training()
# print(len(images[0]))
# print(len(labels))
images_test, labels_test = mndata.load_testing()

# print(len(images))
for i in images_test:
    images.append(i)
# print(len(images))

# print(len(labels))
for i in labels_test:
    labels.append(i)

a_0, a_1, a_2, a_3 = ([] for i in range(4))

for i in range(0, len(images)):
    if labels[i] == 0:
        a_0.append(np.asarray(images[i]))
    elif labels[i] == 1:
        a_1.append(np.asarray(images[i]))
    elif labels[i] == 2:
        a_2.append(np.asarray(images[i]))
```

```
    elif labels[i] == 3:
        a_3.append(np.asarray(images[i]))


################## pick digit images at random
sample_0 = random.sample(range(0, len(a_0)), 30)
sample_1 = random.sample(range(0, len(a_1)), 30)
sample_2 = random.sample(range(0, len(a_2)), 30)
sample_3 = random.sample(range(0, len(a_3)), 30)

data = []
for i in sample_0:
    data.append(a_0[i])
for i in sample_1:
    data.append(a_1[i])
for i in sample_2:
    data.append(a_1[i])
for i in sample_3:
    data.append(a_1[i])
labels = [0]*30 + [1]*30 + [2]*30 + [3]*30
print(len(data), len(data[0]), len(labels))

xTrain, xTest, yTrain, yTest = train_test_split(data, labels, test_size=0.2)
# print(len(xTrain), len(yTrain), yTrain)


###################### two component LDA
lda_2 = LinearDiscriminantAnalysis(n_components=2)
lda_2.fit(xTrain, yTrain)
Y = lda_2.transform(xTrain)
print(Y.shape)

colors = ['teal', 'purple', 'salmon', 'black']
col = [colors[i] for i in yTrain]
p0 = mpatches.Patch(color='teal', label='digit 0')
p1 = mpatches.Patch(color='purple', label='digit 1')
p2 = mpatches.Patch(color='salmon', label='digit 2')
p3 = mpatches.Patch(color='black', label='digit 3')

plt.scatter(Y[:,0], Y[:,1], c=col)
plt.xlabel('V1')
plt.ylabel('V2')
plt.legend(handles=[p0, p1, p2, p3])
plt.show()
```

```
lda_3 = LinearDiscriminantAnalysis(n_components=3)
lda_3.fit(xTrain, yTrain)
Y = lda_3.transform(xTrain)


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(Y[:,0], Y[:,1], Y[:,2], c=col, marker='o')
ax.set_xlabel('V1')
ax.set_ylabel('V2')
ax.set_zlabel('V3')
plt.legend(handles=[p0, p1, p2, p3])
plt.show()
```