ML_HW05_0760406_楊潔生

Dimension Reductions with MNIST

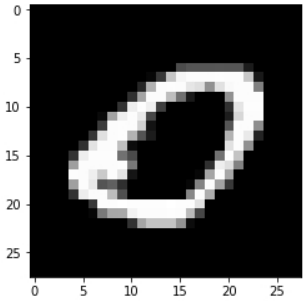
I. Different ways to do dimension reductions (60% in total):
In this assignment, you need to use different ways to do dimension reduction on **mnist_X.csv** and **mnist_label.csv**, including PCA, LDA, S-SNE and T-SNE.

Using Library

```
import numpy as np
from matplotlib import pyplot as plt
```

Read Data

```
mnist = np.genfromtxt('mnist_X.csv', delimiter=',') # mnist_X = 5000*784, ML
mLabel = np.genfromtxt('mnist_label.csv', delimiter=',') # label = 5000*1 label:1-5 = number 0-4
```



Label 1 = number 0

General function – pltshow (visualize)

```
def pltshow(Y, mLabel, fname, xlabel, ylabel, title):
    plt.style.use('ggplot')
    plt.figure(figsize=(8, 6))
    for lab, col in zip(range(5), ('b', 'lightcoral', 'turquoise','violet','gold')):
        plt.scatter(Y[mLabel==(lab+1), 0], Y[mLabel==(lab+1), 1], label=lab, s=20, c=col,alpha=0.8)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.legend(loc='lower left')
    plt.savefig(fname)
    plt.show()
```

- Y: Array of PCA / LDA / S-SNE / T-SNE.
- mLabel: MNIST label, such as 1-5.
- fname: Save the filename of the image.
- Xlabel: The xlabel name of the picture.
- Ylabel: The ylabel name of the picture.
- Title: The title of the picture.

PCA

A. PCA (15%)

Use PCA to project all your data mnist_X.csv onto 2D space and mark the data points into different colors. The color of the data points depends on which cluster it belongs to (mnist_label.csv).

```
def PCA(mnist):
    # Covariance matrix : [(X-meanx)^(T)*(X-meanx)]/(n-1)
    meanVec = np.mean(mnist, axis=0)
    covar = np.dot(((mnist-meanVec).T),(mnist-meanVec)) / (len(mnist)-1)

    #Sort - decrease
    eigVal, eigVec = np.linalg.eig(covar) # eigVal:1*784 ; eigVec:784*784
    eigPair = [(np.abs(eigVal[i]), eigVec[:,i]) for i in range(len(eigVal))] # eigPair=(eigenvalue, eigenvector)
    eigPair.sort(key=lambda x: x[0], reverse=True)
    W = np.hstack((eigPair[0][1].reshape(len(mnist[0]),1), eigPair[1][1].reshape(len(mnist[0]),1)))

    # Projection Matrix
    Y = np.dot(mnist, W)
    return Y
```

```
YPCA=PCA(mnist)
pltshow(YPCA, mLabel, 'PCA.png', 'PCA 1', 'PCA 2','PCA')
```

First step

Computing the mean data matrix: $\text{meanVec} = \frac{\sum pixel_i}{2}$, dimension = 5000*1

Computing the covariance matrix: $\text{covar} = \frac{((X-\bar{X})^T(X-\bar{X}))}{n-1}$

Second step

Eigendecomposition on the covariance matrix by using *np.linalg.eig()*

Make a list of (eigenvalue, eigenvector) tuples.

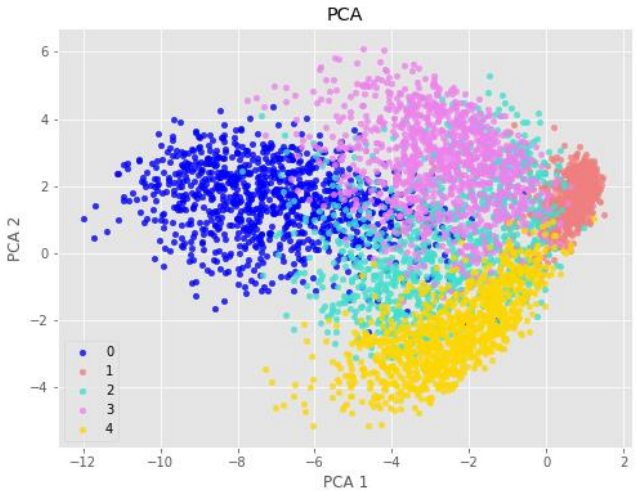
Sort the (eigenvalue, eigenvector) tuples from high to low.

Computing a projection matrix: W

Third step

Projection onto the new feature space: $Y = X \times W$

Visualize



LDA

B. LDA (15%)

Use LDA to project all your data mnist_X.csv onto 2D space and mark the data points into different colors. The color of the data points depends on which cluster it belongs to (mnist_label.csv).

```
def LDA(mnist):
    # mean vector for each class
    meanVector = []
    for cl in range(5):
        meanVector.append(np.mean(mnist[mLabel==(cl+1)], axis=0))

    # meanv: (1/n)*sum(xk); k=0-4
    # scatter: sum((X-meanv)*(X-meanv)^(T)) ; scatter matrix for every class
    # within-class scatter: Sw=sum(scatter)
    Sw = np.zeros((len(mnist[0]),len(mnist[0])))
    for cl,mv in zip(range(5), meanVector):
        scatter = np.zeros((len(mnist[0]),len(mnist[0])))
        for row in mnist[mLabel==(cl+1)]:
            row, mv = row.reshape(len(mnist[0]),1), mv.reshape(len(mnist[0]),1)
            scatter += (row-mv).dot((row-mv).T)
        Sw += scatter

    # between-class scatter: Sb=sum(N*(meanv-overallmean)*(meanv-overallmean)^(T))
    overallM = np.mean(mnist, axis=0)
    SB = np.zeros((len(mnist[0]),len(mnist[0])))
    for i,meanVec in enumerate(meanVector):
        n = mnist[mLabel==i+1,:].shape[0]
        meanVec = meanVec.reshape(len(mnist[0]),1) # make column vector
        overallM = overallM.reshape(len(mnist[0]),1) # make column vector
        SB += n *np.dot((meanVec-overallM),(meanVec-overallM).T)

    # linear discriminants Sw^(-1)*SB
    eigVal, eigVec = np.linalg.eig(np.linalg.pinv(Sw).dot(SB))

    #Sort - decrease
    # eigPair=(eigenvalue, eigenvector)
    eigPair = [(np.abs(eigVal[i]), eigVec[:,i]) for i in range(len(eigVal))]
    eigPair = sorted(eigPair, key=lambda k: k[0], reverse=True)
    W = np.hstack((eigPair[0][1].reshape(len(mnist[0]),1), eigPair[1][1].reshape(len(mnist[0]),1)))

    # Projection Matrix
    Y = np.dot(mnist, W)
    return Y
```

```
YLDA=LDA(mnist)
pltshow(YLDA, mLabel, 'LDA.png', 'LDA 1', 'LDA 2','LDA')
```

First step

Computing the mean vector: $meanVector_i = \begin{bmatrix} \mu_{class1} \\ \mu_{class2} \\ \mu_{class3} \\ \mu_{class4} \\ \mu_{class5} \end{bmatrix}$, $i = 1 - 5$

Second step

Computing the mean vector: $\frac{1}{n_i}\sum_{x\in D_i}^{n} x_k$

Computing the scaling scatter matrix for each class: $S_i = \frac{1}{N_i - 1}\sum_{x\in D_i}^{n}(x - m_i)(x - m_i)^T$

Computing the scatter matrix: $S_W = \sum_{i=1}^{C}(N_i - 1)S_i$; N=2

Third step

Computing the **between-class scatter** matrix: $S_B = \sum_{i=1}^{C}(N_i)(m_i - m)(m_i - m)^T$

m is the overall mean.

Fourth step

Linear discriminants: $S_W^{-1}S_B$

Fifth step

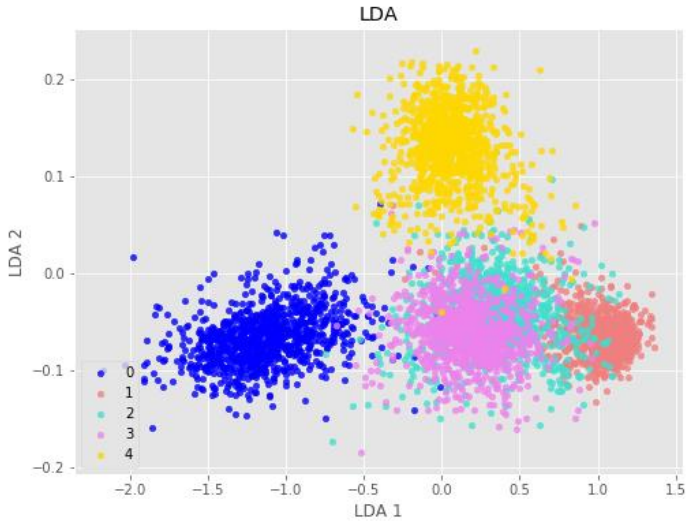
Sort the (eigenvalue, eigenvector) tuples from high to low.

Computing a projection matrix: W

Final step

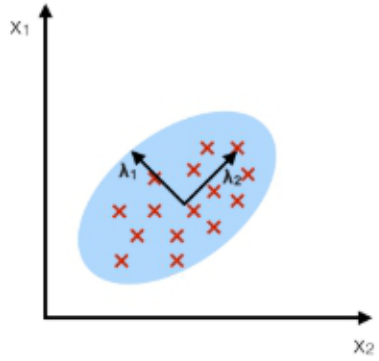
Projection onto the new feature space: $Y = X \times W$

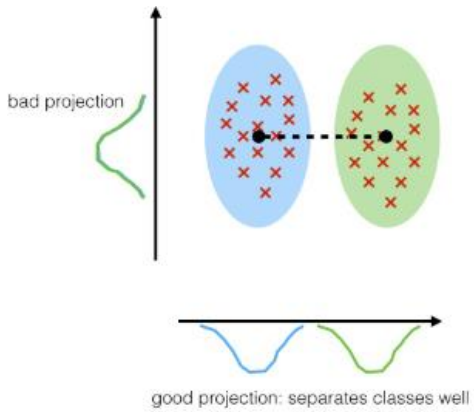
Visualize



Compare PCA & LDA

PCA is component axes that maximize the variance. LDA is maximizing the component axes for class-separation.





Symmetric SNE and T-SNE

C. Symmetric SNE and T-SNE (30%)

Use T-SNE and symmetric SNE to project all your data mnist_X.csv onto 2D space and mark the data points into different colors respectively. The color of the data points depends on which cluster it belongs to (mnist_label.csv). The T-SNE code is provided. You need to modify it to symmetric SNE and discuss their differences. You also have to visualize the distribution of pairwise similarities in both high-dimensional space and low-dimensional space, based on both T-SNE and symmetric SNE.

```
def pca(X=np.array([]), no_dims=50):
    """
        Runs PCA on the NxD array X in order to reduce its dimensionality to
        no_dims dimensions.
    """

    print("Preprocessing the data using PCA...")
    # Covariance matrix : [(X-meanx)^(T)*(X-meanx)]/(n-1)
    meanVec = np.mean(X, axis=0)
    covar = np.dot(((X-meanVec).T),(X-meanVec)) / (len(X)-1)

    #Sort - decrease
    eigVal, eigVec = np.linalg.eig(covar) # eigVal:1*784 ; eigVec:784*784
    eigPair = [(np.abs(eigVal[i]), eigVec[:,i]) for i in range(len(eigVal))]
    eigPair.sort(key=lambda x: x[0], reverse=True)
    W = np.hstack((eigPair[0][1].reshape(len(X[0]),1), eigPair[1][1].reshape(len(X[0]),1)))
    if no_dims > 2:
        for i in range(2,no_dims):
            W = np.hstack((W, eigPair[i][1].reshape(len(X[0]),1)))

    # Projection Matrix
    Y = np.dot(mnist, W)
    return Y
```

T-SNE:

Euclidean distance is converted to conditional probability to express the similarity between points:

$$p_{j|i} = \frac{\exp(-\left||x_i - x_j|\right|^2 / (2\sigma_i^2))}{\sum_{k \neq i} \exp(-\left||x_i - x_j|\right|^2 / (2\sigma_i^2))}$$

For y_i in low dimensions, the similarities between them are as follows:

$$q_{i|j} = \frac{(1 + \left||y_i - y_j|\right|^2)^{-1}}{\sum_{k \neq i}(1 + \left||y_i - y_j|\right|^2)^{-1}}$$

Since the t distribution is a superposition of an infinite number of Gaussian distributions, it is not exponential in calculation. The optimized gradient can be changed as follows:

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \left||y_i - y_j|\right|^2)^{-1}$$

Symmetric SNE

The only difference from the T-SNE code is as follows:

```
# Run iterations
for iter in range(max_iter):

    # Compute pairwise affinities
    sum_Y = np.sum(np.square(Y), 1)
    num = -2. * np.dot(Y, Y.T) #Y = yi-yj; ||yi-yj||^(2)= np.dot(Y, Y.T)
    num = np.exp(-np.add(np.add(num, sum_Y).T, sum_Y))
    num[range(n), range(n)] = 0.
    Q = num / np.sum(num)
    Q = np.maximum(Q, 1e-12)
```

Symmetric SNE assumes that for any$p_{ij} = p_{ji}, q_{ij} = q_{ji}$.

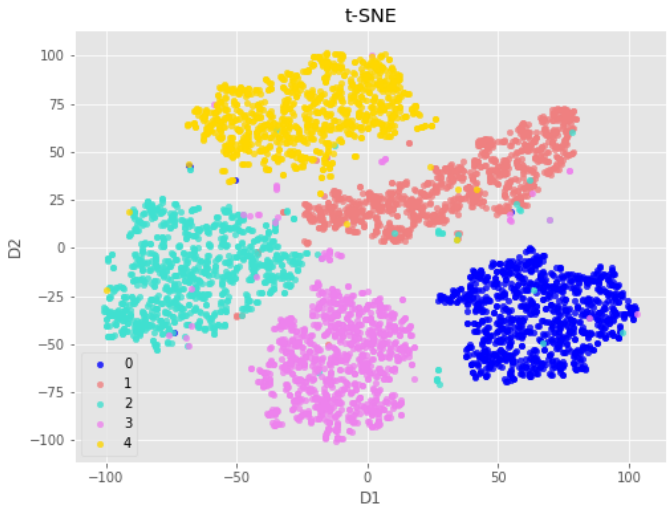
So the probability distribution can be rewritten as:

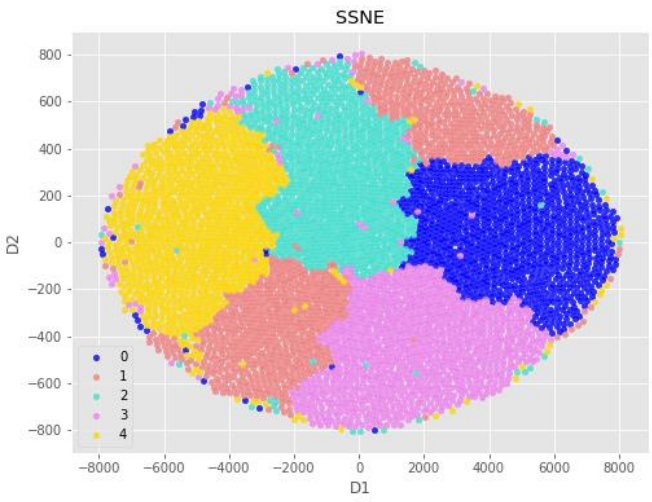
$$q_{ij} = \frac{\exp(-\left||y_i - y_j\right||^2)}{\sum_{k \neq i} \exp(-\left||y_i - y_j\right||^2)}$$

The joint probability distribution is used to replace the conditional probability distribution, that is, P is the joint probability distribution of each point in the high-dimensional space, Q is in the low-dimensional space, and the objective function is

$$C = \mathrm{KL}(\mathrm{P}||\mathrm{Q}) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}$$

Visualize





Eigenface

II. Eigenface (20% in total):
att_faces dataset contains in total 400 images (40 distinct subjects, 10 images per subject). You should use PCA to show the first 25 eigenfaces, and randomly pick 10 images to show their reconstruction (please refer to the lecture slides p.125).

Using Library

```
import numpy as np
from PIL import Image,ImageOps
from copy import deepcopy
from matplotlib import pyplot as plt
%matplotlib inline
```

Read Data

```
pgmf = open('./att_faces/s1/1.pgm', 'rb')
for i in range(3):
    a = pgmf.readline()
    print(a)
```

```
b'P5\n'
b'92 112\n'
b'255\n'
```

Confirm file format.

```
var = 1
directoryName=[]
while var<=40:
    directoryName.append('s'+str(var))
    var+=1

var2 = 1
filename=[]
while var2<=10:
    filename.append(str(var2)+".pgm")
    var2+=1

im = Image.open("./att_faces/"+directoryName[0]+"/"+filename[0])
img = np.asarray(im)
img = img.reshape(img.shape[0]*img.shape[1],1)
orimg = deepcopy(img)

for diname in range(40):
    for fname in range(10):
        im = Image.open("./att_faces/"+directoryName[diname]+"/"+filename[fname])
        img = np.asarray(im)
        img = img.reshape(img.shape[0]*img.shape[1],1)
        orimg = np.hstack((orimg,img))

orimg = np.delete(orimg,0,1)
```

Read the image of the entire “att_faces” folder.

The only difference from the previous PCA is as follows:

```
def pca(X=np.array([]), no_dims=25):
    # Covariance matrix : [(X-meanx)^(T)*(X-meanx)]/(n-1)
    meanVec = np.mean(X, axis=0)
    covar = np.dot(((X-meanVec).T),(X-meanVec)) / (len(X)-1)

    #Sort - decrease
    eigVal, eigVec = np.linalg.eig(covar) # eigVal:1*784 ; eigVec:784*784
    eigPair = [(np.abs(eigVal[i]), eigVec[:,i]) for i in range(len(eigVal))]
    eigPair.sort(key=lambda x: x[0], reverse=True)
    W = np.hstack((eigPair[0][1].reshape(len(X[0]),1), eigPair[1][1].reshape(len(X[0]),1)))
    if no_dims > 2:
        for i in range(2,no_dims):
            W = np.hstack((W, eigPair[i][1].reshape(len(X[0]),1)))

    # Projection Matrix
    Y = np.dot(X-meanVec, W)
    return Y

X = pca(orimg,25)
X /= np.linalg.norm(X,axis=0)
```

We need to let the abs(project matrix length) = 1.

Visualize Eigenface

```
A = X.T
A = A.reshape(((25,112,92)))

iteration=1
for i in range(25):
    fileN = 'eigenface{:s}.png'.format(str(iteration))
    plt.imsave(fileN,A[i],cmap='gray')
    iteration+=1
```

Reconstruct Face image

First step

Computing the "average face"

```
#mean = 10304*1
mean = np.mean(orimg, axis=1).reshape((10304,1))
```

Second step

Computing the weight of each eigenface: $W = (\text{img} - \text{mean}) \times \text{eigenface}$

```
#parameter: weight 400*25
weight = (np.dot((orimg-mean).T,X))
```

Final step

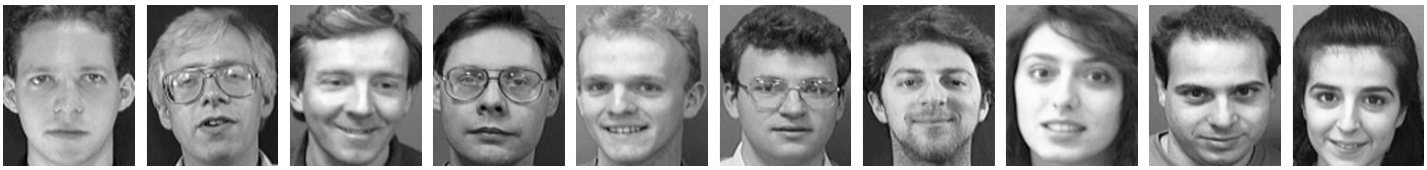
Following the formula of reconstruct face image: $\text{reconimg} = \text{mean} + weight_i \times eigenface.T$

```
var3 = 1
fName=[]
while var3<=10:
    fName.append('Reconstrction_'+str(var3)+'.png')
    var3+=1

var4 = 1
frawName=[]
while var4<=10:
    frawName.append('RAW_'+str(var4)+'.png')
    var4+=1
```

```
orimg = orimg.T
for i in range(10):
    #restrim = 400*25
    reconim = mean.T+np.dot(weight[i*10],X.T)
    im = reconim.reshape((112,92))
    plt.imsave(fName[i],im, cmap='gray')
    rawim = orimg[i*10].reshape((112,92))
    plt.imsave(frawName[i],rawim, cmap='gray')

mmean = mean.reshape((112,92))
plt.imsave("mean.png",mmean, cmap='gray')
```

Visualize Reconstruct Face image

Original image



Reconstruct image

At the stage of PCA, the more features we keep, the more Reconstructed image will look like Original image.