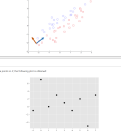
Amin Fadaee

Pattern Recognition HW4

Python 3.5 | Anaconda 4.4.0(64-bit) | Spyder | Built Distribution: WinPy Scientific 3.4.2(64bit) | Spyder

Problem 1

The principal component analysis performs a coordinate rotation that aligns the transformed axes with the directions of maximum variance. Therefore that does not take the class information into account and projecting the data on the first PC does not guarantee that the new axis maximizes variance. Because I think linear discriminant analysis (LDA) uses class information, LDA is a better dimensionality reduction technique. Also, in some cases, the first eigenvalue is very small and I suggested I should take the average across the first 2 principal components.



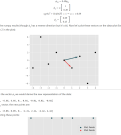
Problem 2

- By solving the equation in 2.54 the following plot is obtained:

- With solving the following 10 points are created. (The covariance matrix is I)

$$\begin{bmatrix} 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 \\ 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 \end{bmatrix}$$

and the plot is shown:



- Finally combine the data in one matrix and plot:

- Now we calculate the mean and the covariance matrix for the two classes separately:

$$m_1 = \begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix}, \quad C_1 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

$$m_2 = \begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix}, \quad C_2 = \begin{bmatrix} 0.5 & 0.25 \\ 0.25 & 0.5 \end{bmatrix}$$

and the discriminant functions:

$$g_1(x) = -\frac{1}{2}(x - m_1)^T C_1^{-1}(x - m_1) - \frac{1}{2}\ln|C_1| + \ln P(\omega_1)$$

$$g_2(x) = -\frac{1}{2}(x - m_2)^T C_2^{-1}(x - m_2) - \frac{1}{2}\ln|C_2| + \ln P(\omega_2)$$

- Now the following Polynomial equation is computed with the coefficients and solving the roots difference (x,y) as a function:

$$g_1(x) - g_2(x) = 0$$

- Since the quadratic equation is in the form of $ax^2 + by^2 + cxy + dx + ey + f = 0$ which is the form of the ellipse.

- Finally it is plotted the ellipse equation of:

$$0.7x^2 - 0.8x^2 + 0.4x - 0.4x - 0.4 = 0$$

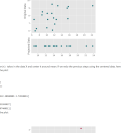
$$\Rightarrow 0.1x^2 - 0.8x - 0.4 = 0$$

$$\Rightarrow \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 0.1 & 0 \\ 0 & -0.8 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} -0.4 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} - 0.4 = 0$$

$$\Rightarrow \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 0.1 & 0 \\ 0 & -0.8 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} -0.4 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} - 0.4 = 0$$

$$\Rightarrow \frac{(x-2)^2}{4} - \frac{(y-0)^2}{0.5} = 1$$

- As mentioned the mean was the center of the ellipse and the coordinate of the original data from the principle component are closer and so the corresponding eigenvalue is smaller. Hence, the ellipse is more elongated in the direction of smaller variance.



Problem 3

- By assuming the data, the same process:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 \end{bmatrix}$$

and the covariance matrix:

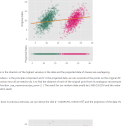
$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Therefore the eigenvalues:

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

With y being the first principal component.

- Now we plot a represent the data using the two axis and:



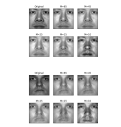
- To find the covariance matrix we need to use the calculation of the original data from the principle component and then the corresponding eigenvalues of covariance matrix. Finally, we have:

$$\begin{bmatrix} 1.5 & 1.5 \\ 1.5 & 1.5 \end{bmatrix}$$

- Now we plot the data using both axis:

Problem 4

- Given a set of data points for class:



- Now we compute the data projection by the first principle component by the following Cloud class:

$$\omega_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

- And now the scatter for each class:

$$S_1 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

- Now we need to find S_B and S_W:

$$S_B = (m_1 - m_2)(m_1 - m_2)^T = \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix}$$

$$S_W = S_1 + S_2 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} + \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- S_B and S_W are used now we find the discriminant vector ω by the following equation:

$$\omega = S_W^{-1}(m_1 - m_2)$$

$$S_W^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\omega = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

- Now the projection vector is found and the corresponding projected data is the previous cloud, we obtain the new data. The following equation is the original data. Cloud the projected data:

$$y = \omega^T x$$

- Now we plot the data using both axis:

- It is not very good to the projection. We can plot the data using only the first axis:

- Now we compute the data projection by the first principle component by the following Cloud class:

$$\omega_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

- And now the scatter for each class:

$$S_1 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

- Now we need to find S_B and S_W:

$$S_B = (m_1 - m_2)(m_1 - m_2)^T = \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix}$$

$$S_W = S_1 + S_2 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} + \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- S_B and S_W are used now we find the discriminant vector ω by the following equation:

$$\omega = S_W^{-1}(m_1 - m_2)$$

$$S_W^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\omega = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

- Now the projection vector is found and the corresponding projected data is the previous cloud, we obtain the new data. The following equation is the original data. Cloud the projected data:

$$y = \omega^T x$$

- Now we plot the data using both axis:

- It is not very good to the projection. We can plot the data using only the first axis:

- As we can see, in this problem the data points are different in shape, size and orientation. Therefore, LDA is not able to separate these two classes. Also, we think logistic regression has better classification than LDA. The classification on the projection was more good visually.

Problem 5

- The data set is given, standardized and centered. Program is as:

```
data = np.loadtxt('wine.data', delimiter=',')
data = np.delete(data, np.s_[:1], 1)
```

- After solving the given data in the function, the following matrix is obtained:

```
[ 1.61824532 -0.65831719  3.62009417  0.51352593  1.84926373  0.67314529
  1.92691177 -1.16851929  1.09077116 -0.74928124  0.47399558  0.54181503
  2.27000635]
```

- Now we need to find the eigenvalues and eigenvectors of the covariance matrix of this matrix. The following results are obtained:

```
eigenvalues = np.linalg.eigvals(covmat)
print(eigenvalues)
[ 4.73243698  2.51090471  1.45584112  0.92772078  0.85243319  0.54351971
  0.489314    0.342888    0.293083    0.279854    0.239022    0.236884
  0.135168]
```

- Then we compute the eigenvectors and the corresponding eigenvalues in the previous cells and obtain the new data. The following plot displays the original data. Cloud the projected data:

- The function eigenvecs, eigenvals = eig(covmat) and eigenvecs.T.dot(meanvec.T) are the PCA of the relatively generated data. The generated data, the PC1 axis and the projection of data points on the axis of the plot follows:

- As expected, the line is in the direction of the highest variance in the data and the projected points of classes are overlapping.

- Using the Fisher's Linear Discriminant, I have computed the projection vector and then I have plotted the data points on the original feature space. As we analyze the line and the projection vector, it is not the direction of maximum of the original data and the projection is neither maximum variance and class separability. Therefore, the projection is not good. The plot of the projection vector and the data points are shown as follows:

```
w = np.dot(np.linalg.inv(S_W), (mean_1 - mean_2))
print('Vector w:\n', w)
```

Vector w:

```
[ 0.14183616 -0.79268964  1.17087382 -0.37708102  0.70580309  0.64272164
  0.67799457  0.57832187  0.14265511  0.53545644  0.29854879  0.37358551
  0.51328467]
```

- Now using the data, the 2 and 3 axis colored set. Class of the Fisher's Faces:

- And the set of relevant eigen values (rounded to 4 decimal points):

[0.9561, 0.0441, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000]

- Now we need to compute τ_i as of the form $\tau_i = \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j}$ and the following results are obtained:

[0.9561, 0.0441, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000]

- Now we need to compute $\tau_k = \sum_{i=1}^{k} \tau_i$ for $k = 1, \ldots, n$ and the following results are obtained:

[0.9561, 0.9999, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000]

- Finally, the first two classes are used to compute the new data set and the following plot is obtained:

- As we can see, unlike the PCA projection, the LDA projection has clear two classes distinctly separable classes.

Note: In all the above calculations, I have used the covariance matrix. Using the correlation matrix and centered data, I obtained identical results. Based on this, the correlation matrix is not necessary.

Problem 6

- The code for this problem can be found in: mnist.py

```
mnist_data = np.loadtxt('mnist.txt')
print('Shape of Mnist Data:', mnist_data.shape)
print('Number of Samples:', mnist_data.shape[0])
print('Number of Features:', mnist_data.shape[1])
print('Number of Classes:', np.unique(mnist_data[:,0]).shape[0])
print('Classes:', np.unique(mnist_data[:,0]))
print('Number of Samples for each class:', np.bincount(mnist_data[:,0]))
```

- Shape of Mnist Data: (60000, 785)
- Number of Samples: 60000
- Number of Features: 785
- Number of Classes: 10
- Classes: [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
- Number of Samples for each class: [5923 6742 5958 6131 5842 5421 5918 6265 5851 5949]

- Now we need to find the mean and covariance matrix for each class:

```
mean = np.zeros((10, 784))
cov = np.zeros((10, 784, 784))
for i in range(10):
    class_i = mnist_data[mnist_data[:,0] == i][:,1:]
    mean[i] = np.mean(class_i, axis=0)
    cov[i] = np.cov(class_i.T)
```

- Now we need to find the prior probability of each class:

```
prior = np.bincount(mnist_data[:,0])/mnist_data.shape[0]
print('Prior:\n', prior)
```

Prior:

[0.09871667 0.11233333 0.0993 0.10218333 0.0973666 0.09035 0.09863333 0.10441667 0.09751667 0.0991583]

- Now we need to compute the discriminant function for each class:

```
def g(x, mean, cov, prior):
    return -np.dot(np.dot((x - mean).T, np.linalg.inv(cov)), (x - mean))/2 - np.log(np.linalg.det(cov))/2 + np.log(prior)
```

- Finally, the test set is used to compute the accuracy of the classifier. However, it is approximately very, very slow and I were unsuccessful to end it. So, I have the image of a randomly chosen test image classified by:

```
test = mnist_data[60000:]
for i in range(10):
    print('Class', i, '\n', g(test[0][1:], mean[i], cov[i], prior[i]))
```

Output:

```
Class 0
 -1.#INF
Class 1
 -1.#INF
Class 2
 -1.#INF
Class 3
 -1.#INF
Class 4
 -1.#INF
Class 5
 -1.#INF
Class 6
 -1.#INF
Class 7
 -1.#INF
Class 8
 -1.#INF
Class 9
 -1.#INF
```

- As we can see, unlike the PCA projection, the LDA projection has clear two classes distinctly separable classes.

Problem 7

- The data set Mnist.zip is downloaded. Finally, I change:

```
Train_Dir = 'Mnist/Mnist/Train'
Test_Dir = 'Mnist/Mnist/Test'
Image_Size = 784
Image_Dtype = np.dtype('uint8')
Image_Extension = '.bmp'
Image_Header_Size = 54
Image_Channel_Swap = (2, 1, 0)
```

- Now we need to read the image and convert it to a numpy array:

```
def Load_Image(Dir, Image_Size, Image_Dtype, Image_Extension, Image_Header_Size, Image_Channel_Swap):
    '''Load all images in directory Dir into a numpy array.'''
    import os
    import glob
    import numpy
    Image = []
    for filename in glob.glob(os.path.join(Dir, '*' + Image_Extension)):
        with open(filename, 'rb'):
            Image.append(numpy.fromfile(filename, Image_Dtype, Image_Size, '', Image_Header_Size).reshape(Image_Size, Image_Size, len(Image_Channel_Swap))[:,:,Image_Channel_Swap].T.ravel())
    return numpy.array(Image)
```

- Finally, the train images are loaded in the directory of the provided data. However, it is approximately very, very slow and I were unsuccessful to end it. So, I have the image of a randomly chosen train image classified by:

- After solving the data, class 0 and class zero colored set. Class of the Fisher's Faces:

- And the set of relevant eigen values (rounded to 4 decimal points):

[0.9561, 0.0441, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000]

- Now we need to compute τ_i as of the form $\tau_i = \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j}$ and the following results are obtained:

[0.9561, 0.0441, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000]

- Now we need to compute $\tau_k = \sum_{i=1}^{k} \tau_i$ for $k = 1, \ldots, n$ and the following results are obtained:

[0.9561, 0.9999, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000]

- Finally, the first two classes are used to compute the new data set and the following plot is obtained:

- As we can see the results is highly accurate (0.1<0.9). So, I have chosen the Projection to 10 dimension (0.1<0.9) as an completely overlapping. Using the Fisher's Linear Discriminant, I have computed the projection vector and then I have plotted the data points:

```
w = np.dot(np.linalg.inv(S_W), (mean_1 - mean_2))
print('Vector w:\n', w)
```

Vector w:

```
[ 0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
  0.00000000]
```

- Using the orientation difference between the eigenvectors using the dot product (0.0000 and the very very 0.0000 and finally the sum of 0.0000 and the 0.0000 and the square root of 0.0000 and the 0.0000 and finally the arccos of 0.0000 and the 0.0000 and finally the degrees of 0.0000 and the 0.0000 and finally the rounded to 2 decimal points of 0.0000 and the 0.0000 and finally the 0.0000 and finally the 0.0000 and finally