M12-L2-P1

November 30, 2024

1 M12-L1 Problem 2

Sometimes the dimensionality is greater than the number of samples. For example,in this problem X has 19 features, but there are only 4 data points. You will need to use the alternate PCA formulation in this case. Follow the steps in the cells below to implement this method.

[303]:
```
import numpy as np
import matplotlib.pyplot as plt

X = np.array([  [-2,  1,  2, -3,  4,  1,  0,  3,  0,  2,  1,  1,  2,  3, -2,␣
 ↪-3, 2,  1,  0],
                [ 1,  2, -4,  2, -4,  2,  5,  2,  2,  1, -3,  0,  0,  1, -2, ␣
 ↪1, 1, -3, -2],
                [ 1, -3,  2,  1,  0, -3, -5, -1,  3,  3, -2, -3, -2, -1,  1, ␣
 ↪0, 5,  4,  2],
                [ 3, -1,  0,  2,  2, -5, -4, -1,  2, -1,  3,  4,  4,  2,  1, ␣
 ↪2, -2,  1, -1]])
```

1.1 Computing Principal Components

1.1.1 The A matrix

First, you should compute the `A` matrix, where A is $(X - \mu)'$. (Note the transpose)

Print this matrix below. It should have size 19×4.

[304]:
```
# YOUR CODE GOES HERE
M = np.mean(X,axis = 0)
A = (X - M).T
print("A = \n", A)
```

```
A =
 [[-2.75  0.25  0.25  2.25]
 [ 1.25  2.25 -2.75 -0.75]
 [ 2.   -4.    2.    0.  ]
 [-3.5   1.5   0.5   1.5 ]
 [ 3.5  -4.5  -0.5   1.5 ]
 [ 2.25  3.25 -1.75 -3.75]
 [ 1.    6.   -4.   -3.  ]
 [ 2.25  1.25 -1.75 -1.75]
```

```
 [-1.75  0.25  1.25  0.25]
 [ 0.75 -0.25  1.75 -2.25]
 [ 1.25 -2.75 -1.75  3.25]
 [ 0.5  -0.5  -3.5   3.5 ]
 [ 1.   -1.   -3.    3.  ]
 [ 1.75 -0.25 -2.25  0.75]
 [-1.5  -1.5   1.5   1.5 ]
 [-3.    1.    0.    2.  ]
 [ 0.5  -0.5   3.5  -3.5 ]
 [ 0.25 -3.75  3.25  0.25]
 [ 0.25 -1.75  2.25 -0.75]]
```

1.1.2 "Small" covariance matrix

By transposing $X - \mu$ to get A, now we can compute a smaller covariance matrix with $A'A$. Compute this matrix, `C`, below and print the result.

[305]:

```
# YOUR CODE GOES HERE
C = A.T@A
print("C = \n", C)
```

```
C =
 [[ 69.875 -18.875 -26.375 -24.625]
 [-18.875 121.375 -53.125 -49.375]
 [-26.375 -53.125  98.375 -18.875]
 [-24.625 -49.375 -18.875  92.875]]
```

1.1.3 Finding nonzero eigenvectors

Next, find the useful (nonzero) eigenvectors of `C`.

For validation purposes, there should be 3 useful eigenvectors, and the first one is `[-0.06628148 -0.79038331 0.47285044 0.38381435]`.

Keep these eigenvectors in a 4×3 array `e`.

[306]:

```
# YOUR CODE GOES HERE
e = np.linalg.eig(C)[1][:,1:]

print(e)
```

```
[[-0.06628148 -0.86249959  0.04124587]
 [-0.79038331  0.34733208 -0.06822502]
 [ 0.47285044  0.22046165 -0.69123739]
 [ 0.38381435  0.29470586  0.71821654]]
```

1.1.4 Calculating "eigenfaces"

Now, we have all we need to compute `U`, the matrix of eigenfaces.

$\mathbf{U_i = Ae_i}$

$(19 \times 3) = (19 \times 4)(4 \times 3)$

Compute and print U. Be sure to normalize your eigenvectors `e` before using the above equation.

[307]:
```
# YOUR CODE GOES HERE

# e = e/np.linalg.norm(e,axis = 0)

U = A@e
U/= np.linalg.norm(U,axis = 0)
print("Eigenfaces, U:\n",U)
```

```
Eigenfaces, U:
 [[ 0.07294372  0.33008441  0.12277459]
 [-0.26034151 -0.11677714  0.11787331]
 [ 0.29998485 -0.27776956 -0.09606164]
 [-0.01067529  0.42516696  0.04536213]
 [ 0.27653993 -0.44157072  0.17530224]
 [-0.37621372 -0.23925816 -0.15082188]
 [-0.59257956 -0.05657115  0.02265222]
 [-0.19897063 -0.250194   -0.0037123 ]
 [ 0.04569305  0.20213547 -0.07236581]
 [ 0.0084373  -0.10504274 -0.25979087]
 [ 0.18948616 -0.1518308   0.35382298]
 [ 0.00380575 -0.03585222  0.46650428]
 [ 0.03449119 -0.10256065  0.40571147]
 [-0.05241297 -0.19442141  0.20419008]
 [ 0.19396809  0.16057937  0.00756997]
 [ 0.01329023  0.36617258  0.11639359]
 [ 0.0508452  -0.08985059 -0.45626561]
 [ 0.3456779  -0.07563409 -0.16842745]
 [ 0.16171488 -0.0569842  -0.18371276]]
```

1.2 Projecting data into 3D

Now project your data into 3 dimensions with the formula:

\$ = U^T A \$

$(3 \times 4) = (3 \times 19)(19 \times 4)$

Call the projected data Ω "`W`". Print `W.T`

[308]:
```
# YOUR CODE GOES HERE
W = U.T@A
print('Projected data in 3 dimensions:\n',W.T)
```

```
Projected data in 3 dimensions:
 [[ -0.8782013   -8.3011616    0.44099733]
 [-10.47224127   3.34291139  -0.72945617]
```

```
 [  6.26506632   2.12184196  -7.39065157]
 [  5.08537624   2.83640825   7.67911041]]
```

1.3 Reconstructing data in 19-D

We can project the transformed data `W` back into the original 19-D space using:

$\Gamma_f = U\Omega + \Psi$
where:
$ _f = $ reconstructed data
$U = $ eigenfaces
$ = $ Reduced data
$ = $ Means

Do this, and compute the MSE between each reconstructed sample and corresponding original points. Report all 4 MSE values.

[309]:
```python
# YOUR CODE GOES HERE

M = M.reshape(-1,1)
Tau = np.dot(U,W) + M
MSE = np.mean((X - Tau.T)**2,axis=0)

for i in range(4):
    print("MSE for sample %d: %e" %(i+1,MSE[i]))
```

```
MSE for sample 1: 4.714677e-31
MSE for sample 2: 1.358936e-30
MSE for sample 3: 1.862960e-30
MSE for sample 4: 7.642090e-31
```

1.4 2-D Reconstruction

What if we had only used the first 2 eigenvectors to compute the eigenfaces? Below, redo the earlier calculations, but use only two eigenfaces. Compute the 4 MSE values that you would get in this case.

(You should get an MSE of 3.626 for the first sample.)

[ ]:
```python
# YOUR CODE GOES HERE
e2 = np.linalg.eig(C)[1][:,0:2]
# e2 = e[:,:2]
U2= A@e2
U2/= np.linalg.norm(U2,axis = 0)

W2 = U2.T@A
M = M.reshape(-1,1)
Tau2 = np.dot(U2,W2) + M
MSE2 = np.mean((X - Tau2.T)**2,axis=0)
```

```python
print("Using only 2 eigenvectors:")
for i in range(4):
    print("MSE for sample %d: %e" %(i+1,MSE2[i]))
```

```
Using only 2 eigenvectors:
MSE for sample 1: 2.501021e+00
MSE for sample 2: 1.435139e+00
MSE for sample 3: 2.966707e+00
MSE for sample 4: 3.228225e+00
```

[ ]:

```python

```