11849060 谭树杰

Lab 7 Report

Class Project 6

Part 1:

- Construct a random matrix **Y** of size 100 X 10. You can use the rand function of **matlab**. Every column of this matrix represents a feature point vector. Compute the mean of these vectors and centralize the points around their mean.
- Compute the eigen vectors of the matrix $\mathbf{S}_1=\mathbf{Y}\mathbf{X}\mathbf{Y}^T$.
- Compute the eigen vectors of the matrix $\mathbf{S}_2=\mathbf{Y}^T\mathbf{X}\mathbf{Y}$.
- Multiply the first eigen vector of $\mathbf{S}_2$ by the matrix **Y**. Normalize the result by dividing by the vector norm. Compare the resulting vector with the first eigen vector of $\mathbf{S}_1$.
- Compare and plot the eigen values of $\mathbf{S}_1$ and $\mathbf{S}_2$. Comment on the results.

1. Construct a random matrix Y of size 100 X 10. Compute the mean of these vectors and centralize the points around their mean.

```
1  Y = np.random.rand(100, 10)
```

```
1  mu = np.mean(Y, axis = 0)
2  mu.shape
```

```
(10,)
```

```
1  YC = Y - mu
2  YC.shape
```

```
(100, 10)
```

2. Compute the eigen vectors of the matrix $S_1 = YXY^T$

```
S1 = YC @ YC.T
```

```
v1, L1 = np.linalg.eigh(S1)
V1 = np.diag(v1)
L1
```

```
array([[ 0.        , -0.00303148, -0.01521311, ...,  0.08080239,
         0.16803024, -0.12045921],
       [ 0.08478053,  0.16634877, -0.06372749, ...,  0.10321607,
        -0.05226445,  0.1632506 ],
       [ 0.04663058, -0.00903322,  0.15788942, ...,  0.05179261,
         0.03279209, -0.05262335],
       ...,
       [-0.03977158, -0.00055669, -0.01334781, ..., -0.00946371,
        -0.09907111, -0.01584245],
       [-0.0531607 ,  0.03182061, -0.07879587, ..., -0.05882959,
        -0.01219975, -0.0358463 ],
       [ 0.03552775, -0.0102358 , -0.03796184, ..., -0.25666014,
        -0.05053933, -0.0730303 ]])
```

3. Compute the eigen vectors of the matrix $S_1 = Y^T XY$

```
S2 = YC.T @ YC
S2
```

```
(10, 10)
```

```
v2, L2 = np.linalg.eigh(S2)
V2 = np.diag(v2)
L2[0:5]
```

```
array([[ 0.28588887,  0.58512644, -0.11067913,  0.07273579,  0.09524833,
        -0.18196544,  0.09871912,  0.47321246,  0.52785898, -0.06212867],
       [-0.46128375, -0.31805364,  0.14436309, -0.33787849, -0.41228874,
        -0.0535433 , -0.02238014,  0.26671754,  0.5219733 ,  0.18469564],
       [ 0.43606027, -0.47969976,  0.47083211,  0.07513904,  0.10117071,
         0.04351602, -0.16766506,  0.46674653, -0.04218891, -0.30420038],
       [ 0.11908872, -0.19543543,  0.12344044, -0.37880471,  0.54253397,
        -0.39719959,  0.43517535, -0.05150762,  0.00496471,  0.38042927],
       [-0.45061167,  0.02426858,  0.11878544,  0.05580406,  0.28004709,
        -0.49332374, -0.1046246 , -0.14768596,  0.08977618, -0.64538685]])
```

4. Multiply the first eigen vector of S_2 by the matrix Y. Normalize the result by dividing by the vector norm. Compare the resulting vector with the first eigen vector of S_1

```python
r1 = YC @ L2[:,0] / np.linalg.norm(L2[:,0])
r2 = L1[:,0]
r1 / r2
```

```
array([-3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684,
       -3.46933684, -3.46933684, -3.46933684, -3.46933684, -3.46933684])
```

```python
v2[0]
```

```
12.036298105345681
```

```python
(-3.46933684)**2
```

```
12.036298109381185
```

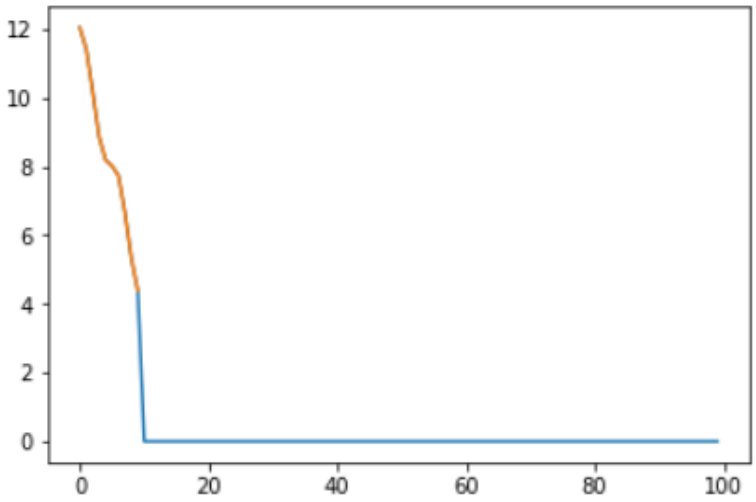
We find the resulting vector are of same values, and the square of this value equals the first eigen value.

5. Compare and plot the eigen values of S_1 and S_2 . Comment on the results.

We could see that the first 10 eigen values of S_1 and S_2 are same.

```
1  x1 = list(range(100))
2  x2 = list(range(10))
3  plt.plot(x1, v1[::-1], x2, v2[::-1])
```

```
[<matplotlib.lines.Line2D at 0x7f6082338fd0>,
 <matplotlib.lines.Line2D at 0x7f6082341160>]
```



Part 2:

- In this part, we aim to use PCA to reduce the dimensionality of a set of feature points. Given a list of 11 binary images (256 X 256) of a fighter jet:
- Read every image to construct a feature vector that represents the intensity values of the image pixels in order. Your feature vector size will be $(256)^2$ X 1. A total of 11 feature vectors will represent the variations of the jet shape as the matrix **Y (of size** $(256)^2$ X 11**)**.
- Use the procedure described in **Part 1** to compute the eigen vectors of the matrix $\mathbf{S=YXY}^T$ (all vectors are centralized around their mean). Note that the matrix size of **S** is $(256)^2$ X $(256)^2$. This size is really a big computational challenge. Comment on this in your report.
- Use the first three eigen vectors of **S** to compute a new vector representation of each fighter jet image.
- Plot the new point vectors in 3D space. Comment on the results describing the images that are very different from the others.
- Use the approach described in the lectures, to select an optimal feature size based on the eigen values and the variance of the projected features. You can plot the variance versus the feature index to pick the point at which variance becomes insignificant.

We construct the feature vectors using following code:

```
f = np.zeros((256**2, 11))
for i, p in enumerate(img_list):
    img = plt.imread(os.path.join(path, p))
    f[:,i] = img.flatten()

```

```
f[:,0].shape
```

```
(65536,)
```

Comment

We could use the SVD to calculate the eigenvectors of $S = Y \times Y^T$, this is the property of SVD. As following code shows, u is eigen matrix of S

```
u, s, vh = np.linalg.svd(Y)
u.shape
```

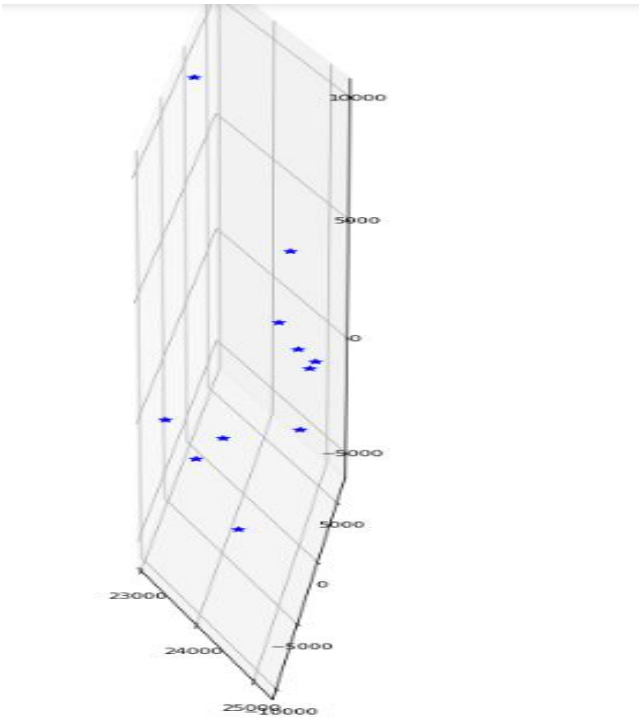
```
(65536, 65536)
```

We project each jet image on the first three eigen vectors of S, i.e. first three columns of u.

```
Y_new = u[:,0:3].T @ Y
Y_new.shape
```

```
(3, 11)
```

Plot new features in below figure.



Part 3 Linear Discriminant Analysis

The objective of this part is to test classification using the linear discriminant analysis.

1- Generate 300 random samples normally distributed with N(μ=200,σ^2=8). Generate another set of 300 sample but with N(μ=400,σ^2=30). Construct a set of 2D samples by grouping these two data sets together. These 2D samples will represent class#1.
2- Repeat **Step 1** but with the following normal distributions: N(μ=100,σ^2=50) and N(μ=-100,σ^2=20). These 2D samples will represent class#2.
3- Use the linear discriminant analysis to compute the direction that gives the best separation between the two classes point projections. Divide the points of each class into training and testing. Compute the error percentage in classification (Use Bayes classification of the projected values). You must visualize your results.
4- Repeat **Step 3** for different values of data variance. Comment on the results.

The generated data required in problem 1 and 2 are shown below:

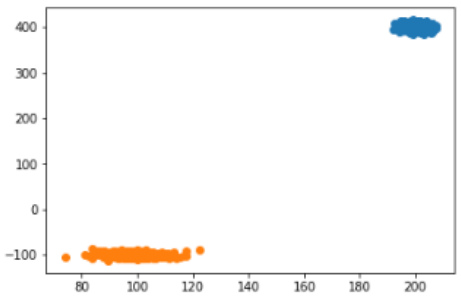
```
mu1 = np.array([200, 400])
n = 300
c11 = mu1[0] + np.random.randn(1, n) * np.sqrt(8)
c12 = mu1[1] + np.random.randn(1, n) * np.sqrt(30)
C1 = np.vstack([c11, c12])
C1.shape
```

```
(2, 300)
```

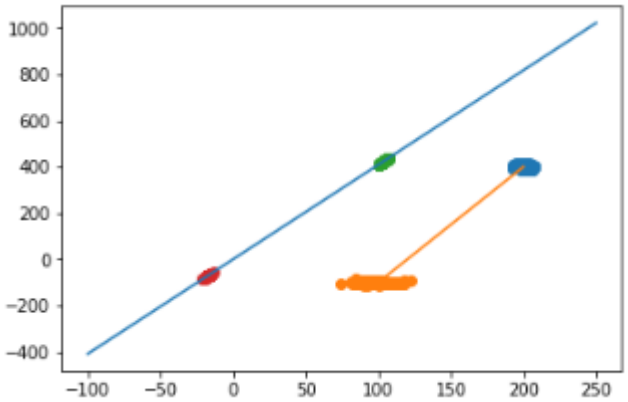
```
mu2 = np.array([100, -100])
n = 300
c21 = mu2[0] + np.random.randn(1, n) * np.sqrt(50)
c22 = mu2[1] + np.random.randn(1, n) * np.sqrt(20)
C2 = np.vstack([c21, c22])
C2.shape
```

```
(2, 300)
```

```
plt.scatter(c11[0], c12[0])
plt.scatter(c21[0], c22[0])
plt.show()
```



Use LDA to compute the direction that gives the best separation between the two classes point projections. We can see that these two classes are separated completely and no point is be misclassified， thus the error percentage is zero.



If we increase pretty large variances of both classes as below code shown. We can see that these two classes are also separated completely and no point is be

misclassified，thus the error percentage is still zero. The result shows the robust of the separation direction.

```
:
# increase covariance
n = 300
c11 = mu1[0] + np.random.randn(1, n) * np.sqrt(1000)
c12 = mu1[1] + np.random.randn(1, n) * np.sqrt(3000)
C1 = np.vstack([c11, c12])
n = 300
c21 = mu2[0] + np.random.randn(1, n) * np.sqrt(2000)
c22 = mu2[1] + np.random.randn(1, n) * np.sqrt(2000)
C2 = np.vstack([c21, c22])

Sw = np.cov(C1) + np.cov(C2)
w = np.linalg.inv(Sw) @ (mu1 - mu2)
m = w[1] / w[0]
```