Final Term Project Report

2016025305 Jihun Kim
jihunkim@hanyang.ac.kr
Oct 16, 2020.

1. Eigenface

First compute means of each feature values and subtract by it to apply PCA.
Then reshape each training samples to a vector.

```
mu = np.mean(x_train, 0)
phi = x_train - mu
phi = np.reshape(phi, (phi.shape[0], -1))
```

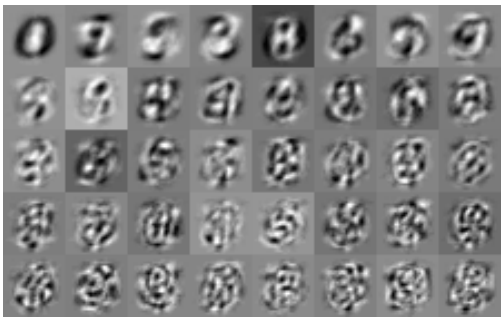
After preprocessing, compute eigenvalue and eigenvector of covariance matrix. Eigenvector represents the direction which maximizes the variance of data when projected, and corresponding eigenvalue represents the variance.

```
C = np.cov(phi.T)
eigenvalues, eigenvec = np.linalg.eig(C)
eigenvec = eigenvec.T
```

Now choose K, the number of eigenvectors that will be used. K=40 is used in this project.

```
K = 40
canvas = np.zeros((5 * 28, 8 * 28))
for i in range(K):
    eigenface = np.reshape(eigenvec[i], (28, 28))
    eigenface = (eigenface - np.min(eigenface)) / (np.max(eigenface) - np.min(eigenface))
    eigenface = np.clip(255 * eigenface, 0, 255)
    canvas[i // 8 * 28: i // 8 * 28 + 28, i % 8 * 28: i % 8 * 28 + 28] = eigenface
show_np_arr(canvas)
```

Here is the visualization result. Left side of the figure denotes the mean of dataset, and right side denotes top-K (left-to-right, and top-to-bottom) eigenvectors. With the right figure, we can observe that eigenvectors with large eigenvalues doesn't contain many details. However, eigenvectors with small eigenvalues does contain details. Therefore, we can say that increasing the number of K helps to preserve details in the original image.



2. Image approximation

Reconstructing image with eigenfaces is straightforward. Given eigenfaces, it works as basis vectors. Approximating the original image is identical to computing vector spanned by basis vectors. Therefore, it can be done with the formula below.

$$\hat{x} = \mu + w_1 u_1 + w_2 u_2 + \cdots + w_K u_K$$

Choosing proper K is also simple. Minimum K is chosen which satisfies following criterion:

$$\frac{\sum_{i=1}^{K} \lambda_i}{\sum_{i=1}^{N} \lambda_i} > Threshold$$

where Threshold is the amount of information of the data we want to preserve. These two formula is easily implementable.

```
while eig_sum < np.sum(eigenvalues) * 0.85:
    w = np.dot(eigenvec[K], (x - mu).reshape(-1, 1))
    eigenface = np.reshape(eigenvec[K], (28, 28))
    x_hat += (w * eigenface).astype('float')
    eig_sum += eigenvalues[K]
    K += 1
```

Here are results. Increasing threshold also increases the K, so approximated image becomes more similar to the original image.

Original data
784 dims
MSE = 0

Thr = 85%
47 dims
MSE = 0.027

Thr = 90%
65 dims
MSE = 0.022

Thr = 95%
99 dims
MSE = 0.017

Thr = 100%
300 dims
MSE = 0.006

This phenomenon can be also measured qualitatively using MSE (Mean Squared Error). The values over on the right denotes MSE when using different value of K. It can be found that MSE steadily decreases while increasing the number of eigenfaces.

```
1 0.07390858899515416
2 0.06852356306909377
...
20 0.03474644924829333
21 0.034704096383500506
...
46 0.028807598717540225
47 0.027992021829003805
```

3. Fast approach

If the dimension of image is much greater than the number of images, there is a better way to compute eigenvectors. Instead of directly computing covariance matrix on original data, computing covariance matrix on transposed data is computationally much more efficient. Overall code is same as above, except how it computes eigenvectors of covariance matrix.

```
C = np.cov(phi)
eigenvalues, eigenvec = np.linalg.eig(C)
eigenvec = phi.T.dot(eigenvec).T
eigenvec /= np.linalg.norm(eigenvec, axis=1).reshape(-1, 1)
```

Results are shown below. Overall appearance is look same with previous implementation of PCA, but there is a slight difference. That is, the intensity of pixel value, MSE value, and the number of dimensions used are also slightly different. I assume that this is because of the accumulation of floating-point error.

Original data
784 dims
MSE = 0

Thr = 85%
47 dims
MSE = 0.030

Thr = 90%
66 dims
MSE = 0.024

Thr = 95%
101 dims
MSE = 0.019

Thr = 100%
300 dims
MSE = 0.011

4. Conclusion

Principle Component Analysis (PCA) finds dominant factor of given data using relatively simple operations. It reduces dimension effectively and capable of restoring lots of original data. Thus, data that needs to be stored can be largely reduced.