Machine Learning Homework 7

309551053 資科工所 黃敏涓

I. Code with detailed explanations (40%)

(1). Kernel Eigenfaces

(a). Part1 (10%)

```python
'''
    function PCA takes image data and the target dimension as inputs. Calculate the
    covariance matrix of image data, and applied eigen-decomposition.
'''
def PCA(data, dim):
    mu = np.mean(data)
    cov = (data - mu) @ (data - mu).T
    eigenvalues, eigenvectors = eig(cov)
    eigenvectors = (data - mu).T @ eigenvectors
    for i in range(eigenvectors.shape[1]):
        eigenvectors[:, i] = eigenvectors[:, i] / norm(eigenvectors[:, i])
    idx = np.argsort(eigenvalues)[::-1]
    W = eigenvectors[:, idx][:, :dim].real

    return W, mu
```

```python
'''
     funtion LDA takes image data, corresponding image label, and the target
     dimension as inputs. Implement the formula listed in slide p.179, then do
     the eigen-decomposition.
'''
def LDA(data, label, dim):
    n, d = data.shape
    C = np.unique(label)
    mu = np.mean(data, axis=0)
    SW = np.zeros((d, d), dtype=np.float64)
    SB = np.zeros((d, d), dtype=np.float64)
    for i in C:
        data_i = data[np.where(label == i)[0], :]
        mu_i = np.mean(data_i, axis = 0)
        SW += (data_i - mu_i).T @ (data_i - mu_i)
        SB += data_i.shape[0] * ((mu_i - mu).T @ (mu_i - mu))
    eigenvalues, eigenvectors = eig(np.linalg.pinv(SW) @ SB)
    for i in range(eigenvectors.shape[1]):
        eigenvectors[:, i] = eigenvectors[:, i] / norm(eigenvectors[:, i])

    idx = np.argsort(eigenvalues)[::-1]
    W = eigenvectors[:, idx][:, :dim].real

    return W
```

Formula used is refered to slide p.179

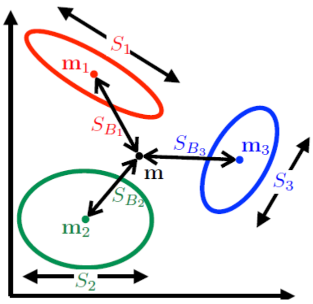
$$\textit{within-class scatter:}\ S_W = \sum_{j=1}^{k} S_j,\ \text{where}\ S_j = \sum_{i \in \mathcal{C}_j} (x_i - \mathbf{m}_j)(x_i - \mathbf{m}_j)^\top$$

$$\text{and}\ \mathbf{m}_j = \frac{1}{n_j} \sum_{i \in \mathcal{C}_j} x_i$$

between-class scatter:

$$S_B = \sum_{j=1}^{k} S_{B_j} = \sum_{j=1}^{k} n_j (\mathbf{m}_j - \mathbf{m})(\mathbf{m}_j - \mathbf{m})^\top$$

$$\text{where}\ \mathbf{m} = \frac{1}{n} \sum x$$



(b). Part2 (5%)

```python
'''
    function classification takes train images, train labels, test images, test labels,
    and the implemented type of method as inputs. Use the concept of K-nearest-neighbor
    to verify the classification results using this type of method.
'''
def Classification(x_train, y_train, x_test, y_test, method):
    res = []
    for i in range(x_test.shape[0]):
        row = []
        for j in range(x_train.shape[0]):
            row.append((np.sum((x_train[j]-x_test[i])**2), y_train[j]))
        row.sort(key = lambda x: x[0])
        res.append(row)
    print(f'face recogonition result using {method}:')
    total = x_test.shape[0]
    for k in K:
        correct = 0
        for i in range(x_test.shape[0]):
            neighbor = np.array([x[1] for x in res[i][:k]])
            nearest, counts = np.unique(neighbor, return_counts = True)
            if nearest[np.argmax(counts)] == y_test[i]:
                correct+=1
        print(f'k = {k}, acc: {correct/total} ({correct}/{total})')
```

Use KNN to calculate the accuracy.

(c). Part3 (10%)

```python
'''
    function KernelPCA takes image data, target dimension, and the kernel type as inputs.
    Use the image data to do the kernel calculation using formula in slide p.128 , then
    use the corresponding result to do eigen-decomposition.
'''
def KernelPCA(data, dim, kernel_type):
    kernel = computeKernel(data, kernel_type)
    n = kernel.shape[0]
    one = np.ones((n, n), dtype=np.float64) / n
    kernel = kernel - one @ kernel - kernel @ one + one @ kernel @ one
    eigen_val, eigen_vec = np.linalg.eig(kernel)
    for i in range(eigen_vec.shape[1]):
        eigen_vec[:, i] = eigen_vec[:, i] / norm(eigen_vec[:, i])
    idx = np.argsort(eigen_val)[::-1]
    W = eigen_vec[:, idx][:, :dim].real

    return kernel @ W
```

After used the preferred kernel function, applied the formula in p.128

$$\rightarrow K^C = K - \mathbf{1}_N K - K\mathbf{1}_N + \mathbf{1}_N K \mathbf{1}_N$$

$\mathbf{1}_N$ is NxN matrix with every element $1/N$ (128

```python
def KernelLDA(data, label, dim, kernel_type):
    C = np.unique(label)
    kernel = computeKernel(data, kernel_type)
    mu = np.mean(kernel, axis=0)
    n = kernel.shape[0]
    SW = np.zeros((n, n), dtype=np.float64)
    SB = np.zeros((n, n), dtype=np.float64)
    for i in C:
        data_i = kernel[np.where(label == i)[0], :]
        m = data_i.shape[0]
        mu_i = np.mean(data_i, axis = 0)
        SW += data_i.T @ (np.identity(m) - (np.ones((m, m), dtype=np.float64) / m)) @ data_i
        SB += m * ((mu_i - mu).T @ (mu_i - mu))
    eigenvalues, eigenvectors = eig(np.linalg.pinv(SW) @ SB)
    for i in range(eigenvectors.shape[1]):
        eigenvectors[:, i] = eigenvectors[:, i] / norm(eigenvectors[:, i])

    idx = np.argsort(eigenvalues)[::-1]
    W = eigenvectors[:, idx][:, :dim].real

    return kernel @ W
```

After used the preferred kernel function, applied the formula in bellow to calculate kernel LDA.

$$\mathbf{w}^{\mathrm{T}}\mathbf{S}_B^{\phi}\mathbf{w} = \mathbf{w}^{\mathrm{T}}\left(\mathbf{m}_2^{\phi} - \mathbf{m}_1^{\phi}\right)\left(\mathbf{m}_2^{\phi} - \mathbf{m}_1^{\phi}\right)^{\mathrm{T}}\mathbf{w} = \alpha^{\mathrm{T}}\mathbf{M}\alpha, \qquad \text{where} \qquad \mathbf{M} = (\mathbf{M}_2 - \mathbf{M}_1)(\mathbf{M}_2 - \mathbf{M}_1)^{\mathrm{T}}.$$

Similarly, the denominator can be written as

$$\mathbf{w}^{\mathrm{T}}\mathbf{S}_W^{\phi}\mathbf{w} = \alpha^{\mathrm{T}}\mathbf{N}\alpha, \qquad \text{where} \qquad \mathbf{N} = \sum_{j=1,2}\mathbf{K}_j(\mathbf{I} - \mathbf{1}_{l_j})\mathbf{K}_j^{\mathrm{T}},$$

(2). t-SNE

(a). Part1 (10%)

```python
144         # Run iterations
145         for iter in range(max_iter):
146             # Compute pairwise affinities
147             if method == 'tsne':
148                 num = 1 / (1 + cdist(Y, Y, 'sqeuclidean'))
149             else:
150                 num = np.exp(-1 * cdist(Y, Y, 'sqeuclidean'))
```

```python
156     # Compute gradient
157     PQ = P - Q
158     for i in range(n):
159         if method =='tsne':
160             # origin.
161             dY[i, :] = np.sum(np.tile(PQ[:, i] * num[:, i], (no_dims, 1)).T * (Y[i, :] - Y), 0)
162         else:
163             dY[i, :] = np.sum(np.tile(PQ[:, i], (no_dims, 1)).T * (Y[i, :] - Y), 0)
```

There's two part have to modify in the original code: line 150 and line 163.

(b). Part2 (2%)

```python
def visualization(Y, labels, idx, interval, method, perplexity):
    fig, ax = plt.subplots()
    scatter = ax.scatter(Y[:, 0], Y[:, 1], 20, labels)
    ax.legend(*scatter.legend_elements(), loc='lower left', title='Digit')
    ax.set_title(f'{method}, perplexity: {perplexity}, iteration: {idx}')
    fig.savefig(f'./{method}_{perplexity}/{idx // interval}.png')
    plt.show()
```

Visualize the results using dimension reduction method.

```python
    gif = []
    files = [int(f.split(".png")[0]) for f in os.listdir(f'{method}_{perplexity}')]
    files.sort()
    for file in files:
        img = Image.open(f'{method}_{perplexity}/'+str(file)+'.png')
        gif.append(img)
    gif[0].save(f'{method}_{perplexity}/{method}_{perplexity}.gif', save_all = True,
                duration = 100, append_images = gif)
    plotSimilarity(P,Q, method, perplexity)
```

Take the images to do the gif animation.

(c). Part3 (2%)

```python
def plotSimilarity(P,Q, method, perplexity):
    pylab.subplot(2, 1, 1)
    pylab.title('SSNE High-dim')
    pylab.hist(P.flatten(), bins = 40, log = True)
    pylab.subplot(2, 1, 2)
    pylab.title('SSNE Low-dim')
    pylab.hist(Q.flatten(), bins = 40, log = True)
    plt.tight_layout()
    plt.savefig(f'./{method}_{perplexity}/{method}_{perplexity}_dimension.png')
    pylab.show()
```

Plot the distribution of pairwise similarities in both high-dimensional space and low-dimensional space.

(d). Part4 (1%)

Modify the command python tsne.py method perplexity to use whatever perplexity you want.

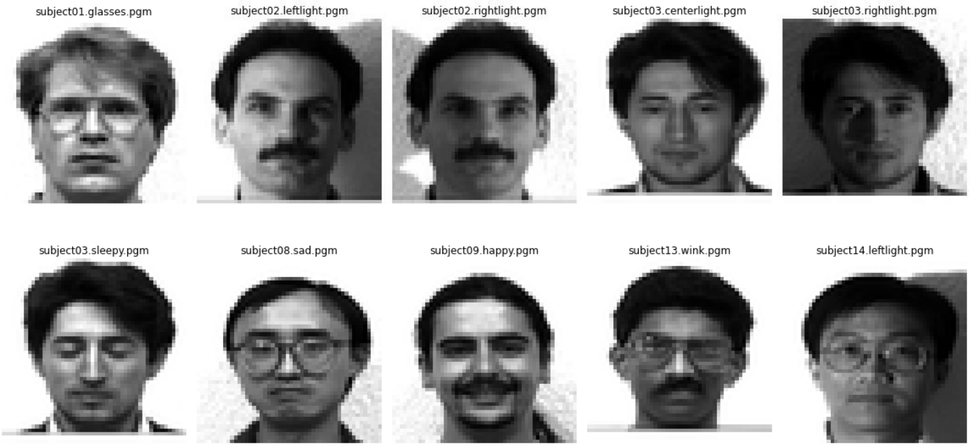
II. Experiments settings and results (35%) & discussion (15%)

(1). Kernel Eigenfaces

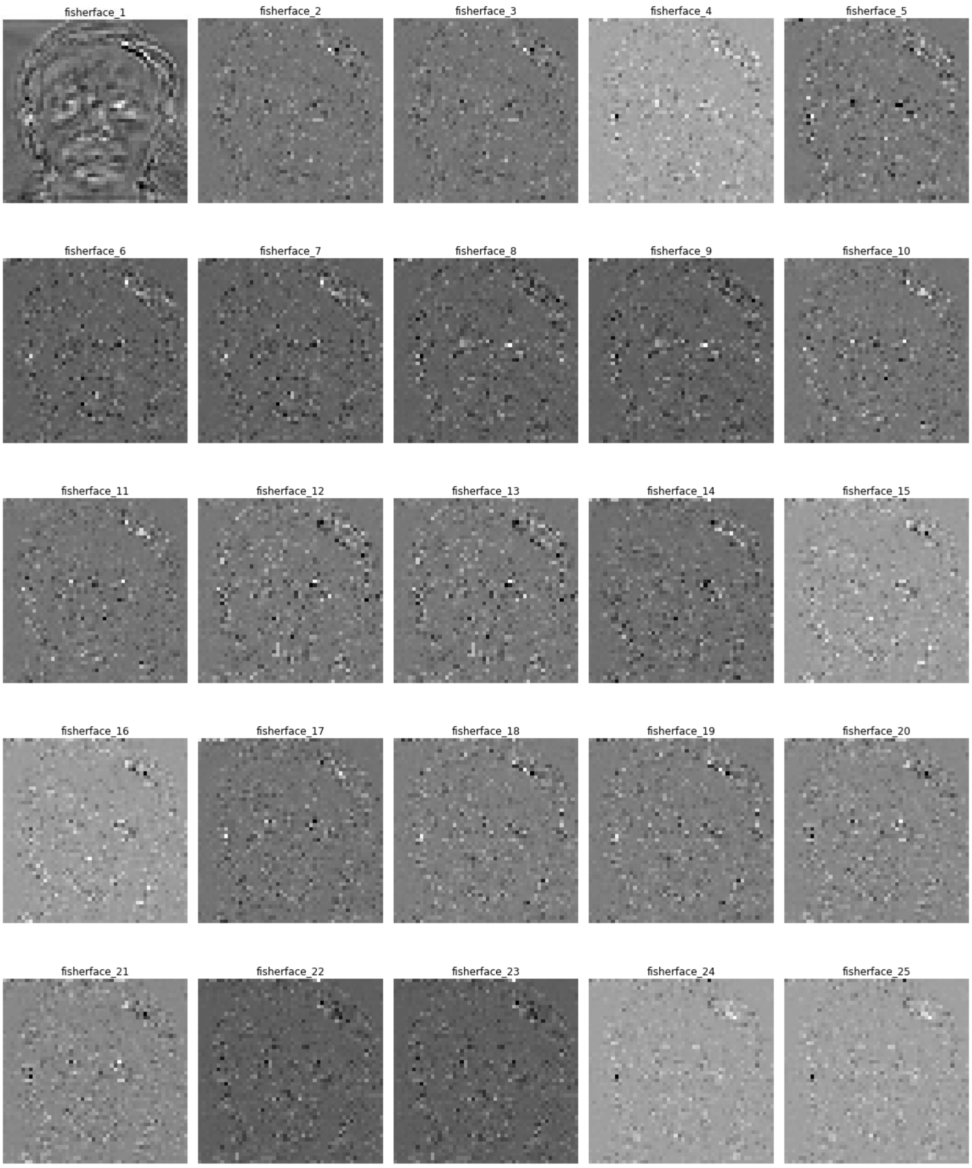
(a). Part1 (5%)

First 25 Eigenfaces:

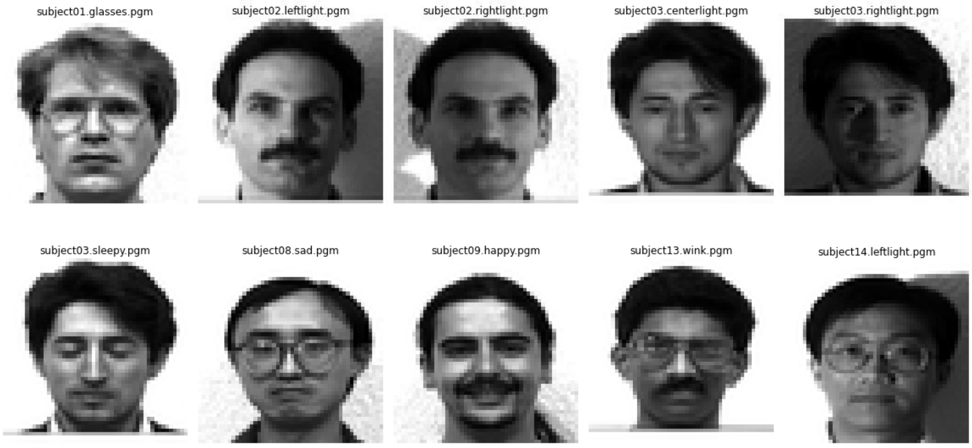
10 reconstruction eigenface:



First 25 Fisherfaces:



10 reconstruction fisherface:



(b). Part2 (5%)

In this part of experiment, I set hyperparameter K in range [1,2,3,4,5,6,7,8,9,10].

Results using PCA and use K-nearest-neighbor to do the classification.

```
face recogonition result using PCA:
k = 1, acc: 0.8333333333333334 (25/30)
k = 2, acc: 0.8333333333333334 (25/30)
k = 3, acc: 0.8333333333333334 (25/30)
k = 4, acc: 0.8333333333333334 (25/30)
k = 5, acc: 0.9 (27/30)
k = 6, acc: 0.8666666666666667 (26/30)
k = 7, acc: 0.9 (27/30)
k = 8, acc: 0.8666666666666667 (26/30)
k = 9, acc: 0.8333333333333334 (25/30)
k = 10, acc: 0.8 (24/30)
```

Results using LDA and use K-nearest-neighbor to do the classification.

```
face recogonition result using LDA:
k = 1, acc: 0.7666666666666667 (23/30)
k = 2, acc: 0.8 (24/30)
k = 3, acc: 0.7666666666666667 (23/30)
k = 4, acc: 0.8 (24/30)
k = 5, acc: 0.8 (24/30)
k = 6, acc: 0.8 (24/30)
k = 7, acc: 0.8 (24/30)
k = 8, acc: 0.8333333333333334 (25/30)
k = 9, acc: 0.8 (24/30)
k = 10, acc: 0.7333333333333333 (22/30)
```

(c). Part3 (5%) & (5%)

Results using kernel PCA and use K-nearest-neighbor to do the classification.

Linear kernel:

```
face recogonition result using KernelPCA_Linearkernel:
k = 1, acc: 0.16666666666666666 (5/30)
k = 2, acc: 0.13333333333333333 (4/30)
k = 3, acc: 0.13333333333333333 (4/30)
k = 4, acc: 0.1 (3/30)
k = 5, acc: 0.13333333333333333 (4/30)
k = 6, acc: 0.13333333333333333 (4/30)
k = 7, acc: 0.13333333333333333 (4/30)
k = 8, acc: 0.13333333333333333 (4/30)
k = 9, acc: 0.1 (3/30)
k = 10, acc: 0.1 (3/30)
```

Polynomial kernel: gamma =3, coefficient= 10, degree = 2

```
face recogonition result using
KernelPCA_PolynomialKernel:
k = 1, acc: 0.1 (3/30)
k = 2, acc: 0.13333333333333333 (4/30)
k = 3, acc: 0.2 (6/30)
k = 4, acc: 0.16666666666666666 (5/30)
k = 5, acc: 0.13333333333333333 (4/30)
k = 6, acc: 0.13333333333333333 (4/30)
k = 7, acc: 0.16666666666666666 (5/30)
k = 8, acc: 0.16666666666666666 (5/30)
k = 9, acc: 0.13333333333333333 (4/30)
k = 10, acc: 0.13333333333333333 (4/30)
```

RBF kernel: gamma = 1e-7

```
face recogonition result using KernelPCA_rbfKernel:
k = 1, acc: 0.8333333333333334 (25/30)
k = 2, acc: 0.8333333333333334 (25/30)
k = 3, acc: 0.8333333333333334 (25/30)
k = 4, acc: 0.8333333333333334 (25/30)
k = 5, acc: 0.8 (24/30)
k = 6, acc: 0.7666666666666667 (23/30)
k = 7, acc: 0.7666666666666667 (23/30)
k = 8, acc: 0.8 (24/30)
k = 9, acc: 0.8333333333333334 (25/30)
k = 10, acc: 0.8 (24/30)
```

Results using kernel LDA and use K-nearest-neighbor to do the classification.

Linear kernel:

```
face recogonition result using KernelLDA_Linearkernel:
k = 1, acc: 0.16666666666666666 (5/30)
k = 2, acc: 0.03333333333333333 (1/30)
k = 3, acc: 0.1 (3/30)
k = 4, acc: 0.1 (3/30)
k = 5, acc: 0.1 (3/30)
k = 6, acc: 0.06666666666666667 (2/30)
k = 7, acc: 0.13333333333333333 (4/30)
k = 8, acc: 0.13333333333333333 (4/30)
k = 9, acc: 0.1 (3/30)
k = 10, acc: 0.06666666666666667 (2/30)
```

Polynomial kernel: gamma =3, coefficient= 10, degree = 2

```
face recogonition result using
KernelLDA_PolynomialKernel:
k = 1, acc: 0.13333333333333333 (4/30)
k = 2, acc: 0.06666666666666667 (2/30)
k = 3, acc: 0.03333333333333333 (1/30)
k = 4, acc: 0.06666666666666667 (2/30)
k = 5, acc: 0.06666666666666667 (2/30)
k = 6, acc: 0.06666666666666667 (2/30)
k = 7, acc: 0.06666666666666667 (2/30)
k = 8, acc: 0.06666666666666667 (2/30)
k = 9, acc: 0.06666666666666667 (2/30)
k = 10, acc: 0.03333333333333333 (1/30)
```

RBF kernel: gamma = 1e-7

```
face recogonition result using KernelLDA_rbfKernel:
k = 1, acc: 0.7666666666666667 (23/30)
k = 2, acc: 0.8 (24/30)
k = 3, acc: 0.7666666666666667 (23/30)
k = 4, acc: 0.7333333333333333 (22/30)
k = 5, acc: 0.7666666666666667 (23/30)
k = 6, acc: 0.7 (21/30)
k = 7, acc: 0.6333333333333333 (19/30)
k = 8, acc: 0.6333333333333333 (19/30)
k = 9, acc: 0.6666666666666666 (20/30)
k = 10, acc: 0.6666666666666666 (20/30)
```

From the results shown above, we can obeserve that using simply PCA and LDA performed better than using Kernel PCA, Kernal LDA (whether using linear kernel, polynomial kernel or rbf kernel) to do the tricks.

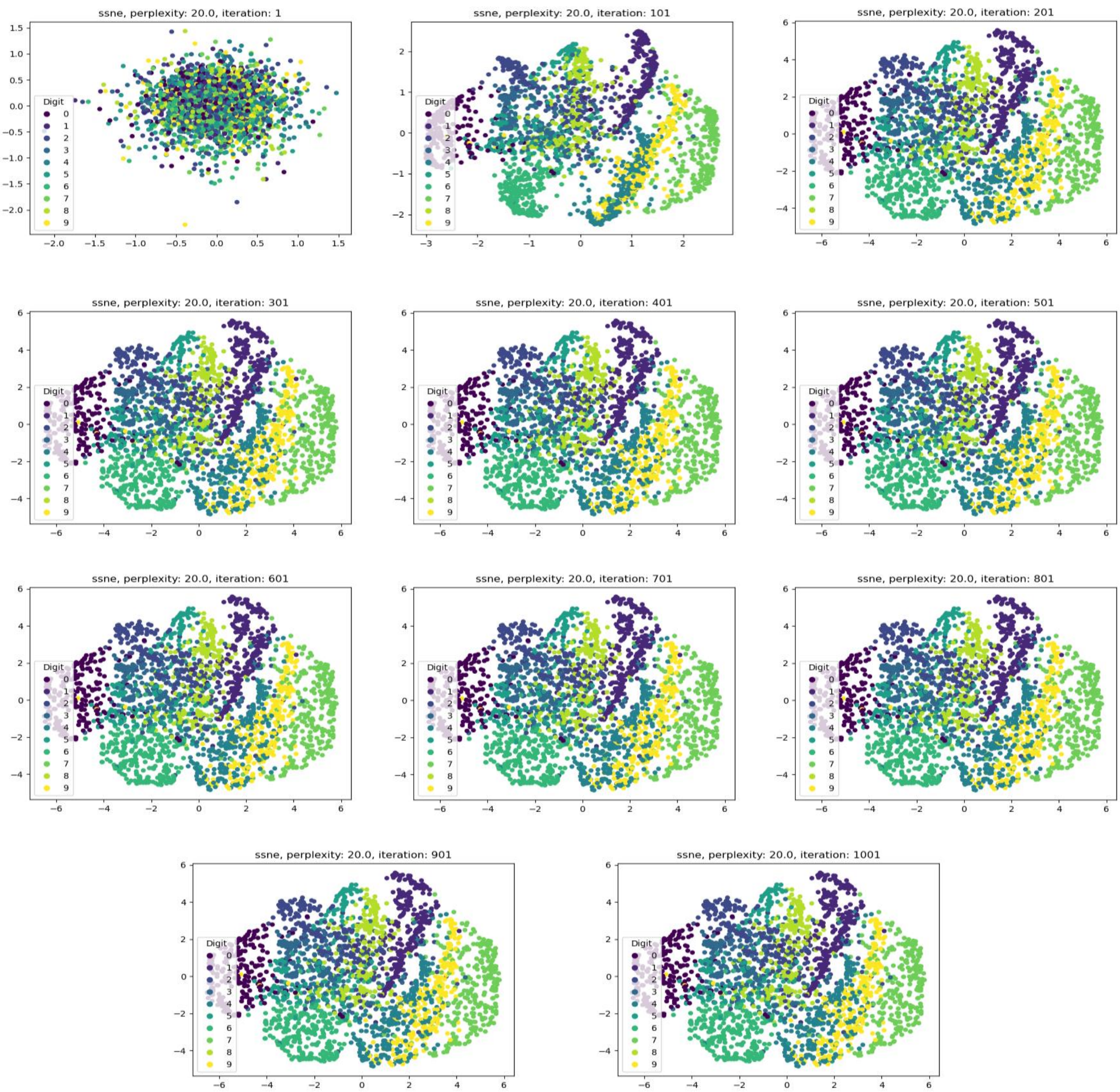
(2) t-SNE

(a). Part1 (5%) & (5%)

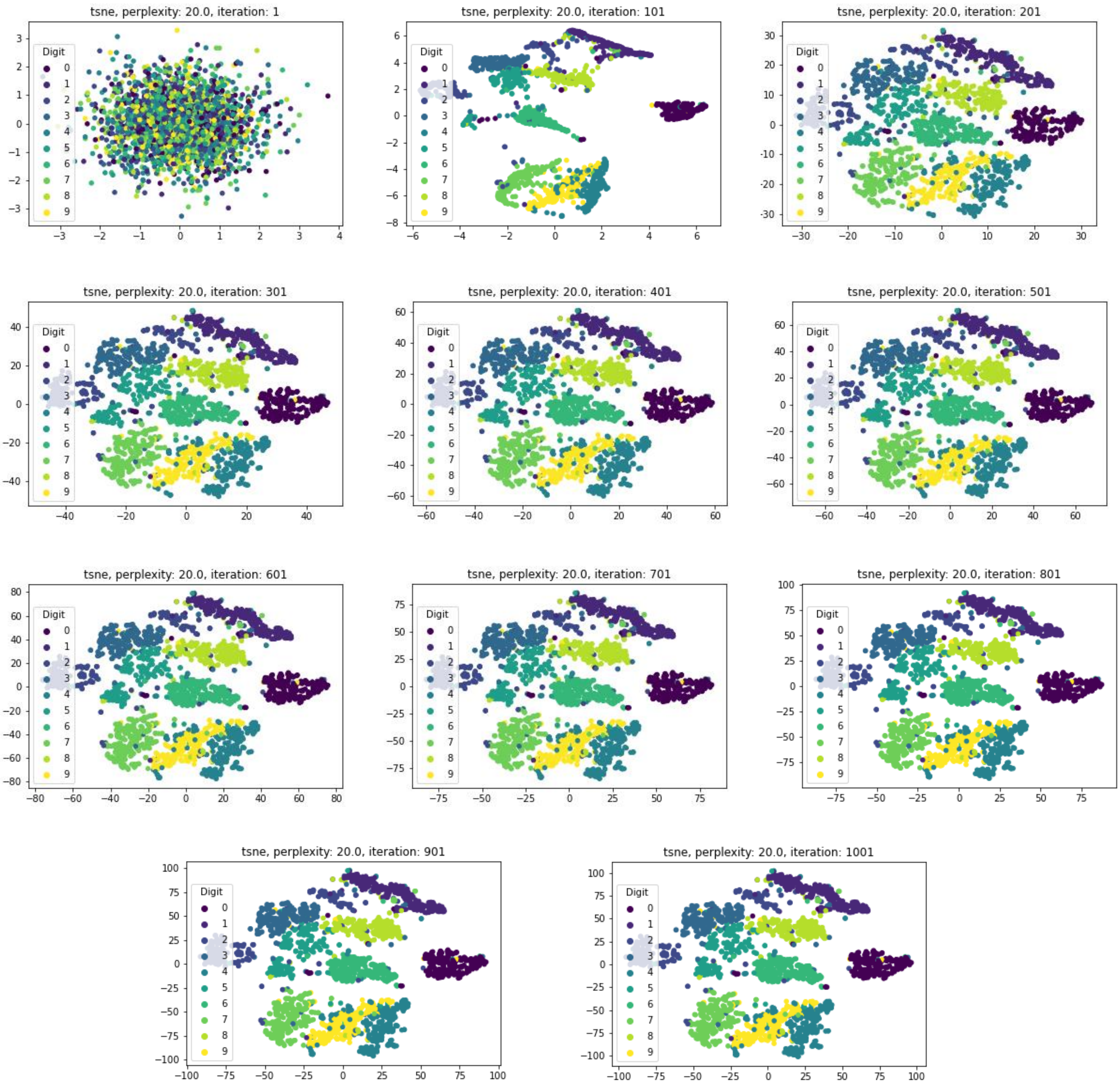
Tsne use **student-t distribution** in low dimension space to alleviate the crowding problem caused by curse of dimensionality

(b). Part2 (5%) – set perplexity = 20

ssne:

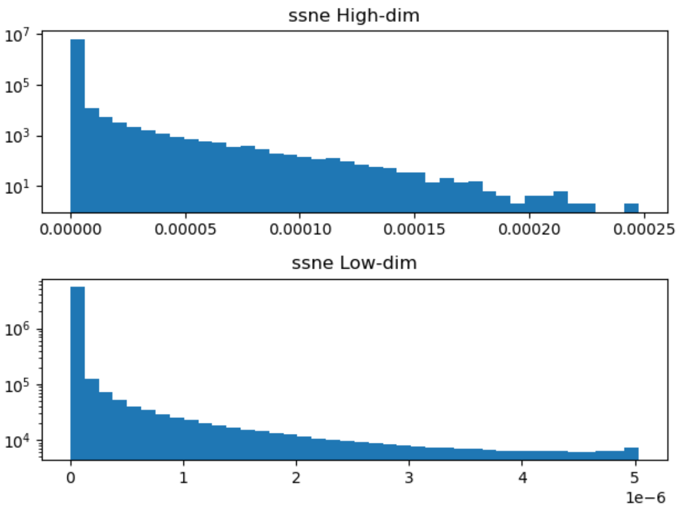


tsne:

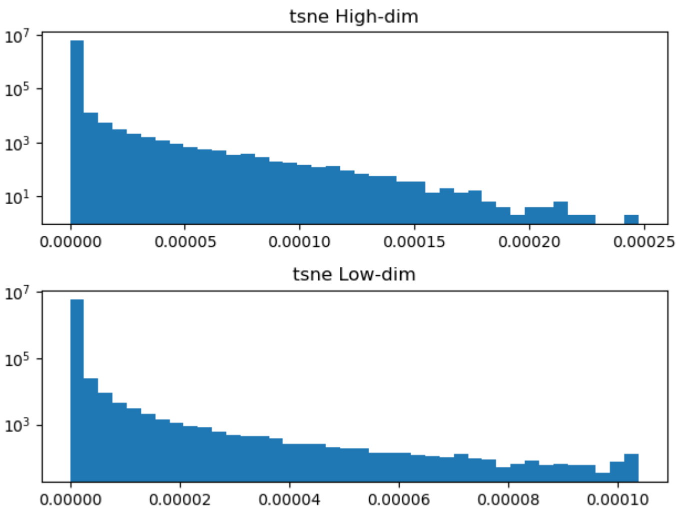

tsne, perplexity: 20.0, iteration: 1
tsne, perplexity: 20.0, iteration: 101
tsne, perplexity: 20.0, iteration: 201
tsne, perplexity: 20.0, iteration: 301
tsne, perplexity: 20.0, iteration: 401
tsne, perplexity: 20.0, iteration: 501
tsne, perplexity: 20.0, iteration: 601
tsne, perplexity: 20.0, iteration: 701
tsne, perplexity: 20.0, iteration: 801
tsne, perplexity: 20.0, iteration: 901
tsne, perplexity: 20.0, iteration: 1001
Digit
0
1
2
3
4
5
6
7
8
9

(c). Part3 (5%) – set perplexity = 20

ssne:

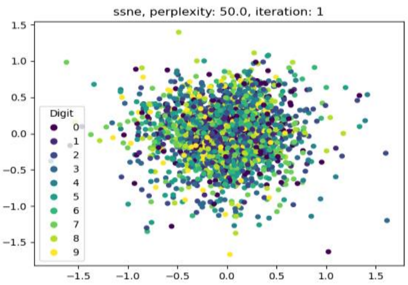


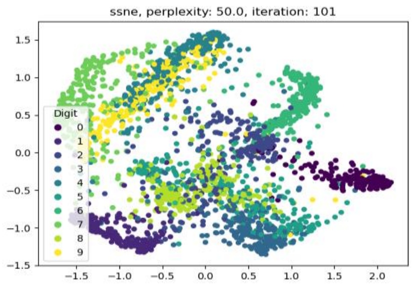
tsne:

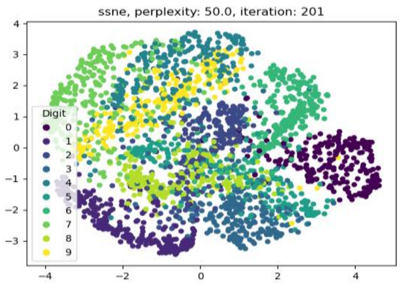


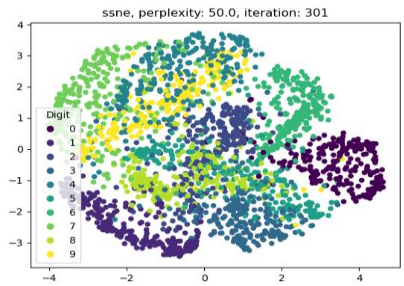
(d). Part4 (5%) & (5%) – set perplexity = 50

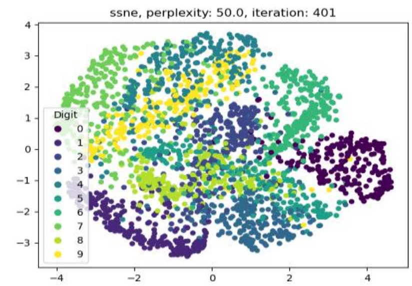
ssne:

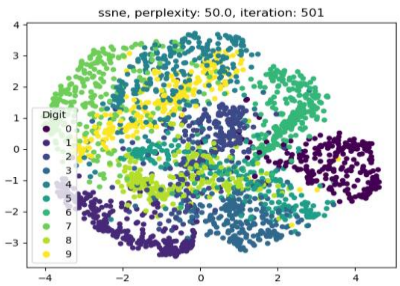


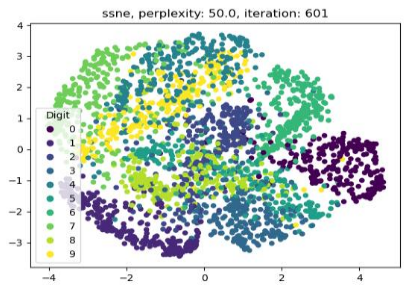


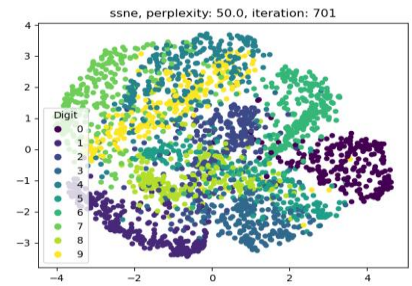


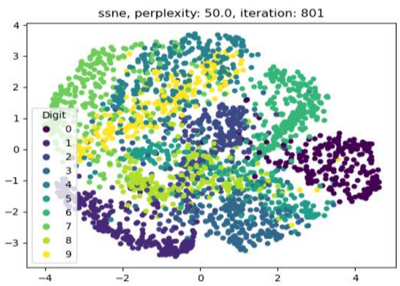


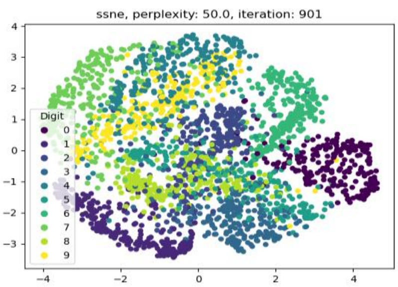


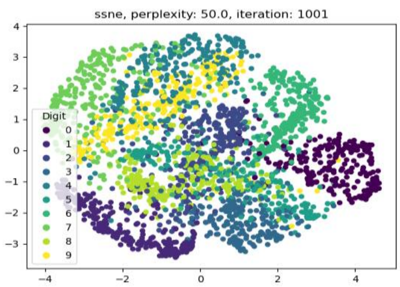




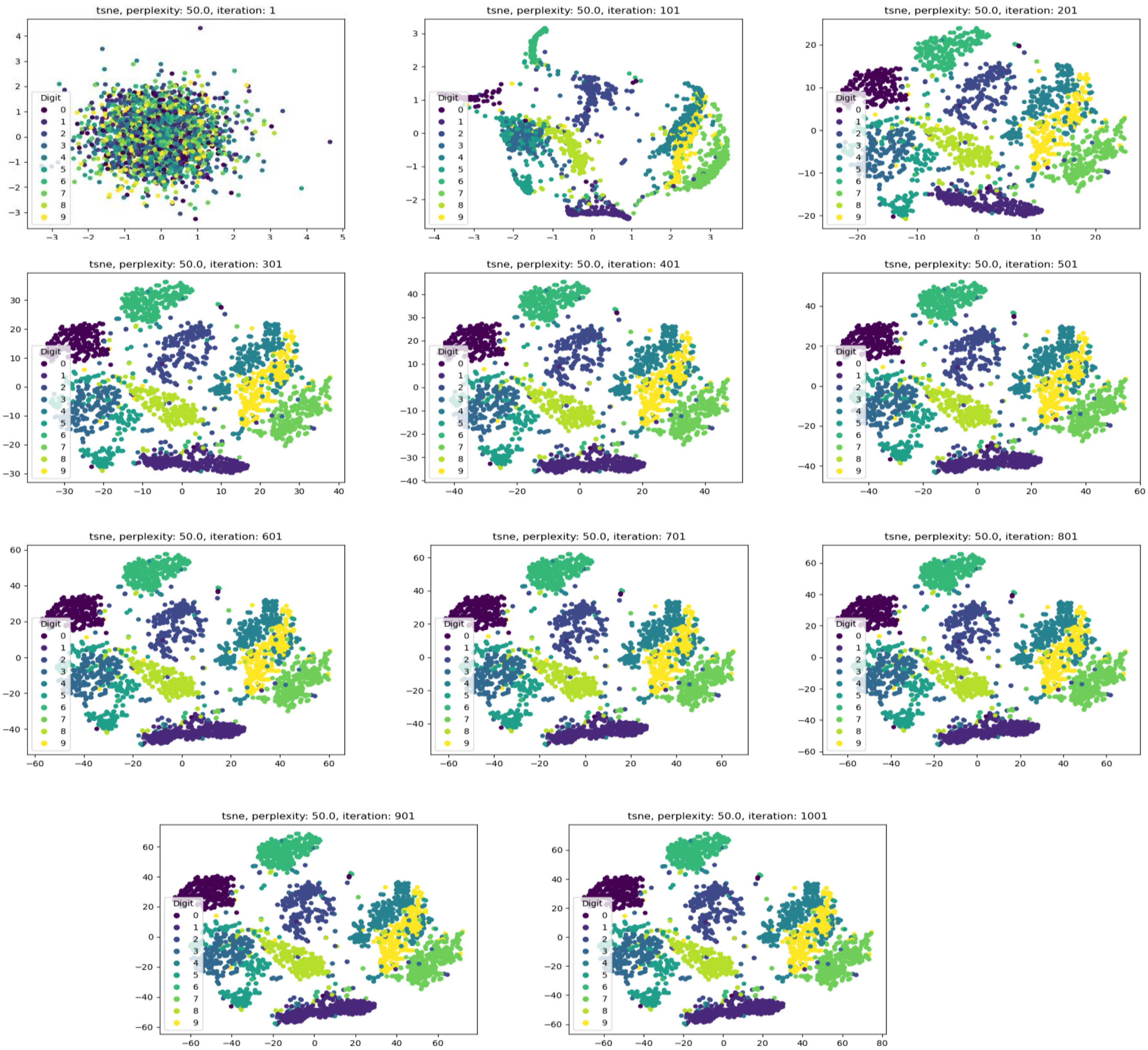








tsne :



We can see that that the global structure is more clear when in high perplexity. When apply in low perplexity, it may bump into the situation that the same group is split into multiple groups.

III. Observations and discussion (10%)

In the eigenface & fisherface experiment 'kernel PCA&LDA' using rbf kernel, I found out that when the hyperparameter gamma set to 1e-7 would give out the best result:

```
face recogonition result using KernelPCA_rbfKernel:
k = 1, acc: 0.8333333333333334 (25/30)
k = 2, acc: 0.8333333333333334 (25/30)
k = 3, acc: 0.8333333333333334 (25/30)
k = 4, acc: 0.8333333333333334 (25/30)
k = 5, acc: 0.8 (24/30)
k = 6, acc: 0.7666666666666667 (23/30)
k = 7, acc: 0.7666666666666667 (23/30)
k = 8, acc: 0.8 (24/30)
k = 9, acc: 0.8333333333333334 (25/30)
k = 10, acc: 0.8 (24/30)
face recogonition result using KernelLDA_rbfKernel:
k = 1, acc: 0.7666666666666667 (23/30)
k = 2, acc: 0.8 (24/30)
k = 3, acc: 0.7666666666666667 (23/30)
k = 4, acc: 0.7333333333333333 (22/30)
k = 5, acc: 0.7666666666666667 (23/30)
k = 6, acc: 0.7 (21/30)
k = 7, acc: 0.6333333333333333 (19/30)
k = 8, acc: 0.6333333333333333 (19/30)
k = 9, acc: 0.6666666666666666 (20/30)
k = 10, acc: 0.6666666666666666 (20/30)
```

But if set gamma to 1e-5 or 1e-6, the results performed much worst.

```
face recogonition result using KernelPCA_rbfKernel:
k = 1, acc: 0.5 (15/30)
k = 2, acc: 0.4666666666666667 (14/30)
k = 3, acc: 0.43333333333333335 (13/30)
k = 4, acc: 0.4 (12/30)
k = 5, acc: 0.4 (12/30)
k = 6, acc: 0.4 (12/30)
k = 7, acc: 0.43333333333333335 (13/30)
k = 8, acc: 0.43333333333333335 (13/30)
k = 9, acc: 0.3333333333333333 (10/30)
k = 10, acc: 0.3333333333333333 (10/30)
face recogonition result using KernelLDA_rbfKernel:
k = 1, acc: 0.3 (9/30)
k = 2, acc: 0.3333333333333333 (10/30)
k = 3, acc: 0.23333333333333334 (7/30)
k = 4, acc: 0.23333333333333334 (7/30)
k = 5, acc: 0.2 (6/30)
k = 6, acc: 0.2 (6/30)
k = 7, acc: 0.2 (6/30)
k = 8, acc: 0.16666666666666666 (5/30)
k = 9, acc: 0.1 (3/30)
k = 10, acc: 0.03333333333333333 (1/30)
```

Gamma = 1e-5

```
face recogonition result using KernelPCA_rbfKernel:
k = 1, acc: 0.7333333333333333 (22/30)
k = 2, acc: 0.6333333333333333 (19/30)
k = 3, acc: 0.6 (18/30)
k = 4, acc: 0.6 (18/30)
k = 5, acc: 0.6 (18/30)
k = 6, acc: 0.6 (18/30)
k = 7, acc: 0.6333333333333333 (19/30)
k = 8, acc: 0.6666666666666666 (20/30)
k = 9, acc: 0.6666666666666666 (20/30)
k = 10, acc: 0.6333333333333333 (19/30)
face recogonition result using KernelLDA_rbfKernel:
k = 1, acc: 0.43333333333333335 (13/30)
k = 2, acc: 0.43333333333333335 (13/30)
k = 3, acc: 0.43333333333333335 (13/30)
k = 4, acc: 0.5 (15/30)
k = 5, acc: 0.43333333333333335 (13/30)
k = 6, acc: 0.4 (12/30)
k = 7, acc: 0.3 (9/30)
k = 8, acc: 0.3333333333333333 (10/30)
k = 9, acc: 0.3333333333333333 (10/30)
k = 10, acc: 0.36666666666666664 (11/30)
```

Gamma = 1e-6