2019/06/13
0780828
Alisher Mukashev

HOMEWORK #7

1. Use PCA to project all your data mnist_X.csv onto 2D space and mark the data points into different colors. The color of the data points depends on which cluster it belongs to (mnist_label.csv).

PCA (homework_7_1_pca.py)

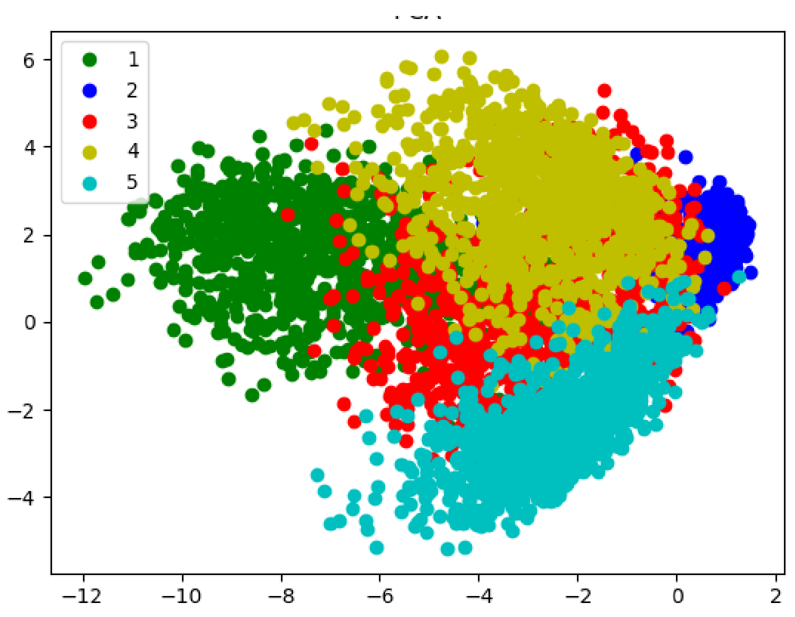
Explanation of code.

First step: compute the covariance matrix of the whole given dataset:
np.cov(mnist_X.T)
Second step: get eigenvectors and corresponding eigenvalues from the covariance of the whole dataset (*weights = np.covariance(mnist_X).T*):
eigenvalues, eigenvectors = np.linalg.eigh(weights)
Third step: the eigenvectors with the lowest eigenvalues bear the least information about the distribution of the data, and those are the ones we want to drop. The common approach is to rank the eigenvectors from highest to lowest corresponding eigenvalue and choose the top 2-eigenvectors (because 2D space):

```
sorted_idx = np.argsort(eigenvalues[::-1])
U = []
for i in range(dim):
   evc = eigenvectors[:, sorted_idx[i]]
   U.append(evc[:, None])
U = np.concatenate(U, axis=1)
```

Fourth step: transform given dataset onto new subspace via the equation *z = W*x*:
PCA = np.dot(mnist_X, W)

Results.



2. Use LDA to project all your data mnist_X.csv onto 2D space and mark the data points into different colors. The color of the data points depends on which cluster it belongs to (mnist_label.csv).

LDA (homework_7_1_lda.py)

Explanation of code.

First step: calculate means of each class and overall mean:

```
m = np.zeros((5000, 5))
for i in range(5):
   indices = (mnist_label == (i+1))
   m[indices, i] = 1
one_class_mean = np.dot(mnist_X.T, m) / 1000
mean = np.mean(mnist_X, axis = 0)
```

Second step: compute the covariance matrix between classes (S_b):

```
diff_mean = np.subtract(one_class_mean, mean[:, np.newaxis])
between_class_scatter = np.dot(diff_mean * 1000, diff_mean.T)
```

Third step: compute the covariance matrix within classes (S_W):

```
within_class = np.subtract(mnist_X.T, np.dot(one_class_mean, m.T))
within_class_scatter = np.zeros([784, 784])
```

```
for i in range(5):
    indices = (mnist_label == (i+1))
    within_class_scatter += (np.dot(within_class[:, indices], within_class[:, indices].T)) / 1000
```

Fourth step: using Lagrangian dual, the problem of maximizing $J = \frac{w^T * S_b * w}{w^T * S_w * w}$ can be transformed into the solution $S_w^{-1} * S_b * w = \lambda * w$, which is an eigenvalue problem for the matrix $S_w^{-1} * S_b$:

(*weights = np.dot(np.linalg.pinv(within_class_scatter), between_class_scatter)*):

```
eigenvalues, eigenvectors = np.linalg.eigh(weights)
```

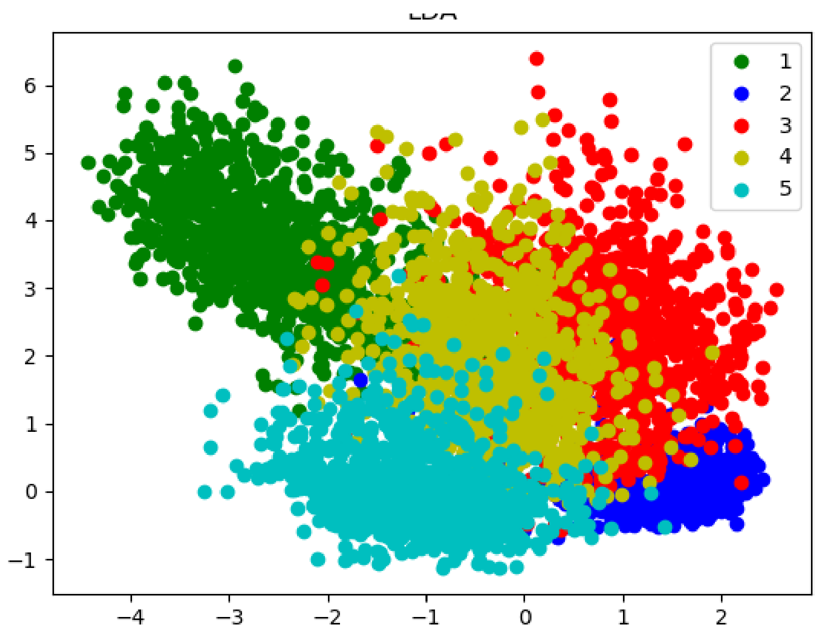
Fifth step: the eigenvectors with the lowest eigenvalues bear the least information about the distribution of the data, and those are the ones we want to drop. The common approach is to rank the eigenvectors from highest to lowest corresponding eigenvalue and choose the top 2-eigenvectors (because 2D space):

```
sorted_idx = np.argsort(eigenvalues[::-1])
U = []
for i in range(dim):
    evc = eigenvectors[:, sorted_idx[i]]
    U.append(evc[:, None])
U = np.concatenate(U, axis=1)
```

Sixth step: transform given dataset onto new subspace via the equation *z = W*x*:

```
LDA = np.dot(mnist_X, W)
```

Results.



Discussion.

I visualized all input data with different colors and show their corresponding labels on left-top corner for PCA and on right-top corner for LDA. Visually, I can tell LDA results in more compact cluster than PCA. I think this phenomenon is consistent to the algorithm, because the LDA not only maximize the distance between cluster, but also minimize the variance in between cluster.

3. Use T-SNE and symmetric SNE to project all your data mnist_X.csv onto 2D space and mark the data points into different colors respectively. The color of the data points depends on which cluster it belongs to (mnist_label.csv). The T-SNE code is provided. You need to modify it to symmetric SNE and discuss their differences. You also have to visualize the distribution of pairwise similarities in both high-dimensional space and low-dimensional space, based on both T-SNE and symmetric SNE.

Explanation of code.

The teacher provided us the code 'tsne.py'. We just need to modify this code to symmetric SNE. According to the definition of symmetric SNE:

$$q_{ij} = \frac{\exp(-\| y_i - y_j \|^2)}{\sum_{k \neq l} \exp(-\| y_l - y_k \|^2)}$$ and $$\frac{\partial C}{\partial y_i} = 2\sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

I added flag *Symmetric* and made it possible to switch from t-sne to s-sne. You must set this value to *True* if you want to work with s-sne. I also implemented PCA by using code from previous task. In 'tsne.pe' I modified function 'tsne' and changed the way to compute Q (the pairwise affinity) and Gradient.

1. Computing Q for symmetric SNE:

```
sum_Y = np.sum(np.square(Y), 1)

num = -2. * np.dot(Y, Y.T)

num = np.exp(-1. * np.add(np.add(num, sum_Y).T,

num[range(n), range(n)] = 0.

Q = num / np.sum(num)

Q = np.maximum(Q, 1e-12)
```

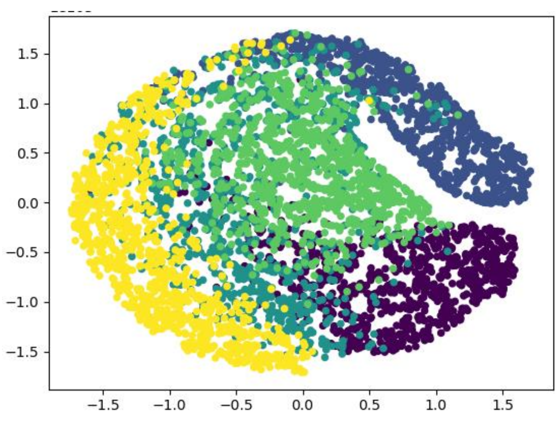
2. Computing Gradient for symmetric SNE:

```
PQ = P - Q

for i in range(n):

    dY[i, :] = np.sum(np.tile(PQ[:, i] * num[:, i], (no_dims, 1)).T * (Y[i, :] - Y), 0)
```

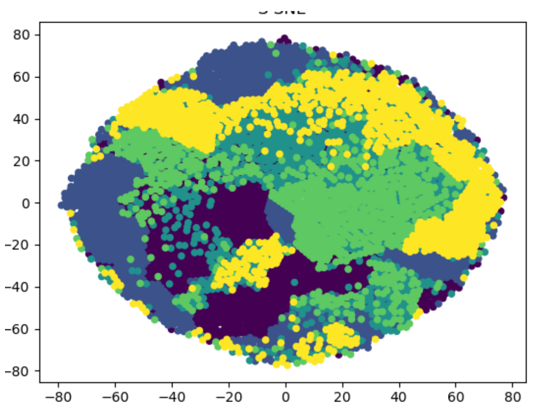
Results.

```
pylab.scatter(Y[:, 0], Y[:, 1], 20, mnist_label)

pylab.show()
```

T-SNE

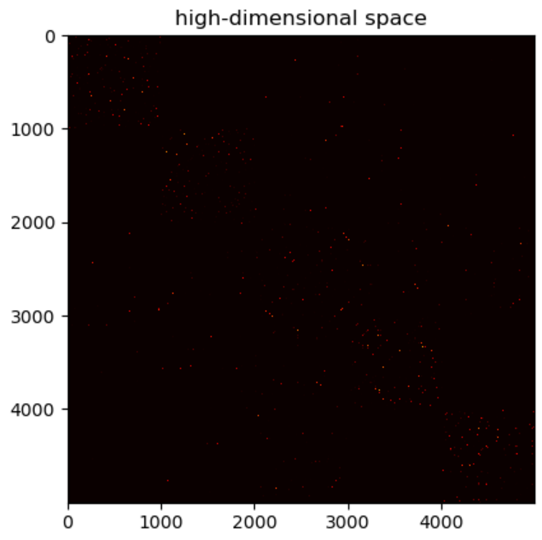


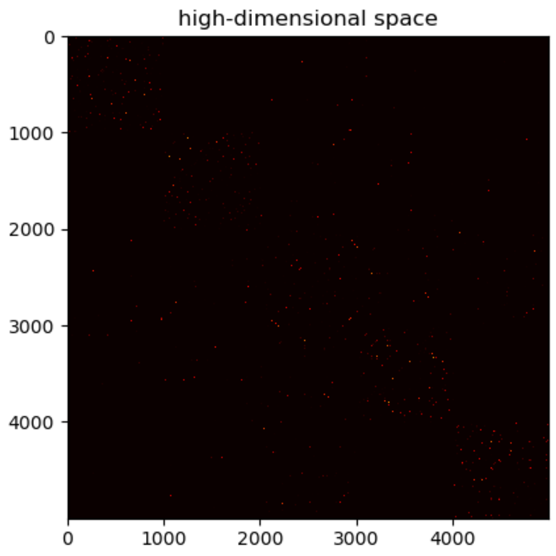
S-SNE



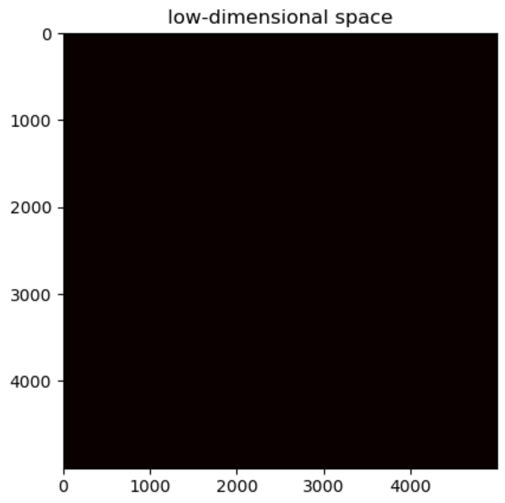
We can see very clearly the embedding of S-SNE has the crowding problems, which resulting in the occlusion problem, and the scale of output are also different, t-SNE ranges from -1,5e5 to 1,5e5 (sorry, I occasionally removed it from image above. It's for both axes); symmetric SNE ranges from -80 to 80. In handwriting digits dataset, the embedding space of tSNE is much larger than symmetric SNE, which is also make sense to property of t-distribution and Gaussian distribution.

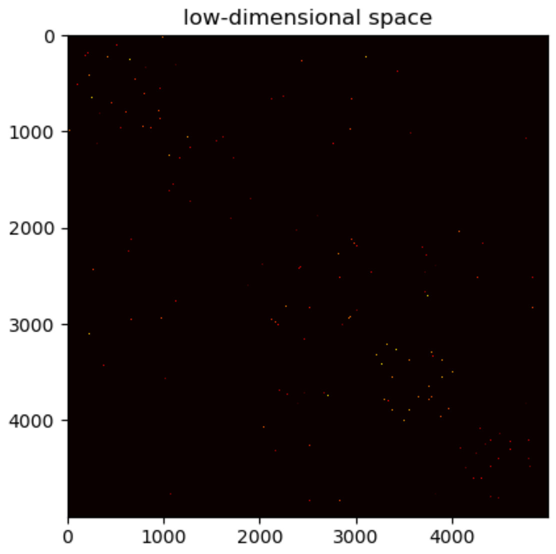
T-SNE **S-SNE**





T-SNE **S-SNE**





For the low dimension representation of S-SNE, besides the diagonal part we can see a lot of relations (compare to T-SNE) between different digits, which will result in the occlusion in the 2D space.

4. att_faces dataset contains in total 400 images (40 distinct subjects, 10 images per subject). You should use PCA to show the first 25 eigenfaces, and randomly pick 10 images to show their reconstruction (please refer to the lecture slides p.125).

Explanation of code.

In this section we use PCA for face-dataset.

First step: read input dataset:

```
att_faces = []
path = './att_faces/'
for i in range(1, 41):
  for j in range(1, 11):
    image = PIL.Image.open(path+'s{}/{}.pgm'.format(i, j)):
    att_faces += [np.array(image).reshape(1,-1)]
att_faces = np.concatenate(att_faces, axis = 0)
```

Second step: apply PCA for obtained data from previous step and get eigenface, which is a name for eigenvectors which are the components of the face itself. (take a look at the first task. The steps are the same.)

Third step: show the first 25 eigenfaces, Figure 4-1:

```
for i in range(5):
  temp = np.concatenate([np.concatenate(face_eigen.T.reshape(-1, 112, 92)[5*i:5*(i+1)], axis=1)], axis=0)
  plt.title('first 25 eigenfaces')
  plt.imshow(temp, cmap='gray')
  plt.show()
```

Fourth step: randomly pick 10 eigenfaces:

```
rand_set = np.random.randint(low=0, high=40, size=10)
rand_set = np.sort(rand_set)
temp = [att_faces[rand_set[i]*10:(rand_set[i]+1)*10] for i in range(10)]
rand_images = np.concatenate(temp, axis=0)
```

Fifth step: show selected images from previous step, Figure 4-2, and their reconstruction by using $y = x * W * W^T$, Figure 4-3:

```
for i in range(10):
  temp = np.concatenate([np.concatenate(rand_images.reshape(-1,112,92)[10*i:10*(i+1)], axis=1)], axis=0)
  plt.title('10 images')
  plt.imshow(temp, cmap='gray')
  plt.show()
for i in range(10):
  temp = np.concatenate([np.concatenate(np.dot(np.dot(rand_images, face_eigen), face_eigen.T).reshape(-1,112,92)[10*i:10*(i+1)], axis=1)], axis=0)
  plt.title('Reconstructed images')
plt.imshow(temp, cmap='gray')
plt.show()
```

Results.

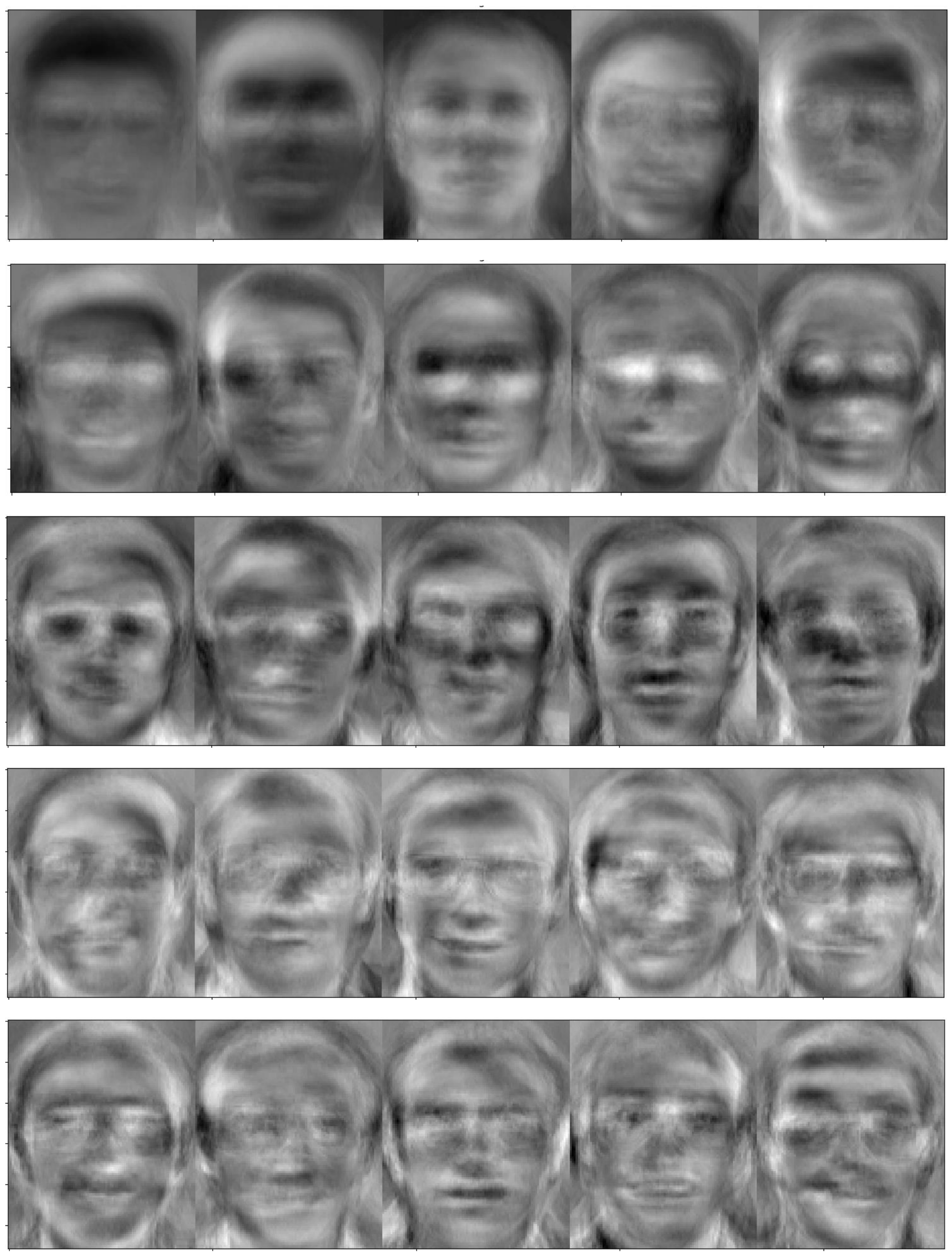


Figure 4-1

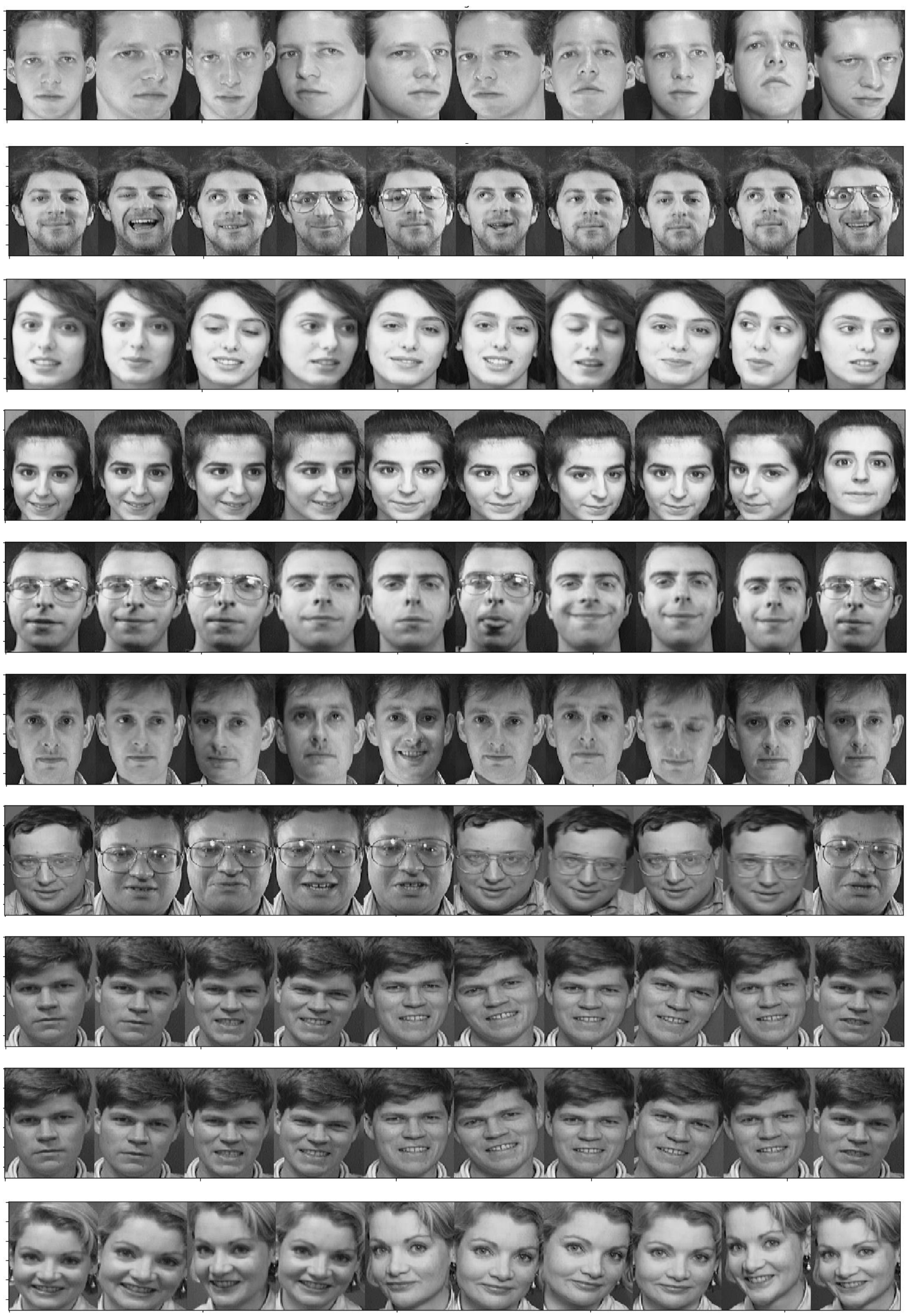


Figure 4-2

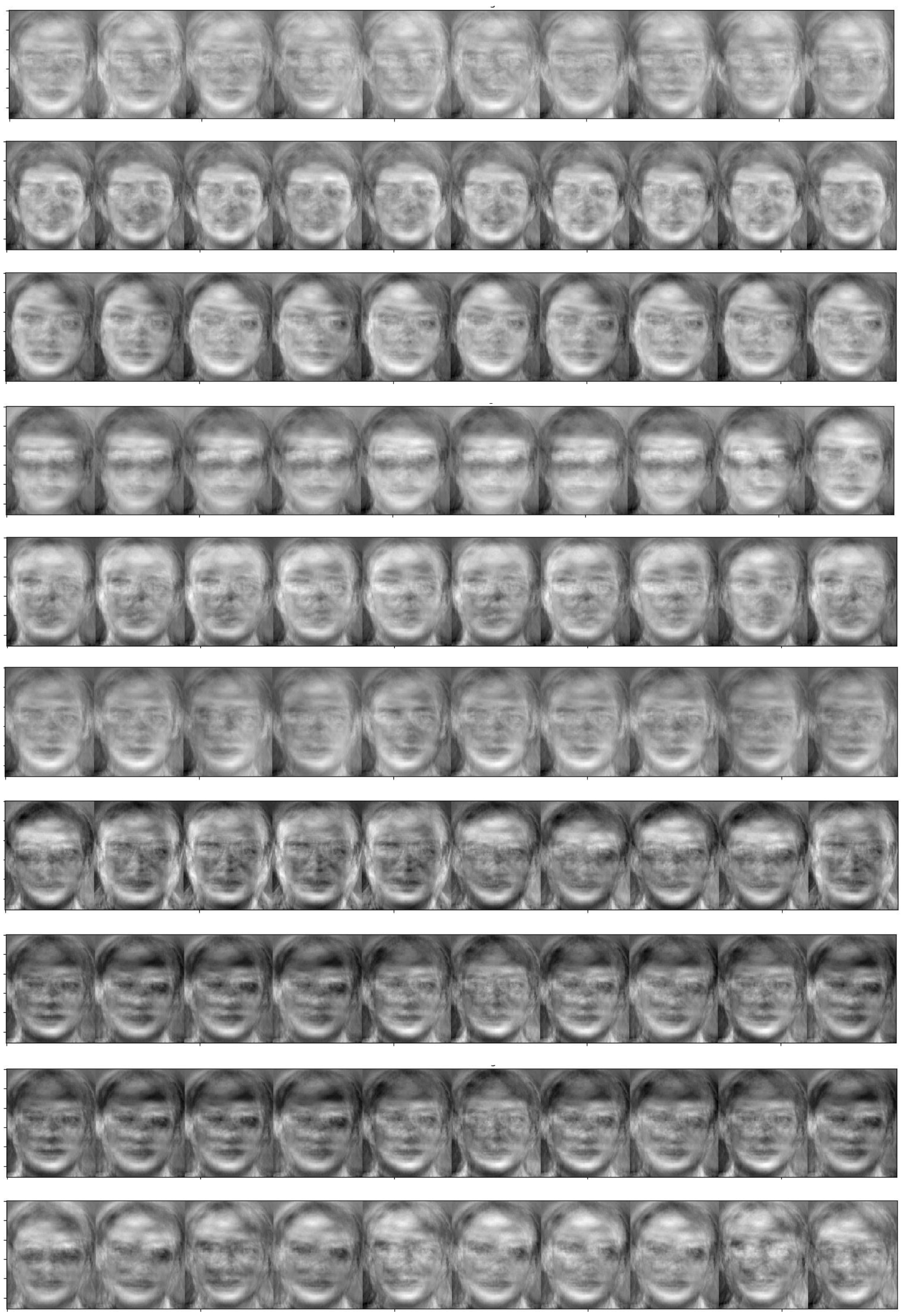


Figure 4-3

Discussion.

We used PCA for dimensionality reduction. It’s a good approach for this dataset because otherwise the work of program may take a lot of time. By using we can keep performance of this dataset.