PCA / LDA

When dataset's feature space grows into very high dimensions, there would be some useless feature. We prefer to extract the more important features for our dataset.

let

$$x_i = [\, x_{i1} \quad x_{i2} \quad \dots \quad x_{id} \,]$$

$$X_{n\times d} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

PCA (`PCA.py`)

`PCA` is a method to find a project matrix $W_{d\times k}$, which could project data $X_{n\times d}$ into $Z_{n\times k}$, i.e.,

$$Z_{n\times k} = X_{n\times d} W_{d\times k}$$

and $W_{d\times k}$ let $Z_{n\times k}$ has maximal variance, which hints that $Z_{n\times k}$ also has minimal square error between $X_{n\times d}$

$$\begin{aligned} \underset{W}{argmax}(\sigma^2(Z)) &= \frac{1}{n}\sum (z_i - \overline{Z})^T (z_i - \overline{Z}) \\ &= \frac{1}{n}\sum (x_i w_i - \overline{x}_i w_i)^T (x_i w_i - \overline{x}_i w_i) \\ &= \frac{1}{n}\sum ((x_i - \overline{x}_i) w_i)^T ((x_i - \overline{x}_i) w_i) \\ &= W^T \frac{1}{n}\sum ((x_i - \overline{x}_i)^T (x_i - \overline{x}_i)) W \\ &= W^T S W \ (\ diagonal\ value\ ) \end{aligned}$$

By `Rayleigh Quotient`, the first `k` large eigenvector of S is our W, (i.e., solving $Sv = \lambda v$)

Look at $S_{d\times d}$ matrix, it is time consuming in solving its eigen-problem when d is very large. We could do a little trick on it.

Let

$$\begin{aligned} \phi(X) &= X - \overline{X} \\ C_{n\times n} &= \frac{1}{n}(X - \overline{X})(X - \overline{X})^T = \frac{1}{n}\phi(X)\phi(X)^T \\ S_{d\times d} &= \frac{1}{n}(X - \overline{X})^T (X - \overline{X}) = \frac{1}{n}\phi(X)^T \phi(X) \end{aligned}$$

Then, solving

$$
\begin{aligned}
Cv &= \lambda v \\
\phi(X)^T Cv &= \lambda \phi(X)^T v \\
\phi(X)^T \frac{1}{n} \phi(X) \phi(X)^T v &= \lambda \phi(X)^T v \\
(\frac{1}{n} \phi(X)^T \phi(X)) \phi(X)^T v &= \lambda \phi(X)^T v \\
S(\phi(X)^T v) &= \lambda (\phi(X)^T v) \\
SW &= \lambda W
\end{aligned}
$$

So, we could solve eigen-problem in $n \times n$ dimension to get $W = \phi(X)^T V = (X - \overline{X})V$.

Then, we could project data such like XW and $X_{test}W$.

```
def __get_covariance(self, datas):
        n = datas.shape[0]
        if self.is_kernel:
            ...
        else:
            # This part use a little trick
            S = datas - np.sum(datas, axis=0) / n
            S = np.matmul(S, S.T) / n
            return S
...
def run(self):
        S = self.__get_covariance(self.datas)
        e_vals, e_vecs = self.__get_sorted_eigen(S, self.k)
        mean_datas = self.datas - np.sum(
            self.datas, axis=0) / self.datas.shape[0]
        if self.is_kernel:
            ...
        else:
            W = np.matmul(mean_datas.T, e_vecs)
            W /= np.sqrt(e_vals)
            pca_space = np.matmul(self.datas, W)
        return pca_space, W
```

```
...
pca = PCA(train_faces, k=25)
train_space, W_train = pca.run()
test_space = np.matmul(test_faces, W_train)
...
```

kernel PCA (`PCA.py`)

At the first, mapping datas into feature space.

$$
X_{n \times d} \rightarrow \phi(X)_{n \times m}
$$

and then try to calculate the covariance as `PCA` do.

$$
\begin{aligned}
cov(\phi(X)) &= \frac{1}{n}(\phi(X) - \frac{1}{n}\sum \phi(X))^T(\phi(X) - \frac{1}{n}\sum \phi(X)), (\ let\ \Psi(X) = \phi(X) - \frac{1}{n}\sum \phi(X)) \\
&= \frac{1}{n}\Psi(X)^T\Psi(X) \\
&= S
\end{aligned}
$$

We want to solve $SW = \lambda W$, but $\Psi(X)$ is not explict.

Assume X is $d \times n$,

$$
\begin{aligned}
Sw &= \lambda w \\
\frac{1}{n}XX^Tw &= \lambda w \\
\frac{1}{n\lambda}\sum x_i(x_i^Tw) &= w \\
\sum[\frac{1}{n\lambda}(x_i^Tw)]x_i &= w \\
\sum \alpha_i x_i &= w \\
XA &= w
\end{aligned}
$$

We could observ that the eigenvector is linear combinition of X, so we could rewrite the eigen-problem as following:
(here X is $n \times d$)

$$
\begin{aligned}
\frac{1}{n}\Psi(X)^T\Psi(X)w &= \lambda w \\
\frac{1}{n}\Psi(X)^T\Psi(X)\Psi(X)^TA &= \lambda\Psi(X)^TA \\
\frac{1}{n}\Psi(X)^T\Psi(X)\Psi(X)^TA &= \lambda\Psi(X)^TA \\
\frac{1}{n}\Psi(X)\Psi(X)^T\Psi(X)\Psi(X)^TA &= \lambda\Psi(X)\Psi(X)^TA \\
\frac{1}{n}K^cK^cA &= \lambda K^cA \\
\frac{1}{n}K^cA &= \lambda A
\end{aligned}
$$

What is K^c ?

$$
\begin{aligned}
let\ K &= \phi(X)\phi(X)^T \\
K^c &= \Psi(X)\Psi(X)^T \\
&= (\phi(X) - \frac{1}{n}\sum \phi(X))(\phi(X) - \frac{1}{n}\sum \phi(X))^T \\
&= \phi(X)\phi(X)^T - (\frac{1}{n}\sum \phi(X))\phi(X)^T - \phi(X)(\frac{1}{n}\sum \phi(X)^T) + \frac{1}{n^2}\sum \phi(X)\phi(X)^T \\
&= K - K1_n - 1_nK + 1_nK1_n,\ (\ 1_n\ is\ a\ n \times n\ matrix\ whose\ elements\ are\ all\ \frac{1}{n}\ )
\end{aligned}
$$

```
def __get_covariance(self, datas):
        n = datas.shape[0]
        if self.is_kernel:
            N1 = np.ones((n, n)) / n
            S = (datas - np.matmul(N1, datas) - np.matmul(datas, N1) +
                 np.matmul(np.matmul(N1, datas), N1))
            return S
        else:
            ...
```

How to project data?

$$
\begin{aligned}
\phi(X)W &= \phi(X)\Psi(X)^T A \\
&= \phi(X)(\phi(X) - \frac{1}{n}\sum \phi(X))^T A \\
&= (\phi(X)\phi(X)^T - \phi(X)(\frac{1}{n}\sum \phi(X)^T))A \\
&= (K - 1_n K)A
\end{aligned}
$$

```
def run(self):
        S = self.__get_covariance(self.datas)
        e_vals, e_vecs = self.__get_sorted_eigen(S, self.k)
        mean_datas = self.datas - np.sum(
            self.datas, axis=0) / self.datas.shape[0]
        if self.is_kernel:
            W = e_vecs
            W /= np.sqrt(e_vals)
            N1 = np.ones(self.datas.shape) / self.datas.shape[0]
            pca_space = np.matmul((self.datas - np.matmul(N1, self.datas)), W)
        else:
            ...

...

K_train = kernel(train_faces, train_faces)
K_test_train = kernel(test_faces, train_faces)
kpca = PCA(K_train, k=25, is_kernel=True)
train_space, alpha = kpca.run()
```

How about projecting new data X_{test} ?

$$
\begin{aligned}
\phi(X_{test})W &= \phi(X_{test})\Psi(X)^T A \\
&= \phi(X_{test})(\phi(X) - \frac{1}{n}\sum \phi(X))^T A \\
&= (\phi(X_{test})\phi(X)^T - \phi(X_{test})(\frac{1}{n}\sum \phi(X)^T))A \\
&= (K_{test\_train} - 1_{n:m\times n}K)A
\end{aligned}
$$

```
K_train = kernel(train_faces, train_faces)
K_test_train = kernel(test_faces, train_faces)
kpca = PCA(K_train, k=25, is_kernel=True)
train_space, alpha = kpca.run()
NM1 = np.ones(K_test_train.shape) / K_train.shape[0]
### Project testing data at next line
test_space = np.matmul(K_test_train - np.matmul(NM1, K_train), alpha)
```

LDA (`LDA.py`)

Not like `PCA` is projecting data into eigen-features space. `LDA` projecting data for maximize the distance between different labels/clusters, and minimize the distance whthin labels/clusters.

solving project matrix W as eigen-problem below:

$$S_W^{-1} S_B w = \lambda w, \ where$$

$$S_W = \sum_{j=1}^{k} S_j, \ S_j = \sum (x_i - m_j)(x_i - m_j)^T, \ m_j = \frac{1}{n_j} \sum_{i \in C_j} x_i$$

$$S_B = \sum_{j=1}^{k} S_{B_j} = \sum_{j=1}^{k} n_j (m_j - m)(m_j - m)^T$$

But S_W might singular when $n < d$, So We could find its pseudo inverse, instead.
In our work, n is actually smaller than d, so I find pseudo inverse directly.

```
class LDA():
    def __init__(self, datas, labels, k, is_kernel=False):
        self.datas = datas
        self.n = datas.shape[0]
        self.labels = labels
        self.k = k
        self.is_kernel = is_kernel

    def __count_labels(self):
        C = np.zeros((self.n, len(np.unique(self.labels))))
        for idx, j in enumerate(np.unique(self.labels)):
            C[self.labels == j, idx] = 1
        return C

    def __get_Sb_Sw(self, C):
        Mj = np.matmul(self.datas.T, C) / np.sum(C, axis=0)
        M = np.sum(self.datas.T, axis=1) / self.datas.shape[0]

        B = Mj - M[:, None]
        Sb = np.matmul(B * np.sum(C, axis=0), B.T)

        W = self.datas.T - np.matmul(Mj, C.T)
        Sw = np.zeros(Sb.shape)
```

```
        for group in np.unique(self.labels):
            w = W[:, self.labels == group]
            Sw += (np.matmul(w, w.T) / w.shape[1])
        return Sb, Sw

    def __get_sorted_eigen(self, A, k):
        eigenvalues, eigenvectors = np.linalg.eigh(A)
        sorted_idx = np.flip(np.argsort(eigenvalues))
        sorted_eigenvalues = []
        sorted_eigenvectors = []
        for i in range(k):
            vector = eigenvectors[:, sorted_idx[i]]
            sorted_eigenvectors.append(vector[:, None])
            sorted_eigenvalues.append(eigenvalues[sorted_idx[i]])
        sorted_eigenvalues = np.array(sorted_eigenvalues)
        sorted_eigenvectors = np.concatenate(sorted_eigenvectors, axis=1)

        return sorted_eigenvalues, sorted_eigenvectors

    def run(self):
        C = self.__count_labels()
        Sb, Sw = self.__get_Sb_Sw(C)
        #try:
        #    Sw_inv = np.linalg.inv(Sw)
        #except:
        #    Sw_inv = np.linalg.pinv(Sw)
        Sw_inv = np.linalg.pinv(Sw)
        obj_matrix = np.matmul(Sw_inv, Sb)
        value, vector = self.__get_sorted_eigen(obj_matrix, self.k)
        return np.matmul(self.datas, vector), vector
```

Kernel LDA (`LDA.py`)

Referrence from wikipedia.

> But I could not map testing data to the training's space correctly until the deadline coming, so my accurancy of face-reconition is low.....

K-Nearest Neighbor

Decide the data belone to the cluster which has the largest numbers datas in its k-nearest neighbors.

```
import numpy as np


class KNN():
    def __init__(self, train_datas, test_datas, labels, k=5):
```

```
        self.train_datas = train_datas
        self.test_datas = test_datas
        self.labels = labels
        self.k = k

    def __eucidiance(self, U, V):
        return np.matmul(U**2, np.ones(
            (U.shape[1], V.shape[0]))) - 2 * np.matmul(U, V.T) + np.matmul(
                np.ones((U.shape[0], V.shape[1])), (V.T)**2)

    def run(self):
        dist = self.__eucidiance(self.test_datas, self.train_datas)
        closet = np.argsort(dist, axis=1)

        y = []
        for i in range(self.test_datas.shape[0]):
            for j in range(self.k):
                y.append(self.labels[closet[i][j]])
        y = np.array(y)
        y = y.reshape(self.test_datas.shape[0], self.k)

        count = []
        for i in range(len(self.labels)):
            count.append(np.count_nonzero(y == self.labels[i], axis=1))
        count = np.vstack(count)
        y = np.argmax(count, axis=0)
        predict = [None] * self.test_datas.shape[0]
        for i in range(self.test_datas.shape[0]):
            predict[i] = self.labels[y[i]]

        return predict

...

knn = KNN(train_space, test_space, train_labels, k=5)
predict = knn.run()
```

Kernel (in `util.py`)

I use three different kernel, linear, polynomial and rbf.

```
def euclidean(u, v):
    return np.matmul(u**2, np.ones(
        (u.shape[1], v.shape[0]))) - 2 * np.matmul(u, v.T) + np.matmul(
            np.ones((u.shape[0], v.shape[1])), (v.T)**2)


def rbf(u, v, g=10**-11):
    return np.exp(-1 * g * euclidean(u, v))
```

```
def linear(u, v):
    return np.matmul(u, v.T)


def polynomial(u, v, g=0.7, coef0=10, d=5):
    return ((g * np.matmul(u, v.T)) + coef0)**d
```

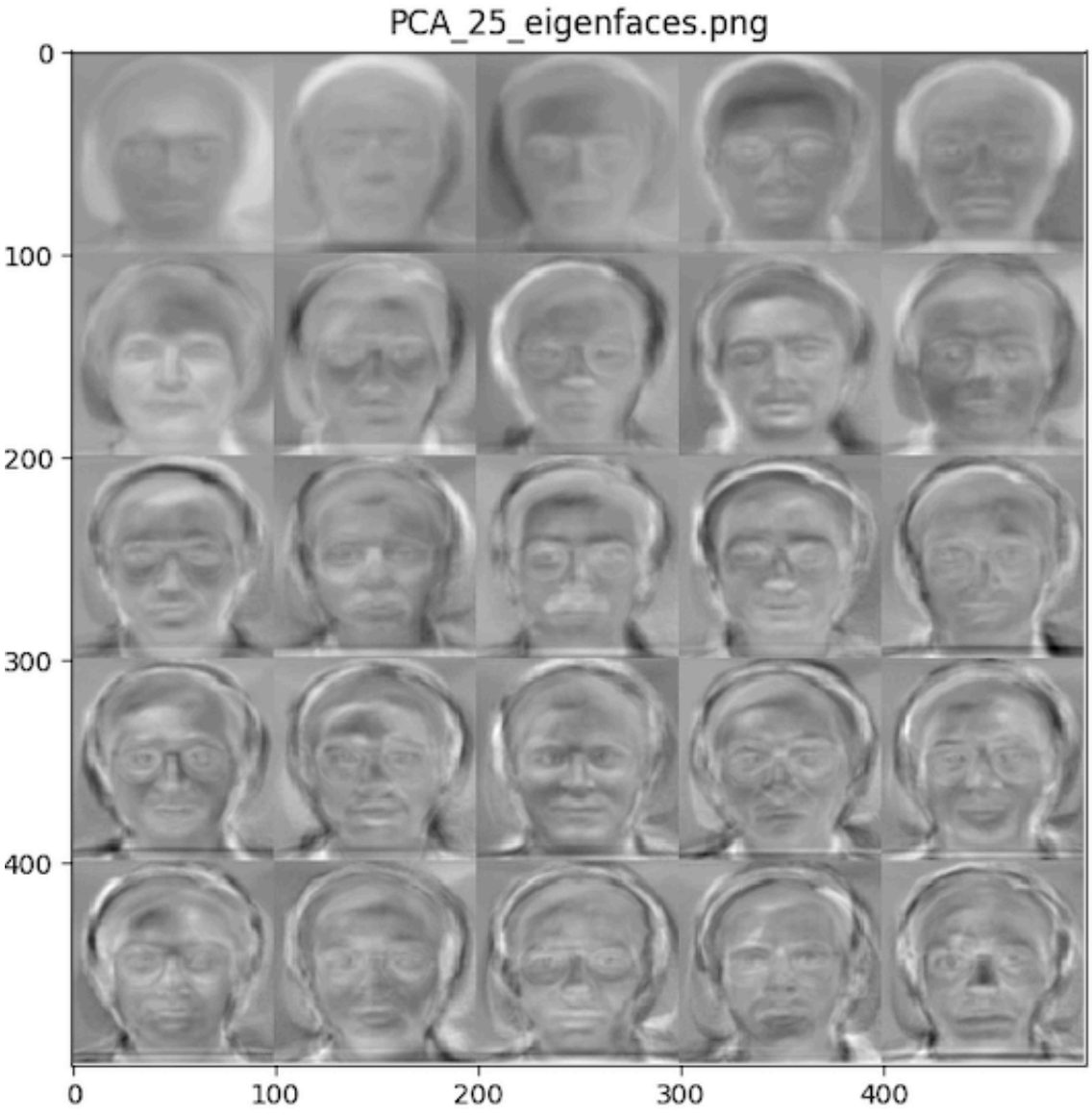
Other Methods in My Implementation (in `util.py`)

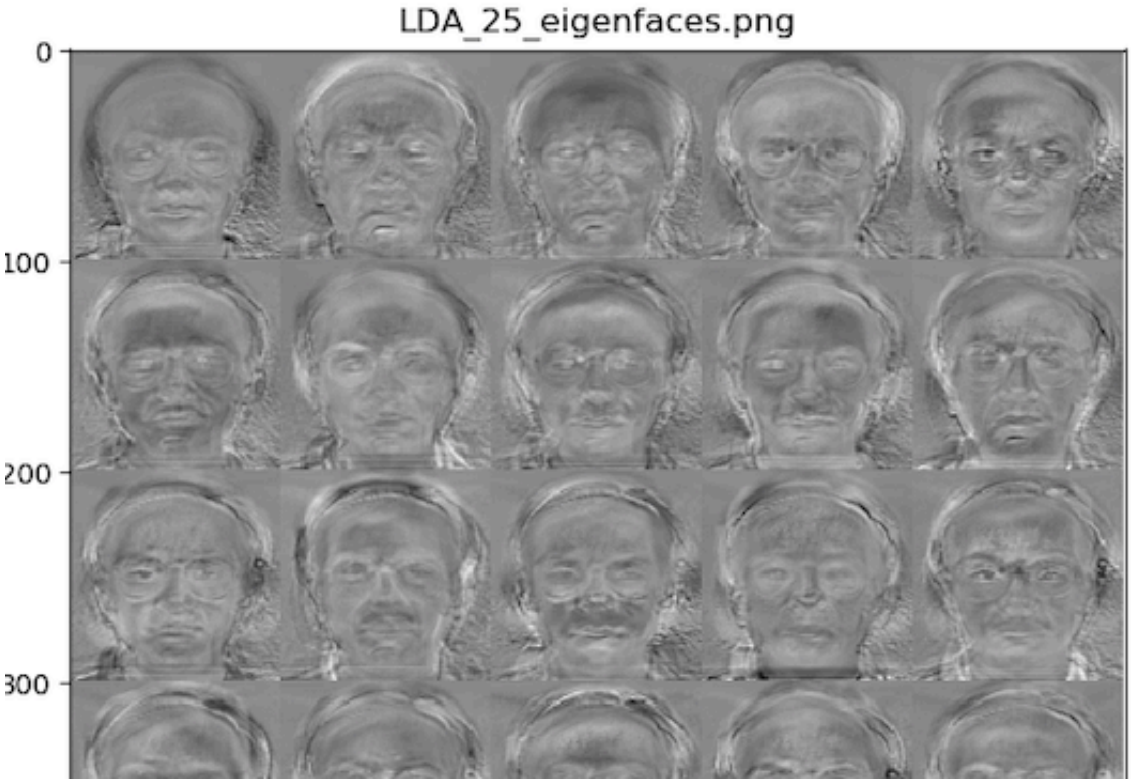
`read_face` method read the dataset and resize the image into `100*100`.
`show_face` method combine each face matrix in column and the row for visualize.

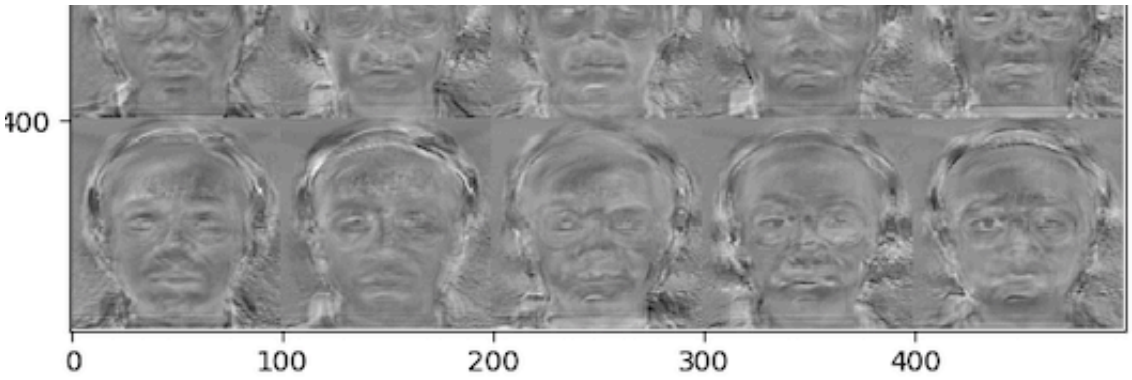
```
def read_faces(dir_path):
    faces = []
    labels = []
    for filename in os.listdir(dir_path):
        with PIL.Image.open(dir_path + filename) as im:
            im = im.resize((100, 100), PIL.Image.BILINEAR)
            faces.append([np.array(im).reshape(1, -1)])
            labels.append(filename.split('.', 1)[0])
    faces = np.concatenate(faces, axis=0)
    faces = faces.reshape(faces.shape[0], faces.shape[2])
    faces = faces.astype('int64')
    return faces, labels


def show_faces(faces, filename=None, col=5):
    #f = faces.reshape(-1,231,195)
    f = faces.reshape(-1, 100, 100)
    n = f.shape[0]
    all_faces = []
    for i in range(int(n / col)):
        all_faces.append([np.concatenate(f[col * i:col * (i + 1)], axis=1)])

    all_faces = np.concatenate(all_faces[:], axis=1)
    all_faces = all_faces.reshape(all_faces.shape[1], all_faces.shape[2])

    plt.figure(figsize=(1.5 * col, 1.5 * n / col))
    plt.title(filename)
    plt.imshow(all_faces, cmap='gray')
    plt.subplots_adjust(left=0.05, right=0.95, top=0.85, bottom=0.15)

    if (filename):
        plt.savefig('./output/' + filename)
    else:
        plt.show()
```

Result and Discussion

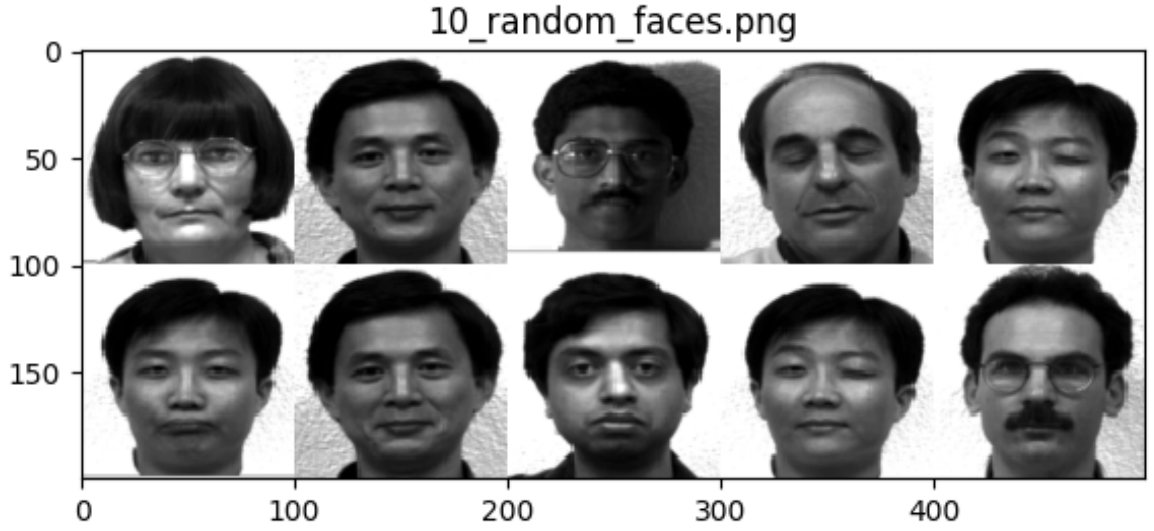
1. The first 25 eigenfaces and fisherfaces

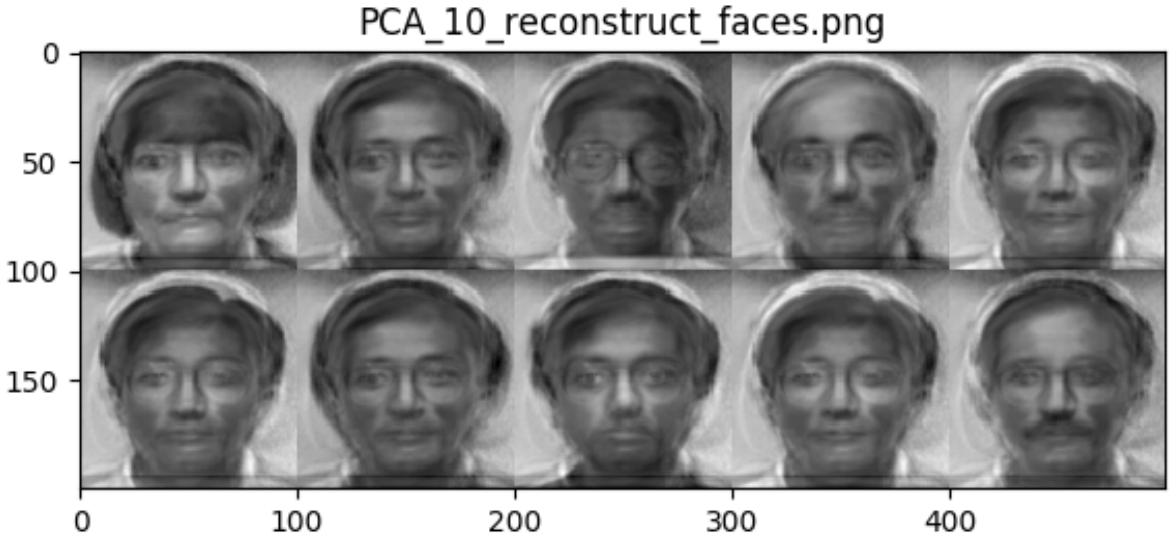


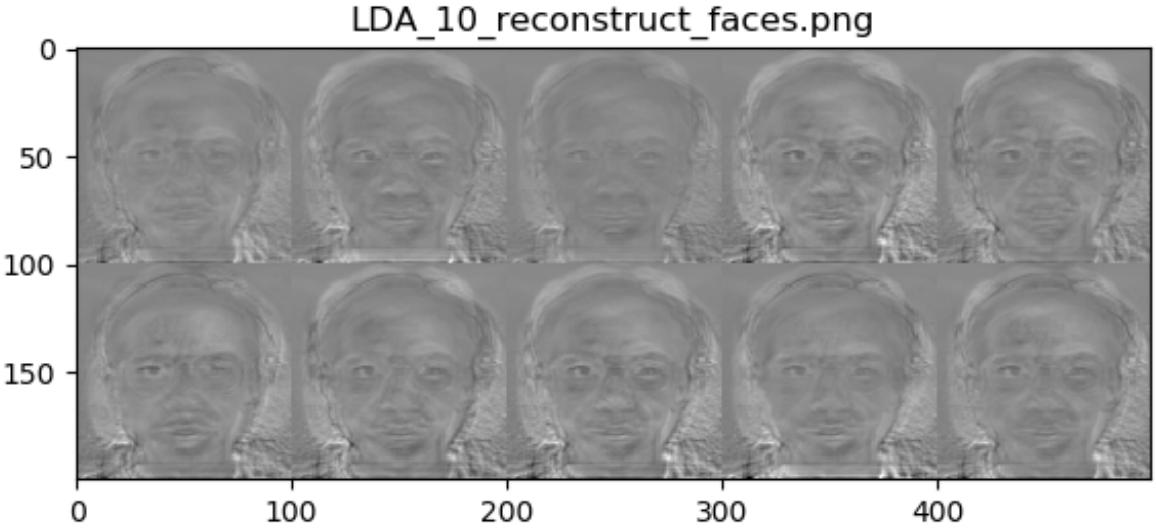



400
0
100
200
300
400

2. Reconstruct random 10 faces







3. and 4. Doing face recognition via PCA/KPCA and LDA/KLDA

```
(base) swchiu@gpuserv1:~/Embedding$ python3 main.py
PCA
Face-reconition accuracy: 27/30 = 90.00%

Kernel PCA
```

```
(rbf) Face-reconition accuracy: 27/30 = 90.00%
(polynomial) Face-reconition accuracy: 25/30 = 83.33%
(linear) Face-reconition accuracy: 27/30 = 90.00%

LDA
Face-reconition accuracy: 28/30 = 93.33%

Kernel LDA
(rbf) Face-reconition accuracy: 24/30 = 80.00%
(polynomial) Face-reconition accuracy: 18/30 = 60.00%
(linear) Face-reconition accuracy: 18/30 = 60.00%
```

`LDA` is better than `PCA`, I think the reason is that `LDA` could split the data more widly in its concept, but `PCA` only to mapping data for preserve the maximum variance.
In this case, we use `KNN` to classify data, which is based on euclidiean distance, and `LDA` could split widly in euclidiean distance.

`RBF` and `Linear` looks similar in `KPCA`, and `RBF` performs best in `KLDA`.
`Polynomial` performs worst in both `KPCA` and `KLDA`.

But the results of `KLDA` were not in my expectation. I think the problem is that I use the wrong mapping way to testing data. So I tried to derive the formula on my own, but I could not make a sence until the deadline coming.