SCHOOL OF ELECTRONICS & COMPUTER ENGINEERING

Computer Vision

FACE RECOGNITION WITH EIGENFACES AND FISHERFACES

Submitted By:
Huynh Van Thong
휴 완 텅
188261

Submitted To:
Prof. 이 칠 우

19 April, 2018

Contents

1 Eigenfaces

Let $X = \{x_1, x_2, \ldots, x_N\}, x_i \in \mathbb{R}^d$ for $i = \overline{1, n}$, be a set of N sample images, and assume that each image belongs to one of c classes $\{\omega_1, \omega_2, \ldots, \omega_c\}$

$$X = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1N} \\ x_{21} & x_{22} & \ldots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{d1} & x_{d2} & \ldots & x_{dN} \end{bmatrix}$$

Meanwhile, we have d variables, and each variable has N observation. The mean vector μ consists of the means of each variable

$$\mu_i = \frac{1}{N} \sum_{j=1}^{N} x_{ij},$$

and the variance-covariance matrix (total scatter matrix) S consists of the variances of the variables along the main diagonal and the covariances between each pair of variables in the other matrix positions.

$$S = \frac{1}{N-1}(x - \mu)(x - \mu)^T$$

The Eigenfaces use a linear transformation $W^T = \{w_1, w_2, \ldots, w_k\}$, k eigenvectors corresponding to k largest eigenvalues of S, mapping the original d-dimensional image space into an k-dimensional space, where $k < d$. The new feature vector $y_k \in \mathbb{R}^k$ are defined by

$$y_k = W^T(x - \mu). \tag{1}$$

```
    def pca(self, x_data, n_comp):
        """

        :param x_data: dxn array, n is number of sample, d is number of variable
        :param n_comp: number of components will be kept
        :return:
        """
        # Get mean of variables (by rows)
        x_mean = np.mean(x_data, axis=1).reshape(-1, 1)
        x_adjusted = x_data - x_mean
```

```python
        # x_adj_cov covariance matrix, size dxd
        x_adj_cov = np.cov(x_adjusted, rowvar=True)

        # Each column is a eigenvector corresponding to a eigenvalue, size dxd
        eg_value, eg_vector = np.linalg.eig(x_adj_cov)

        idx_inc_sort = np.argsort(eg_value)
        idx_dec_sort = np.flip(idx_inc_sort, axis=-1)

        # Matrix for projective, size dxn_comp
        self.mat_transf = eg_vector[:, idx_dec_sort[: n_comp]]
        # print("Size Matrix for pca transform ", self.mat_transf.shape)

        self.mu = x_mean

    def eigenface_fit(self, x_data, y_labels, n_comp):
        """

        :param x_data: dxn array, n is number of sample, d is number of variable
        :param n_comp: number of component will be kept
        :return:
        """
        self.pca(x_data, n_comp)
        x_adjusted = x_data - self.mu
        self.x_eg_projected = np.matmul(np.transpose(self.mat_transf), x_adjusted)
        self.y_eg_labels = y_labels
```

The simple method for determining which face class provides the best description of an input face image is to find the face class ω_ℓ that minimizes the Euclidian distance $\epsilon_\ell = ||y - y_\ell||$, where y, y_ℓ are new feature vector transformed using Equation 1 of input image and observed images.

```python
        self.y_eg_labels = y_labels

    def ef_face_predict(self, x_pred):
        """

        :param x_pred: dxn array, n is number of sample, d is number of variable

        :return: labels for n samples use eigen face or fisher face
        """
        # print(self.x_eg_projected.shape)
```

```python
# Project query image to eigen/fisher face space
x_pred_adjusted = x_pred - self.mu
x_pred_projected = np.matmul(self.mat_transf.transpose(), x_pred_adjusted)

idx = np.arange(x_pred.shape[1])
y_pred = np.zeros((x_pred.shape[1]), dtype=np.int32) - 1

for ix in idx:
    cur_x = x_pred_projected[:, ix].reshape(-1, 1)
```

2 Fisherfaces

The idea of this method is same classes should cluster tightly together, while different classes are as far away as possible from each other.
The within-class scatter S_w and between-class scatter S_b are calculated as:

$$S_w = \sum_{i=1}^{c} \sum_{x_j \in \omega_i} (x_j - \mu_i)(x_j - \mu_i)^T$$

$$S_b = \sum_{i=1}^{c} N_i(\mu_i - mu)(\mu_i - mu)^T$$

where μ is total mean

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i,$$

and N_i is the number of samples in class ω_i, μ_i is the mean of class ω_i, $i = \overline{1, c}$

$$\mu_i = \frac{1}{N_i} \sum_{x_j \in \omega_i} x_j$$

Fisher's classic algorithm now looks for a projection W that maximizes the class separability criterion:

$$W_{opt} = \arg\max_{W} \frac{|W^T S_b W|}{|W^T S_w W|} = [w_1, w_2, \ldots, w_m]$$

where $\{w_i | i = 1, 2, \ldots, m\}$ is the set of generalized eigenvectors of S_b and S_w corresponding to the m largest generalized eigenvalues $\{\lambda_i | i = 1, 2, \ldots, m\}$, i.e.,

$$S_b w_i = \lambda_i S_w w_i, \qquad i = \overline{1, m}. \tag{2}$$

The rank of S_w is at most $N - c$, and, in general, the number of images in the learning set N is much smaller than the number of pixels in each image d. Fisherfaces project the image set to a lower dimensional space so that the resulting within class scatter matrix S_w is non-singular. This is achieved by using PCA to reduce the dimension of the feature space to $N - c$ and then applying the standard FLD defined by (2) to reduce the dimension to $c - 1$. More formally, W_{opt} is given by

$$W_{opt}^T = W_{fld}^T W_{pca}^T,$$

where

$$W_{pca} = \arg\max_W |W^T S_T W|$$
$$W_{fld} = \arg\max_W \frac{|W^T W_{pca}^T S_b W_{pca} W|}{|W^T W_{pca}^T S_w W_{pca} W|}.$$

```python
    def fisherface_fit(self, x_data, y_labels):
        """

        Data dimension is much larger then the number of samples d >> n
        :param x_data: dxn array, n is number of sample, d is number of variable
        :param y_labels: labels of samples
        :return:
        """
        lb_unqiue, lb_unique_cnt = np.unique(y_labels, return_counts=True)
        n_classes = np.size(lb_unqiue)
        n_samples = x_data.shape[1]

        # Project with PCA to n_samples - n_classes feature space
        self.pca(x_data, n_samples - n_classes)
        x_adjusted = x_data - self.mu

        # Size of x_adjusted_pca: (n_samples - n_classes) x n
        x_adjusted_pca = np.matmul(np.transpose(self.mat_transf), x_adjusted)

        # Current dimension (n_samples - n_classes)
        cur_dimen = x_adjusted_pca.shape[0]
```

```
mu_total = np.mean(x_adjusted_pca, axis=1).reshape(-1, 1)

# Within-class scatter matrix
Sw = np.zeros((cur_dimen, cur_dimen))
# Betweem-class scatter matrix
Sb = np.zeros((cur_dimen, cur_dimen))

for c in lb_unqiue:
    c_idx = (y_labels == c)
    cur_samples = x_adjusted_pca[:, c_idx]

    if cur_samples.ndim == 1:
      cur_samples = cur_samples.reshape(-1, 1)

    # Mean of samples in class c
    mu_c = np.mean(cur_samples, axis=1).reshape(-1, 1)

    Sw = Sw + np.cov(cur_samples, rowvar=True)

    mu_c_total = mu_c - mu_total
    Sb = Sb + lb_unique_cnt[c] * np.matmul(mu_c_total, np.transpose(mu_c_tot

# Calculate eigenvectors, eigenvalues in Equation (2)
invSw = np.linalg.inv(Sw)
invSwSb = np.matmul(invSw, Sb)

eg_value, eg_vector = np.linalg.eig(invSwSb)

idx_inc_sort = np.argsort(eg_value)
idx_dec_sort = np.flip(idx_inc_sort, axis=-1)

self.mat_transf = np.matmul(np.transpose(eg_vector[:, : n_classes-1]),
                            np.transpose(self.mat_transf)).transpose()

# Calculate transformation matrix
```

3 Experiments

In this homework, *Labeled Faces in the Wild (LFW)* people dataset is used for training and testing. The dataset is loaded by class $fetch\_lfw\_people$ from module $sklearn.datasets$. Each picture is cropped and centered on a single face, resized to 62×47. There are 966 image in training set and 322 image in testing set.

	Egienfaces			Fisherfaces
Feature space	150	50	6	6
Accuracy	57.76	54.35	33.85	66.46

Table 1: Performance of eigenfaces and fisherfaces on test set.

Figure 1, 2, 3, 4 are the result of the prediction on a portion of the test set

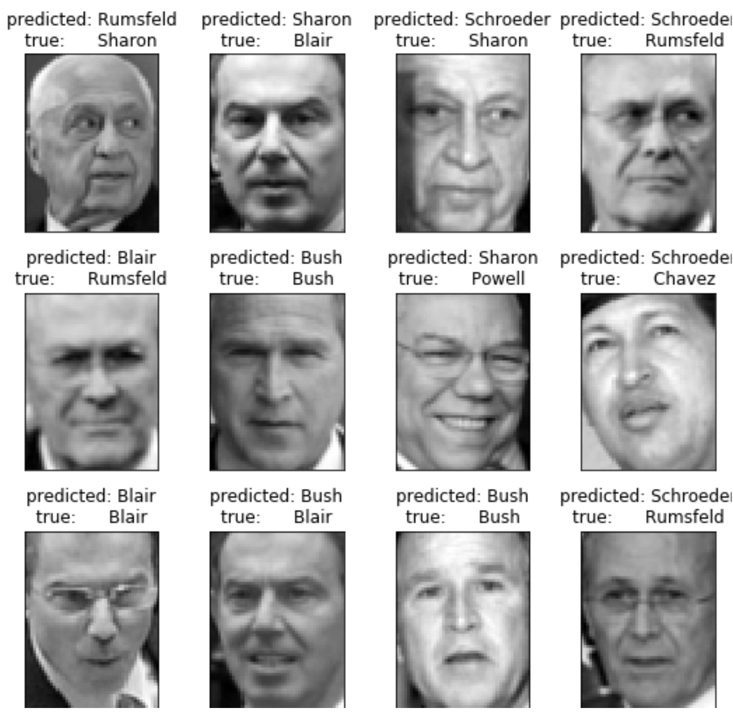

Figure 1: Eigenfaces with 6 feature space.

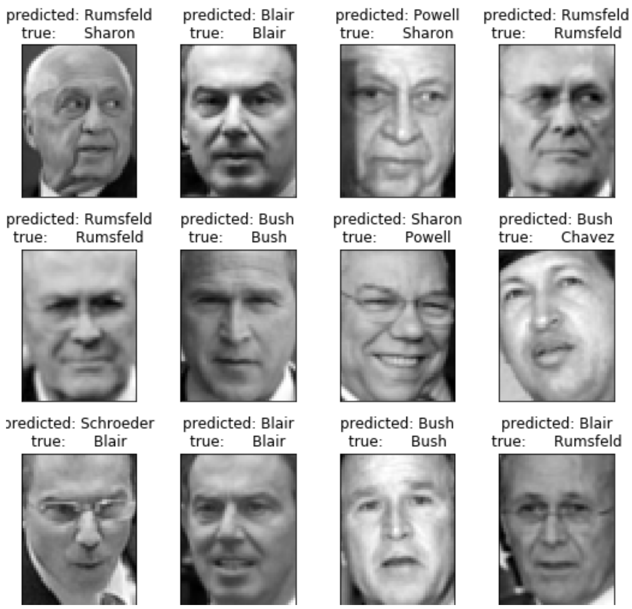

Figure 2: Eigenfaces with 50 feature space.

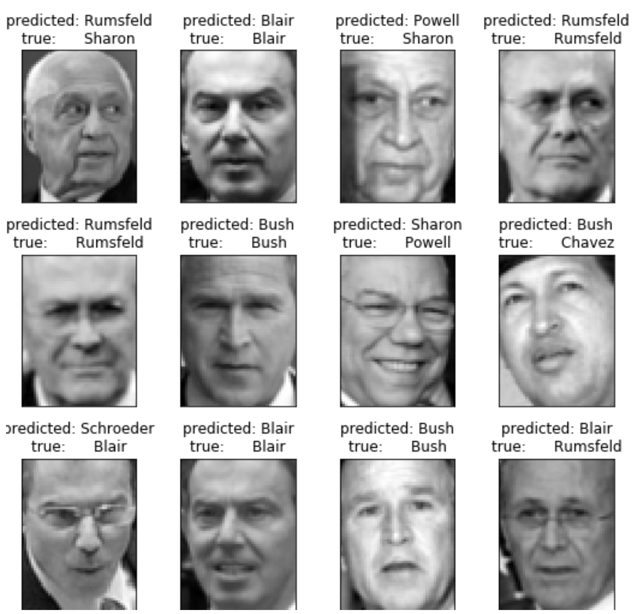

Figure 3: Eigenfaces with 150 feature space.

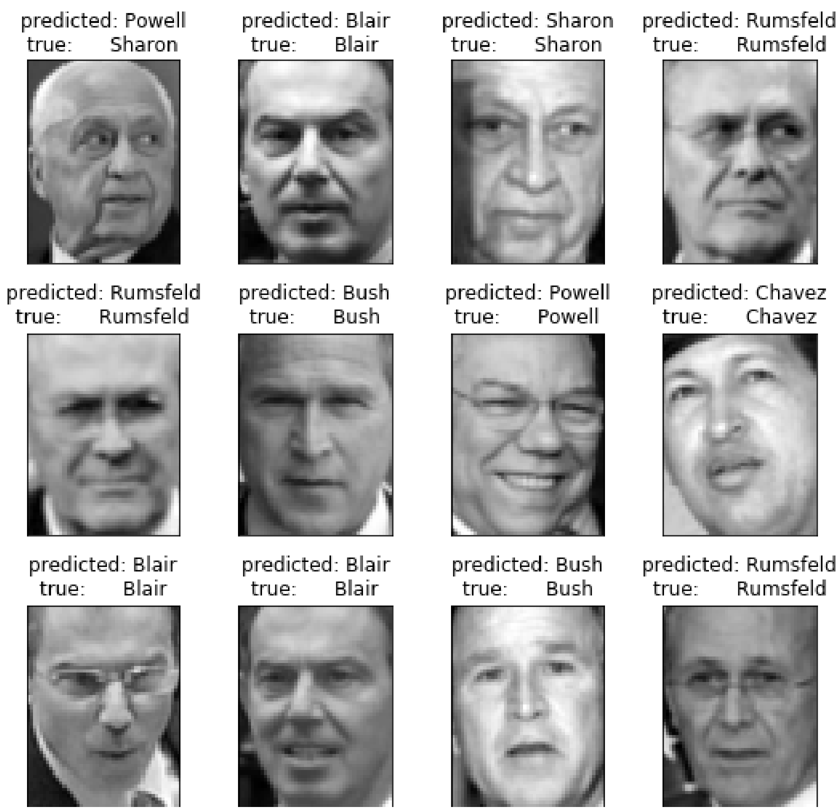

Figure 4: Fisherface.

References

[1] Turk, M. A., & Pentland, A. P. (1991, June). Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on* (pp. 586-591). IEEE.

[2] Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on pattern analysis and machine intelligence, 19*(7), 711-720.

[3] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct), 2825-2830.