Pattern Recognition - Dimension Reduction

0756079 陳冠聞

May 15, 2019

1 Introduction

Dimensionality reduction is the task that attempts to describe the data in lower dimensionality while keeping the characteristics of data, which is very useful for data compression, data visualization and can help the model to avoid the **curse of dimensionality**. In this program assignment, I will implement three methods of the dimensionality reduction, and use the classifiers and dataset in previous homework to evaluate the effect of dimensionality reduction.

2 LDA

Linear Discriminant Analysis (LDA) or Fisher Linear Discriminant (FLD) is aimed to project the high dimensional data to the lower dimensional space, and maximize the **class separability measure** after projection. The class separability measure in LDA is defined as the between-class scatter divided by the within-class scatter after projection. That is, LDA tries to maximize

$$J(w) = \frac{S^{Between}}{S^{Within}} = \frac{w^T S_B w}{w^T S_W w}. \tag{1}$$

And it can be formulated as constrained optimization problem

$$\text{maximize } w^T S_B w \tag{2}$$

$$\text{subject to } w^T S_W w = 1 \tag{3}$$

The solution can be obtained using Lagrange multiplier

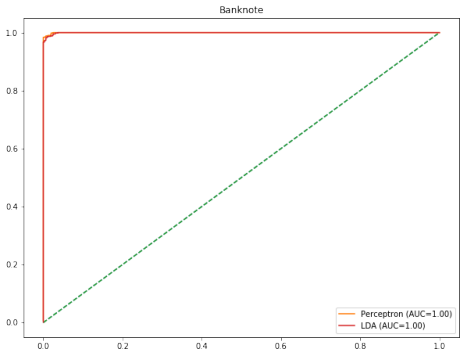
$$L(w, \lambda) = w^T S_B w - \lambda(w^T S_W w - 1) \tag{4}$$

$$\frac{\partial L(w, \lambda)}{\partial w} = 0 \Rightarrow 2S_B w - 2\lambda S_W w = 0 \tag{5}$$

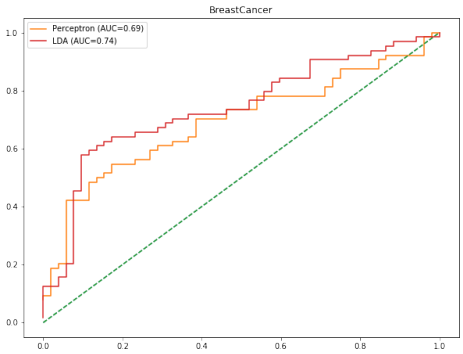
$$\Rightarrow S_B^{-1} S_W = \lambda w \tag{6}$$

That is, we can maximize the class separability measure by selecting the eigenvector corresponding to the max eigenvalue of $S_B^{-1} S_W$ as w.

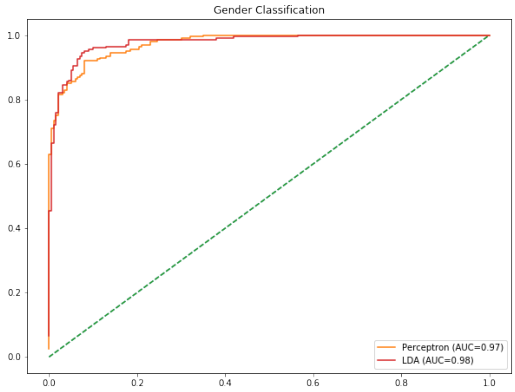
In fact, except for the dimension reduction, LDA can also be used as a classifier. Because it's a linear classifier, I compare the result of it with the Perceptron classifier implemented in previous homework on *Banknote* and *Breast Cancer* dataset. In this homework, the LDA classifier is first project the data to 1-D space, then determine the output class by the distance to the mean of each class, or it can be thought as first use LDA to do dimension reduction to 1-D then using naive-Bayes classifier to do classification. The result is shown in Figure 1. Unfortunately, we can observe nothing in *Banknote* dataset because it's too simple for both classifiers and they both achieve AUC=1.0. For the Breast Cancer dataset, the performance of LDA (AUC=0.74) is better than the Perceptron classifier (AUC=0.69).



(a) Banknote dataset



(b) Breast Cancer dataset



(c) Gender Recognition

Figure 1: Comparison of LDA and Perceptron Classifier

Table 1: Class separability measures for Banknote

Banknote	J1	J2	J3
Before Projection	1.1505	1.4714	4.4714
After Projection	1.4713	1.4713	1.4713

Table 2: Class separability measures for Breast Cancer

Breast Cancer	J1	J2	J3
Before Projection	1.0093	1.2358	9.2358
After Projection	1.2357	1.2357	1.2357

On the other hand, LDA is trying to preserves the most class separability, so I compare the class separability measures before and after the projection. I use three metrics to evaluate the class separability measure, which are $J_1 = \frac{trace(S_M)}{trace(S_W)}$, $J_2 = \frac{det(S_M)}{det(S_W)}$ and $J3 = trace(S_W^{-1} S_M)$

From the above table, we can see that the LDA preserve the class separability measure in term of J_1 and J_2. Note the the value of J_1, J_2 and J_3 is equal after projection since I project the data to 1-D space, and the value of J_1, J_2 and J_3 is all equal to S_M/S_W because it's a 1x1 matrix.

3 PCA

Principal Component Analysis (PCA) is aimed to project the high dimensional data to lower dimension space, and maximize the variance after the projection. The goal of maximizing the variance is because the higher the variance, the more likely the characteristics is preserved, but it is not always.

In this homework, I use the PCA to project the *Banknote*, *Breast Cancer* and *Iris* dataset to half as well as one-fourth of their original feature dimensions. Furthermore, I compare those features with three classifier implemented in previous homework.

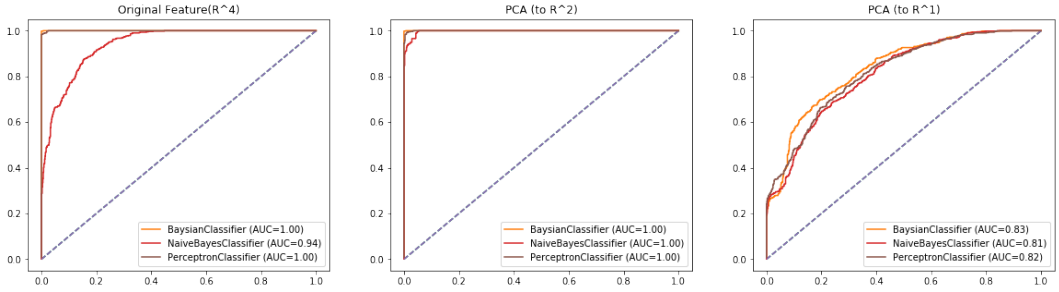


Figure 2: Result of Banknote

From the Figure 2, we can see that the model performances are similar in original feature space R^4 and projected feature space R^2. But the models perform worse in projected feature space R^1. What's interesting is that LDA

can still perfectly classify the Banknote after projected to 1-D space, which means that maximize the variance is not always the best choice to reduce data dimension, especially in the task of classification.

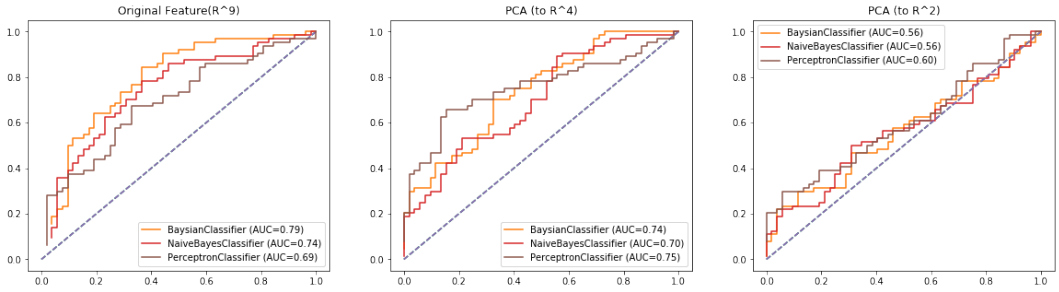


Figure 3: Result of Breast Cancer

From the Figure 3, we can see that both Naive-Bayes and Bayesian classifier perform worse in projected feature space R^4 than original featue space R^9 while Perceptron classifier perform better. For the projected feature space R^3, three classifier all perform worse than R^9 and r^4. Agian, LDA can achieve AUC=0.74 even in R^1 while PCA can only reach AUC=0.6 in R^2 imply that maximizing the variance is not the best choice for classification in lower dimensional space.

Table 3: 5-folds Cross-Validation Accuracy for Iris

Iris	Naive-Bayes	Bayesian	Perceptron
Original (R^4)	96.67%	96.00%	66.66%
PCA (R^2)	97.33%	90.00%	66.00%
PCA (R^1)	93.33%	93.00%	66.00%

As for the *Iris* dataset, it seems that the PCA performs very well since the classifiers can perform similar in lower dimension, which means that the useful information to do the classification is mostly preserved.

4 EigenFace

Eigenface is an computer vision algorithm proposed to deal with the face recognition problem. The basic idea of EigenFace is to do the eigenvalue decomposition of covariance of all face images, and use the eigenvector as eigenFace, then we can measure the similarity between two images in eigenspace by using eigenFaces to project image. In test time, we can classify the class of query image as the label of most similar face in training data.

In EigenFace, we treat each face as a feature vector $\vec{f}$. The first step of EigenFace is to compute the mean face $\vec{m}$ by compute mean of the sum of all face.

$$\vec{m} = \frac{1}{N}\sum_{i=1}^{N}\vec{f}_i \tag{7}$$

Then subtract each face from the mean face

$$\tilde{f}_i = \vec{f}_i - \vec{m} \tag{8}$$

Afterward, compute the covariance matrix C

$$C = \frac{1}{N}\sum_{i=1}^{N}\tilde{f}_i\tilde{f}_i^T \tag{9}$$

$$= \frac{1}{N}AA^T \tag{10}$$

where $A = [\tilde{f}_1\tilde{f}_2...\tilde{f_N}]$. Ideally, we do the eigenvalue decomposition of C, and we can get $AA^Tu_i = \lambda_i u_i$. But in most cases, it's computational expensive to solve the eigenvalue decomposition of the high dimensional $D \times D$ matrix AA^T (in this case, D=1600). So we do eigenvalue decomposition of $N \times N$ matrix A^TA instead, because the eigenvalues of A^TA are the same as the top-N eigenvalue of AA^T, and we only pick top-k eigenvectors as the EigenFace.

$$A^TAv_i = \mu_i v_i \tag{11}$$

$$\Rightarrow AA^TAv_i = \lambda_i Av_i \tag{12}$$

$$\Rightarrow u_i = Av_i \text{ and } \lambda_i = \mu_i \tag{13}$$

Figure 4: Top 25 EigenFaces (facesP1.bmp)

For the task of recognition, we first subtract the test face image $\vec{f}$ from mean face $\vec{m}$ in training data $\tilde{f} = \vec{f} - \vec{m}$. Then we project the vector to eigenspace by $w_i = u_i^T \tilde{f}$. Finally, we choose the label of training image with lowest Mahalanobis distance in eigenspace as the prediction for test image.

4.1 Face Recognition

For the face recognition, I use the data provided by TA, which is eighty 40x40 images of face (16 people, each have 5 face images). I use one face image for each person as test data (total 16 test data), and I generate horizontally flipped version of the face images for training data, resulting $8 \times 16 = 128$ training data. From the Figure 5, we can see that as the recognition accuracy grows when the number of eigenvectors K for the comparison of distance of two faces increases, and the accuracy saturate around 0.6 after K=10.

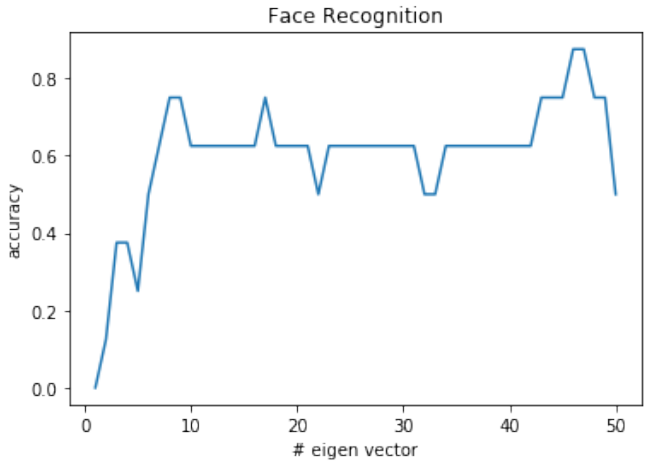


Figure 5: Face Recognition using EigenFace

4.2 Gender Recognition

For the gender recognition, I use the data provided by TA, which is one hundred 40x40 images for each gender. I use ten face image for each gender as test data (total 20 test data), and I generate horizontally flipped version of the face images for training data, resulting $90 \times 2 \times 2 = 360$ training data. From the Figure 6, we can see that as the recognition accuracy grows when the number of eigenvectors K for the comparison of distance of two faces increases, and the accuracy oscillate around 0.85 after K=10.

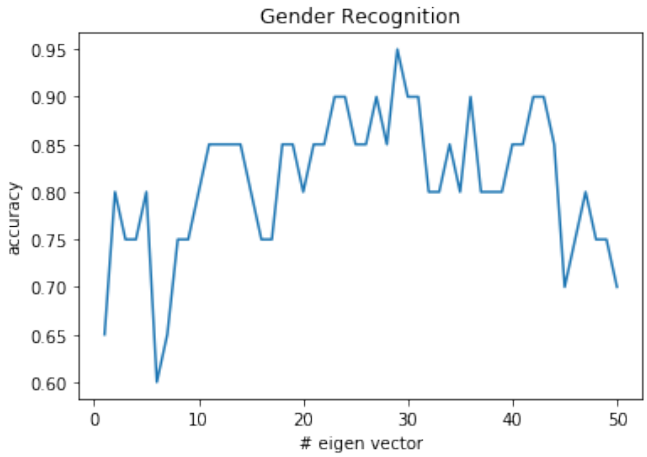


Figure 6: Gender Recognition using EigenFace

4.3 Face Reconstruction

Furthermore, EigenFace can also be used to do the Face Reconstruction. For a given face image, we first subtract it from the mean face, and project it to the eigenspace, then we add the mean face with each EigenFace multiply by the weight or coordinate in eigenspace to get the reconstructed face. From the Figure 7, we can see that with more eigenvectors used to reconstruct, the closer the reconstructed face to the original face.

Figure 7: Face Reconstruction using EigenFace

5 Appendix

```python
class LDA:
    def fit(self, X, Y):
        N, dim = X.shape
        X_high = np.copy(X)
        self.mean = X_high.mean()
        self.std = X_high.std()
        X_high = (X_high - self.mean) / self.std

        # Compute mean for each class (mj, nj)
        mean_vectors = []
        for c in set(Y):
            mean_vectors.append( np.mean(X_high[Y==c], axis=0) )
        self.mean_vectors = mean_vectors

        # Compute within-class scatter
        SW = np.zeros( (dim,dim) )
        for c, mv in zip(set(Y), mean_vectors):
            within_class_scattter = np.zeros((dim, dim))
            for xi in X_high[Y==c]:
                xi = xi.reshape(-1, 1) # make vec to mat
                mj = mv.reshape(-1, 1) # make vec to mat
                within_class_scattter += np.matmul(xi-mj, (xi-mj).T)
            SW += within_class_scattter

        # Compute between-class scatter
        SB = np.zeros( (dim,dim) )
        m = np.mean(X_high, axis=0).reshape(-1, 1)
        for c, mv in zip(set(Y), mean_vectors):
            nj = X_high[Y==c].shape[0]
            mj = mv.reshape(-1, 1) # make vec to mat
            SB += nj * np.matmul((mj-m), (mj-m).T)

        # Compute W using first k eigenvetor of inv(SW)*SB
        mat = np.dot(np.linalg.pinv(SW), SB)
        eigenValues, eigenVectors = np.linalg.eig(mat)
        idx = eigenValues.argsort()[::-1]
        eigenValues = eigenValues[idx]
        eigenVectors = eigenVectors[:,idx]
        W = np.real(eigenVectors[:, 0:self.n_components])
        W /= np.linalg.norm(W, axis=0)
        self.W = W
        return self
```

Figure 8: Code Snipper for LDA

```
class PCA:
    def __init__(self, n_components=2):
        self.n_components = n_components

    def transform(self, X):
        X_high = np.copy(X)
        mean_mat = np.tile(self.mean_vec, (X.shape[0],1))
        diff_mat = X_high - mean_mat
        # Project from high to low
        X_low = np.matmul(diff_mat, self.W)
        return np.real(X_low)

    def fit(self, X):
        X_high = np.copy(X)
        mean_vec = np.mean(X_high, 0)
        mean_mat = np.tile(mean_vec, (X.shape[0],1))
        diff_mat = X_high - mean_mat
        cov_mat = np.cov(diff_mat.T)
        self.mean_vec = mean_vec

        # Compute eigenpairs of cov mat
        eigenValues, eigenVectors = np.linalg.eig(cov_mat)
        idx = eigenValues.argsort()[::-1]
        W = eigenVectors[:,idx][:, :self.n_components]
        W = W * -1
        self.W = W
        return self`
```

Figure 9: Code Snipper for PCA

```python
class EigenFace:
    def fit(self, X, y, k=25):
        self.X = X # (num_data, vec_len)
        self.y = y # (num_data, )
        N, D = X.shape[0], X.shape[1]
        imgs_vec = X.T
        # Find mean vector
        mean_vector = X.mean(0).T
        self.mean_vector = mean_vector
        # plt.imshow(vec2img(mean_vector), cmap='gray'), plt.show() # Mean Face

        diff_imgs = imgs_vec - np.tile(np.array([mean_vector]).T, (1, N))
        T_trans_T = np.cov(diff_imgs.T)

        eigenValues, eigenVectors = np.linalg.eig(T_trans_T)
        idx = eigenValues.argsort()[::-1]
        eigenValues = eigenValues[idx]
        eigenVectors = eigenVectors[:,idx]
        # First K eigenvector
        eigenValues = eigenValues[:k]
        eigenVectors = eigenVectors[:, 0:k]

        # Get EigenFace
        A = diff_imgs
        eigenVectors = np.matmul(A, eigenVectors)
        eigen_faces_vec = np.copy(eigenVectors)

        for i in range(k):
            eigen_faces_vec[:, i] =  eigenVectors[:,i] / np.linalg.norm(eigenVectors[:,i])

        eigen_faces_vec = eigen_faces_vec.T  # (K, vec_length)
        self.eigen_faces_vec = eigen_faces_vec
        self.eigen_values = eigenValues
```

Figure 10: Code Snipper for EigenFace (part1)

```python
def predict(self, img, n_eigenvec=10):
    img = img.copy()
    k = self.eigen_faces_vec.shape[0]

    # Find input weight
    img_vec = img2vec(img)
    diff_vec = img_vec - self.mean_vector
    diff_face = vec2img(diff_vec)
    input_weights = []
    for i in range(n_eigenvec):
        input_weights.append(np.dot(diff_vec, self.eigen_faces_vec[i]))

    best_match_img = None
    best_match_error = 1e+10
    best_y = 0
    # Compare with training data
    for x, y  in zip(self.X, self.y): # (num_data, vec_len)
        img = vec2img(x)
        img_vec = img2vec(img)
        diff_vec = img_vec - self.mean_vector
        diff_face = vec2img(diff_vec)
        target_weights = []
        for i in range(n_eigenvec):
            target_weights.append(np.dot(diff_vec, self.eigen_faces_vec[i]))
        error = weight_distance(input_weights, target_weights, self.eigen_values)
        if error < best_match_error:
            best_match_img = img.copy()
            best_match_error = error
            best_y = y
    return best_match_img, best_y
```

Figure 11: Code Snipper for EigenFace (part2)