ENEE633 Project 1 Report: Face Recognition

Fengyu Liu (#116789028)
Department of Electrical and Computer Engineering, University of Maryland

Abstract

In this report, I describe how to realize face recognition through various algorithms, show some of the experiment results, and discuss their advantages, disadvantages and the various parameters involved. I also analyzed the general principles of selecting parameters in different classifiers, and completed two tasks: Identifying the subject label from a test image, and neutral vs. facial expression classification.

I. Introduction

The aim of this project is to implement different classifiers to achieve face recognition. I split the given set of faces and their corresponding labels into training and testing sets and use the training data to train my classifiers. Firstly I implement the Bayes' Classifier, k-NN rule, PCA and MDA. They can be trained to a classifier that can deal with more than 2 classes. For the classifier of 2 classes, I use Kernel SVM and Boosted SVM methods. I also did some experiments with them and the results are shown in the end of this report.

II. Datasets

In this project, there are three datasets: data.mat, pose.mat and illumination.mat. They are used in different tasks.
(1) data.mat: 200 subjects, 3 faces per subject, size: 24 x 21. It is used in both tasks.
(2) pose.mat: 68 subjects, 13 images per subject (13 different poses), size: 48 x 40. It is only used in the task of identifying the subject label.
(3) illumination.mat: 68 subjects, 21 images per subject (21 different illuminations), size: 48x40. It is only used in the task of identifying the subject label.

III. Task 1: Identifying the subject label from a test image

In the task of identifying the subject label, every subject is a class, so the three datasets have 200, 68, and 68 classes respectively. In each class, we take out a part of the data as the training set, and our task is to identify the remaining pictures as accurately as possible.

A. Bayes' Classifier

If we assume the underlying distribution is Gaussian, the Maximum Likelihood estimation gives us estimated values of mean values and covariance matrices (Homework 3.1), following

$$\hat{\mu}_{ML} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \,, \hat{\mathbf{\Sigma}}_{ML} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \hat{\mu}_{ML})(\mathbf{x}_n - \hat{\mu}_{ML})^T \tag{1}$$

Then, we can take these values to the discriminant functions of the Bayes' Classifier under Gaussian distribution, which can be written as

$$g_i(\mathbf{x}) = \mathbf{x}^T W_i \mathbf{x} + \omega_i^T \mathbf{x} + \omega_{i0} \tag{2}$$

where

$$W_i = -\frac{1}{2}\Sigma_i^{-1} \,, \omega_i = \Sigma_i^{-1}\mu_i \,, \omega_{i0} = -\frac{1}{2}\mu_i^T\Sigma_i^{-1}\mu_i - \frac{1}{2}ln|\Sigma_i| + lnP(\omega_i) \tag{3}$$

$P(\omega_i)$ can be ignored because I assume each class has the same probability. A problem in the simulation is that there are very few data points in each class, which means its number is smaller than the dimension. As a result, the covariance matrices are not full rank, and it makes us unable to find its inverse matrix. My solution of this problem is to add an identity matrix representing the noise in the covariance matrices. In the simulation, I can show that an appropriate Gaussian noise will not greatly affect the performance of the classifier.

B. k-Nearest Neighbors (k-NN) rule

In kNN rule, we need to calculate the Euclidean distance between a testing data point and every training point. Then, we assign it to the class corresponding to the mode of the labels of the nearest k data points. It should be noted that because our number of training samples is small, we should not choose a large value of k, so I choose $k = 1$ in most calculations. At the same time, if there are more than 1 modes, the mode function in MATLAB will give the smallest one. This will not affect our analysis and conclusions.

C. Principal Component Analysis (PCA)

In PCA, we need to follow the following 4 steps:
(1) Center the training data, and find the eigenvalues and eigenvectors of $\hat{\Sigma}$, following

$$\hat{\Sigma} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T \tag{4}$$

(2) Choose the largest m eigenvalues and the corresponding eigenvectors.
(3) Reduce the dimension of the training data to m using the eigenvectors obtained in the previous step.
(4) Use Bayes' classifier or k-NN rule to process the projected training data.

D. Multiple discriminant analysis (MDA)

Similar with PCA, in MDA, we need to follow the following 3 steps:
(1) Find the m largest eigenvalues and eigenvectors in the equation following

$$\Sigma_b \omega_i = \lambda_i \Sigma_w \omega_i \tag{5}$$

where Σ_w is the within-scatter matrix and Σ_b is the between-scatter matrix.
(2) Reduce the dimension of the data to m using the eigenvectors obtained in the previous step.
(3) Use Bayes' classifier or k-NN rule to process the projected training data.

Similar with the problem in Bayes' classifier, Σ_w is not full rank so it's impossible to calculate its inverse matrix. Here, I use two different methods:
1) As usual, we can add a Gaussian noise in the matrix. It will make Σ_w invertible.
2) We can calculate the eigenvectors of Σ_w first, and project the data to the subspace corresponding to Σ_w. In this space, Σ_w must be full rank, so we can easily calculate its inverse. What we need to pay attention to is that the dimension of the projected space is smaller than the dimension of the original space, thus the largest dimension of the result is M-1, where M is the rank of Σ_w.
Both methods can be implemented in my code. We can find the first method is better because the projection in the second method may be facing a very wrong direction. If so, the accuracy will drop significantly.

IV. Neutral vs. facial expression classification

In the task of neutral vs. facial expression classification, we only use data.mat. There are 2 classes, so we have 200 data points in each class. Then, we take out a part of the data as the training set, and our task is to determine as accurately as possible whether the remaining pictures are natural faces or expressive faces.

A. Kernel SVM classifier

In Kernel SVM classifier, we need to solve a dual optimization problem, following

$$\begin{aligned} min_{\alpha_i} & \frac{1}{2}\left(\sum_{i,j=1}^{N} \alpha_i y_i \alpha_j y_j x_i x_j\right) - \sum_{i=1}^{N} \alpha_i \\ s.t.\ & 0 \le \alpha_i \le C \\ & \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{6}$$

I use a famous SMO (Sequential minimal optimization) algorithm in the calculation, and the cross-validation can tell us the optimal values of the parameters.

B. Boosted SVM

In the AdaBoost algorithm, The problem for me is how to get N weak classifiers. My idaea is that every subset of the training data can give us a classifier by linear SVMs. As a result, In my calculations, I follow the following steps.
(1) Start from a small subset of the training data. (5-50 data points in each class). Get a weak classifier, and confirm it meets the requirements of AdaBoost.
(2) Find the weight of each data point in the next iteration by AdaBoost.
(3) Choose the training data with the largest weights $\omega_n^{(i)}$ as the training data for the support vector machine, and it can always give us a different weak classifiers in different interations because the weights are updated in every interation.
(4) Stop the interation at a fixed number or when all the training data is correctly classified.
In the end, we can get a strong classifier by the combination of several weak classifiers. When there is a lot of training data, we can see that the speed of this algorithm is much faster than the previous one, because only a small number of data points are used in each iteration.

V. Experiments

A. Identifying the subject label from a test image

#	Data	Train	Test	Bayes	1-NN rule	PCA&Bayes	PCA&1-NN	MDA&Bayes	MDA&1-NN
1	data	neutral&facial	illum.	64.0%	59.5%	63.5%	59.0%	62.5%	62.0%
2	data	facial&illum.	neutral	72.0%	55.5%	71.0%	55.5%	83.5%	79.5%
3	data	neutral&illum.	facial	66.5%	65.0%	66.5%	65.5%	75.0%	69.5%
4	pose	1∼6	7∼13	75.4%	69.8%	75.4%	69.8%	75.4%	75.6%
5	pose	7∼13	1∼6	96.6%	91.2%	96.6%	91.2%	93.9%	93.9%
6	illum.	1∼10	11∼21	86.2%	46.4%	86.2%	46.4%	69.5%	70.6%
7	illum.	11∼21	1∼10	99.9%	91.0%	99.9%	91.0%	99.7%	99.7%
	average	∼	∼	80.1%	68.3%	79.9%	68.3%	79.9%	78.7%

The above table is some data I got in my simulation. In PCA and MDA, I cut the dimension in half (floor(Nd/2) in MATLAB, where Nd is the dimension). Here are some discussions about the results:
(1) Bayes' classifiers is more acurrate in most cases, but it always take much more time than 1-NN rule classifier as well. This is because calculating the inverse of the matrix takes much more time than calculating the distance between data points.
(2) k-NN classifiers can save lots of time, but its accuracy will be lower than Bayes' classifiers in the same condition. This is not surprising, because we have already seen this in theory. Its accuracy is more unstable, too. I think that's due to the extent to the data in different classes cover each other. We can also see that its accuracy is related to the number k, which will be discussed in a later part.
(3) PCA and MDA can speed up calculation effectively. In my simulation, the dimension is much larger than the number of the samples in each class (Thousands of dimensions, but only tens or hundreds of data), so if we reduce the dimension to half, we can see that the accuracy of the two classifiers has no significant change or has improved. This tells us that dimensions should be appropriately compressed before the training data is processed.
(4) The accuracy of Bayes' classifier does not change much under PCA and MDA, but sometimes, the accuracy of 1-NN classifier does increase a lot under PCA and MDA. This may be because under the projection, the distances in different dimensions have different weights. Sometimes, it make 1-NN classifiers more accurate.

B. Neutral vs. Facial expression classification

#	Data	Train	Test	Kernel-RBF	Kernel-Poly	Linear-AdaBoost
1	data	1∼50	51∼100	90.0%	88.0%	89.0%
2	data	51∼100	1∼50	92.0%	88.0%	90.0%
3	data	1∼20	21∼100	87.5%	87.5%	86.3%
4	data	21∼100	1∼20	92.5%	92.5%	90.0%
	average	∼	∼	90.5%	89.0%	88.8%

The above table is some data I got in my simulation. Because there is not much training data, I set C to a relatively large number and used cross-validation to get the optimal parameters ($\sigma \approx 2.5$ and $\gamma \approx 1.2$). Here are some discussions about the results:
(1) The above three classifiers all have very good results. The difference of accuracy between the three is not obvious.
(2) As we can see, in general, RBF or polynomial kernel can give us a really good classification. However, when there is a lot of training data, it may take a lot of time to find the solution. At that time, we can use Linear Kernel with AdaBoost, which only consider the incorrectly classified elements (or elements with the largest weight) and get a weak classifier at each time. By AdaBoost, we can combine these weak classifiers to get a strong classifier and classify all the training data correctly.
(3) When there is not much training data, there is not much difference between RBF and polynomial kernel. When the amount of training data increases, RBF will have some advantages over the polynomial kernel. I think the reason is that when the amount of data is very small, there is no need to map the data to too high dimensions. In fact, at that time, the difference between polynomial kernel and linear kernel is also very small.
(4) For the case of Boosted SVM, because we only find and use the data with the largest weight in each step, maybe some of the data is missed, and the corresponding accuracy is slightly lower, but it is still in a satisfactory range. This algorithm is very suitable for situations with a lot of training data because it can save a lot of time and the weak classifiers generated in different iterations are almost impossible to be the same.

C. Influence of k in k-NN classifers

When we are considering the case $k \geq 2$, first, we must pay attention to the total number of training data in each class. For example, In data.mat, each class only have 2 training data points. Therefore, if k is set to a large value, it will bring very inaccurate results.

In the figure below, the blue, red and yellow lines respectively represent the case 1, case 4 and case 6 in part A. The purple line represents a new case which is similar with case 6, but there are 18 data points in training dataset and 3 data points in testing dataset for each class. (As mentioned earlier, it should be noted that when there are many modes, the mode function in MATLAB will select the smallest one, but this does not affect our conclusion.) From this figure, I believe when there is enough training data, we can increase the value of k appropriately, and when the amount of training data is insufficient, it should be set to $k = 1$.

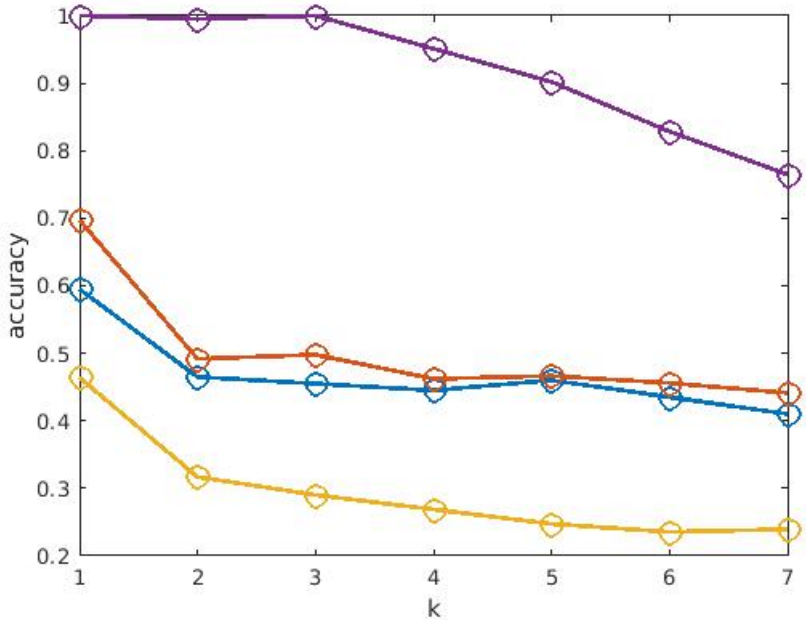

Fig. 1: The relationship between accuracy and k

D. Influence of the ratio between training data and testing data

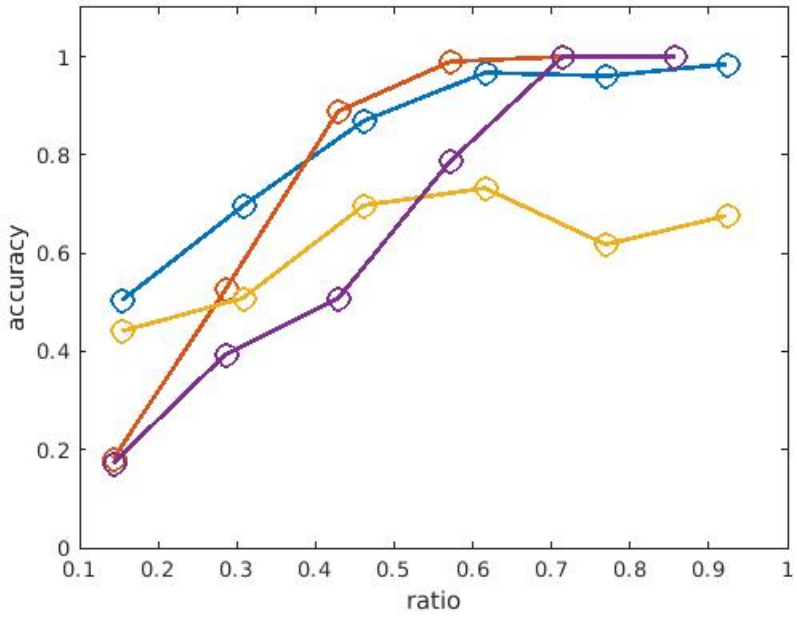

Fig. 2: The relationship between accuracy and ratio

In the figure above, the purple and yellow lines are calculated by 1-NN rule using pose.mat and illumination.mat, respectively. The blue and red lines are calculated by Bayes' Classifier using pose.mat and illumination.mat, respectively. In order to reduce the calculation time, I first use the PCA algorithm to reduce the data dimension to 200.

In principle, more training data will definitely lead to higher accuracy. However, sometimes, because we don't have much data, when most of the data is used as training data, the accuracy of the remaining small amount of testing data does not well represent the classifier's ability. Despite this, we can still see in the figure, for different classifiers, the accuracy generally increases with the increase of training data.

E. Influence of the noise matrix

As we mentioned earlier, in order to calculate the inverse of the matrix, we have added equal and incoherent Gaussian noise to each dimension. It is represented as an identity matrix. In the figure below, we can see the noise hardly affects the final accuracy.

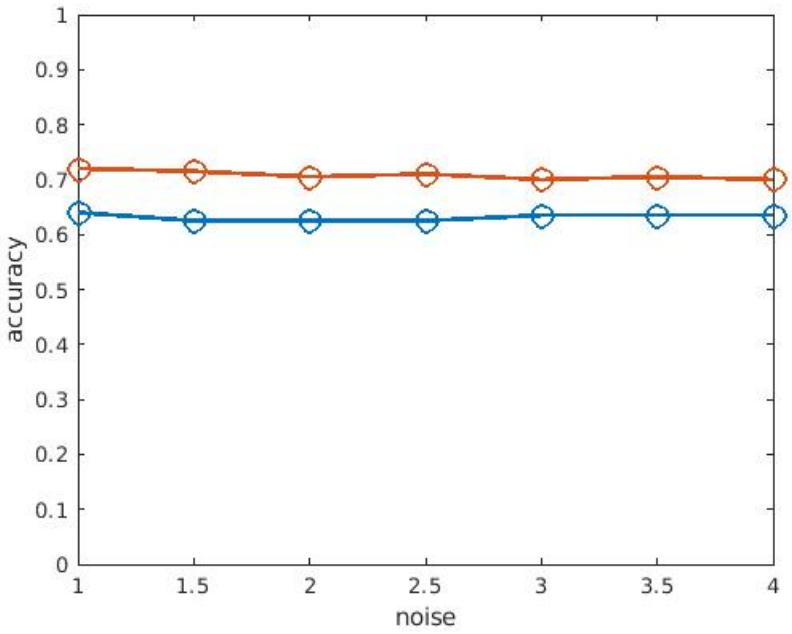

Fig. 3: The relationship between accuracy and noise

VI. Conclusion

In this project, I successfully used the Bayes' Classifier, k-NN rule, PCA and MDA algorithms to identify the subject label, and used Kernel SVM and Boosted SVM methods to get facial expression classifications. The experimental results have shown that these algorithms are all effective, but the calculation time and accuracy are different. The analysis of k in k-NN classifiers showed that we should choose the value of k appropriately according to the size of the training set, and the analysis of the ratio of training set to testing set showed that more data will make the classifier more accurate. Finally, I proved that for an irreversible matrix, we can make it invertible by adding noise and it will not influence its accuracy very much.