Chapter 02 패턴인식기 개발

[학습목표]
1장에서 패턴인식의 전반적인 개념에 대하여 알아보았다. 이 장에서는 이를 바탕으로 간단한 패턴인식기를 직접 만들어 보겠다. 먼저 간단한 2차원 데이터를 생성하고 분류해 봄으로써 1장에서 배운 개념들을 복습하고, 이어서 숫자를 인식하는 패턴인식기를 개발해 본다. 개발과정은 매트랩 코드로 제공되어 다음 장부터 본격적으로 제공되는 다양한 매트랩 코드들을 이해하기 위한 기초 연습을 수행한다.

2. 패턴인식기 개발

2.1. 2차원 데이터의 분류

먼저 간단한 패턴인식기를 만들어 보면서 1장에서 배운 개념들을 다시 한 번 돌아보고, 매트랩을 이용한 패턴인식에의 첫발을 내딛어보자. 이 절에서는 간단한 2차원 데이터에 대한 인식기를 만들어 볼 것이지만, 비록 간단한 문제라 하더라도 기본적으로 [그림 1-5]에 소개된 패턴인식기의 개발 과정을 따르게 되므로 전체적인 개념을 이해하기에는 충분할 것이다. 또한 2차원 데이터를 이용한 분류는 새로운 패턴인식 방법론을 개발하는 연구에 있어서도 매우 중요하다. 데이터의 분포 특성과 인식기에 의해 찾아지는 결정경계가 눈으로 확인될 수 있으므로, 문제에 적합한 분포를 가진 데이터 집합을 생성하여 방법론의 특성을 분석하기 위한 가장 기본적이면서 명확한 방법이기 때문이다. 따라서 이 절에서 독자들이 경험하게 되는 패턴인식기의 개발 경험은 이후 심화된 연구를 함에 있어서의 중요한 기초가 될 것이다.

2.1.1 데이터의 생성

먼저 [그림 1-5]의 첫 단계에 표시된 데이터 수집 과정이 필요하다. 그런데 여기서 대상으로 하는 2차원 데이터는 외부에서 주어지는 것이 아니라 목적에 맞추어 적절히 생성하면 되는 것으로, 여기서는 매트랩을 이용하여 간단히 생성해 보겠다. 한 가지 명확히 해야 할 것은, 이 책에서 소개하는 통계적 패턴인식기의 데이터의 집합은 기본적으로 어떤 확률분포를 따르고 관찰되는 데이터들은 그 분포를 따르는 시스템에서 확률적으로 생성되어진다는 가정을 바탕으로 하고 있다는 점이다. 따라서 인공적으로 데이터를 생성함에 있어서도, 먼저 연구 목적에 맞추어 데이터의 확률분포를 정의하고 이로부터 랜덤하게 데이터를 추출하는 과정을 거친다.

이 절에서는 간단한 분류 문제를 다루어보는 것을 목적으로 하므로, 가장 널리 사용되는 가우시안 분포를 따르는 데이터 집합을 생성해 보겠다. 두 개의 서로 다른 패턴을 분류하는 것이 목적이라면, 두 클래스 C_1과 C_2에 대하여 각각 서로 다른 확률분포를 정의해야 할 것이다. 여기서는 두 클래스가 모두 가우시안 분포를 따르되, 그 평균이 서로 다르도록 다음 식과 같이 정의한다.

$$p(\boldsymbol{x}|C_1) \sim G(\boldsymbol{\mu}_1, \Sigma_1), \quad p(\boldsymbol{x}|C_2) \sim G(\boldsymbol{\mu}_2, \Sigma_2) \qquad \text{[식 2-1]}$$
$$\boldsymbol{\mu}_1 = \begin{pmatrix} 3 \\ 5 \end{pmatrix},\ \Sigma_1 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix},\ \boldsymbol{\mu}_2 = \begin{pmatrix} 5 \\ 3 \end{pmatrix},\ \Sigma_2 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$$

데이터는 매트랩에서 제공하는 랜덤함수를 이용하여 간단하게 생성할 수 있는데, 관련된 매트랩 코드를 [프로그램 2-1]에 나타내었다. 매트랩에서는 C언어에서와 마찬가지로 균등분포(uniform distribution)를 따르는 데이터를 생성하는 함수 rand()를 제공할 뿐 아니라, 가우시안 분포를 따르는 데이터를 생성하는 함수 randn()을 제공한다. 그런데, 함수 randn()의 경우

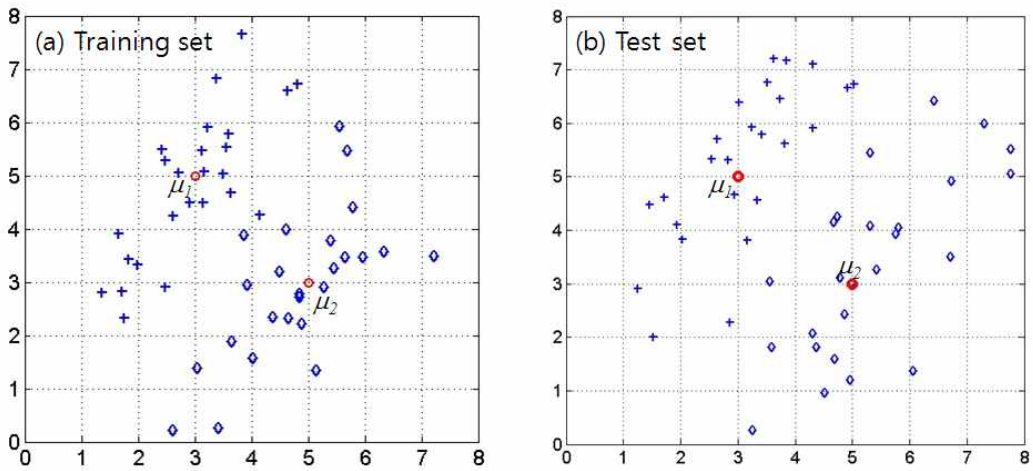
평균이 0이면서 공분산이 단위행렬인 표준정규분포를 따르는 데이터를 생성하게 되므로, 이것을 우리가 원하는 평균과 공분산을 가질 수 있도록 변환하는 과정이 필요하다. 평균의 경우 각 데이터에 평균벡터를 더해줌으로써 간단히 해결될 수 있으나, 공분산의 경우 공분산 행렬의 제곱근행렬을 데이터에 곱해주어야 한다. 여기서는 이를 위하여 매트랩에서 제공하는 제곱근함수 sqrtm()을 사용하였다.

프로그램 2-1 Data Generation

가우시안 분포를 따르는 데이터를 생성하고, 2차원 평면에 플롯하는 프로그램

```
001  N=25;                                %각 클래스의 데이터 개수
002  m1=repmat([3,5], N,1);               %클래스 C1의 평균을 가지는 행렬 m1
003  m2=repmat([5,3], N,1);               %클래스 C2의 평균을 가지는 행렬 m2
004  s1=[1 1; 1 2];                     %클래스 C1의 공분산 행렬 s1
005  s2=[1 1; 1 2];                     %클래스 C2의 공분산 행렬 s2
006  X1=randn(N,2)*sqrtm(s1) + m1;       %클래스 C1의 데이터 생성
007  X2=randn(N,2)*sqrtm(s2) + m2;       %클래스 C2의 데이터 생성
008  plot(X1(:,1), X1(:,2), '+');        %클래스 C1의 데이터 플롯("+" 모양)
009  hold on;
010  plot(X2(:,1), X2(:,2), 'd');        %클래스 C2의 데이터 플롯("◇" 모양)
011  save data2_1 X1 X2;
```

이렇게 생성된 각 클래스별 25개씩의 데이터를 2차원 평면에 점들로 표시한 산점도가 [그림 2-1a]에 나타나 있다. 그림에서 μ_1, μ_2로 표시된 두 클래스의 평균을 중심으로 각 데이터들이 공분산행렬에 맞추어 퍼져있는 것을 확인할 수 있다. 이 데이터 집합은 이후 단계에서 학습 데이터로 사용될 것이다. 같은 과정을 한 번 더 반복하면 같은 분포를 따르는 새로운 데이터 집합을 만들 수 있고, 이것을 테스트 데이터로 사용하여 개발한 인식기의 성능을 평가하는데 사용할 수 있다. 테스트 데이터는 [그림 2-1b]에 나타나있다.



[그림 2-1] 가우시안 분포를 따르는 두 클래스로 이루어진 데이터 집합

2.1.2 학습 : 데이터의 분포 특성 분석

학습 데이터와 테스트 데이터의 생성을 통해 데이터 집합이 주어지면, 이제 학습 데이터 집합의 분포 특성을 분석하여 분류를 위한 결정경계를 찾는 단계를 수행한다. [그림 1-5]에서는 이에 앞서 데이터 전처리와 특징추출 과정이 존재하나, 이 절에서 살펴보는 간단한 2차원 데이터에 대해서는 생략하겠다.
최적의 결정 경계를 얻기 위하여 데이터의 분포 특성을 분석하는 방법은 매우 다양하다. 기계학습 분야에서는 이 과정을 <학습(learning)>이라고 부르는데, 간단한 통계량을 계산하는 것에서부터 복잡한 결정경계를 반복탐색에 의해 찾아가는 방법에 이르기까지 다양한 학습방법들이 개발되어 왔다. 이 절에서는 가장 간단하고 직관적인 방법을 소개할 것이며, 이 후 다양한 방법들이 이 책 전체에서 걸쳐 소개될 것이다.

프로그램 2-2 Data Analysis
학습 데이터에 대한 평균과 공분산을 계산하는 프로그램

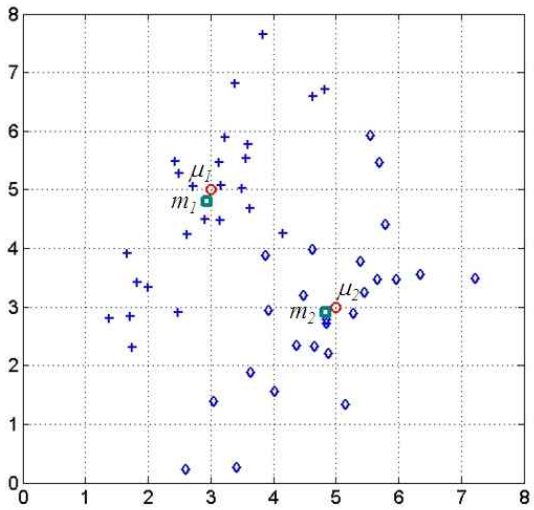
```
001 load data2_1                    %저장된 데이터를 불러들임
002 m1 = mean(X1);                  %데이터 행렬의 평균 계산
003 m2 = mean(X2);
004 s1 = cov(X1);                   %데이터 행렬의 공분산 계산
005 s2 = cov(X2);
006 save mean2_1 m1 m2 s1 s2;       %계산된 평균과 공분산을 저장해둠
```

데이터의 분포 특성을 나타내는 가장 대표적인 통계량으로는 평균과 분산을 들 수 있겠다. 특히 데이터 집합이 가우시안 분포를 따른다면 두 통계량만으로 구체적인 확률밀도함수가 얻어진다. 이 절에서는 학습 데이터 집합에 대한 표본 평균과 표본 분산을 계산하여 이를 바탕으로 분류를 위한 결정경계를 정해 보겠다. [그림 2-1a]에 나타난 학습 데이터는 입력값과 함께 각 데이터가 어떤 클래스에 속하는지도 함께 주어진 경우이므로, 각 클래스 별로 평균과 공분산 행렬을 계산할 수 있을 것이다. [프로그램 2-2]는 파일(data2_1.mat)에 저장된 학습 데이터를 불러들여 각 클래스의 평균과 공분산을 계산하는 것이다. 이를 통해 얻어진 평균($\boldsymbol{m}_1, \boldsymbol{m}_2$)과 공분산($\mathrm{S}_1, \mathrm{S}_2$)은 다음과 같다.

$$\boldsymbol{m}_1 = \begin{pmatrix} 2.94 \\ 4.80 \end{pmatrix},\ \mathrm{S}_1 = \begin{pmatrix} 0.86 & 0.99 \\ 0.99 & 1.93 \end{pmatrix},\ \boldsymbol{m}_2 = \begin{pmatrix} 4.83 \\ 2.91 \end{pmatrix},\ \mathrm{S}_2 = \begin{pmatrix} 1.14 & 0.97 \\ 0.97 & 1.89 \end{pmatrix} \qquad \text{[식 2-2]}$$

이 값을 보면, 데이터 생성 시에 가정한 확률분포의 실평균($\boldsymbol{\mu}_1, \boldsymbol{\mu}_2$)과 공분산($\Sigma_1, \Sigma_2$)과는 어느 정도 차이가 있음을 알 수 있는데([그림 2-2]참조), 이러한 확률적 불확실성이 학습과 관련된 여러 문제(예를 들어 학습오차와 테스트오차의 차이나 과다 적합의 문제 등)를 야기하게 된다. 그러나 인식기의 개발 단계에서 얻을 수 있는 데이터는 학습 데이터뿐이므로, 확률적 불확실성을 감수하면서도 학습 데이터로부터 추정된 값을 사용하여 결정경계를 찾아주어야 하며, 따라서 학습데이터만을 이용하여 과다적합을 피하면서 일반화 오차를 줄이는 결절

경계를 찾는 여러 방법들이 연구되고 있다.



[그림 2-2] 학습 데이터로부터 추정된 평균($\boldsymbol{m}_1, \boldsymbol{m}_2$)과 실평균($\boldsymbol{\mu}_1, \boldsymbol{\mu}_2$)

2.1.3 분류 : 결정경계의 설정

이어서 학습 데이터에 대한 분석 결과를 바탕으로 간단한 분류 기준(결정경계)을 찾는 단계를 거친다. 앞 절에서 얻어진 통계량은 평균과 공분산뿐이지만, 평균만으로도 간단한 결정경계를 찾을 수 있다. 먼저 [그림 2-3]과 같이 새로운 데이터 하나가 주어졌다고 가정하자. 새로운 데이터 $\boldsymbol{x}_{new}$가 두 클래스 중 어디에 속하는지 판단하는 가장 직관적인 방법으로는, 각 클래스의 평균($\boldsymbol{m}_1, \boldsymbol{m}_2$)과의 거리를 계산하여 거리값이 적은 쪽의 클래스로 할당하는 것을 생각해 볼 수 있다. 이차원 공간상의 두 점 $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ 간의 거리를 $d(\boldsymbol{x}_1, \boldsymbol{x}_2)$로 나타내면, 결정경계를 나타내는 함수 $g(\boldsymbol{x})$=0는 다음 식과 같이 정의할 수 있다.

$$g(\boldsymbol{x}) = d(\boldsymbol{x}, \boldsymbol{m}_2) - d(\boldsymbol{x}, \boldsymbol{m}_1) = 0 \qquad \text{[식 2-3]}$$

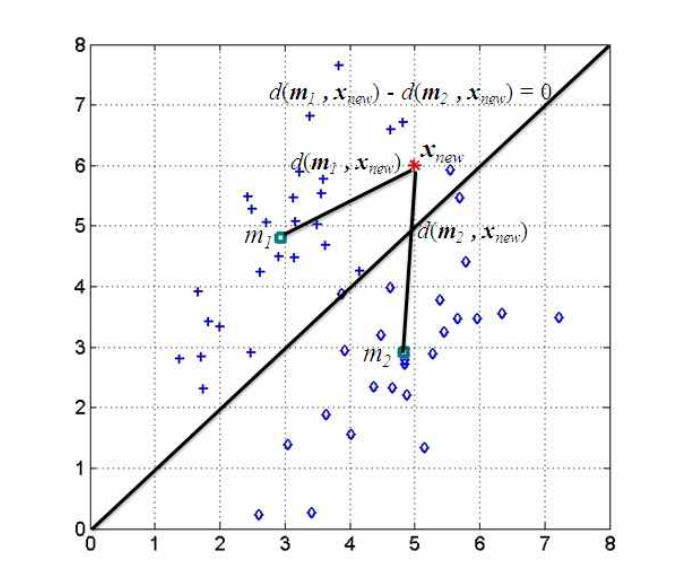
이 결정경계에 의하여 분류를 수행하는 것은, 다음과 같이 클래스 라벨을 결정해 주는 것을 의미한다.

$$y(x) = \begin{cases} 1 & \text{if } g(\boldsymbol{x}) > 0 \\ -1 & \text{if } g(\boldsymbol{x}) < 0 \end{cases} \qquad \text{[식 2-4]}$$

또한, 거리함수 $d(\boldsymbol{x}_1, \boldsymbol{x}_2)$를 두 벡터 차의 이차노름(유클리디안 거리의 제곱)으로 정의하면 [식 2-3]의 결정경계는 다음과 같은 구체적인 식으로 풀어 쓸 수 있다.

$$\begin{aligned} d(\boldsymbol{x}, \boldsymbol{m}_2) - d(\boldsymbol{x}, \boldsymbol{m}_1) &= (\boldsymbol{x} - \boldsymbol{m}_1)^T(\boldsymbol{x} - \boldsymbol{m}_1) - (\boldsymbol{x} - \boldsymbol{m}_2)^T(\boldsymbol{x} - \boldsymbol{m}_2) \\ &= -2(\boldsymbol{m}_1 - \boldsymbol{m}_2)^T\boldsymbol{x} + \boldsymbol{m}_1^T\boldsymbol{m}_1 - \boldsymbol{m}_2^T\boldsymbol{m}_2 = 0 \end{aligned} \qquad \text{[식 2-5]}$$

즉, 이를 2차원 공간상의 그래프로 나타내면 두 평균의 차로 이루어진 벡터 $\boldsymbol{m}_1 - \boldsymbol{m}_2$를 법선벡터로 가지고 두 벡터의 중점을 지나는 직선이 됨을 알 수 있다. [그림 2-3]에는 [식 2-2]에서 주어진 구체적인 평균값을 [식 2-5]에 대입하여 얻어지는 결정경계를 나타내었다. 이 결정경계를 이용하여 새로운 데이터 $\boldsymbol{x}_{new}$를 분류하면, $g(\boldsymbol{x})$=0 보다 위쪽에 있으므로, $y(\boldsymbol{x}_{new}) = 1$, 즉 클래스 C_1에 할당됨을 알 수 있다.



[그림 2-3] 각 클래스의 평균($\boldsymbol{m}_1, \boldsymbol{m}_2$) 을 이용하여 정의된 결정경계

2.1.4 성능 평가

마지막으로, 얻어진 결정 경계를 이용하여 분류를 수행하였을 때의 성능을 평가해 본다. 성능 평가의 과정은 개발된 인식기의 특성을 반영해 줄 뿐 아니라 추후 성능 개선을 위한 연구를 위해서도 매우 중요하다. 성능을 평가하는 기준으로는 1장에서 설명한 바와 같이 분류율 혹은 분류오차를 주로 사용한다. 분류오차는 학습오차, 테스트오차, 그리고 일반화오차가 있는데, 여기서는 학습오차와 테스트오차에 대하여 평가해 본다.

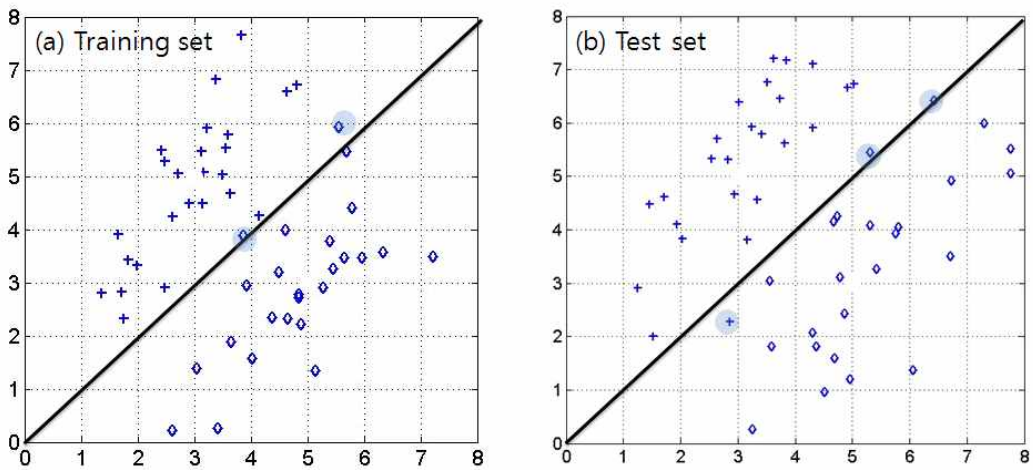
[프로그램 2-3]에 학습오차를 계산하는 프로그램을 나타내었다. 결정경계를 정의하기 위한 통계량 $\boldsymbol{m}_1, \boldsymbol{m}_2$를 미리 파일에 저장해 두었으므로, 이것을 불러와서 간단히 결정경계를 계산할 수 있다. 각각의 학습데이터에 대하여 두 평균 간의 거리를 계산한 후, 결정규칙([식 2-4])에 따라 클래스를 할당해 주고, 그 결과가 올바르지 못한 경우 학습오차를 증가시키는 과정을 반복한다. 현재 사용된 학습 데이터의 경우, [그림 2-4a]에서도 확인할 수 있듯이 총 50개중 2개의 데이터가 오분류되어 학습오차는 4%이다. 테스트 데이터에 대해서도 같은 과정을 거치면 [그림 2-4b]에서와 같이 테스트오차가 6%가 됨을 확인할 수 있다. 보다 정확하게 성능을 평가하기 위해서는 1장에서 설명한 일반화오차를 계산할 필요가 있는데, 이 문제의 경우 데이터를 생성하는데 사용한 실제 확률분포를 알 수 있으므로 수리적으로 계산하는

것도 가능하다. 또는 테스트 데이터를 생성할 때와 같은 방식으로 충분히 많은 양의 데이터를 생성하여 오차를 계산하면 일반화오차에 근사한 값을 얻을 수 있다. 이 문제의 경우 10^6 개의 데이터를 생성하여 오차를 계산해 본 결과, 그 값이 2.3%로 나타났다.

프로그램 2-3 Evaluating Performance

학습오차를 계산하는 프로그램

```
001  load data2_1                        %저장된 데이터를 불러들임
002  load mean2_1                        %저장된 평균/공분산의 추정치를 불러들임
003  Etrain=0;                           %학습데이터에 대한 분류 시작
004  N = size(X1,1);
005  for i=1:N
006      d1=norm(X1(i,:)-m1);            %클래스 1의 평균과의 거리 계산
007      d2=norm(X1(i,:)-m2);            %클래스 2의 평균과의 거리 계산
008      if (d1-d2)>0
009          Etrain = Etrain+1;          %오분류된 경우 학습오차 증가
010      end
011      d1=norm(X2(i,:)-m1);            %클래스 1의 평균과의 거리 계산
012      d2=norm(X2(i,:)-m2);            %클래스 2의 평균과의 거리 계산
013      if (d1-d2)<0
014          Etrain = Etrain+1;          %오분류된 경우 학습오차 증가
015      end
016  end
017  fprintf(1,'Training Error = %.3fWn', Etrain/50);   %학습오차 출력
```



[그림 2-4] 학습 데이터와 테스트 데이터에 대한 분류오차 - 음영 표시된 부분이 오분류된 데이터를 나타냄. (학습오차: 4%, 테스트오차: 6%)

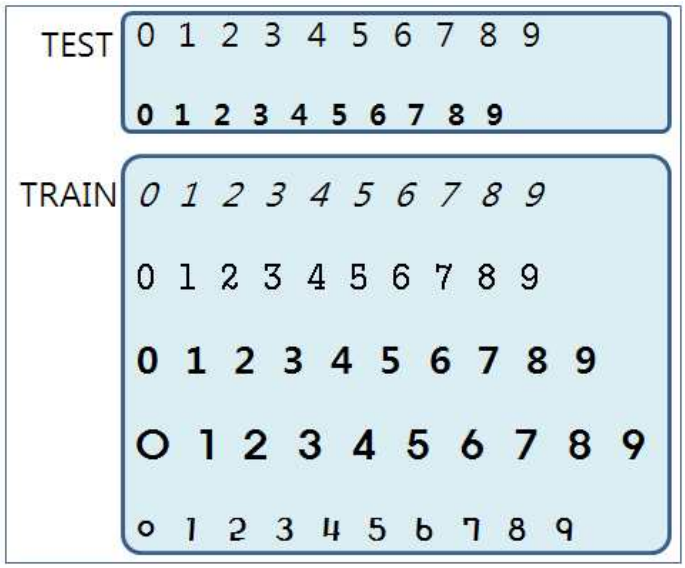
이 실험에서 사용한 분류 방식은 매우 간단하면서도 학습 데이터와 테스트 데이터 모두에

대해 비교적 좋은 성능을 보이고 있음을 알 수 있다. 또한, 학습에 의해 평균이 추정되고 결정경계가 정해지고 나면, 더 이상 학습데이터를 가지고 있지 않아도 새로운 데이터에 대하여 분류가 가능하다는 장점도 가지고 있다. 물론 이 간단한 방법은 여러 가지 제약점도 가지고 있는데, 이에 대한 자세한 설명은 3장 이후에서 그 이론적 배경과 함께 설명하기로 하겠다.

2.2 숫자 데이터의 분류

2.2.1 데이터 수집과 전처리

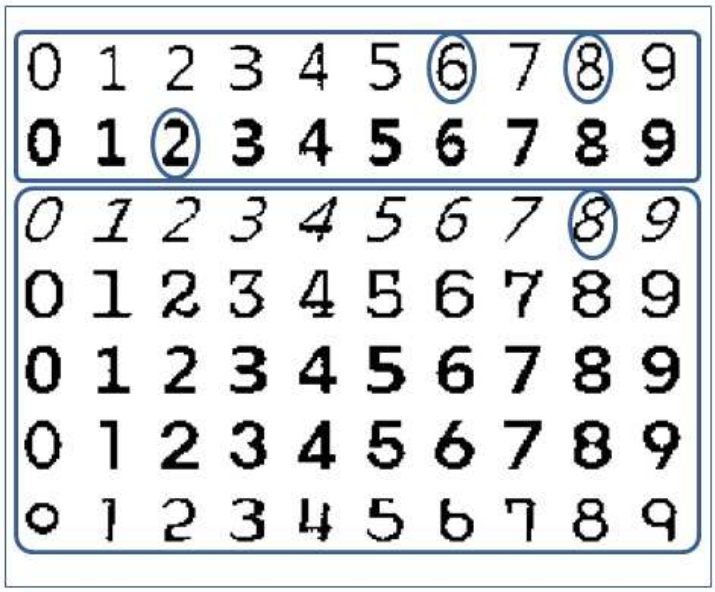
앞 절에서 소개한 평균값을 이용한 분류는 매우 간단하지만 실세계 데이터에도 충분히 적용 가능하다. 이 절에서는 숫자인식 문제에 적용한 예를 소개한다. [그림 2-5]에 여러 가지 글씨체와 크기의 변형을 가진 70개의 숫자 데이터가 나타나있다. 이 중 처음 두 행의 20개 데이터를 테스트 데이터 집합으로, 나머지 다섯 행의 50개 데이터를 학습 데이터 집합으로 사용하여 인식기를 개발한다.



[그림 2-5] 수집된 데이터집합

먼저 수집된 데이터들에 대하여 간단한 전처리를 수행한다. 글씨체가 가지는 크기 변형을 처리하기 위하여 모두 같은 크기로 정규화하고, 이진화 (binarization) 과정을 거쳐 흑백 영상으로 바꾸는 전처리를 수행하여 얻어진 영상을 [그림 2-6]에 나타내었다. 정규화와 이진화는 영상 데이터에서 널리 사용되는 전처리 과정이지만, 이 책의 범위를 벗어나므로 자세한 설명은 생략한다. 이밖에 숫자나 문자 데이터의 경우에는 글씨체의 굵기를 동일하게 만드는 등의 전처리를 수행하기도 한다. 전처리를 거친 데이터에 대해서는 특징추출 단계를 수행해야 하지만, 여기서는 이를 생략하고 전처리된 데이터를 이용하여 바로 학습단계를 수행한다.

특징추출 방법에 대하여서는 8장에서 자세히 설명하겠다.



[그림 2-6] 전처리 후의 데이터

프로그램 2-4 Reading image data

영상 데이터를 읽어 데이터 벡터로 표현하고 저장하는 프로그램

```
001 for i=0:1:9
002     % 학습데이터 집합 생성
003     for j=3:7
004         fn=sprintf('digit%d_%d.bmp',i,j);      %영상이 저장된 파일을 읽음
005         xi=imread(fn);
006         x = reshape(double(xi),16*20,1);       %벡터 형태로 변환
007         Xtrain(:,i*5+j-2)=x;                   %학습데이터 행렬에 저장
008         Ttrain(i*5+j-2,1)=i;                   %데이터의 클래스 라벨 저장
009     end
010     % 테스트데이터 집합 생성
011     for j=1:2
012         fn=sprintf('digit%d_%d.bmp',i,j);      %영상이 저장된 파일을 읽음
013         xi=imread(fn);
014         x = reshape(double(xi),16*20,1);       %벡터 형태로 변환
015         Xtest(:,i*2+j)=x;                      %테스트 데이터 행렬에 저장
016         Ttest(i*2+j,1)=i;                      %데이터의 클래스 라벨 저장
017     end
018 end
019 save digitdata Xtrain Ttrain Xtest Ttest   %학습 데이터와 테스트 데이터 저장
```

전처리가 끝난 영상 데이터들은 패턴인식기에서 다루기 좋은 벡터 형태로 표현해 주어야 한다. [프로그램 2-4]에 숫자영상 데이터를 읽어서 학습 데이터 집합과 테스트 데이터 집합으로 나누어 행렬로 표현하고 저장하는 프로그램을 나타내었다. 크기 정규화를 통해 20×16 크기로 저장되어 있는 영상들을 하나씩 읽어 들인다. 각 영상들은 하나의 2차원 행렬로 주어지는데, 이를 패턴인식 문제에 적용하기 위해서는 벡터 형태로 바꾸어 주어야 한다. 즉, 20×16 크기의 2차원 행렬은 각 행들을 일렬로 붙여서 만들어지는 320차원의 행벡터로 변환하여 표현하는데, 이는 매트랩에서 제공하는 행렬형태 변환 함수 reshape()을 이용하여 간단히 수행할 수 있다. 모든 데이터에 대해 같은 처리를 수행함으로써 50개의 학습 데이터 집합은 320×50크기의 행렬 X_{train}으로 표현된다. 이와 함께 각 데이터는 10개의 클래스 중 어디에 속하는지가 미리 결정되어 있으므로, 이를 클래스라벨 T_{train}로 나타내었다. 마찬가지로 테스트데이터 영상들에 대해서도 같은 처리를 수행하여 X_{test}와 T_{test}도 얻은 후, 이후 인식과정을 위하여 파일 "digitdata.mat"로 저장해 둔다.

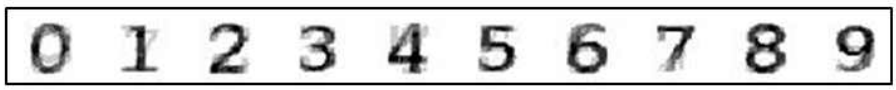
2.2.2 학습과 결정경계

학습을 통해 결정경계를 얻기 위해서는 앞 절의 2차원 데이터와 같이 평균을 이용하는 방법을 적용한다. [프로그램 2-5]에 학습 데이터에 대하여 평균을 계산하는 프로그램을 나타내었다. X_{train}에 저장된 데이터들은 각 숫자패턴 별로 5개씩, 총 50개의 데이터이다. 이에 대해 각 숫자 패턴별로 평균을 취하여 총 10개의 평균 벡터 ($\boldsymbol{m}_i$,$i=0,\ldots,9$)를 얻는다. [그림 2-7]에는 얻어진 평균 벡터들을 다시 20×16크기의 2차원 행렬로 재변환하여 영상으로 보여준 것이다.

프로그램 2-5 Calculating mean vector

학습데이터를 읽어서 평균을 계산하는 프로그램

```
001 load digitdata                                  %저장된 데이터를 읽어 들임
002 for i=0:1:9
003     mX(:,i+1)=mean(Xtrain(:,i*5+1:i*5+5)')';    %각 클래스의 평균 계산
004 end
005 save digitMean mX                               %각 클래스의 평균 저장
006 %평균영상을 그림으로 나타내는 과정
007 mXi = uint8(mX*255);                            %양의 정수 자료유형으로 변환
008 for i=0:1:9
009     subplot(1,10,i+1);
010     imshow(reshape(mXi(:,i+1),20,16));          %평균영상을 화면에 표시
011 end
```



[그림 2-7] 평균영상 데이터

이렇게 얻어진 평균 벡터를 이용하여 결정경계를 정해주는데, 이 문제의 경우 앞 절의 2차원 데이터와는 달리 모두 10개의 클래스가 존재하므로 두 클래스를 양분하는 형태의 결정경계 $g(\boldsymbol{x})=0$은 찾을 수 없다. 그러나 분류 대상이 되는 데이터 $\boldsymbol{x}$와 각 평균 벡터 ($\boldsymbol{m}_i$,$i=0,\ldots,9$)와의 거리 $d(\boldsymbol{x},\boldsymbol{m}_i)$를 기준으로 클래스가 결정되는 것은 동일하기 때문에 다음과 같은 결정규칙을 정의할 수 있겠다. 식에서 argmin은 중괄호 안의 값이 최소가 될 때의 매개변수(argument)(여기서는 i)의 값을 의미한다.

$$y(\boldsymbol{x}) = \mathrm{argmin}_i\{d(\boldsymbol{x},\boldsymbol{m}_i)\} \qquad \text{[식 2-6]}$$

2.2.3 분류와 성능 평가

[식 2-6]의 결정규칙을 이용하여 학습 데이터와 테스트 데이터에 대하여 분류를 수행하고, 성능을 평가한다. [프로그램 2-6]에 분류를 수행하여 학습오차와 테스트오차를 계산하는 프로그램을 나타내었다. 분류 결과, 학습 데이터에 대해서는 총 50개 중 1개를 오분류하여 2%의 학습오차를 나타내었으며, 테스트 데이터에 대해서는 총 20개 중 3개를 오분류하여 15%의 테스트오차를 나타내었다. 오분류된 데이터는 [그림 2-6]에 원으로 표시해 두었다. 테스트오차를 줄이기 위해서는 보다 많은 학습 데이터를 사용하거나 보다 정교한 전처리나 특징 추출 방법을 사용할 수 있을 것이다. 또한 이 장에서 간단한 소개한 분류방법이 아닌 보다 정교한 분류기를 사용함으로써 오차율을 줄이는 노력들이 많이 행해지고 있다. 이 책에서는 이러한 다양한 분류방법들에 대하여 소개할 것이다.

프로그램 2-6 Classifying data

데이터를 분류하여 학습오차와 테스트오차를 계산

```
001 load digitdata                          %데이터를 읽어 들임
002 load digitMean                          %저장된 평균을 읽어 들임
003 Ntrain=50;                              %학습 데이터의 수
004 Ntest=20;                               %테스트 데이터의 수
005 Etrain=0;
006 Etest=0;
007 for i=1:50                              %학습오차의 계산
008     x=Xtrain(:,i);
009     for j=1:10
010         dist(j)=norm(x-mX(:,j));        %평균과 학습데이터의 거리계산
011     end
012     [minv,minc_train(i)]=min(dist);     %가장 가까운 평균을 찾음
013     if(Ttrain(i)~=(minc_train(i)-1))
014         Etrain=Etrain+1;                %오분류시 학습오차 증가
015     end
016 end
017 for i=1:20                              %테스트오차의 계산
018     x=Xtest(:,i);
019     for j=1:10
020         dist(j)=norm(x-mX(:,j));        %평균과 테스트 데이터의 거리계산
021     end
022     [minv,minc_test(i)]=min(dist);      %가장 가까운 평균을 가진 클래스 찾기
023     if(Ttest(i)~=(minc_test(i)-1))
024         Etest=Etest+1;                  %오분류시 학습오차 증가
025     end
026 end
```

연습 문제

1. 매트랩을 이용하여 다음 단계를 따라 간단한 분류 실험을 해 보시오.
(1) 아래에 정의된 평균과 공분산을 가진 가우시안 분포를 따르는 두 클래스에 대하여 각각 25개의 학습 데이터와 25개의 테스트 데이터를 생성하여 파일에 저장하고, 2차원 평면상에 산점도를 그려보시오.

$$\mu_1 = \begin{pmatrix} 3 \\ 5 \end{pmatrix},\ \Sigma_1 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix},\ \mu_2 = \begin{pmatrix} 5 \\ 3 \end{pmatrix},\ \Sigma_2 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$$

(2) 학습 데이터를 이용하여 각 클래스의 평균과 공분산을 추정해 보고, (1)에서 정의한 실 평균과 실공분산 값과 비교해 보시오.

(3) (2)에서 계산한 평균을 이용하여 결정경계식을 찾아보고, 그래프로 표시해 보시오.

(4) (3)에서 찾은 결정경계를 이용하여 학습 데이터와 테스트 데이터에 대해 분류를 수행하고, 분류오차를 계산해 보시오.

(5) 테스트 데이터와는 별도로 충분히 많은 양의 데이터를 생성하여 일반화오차의 근사값을 계산해 보시오.

2. 이 장에서 사용한 간단한 분류기가 가지는 문제점으로는 어떤 것이 있을지 생각해 보시오.

3. 숫자 데이터에 적합한 특징추출 방법으로 어떤 것들이 있을지 생각해 보고, 매트랩으로 구현해 보시오.

4. 3번 문제에서 추출된 특징을 이용하여 숫자인식을 수행해 보고, 학습오차와 테스트오차를 계산해 보시오.

참고 자료

이 책에서는 주로 영상데이터를 응용으로 다루고 있는데, 영상데이터에 대한 패턴인식을 수행하기 위해서는 이 장에서 보여준 것과 같은 크기정규화나 이진화 등의 적절한 전처리 과정이 필요하다. 이에 관련된 내용은 이 책에서는 자세히 언급하지 않으나 [Gonzalez 07]을 참고할 수 있을 것이다. 또한 최근에는 영상 처리와 관련된 시스템 개발을 위해서는 [OpenCV] 라이브러리를 많이 활용하고 있다. 매트랩을 익히기 위해서는 매트랩 프로그램에서 제공하는 Help Document를 적극적으로 활용할 것을 권장한다.

[Gonzalez 07] R. C. Gonzalez and R. E. Woods, Digital Image Processing (3rd ed.).

Prentice Hall, 2007.
[OvenCV] Open Computer Vision Library (http://sourceforge.net/projects/opencvlibrary/)