

파이썬을 활용한 컴퓨터 비전 입문

Chapter 15. 머신 러닝

동양미래대학교
인공지능소프트웨어학과
권 범

본 강의자료는 길벗 출판사의 『OpenCV 4로 배우는 컴퓨터 비전과 머신 러닝』
교재 내용을 토대로 작성되었습니다.

❖ 15장 머신 러닝

- 15.1 머신 러닝과 OpenCV
- 15.2 k 최근접 이웃
- 15.3 서포트 벡터 머신

15.1 머신 러닝과 OpenCV

15.2 k 최근접 이웃

15.3 서포트 벡터 머신

❖ 머신 러닝 개요

- **머신 러닝(machine learning)**이란 주어진 데이터를 분석하여 규칙성, 패턴 등을 찾는
- 이를 이용하여 의미 있는 정보를 추출하는 과정을 말함
- 예를 들어 다수의 사과와 바나나 사진으로부터 사과와 바나나를 구분할 수 있는 특징 또는 규칙을 찾는
- 이를 이용하여 새로운 사진이 들어 왔을 때 이것이 사과인지 또는 바나나인지를 판별하는 작업이 머신 러닝이 하는 일임
- 이때 데이터로부터 규칙을 찾아내는 과정을 **학습(train)** 또는 훈련이라고 함
- 학습에 의해 결정된 규칙을 모델(model)이라고 함
- 새로운 데이터를 학습된 모델에 입력으로 전달하고 결과를 판단하는 과정을 **예측(predict)** 또는 **추론(inference)**이라고 함

❖ 머신 러닝 개요: ① 지도 학습 (1/7)

- 머신 러닝은 크게 **지도 학습**(supervised learning)과 **비지도 학습**(unsupervised learning)으로 구분함
- 지도 학습은 정답을 알고 있는 데이터를 이용하여 학습을 진행하는 방식임
- 이때 학습 데이터에 대한 정답에 해당하는 내용을 **레이블**(label)이라고 함

❖ 머신 러닝 개요: ① 지도 학습 (2/7)

- 예를 들어 사과 사진과 바나나 사진을 구분하기 위해 지도 학습을 수행하려면 각각의 사진이 사과 사진인지 바나나 사진인지를 함께 알려 주어야 함
- 이 경우 머신 러닝 알고리즘은 사과 사진과 바나나 사진을 구분 지을 수 있는 규칙을 찾기 위해 수학적 또는 논리적 연산을 수행함

❖ 머신 러닝 개요: ① 지도 학습 (3/7)

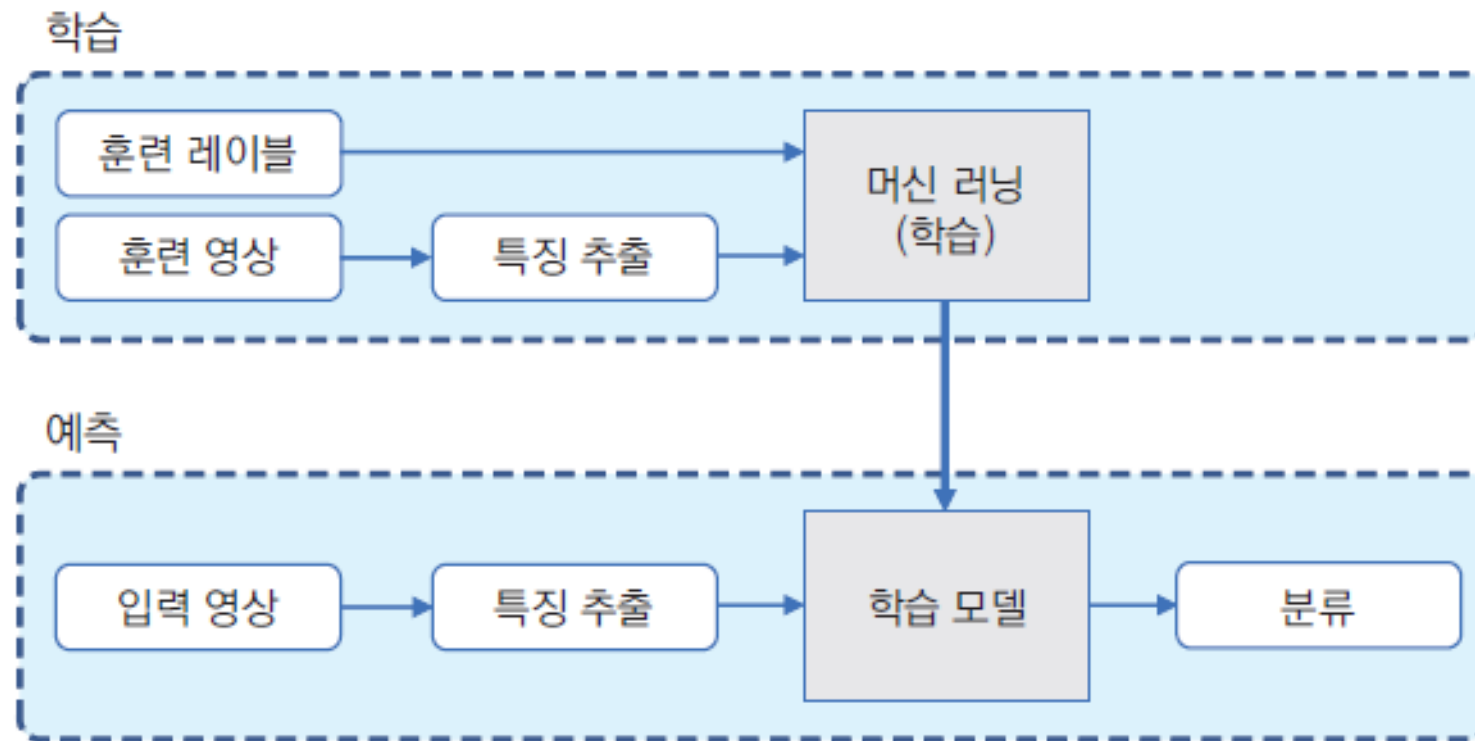
- 영상 데이터는 픽셀로 구성되어 있지만, 이 픽셀 값을 그대로 머신 러닝 입력으로 사용하는 것은 **그다지 흔치 않음**
- 왜냐하면 영상의 픽셀 값은 조명 변화, 객체의 이동 및 회전 등에 의해 매우 민감하게 변화하기 때문임
- 많은 머신 러닝 응용에서는 **영상의 다양한 변환에도 크게 변경되지 않는 특징 정보를 추출하여 머신 러닝 입력으로 전달함**
- 사과와 바나나 사진을 구분하는 용도라면 영상의 주된 색상(hue) 또는 객체 외곽선과 면적 비율 등이 유효한 특징으로 사용될 수 있음

❖ 머신 러닝 개요: ① 지도 학습 (4/7)

- 이처럼 영상 데이터를 사용하는 지도 학습에서는 먼저 다수의 학습 영상에서 특징 벡터를 추출함
- 이를 이용하여 머신 러닝 모델을 학습시킴
- 시험 영상을 이용한 예측 과정에서도 입력 영상으로부터 특징 벡터를 추출함
- 이 특징 벡터를 학습 모델 입력으로 전달하면 입력 영상이 어떤 영상인지에 대한 예측 결과를 얻을 수 있음

❖ 머신 러닝 개요: ① 지도 학습 (5/7)

▼ 그림 15-1 지도 학습에 의한 영상 분류 과정



❖ 머신 러닝 개요: ① 지도 학습 (6/7)

- 지도 학습은 주로 회귀(regression) 또는 분류(classification)에 사용됨
- 회귀는 연속된 수치 값을 예측하는 작업임
- 예를 들어 학생들의 키와 몸무게의 상관관계를 머신 러닝으로 학습함
- 새로운 학생의 키를 입력으로 주었을 때 몸무게를 예측하는 것은 회귀임

❖ 머신 러닝 개요: ① 지도 학습 (7/7)

- 반면에 **분류는 이산적인 값을 결과로 출력하는 머신 러닝임**
- 예를 들어 사과를 0번 클래스, 바나나를 1번 클래스라고 설정함
- 새로운 사진이 머신 러닝 입력으로 들어오면 결과를 0 또는 1로 나오게 설정하는 것은 분류임
- 입력 영상이 사과인지 바나나인지를 구분하는 것을 **인식(recognition)**이라고도 부르지만 결국은 분류 문제에 해당함

❖ 머신 러닝 개요: ② 비지도 학습

- 비지도 학습은 학습 데이터의 정답에 대한 정보 없이 **오로지 데이터 자체만을 이용하는 학습 방식임**
- 예를 들어 무작위로 섞여 있는 사과와 바나나 사진을 입력으로 전달함
- 전체 사진을 두 개의 그룹으로 나누도록 학습시키는 방식임
- 색상 정보만 적절하게 이용하여도 전체 사진을 사과 사진과 바나나 사진으로 구분할 수 있을 것임
- 다만 비지도 학습의 경우 분리된 두 개의 사진 집합이 무엇을 의미하는지는 알 수 없음
- 단지 두 사진 집합에서 **서로 구분되는 특징을 이용하여 서로 분리하는 작업만 수행하는 것임**
- 비지도 학습은 주로 **군집화(clustering)**에 사용됨

❖ 머신 러닝 개요 (1/7)

- 많은 머신 러닝 알고리즘이 지도 학습을 이용한 영상 분류 문제에 사용되고 있음
- 사과와 바나나 영상을 구분하는 것도 분류이고, 0부터 9까지의 필기체 숫자를 인식하는 것도 열 개의 클래스 분류 문제에 해당함
- 이처럼 머신 러닝을 이용하여 분류를 수행할 경우, 학습된 분류 모델이 얼마나 제대로 동작하는지를 확인해야 하는 경우가 있음
- 학습된 모델의 성능이 좋지 않다면 다른 머신 러닝 알고리즘을 선택하거나 영상에서 다른 특징 벡터를 추출하는 것을 고려해야 하기 때문임

❖ 머신 러닝 개요 (2/7)

- 사용할 수 있는 영상 데이터 전체를 학습에 사용하지 않음
- 일부는 성능 측정을 위한 시험 용도로 사용함
- 예를 들어 주어진 1만 개의 영상 중에서 8000개만 머신 러닝 모델의 학습에 사용함
- 나머지 2000개 영상으로는 학습 완료된 머신 러닝 모델의 성능 평가를 수행하여 분류 정확도를 계산하는 방식임

❖ 머신 러닝 개요 (3/7)

- 머신 러닝 알고리즘 종류에 따라서는 **내부적으로 사용하는 많은 파라미터에 의해 성능이 달라지기도 함**
- 최적의 파라미터를 찾는 것이 또 하나의 해결해야 할 문제가 되기도 함

❖ 머신 러닝 개요 (4/7)

- 이런 경우에는 학습 데이터를 k개의 부분 집합으로 분할함
- 학습과 검증(validation)을 반복하면서 최적의 파라미터를 찾을 수도 있음
- 예를 들어 8000개의 학습 영상을 800개씩 열 개의 부분 집합으로 분할함
- 이 중 아홉 개의 부분 집합으로 학습하고 나머지 한 개의 집합을 이용하여 성능을 검증함
- 검증을 위한 부분 집합을 바꿔 가면서 여러 번 학습과 검증을 수행함
- 이러한 작업을 다양한 파라미터에 대해 수행하면서 가장 성능이 높게 나타나는 파라미터를 찾을 수 있음
- 학습 데이터를 k개의 부분 집합으로 분할하여 학습과 검증을 반복하는 작업을 **k-폴드 교차 검증(k-fold cross-validation)**이라고 함

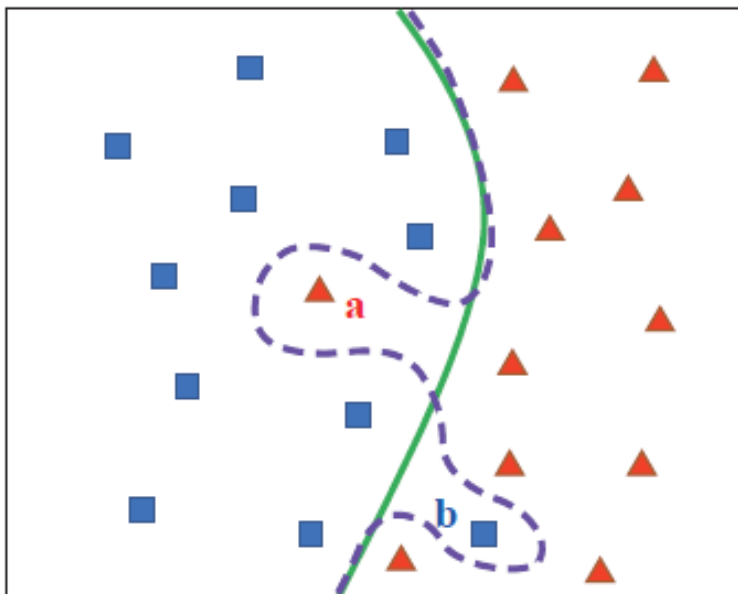
❖ 머신 러닝 개요 (5/7)

- 머신 러닝 알고리즘으로 학습 데이터를 학습할 경우 학습 데이터에 포함된 잡음 또는 이상치(outlier)의 영향을 고려해야 함
- 많은 머신 러닝 분류 알고리즘이 학습 데이터를 효과적으로 구분하는 경계면을 찾으려고 함
- 만약 학습 데이터에 잘못된 정보가 섞여 있다면 경계면을 어떻게 설정하는 것이 좋은지 모호해질 수 있음

❖ 머신 러닝 개요 (6/7)

- 그림 15-2는 빨간색 삼각형과 파란색 사각형 점을 구분하는 분류 문제임
- 두 가지 종류의 점을 완벽하게 구분하는 경계면은 분명 보라색 점선임
- 보라색 경계면은 학습 데이터에 대해서는 100% 정확하게 동작함

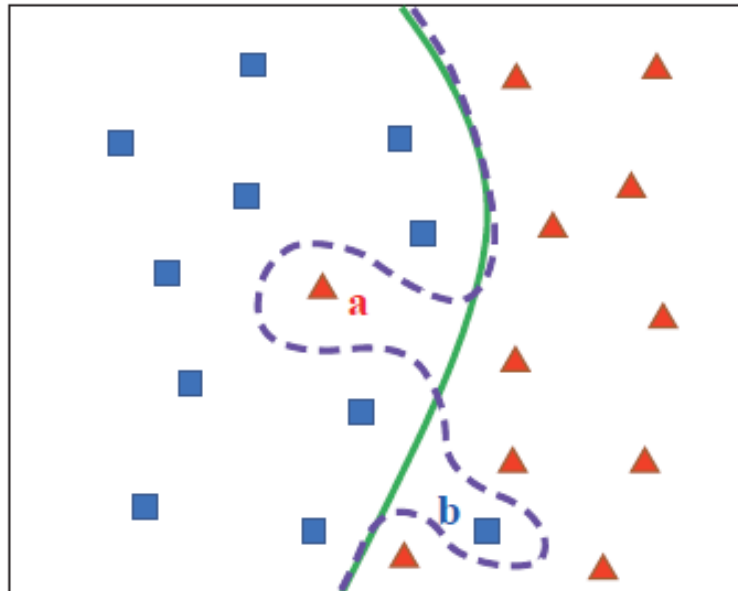
▼ 그림 15-2 학습 데이터에 이상치가 있을 경우의 분류 경계면



❖ 머신 러닝 개요 (7/7)

- 실제 새로운 입력 데이터에 대해서는 오히려 정확도가 떨어질 수 있는 가능성을 포함하게 됨
- a와 b 점은 각각 빨간색 삼각형과 파란색 사각형 분포와 동떨어진 위치에 존재함
- 잡음 또는 잘못 측정된 이상치일 가능성이 높기 때문임
- 이때는 보라색 점선 경계면보다는 오히려 녹색 실선 경계면을 사용하는 것이 실제 시험에서 더 좋은 성능을 보여 줄 수 있음

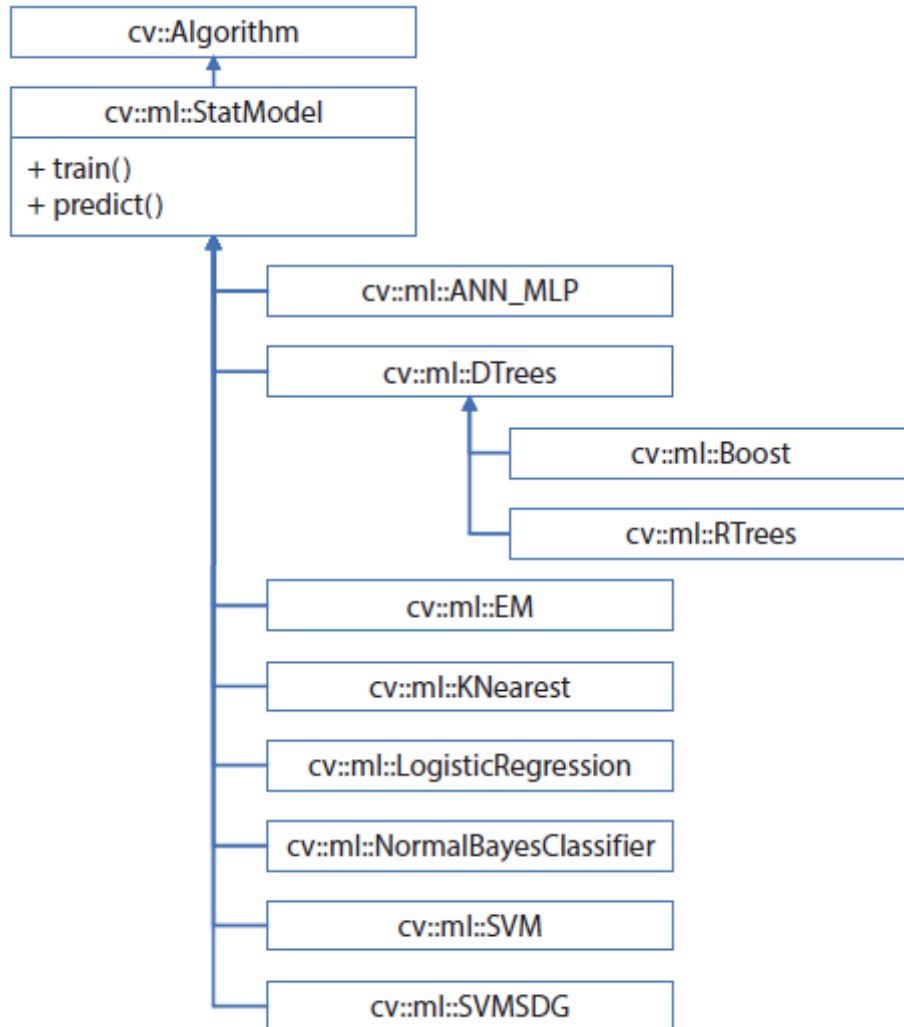
▼ 그림 15-2 학습 데이터에 이상치가 있을 경우의 분류 경계면



❖ OpenCV 머신 러닝 클래스 (1/8)

- OpenCV에서 제공하는 머신 러닝 클래스는 주로 ml 모듈에 포함되어 있음
- 학습 데이터로부터 머신 러닝 모델을 학습시킬 때는 `train()` 메서드를 이용함
- 학습된 모델에 대해서 시험 데이터에 대한 결과를 예측할 때는 `predict()` 메서드를 이용함

❖ OpenCV 머신 러닝 클래스 (2/8)



◀ 그림 15-3 OpenCV 머신 러닝 클래스

❖ OpenCV 머신 러닝 클래스 (3/8)

- 머신 러닝 클래스에서 학습을 수행하는 `train()` 메서드 원형은 다음과 같음

`train(samples, layout, responses)`

<code>samples</code>	학습 데이터 행렬
<code>layout</code>	학습 데이터 배치 방법. ROW_SAMPLE 또는 COL_SAMPLE를 지정합니다.
<code>responses</code>	각 학습 데이터에 대응되는 응답(레이블) 행렬
반환값	정상적으로 학습이 완료되면 True를 반환합니다.

❖ OpenCV 머신 러닝 클래스 (4/8)

- layout에는 표 15-1에 나타난 SampleTypes 열거형 상수 중 하나를 지정할 수 있음
- 많은 경우 ROW_SAMPLE 방법을 사용함

▼ 표 15-1 SampleTypes 열거형 상수

SampleTypes 열거형 상수	설명
ROW_SAMPLE	각 훈련 데이터가 samples 행렬에 행 단위로 저장되어 있습니다.
COL_SAMPLE	각 훈련 데이터가 samples 행렬에 열 단위로 저장되어 있습니다.

❖ OpenCV 머신 러닝 클래스 (5/8)

- 이미 학습된 모델에 대해 시험 데이터의 응답을 얻고 싶으면 `predict()` 함수를 사용함

`predict(samples, results)`

<code>samples</code>	입력 벡터가 행 단위로 저장된 행렬
<code>results</code>	각 입력 샘플에 대한 예측 결과가 저장된 행렬
반환값	입력 벡터가 하나인 경우에 대한 응답이 반환됩니다.

❖ OpenCV 머신 러닝 클래스 (6/8)

- OpenCV에서 제공하고 있는 머신 러닝 알고리즘 구현 클래스에 대한 간략한 설명을 표 15-2에 정리함

▼ 표 15-2 OpenCV 머신 러닝 클래스 이름과 의미

클래스 이름	설명
ANN_MLP	인공 신경망(artificial neural network) 다층 퍼셉트론(multi-layer perceptrons). 여러 개의 은닉층을 포함한 신경망을 학습시킬 수 있고, 입력 데이터에 대한 결과를 예측할 수 있습니다.
DTrees	이진 의사 결정 트리(decision trees) 알고리즘. DTrees 클래스는 다시 부스팅 알고리즘을 구현한 <code>ml::Boost</code> 클래스와 랜덤 트리(random tree) 알고리즘을 구현한 <code>ml::RTree</code> 클래스의 부모 클래스 역할을 합니다.
Boost	부스팅(boosting) 알고리즘. 다수의 약한 분류기(weak classifier)에 적절한 가중치를 부여하여 성능이 좋은 분류기를 만드는 방법입니다.
RTrees	랜덤 트리(random tree) 또는 랜덤 포레스트(random forest) 알고리즘. 입력 특징 벡터를 다수의 트리로 예측하고, 그 결과를 취합하여 분류 또는 회귀를 수행합니다.

❖ OpenCV 머신 러닝 클래스 (7/8)

▼ 표 15-2 OpenCV 머신 러닝 클래스 이름과 의미

클래스 이름	설명
EM	기댓값 최대화(Expectation Maximization). 가우시안 혼합 모델(Gaussian mixture model)을 이용한 군집화 알고리즘입니다.
KNearest	k 최근접 이웃(k-Nearest Neighbor) 알고리즘. k 최근접 이웃 알고리즘은 샘플 데이터와 인접한 k개의 훈련 데이터를 찾고, 이 중 가장 많은 개수에 해당하는 클래스를 샘플 데이터 클래스로 지정합니다.
LogisticRegression	로지스틱 회귀(logistic regression). 이진 분류 알고리즘의 일종입니다.
NormalBayesClassifier	정규 베이지 분류기. 정규 베이지 분류기는 각 클래스의 특징 벡터가 정규 분포를 따른다고 가정합니다. 따라서 전체 데이터 분포는 가우시안 혼합 모델로 표현 가능합니다. 정규 베이지 분류기는 학습 데이터로부터 각 클래스의 평균 벡터와 공분산 행렬을 계산하고, 이를 예측에 사용합니다.

❖ OpenCV 머신 러닝 클래스 (8/8)

▼ 표 15-2 OpenCV 머신 러닝 클래스 이름과 의미

클래스 이름	설명
SVM	서포트 벡터 머신(support vector machine) 알고리즘. 두 클래스의 데이터를 가장 여유 있게 분리하는 초평면을 구합니다. 커널 기법을 이용하여 비선형 데이터 분류에도 사용할 수 있으며, 다중 클래스 분류 및 회귀에도 적용할 수 있습니다.
SVMMSG	통계적 그래디언트 하향(stochastic gradient descent) SVM. 통계적 그래디언트 하향 방법을 SVM에 적용함으로써 대용량 데이터에 대해서도 빠른 학습이 가능합니다[Bottou10].

15.2 k 최근접 이웃

15.1 머신 러닝과 OpenCV

15.3 서포트 벡터 머신

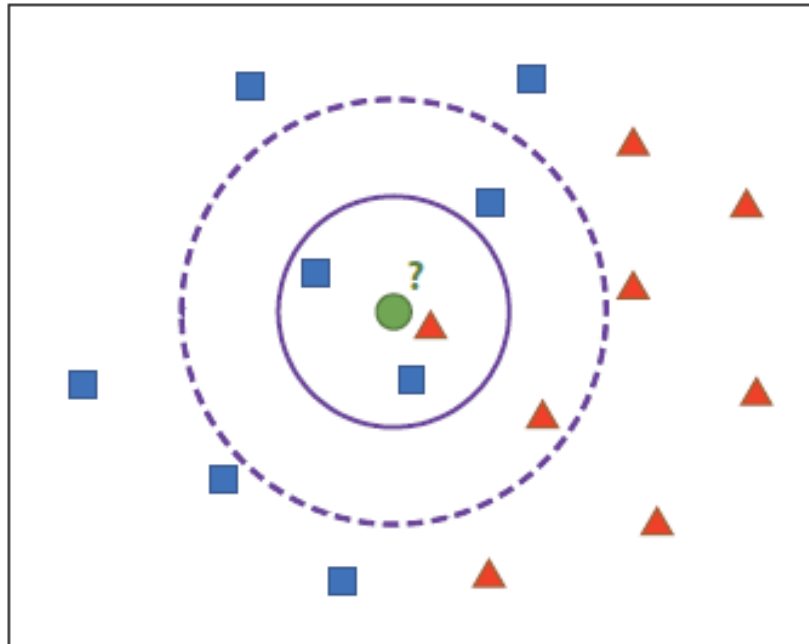
❖ k 최근접 이웃 알고리즘 (1/6)

- **k 최근접 이웃**(kNN, k-Nearest Neighbors) 알고리즘은 분류 또는 회귀에 사용되는 지도 학습 알고리즘의 하나임
- kNN 알고리즘을 **분류에 사용할 경우**, 특징 공간에서 시험 데이터와 가장 가까운 k개의 학습 데이터를 찾음
- k개의 학습 데이터 중에서 가장 많은 클래스를 시험 데이터의 클래스로 지정함
- kNN 알고리즘을 **회귀 문제에 적용할 경우**, 시험 데이터에 인접한 k개의 학습 데이터 평균을 시험 데이터 값으로 설정함

❖ k 최근접 이웃 알고리즘 (2/6)

- 그림 15-4는 2차원 평면상에 파란색 사각형과 빨간색 삼각형 두 종류의 데이터가 분포되어 있음
- 이들 파란색과 빨간색 점들이 학습 데이터이고, 이 학습 데이터는 두 개의 클래스로 구분되어 있음
- 각 점들은 (x, y) 좌표로 표현되므로, 이들 데이터는 2차원 특징 공간에 정의되어 있다고 표현할 수 있음

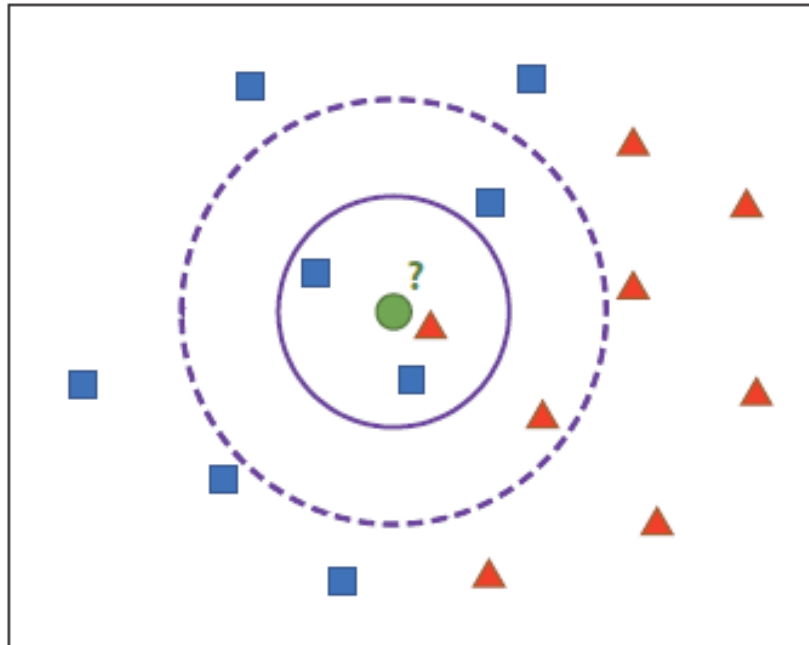
▼ 그림 15-4 kNN 알고리즘에 의한 점 분류



❖ k 최근접 이웃 알고리즘 (3/6)

- 여기에 녹색으로 표시한 새로운 점을 추가할 경우, 이 점을 파란색 사각형 클래스로 넣을 것인지, 아니면 빨간색 삼각형 클래스로 넣을 것인지를 결정해야 함
- 간단한 방법은 새로 들어온 점과 가장 가까이 있는 학습 데이터 점을 찾아, 해당 학습 데이터 클래스와 같게 설정하는 것임

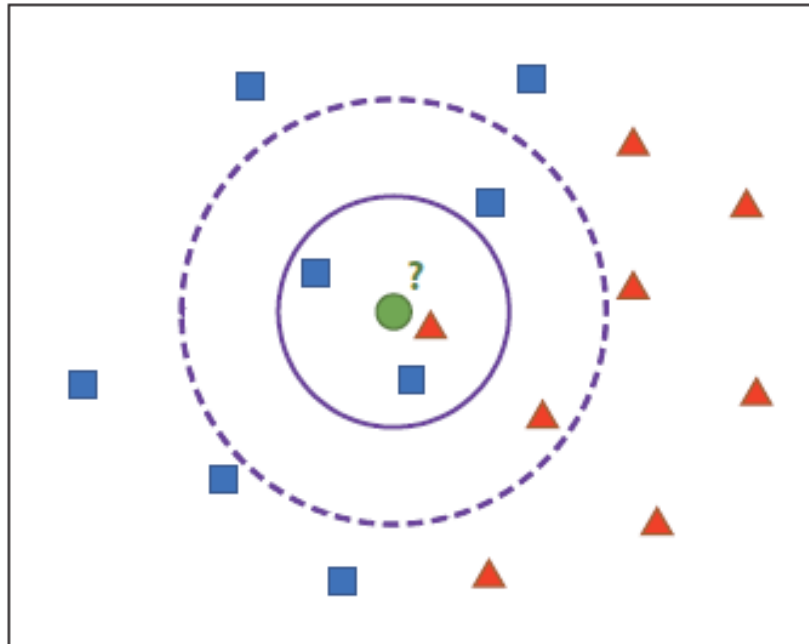
▼ 그림 15-4 kNN 알고리즘에 의한 점 분류



❖ k 최근접 이웃 알고리즘 (4/6)

- 그림 15-4에서 녹색 점과 가장 가까운 점은 빨간색 삼각형이므로 녹색 점을 빨간색 삼각형 클래스로 지정할 수 있음
- 이러한 방법을 **최근접 이웃**(NN, Nearest Neighbors) 알고리즘이라고 함

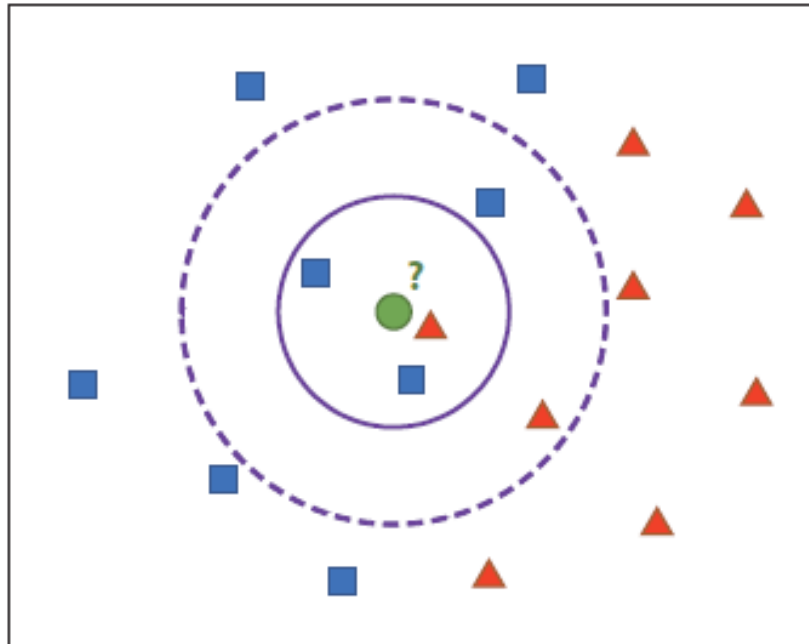
▼ 그림 15-4 kNN 알고리즘에 의한 점 분류



❖ k 최근접 이웃 알고리즘 (5/6)

- 녹색 점에서 가장 가까운 도형은 빨간색 삼각형이지만, 이 지점은 파란색 사각형이 더 많이 분포하는 지역이라고 판단할 수 있음
- 녹색 점을 파란색 사각형 클래스로 지정하는 것이 더욱 합리적일 수 있음
- 이러한 방식으로 분류하는 방법을 **kNN 알고리즘**이라고 함

▼ 그림 15-4 kNN 알고리즘에 의한 점 분류



❖ k 최근접 이웃 알고리즘 (6/6)

- kNN 알고리즘에서 k를 1로 설정하면 최근접 이웃 알고리즘이 됨
- 보통 k는 1보다 큰 값으로 설정함
- **k 값을 어떻게 설정하느냐에 따라 분류 및 회귀 결과가 달라질 수 있음**
- 최선의 k 값을 결정하는 것은 주어진 데이터에 의존적임
- 보통 k 값이 커질수록 잡음 또는 이상치 데이터의 영향이 감소함
- k 값이 어느 정도 이상으로 커질 경우 오히려 분류 및 회귀 성능이 떨어질 수 있음

❖ KNearest 클래스 사용하기 (1/13)

- OpenCV에서 k 최근접 이웃 알고리즘은 KNearest 클래스에 구현되어 있음
- KNearest 클래스는 ml 모듈에 포함되어 있음
- KNearest 클래스를 이용하려면 먼저 KNearest 객체를 생성해야 함
- KNearest 객체는 KNearest_create() 함수를 사용하여 생성할 수 있음

```
knn = cv2.KNearest_create()
```

반환값

KNearest 객체

❖ KNearest 클래스 사용하기 (2/13)

- KNearest 클래스는 기본적으로 k 값을 10으로 설정함
- 이 값을 변경하려면 `setDefaultK()` 메서드를 이용하여 변경할 수 있음
- 다만 `findNearest()` 메서드를 이용하여 시험 데이터의 응답을 구할 경우에는 k 값을 `findNearest()` 메서드 인자로 명시적으로 지정할 수 있음

```
knn.setDefaultK(val)
```

val

kNN 알고리즘에서 사용할 k 값.

`predict()` 함수를 사용할 경우 미리 k 값을 적절하게 설정해야 합니다.

❖ KNearest 클래스 사용하기 (3/13)

- KNearest 객체는 기본적으로 분류를 위한 용도로 생성됨
- KNearest 객체를 분류가 아닌 회귀에 적용하려면 `setIsClassifier()` 메서드에 `False`를 지정하여 호출해야 함

```
knn.setIsClassifier(val)
```

`val` 이 값이 `True`이면 분류로 사용하고, `False`이면 회귀로 사용합니다.

- KNearest 객체를 생성하고 속성을 설정한 후에는 `train()` 메서드를 이용하여 학습을 진행할 수 있음
- KNearest 클래스의 경우에는 `train()` 메서드에서 실제적인 학습이 진행되지는 않음
- 단순히 학습 데이터와 레이블 데이터를 KNearest 클래스 변수에 모두 저장하는 작업이 이루어짐

❖ KNearest 클래스 사용하기 (4/13)

- KNearest 클래스에서 학습 데이터를 학습함
- 시험 데이터에 대한 예측을 수행할 때에는 주로 `findNearest()` 메서드를 사용함
- KNearest 클래스에서도 `predict()` 메서드를 사용할 수 있음
- `findNearest()` 메서드가 예측 결과와 관련된 정보를 더 많이 반환하기 때문에 유용함

```
results, neighborResponses, dist = knn.findNearest(samples, k)
```

<code>samples</code>	시험 데이터 벡터가 행 단위로 저장된 행렬
<code>k</code>	사용할 최근접 이웃 개수 (1보다 같거나 커야 합니다)
<code>results</code>	각 입력 샘플에 대한 예측(분류 또는 회귀) 결과를 저장한 행렬
<code>neighborResponses</code>	예측에 사용된 k개의 최근접 이웃 클래스 정보를 담고 있는 행렬
<code>dist</code>	입력 벡터와 예측에 사용된 k개의 최근접 이웃과의 거리를 저장한 행렬
반환값	입력 벡터가 하나인 경우에 대한 응답이 반환됩니다.

❖ KNearest 클래스 사용하기 (5/13)

- 2차원 평면에서 세 개의 클래스로 구성된 점들을 kNN 알고리즘으로 분류함
- 그 경계면을 화면에 표시하는 예제 프로그램 소스 코드를 코드 15-1에 나타냄
- 코드 15-1은 (150, 150), (350, 150), (250, 400) 좌표를 중심으로 하는 가우시안 분포의 점을 각각 30개씩 생성하여 kNN알고리즘 학습 데이터로 사용함
- (0, 0) 좌표부터 (499, 499) 좌표 사이의 모든 점에 대해 kNN 분류를 수행하여 그 결과를 빨간색, 녹색, 파란색 색상으로 나타냄
- kNN 알고리즘의 k 값은 트랙바를 이용하여 프로그램 실행 중 변경할 수 있도록 함

❖ KNearest 클래스 사용하기 (6/13)

코드 15-1 kNN 알고리즘을 이용한 2차원 점 분류 (knnplane.py)

```
1  import numpy as np
2  import cv2
3
4  train = []
5  label = []
6  k_value = 1
7
8  def on_k_changed(pos):
9      global k_value
10
11      k_value = pos
12      if k_value < 1:
13          k_value = 1
14
15      trainAndDisplay()
16
17  def addPoint(x, y, c):
18      train.append([x, y])
19      label.append([c])
20
```


❖ KNearest 클래스 사용하기 (7/13)

코드 15-1 kNN 알고리즘을 이용한 2차원 점 분류 (knnplane.py)

```
21 def trainAndDisplay():
22     train_array = np.array(train).astype(np.float32)
23     label_array = np.array(label)
24     knn.train(train_array, cv2.ml.ROW_SAMPLE, label_array)
25
26     for j in range(img.shape[0]):
27         for p in range(img.shape[1]):
28             sample = np.array([[p, j]]).astype(np.float32)
29
30             ret, res, _, _ = knn.findNearest(sample, k_value)
31
32             response = int(res[0, 0])
33             if response == 0:
34                 img[j, p] = (128, 128, 255)
35             elif response == 1:
36                 img[j, p] = (128, 255, 128)
37             elif response == 2:
38                 img[j, p] = (255, 128, 128)
39
```

❖ KNearest 클래스 사용하기 (8/13)

코드 15-1 kNN 알고리즘을 이용한 2차원 점 분류 (knnplane.py)

```
40     for p in range(len(train)):
41         x, y = train[p]
42         l = label[p][0]
43
44         if l == 0:
45             cv2.circle(img, (x, y), 5, (0, 0, 128), -1, cv2.LINE_AA)
46         elif l == 1:
47             cv2.circle(img, (x, y), 5, (0, 128, 0), -1, cv2.LINE_AA)
48         elif l == 2:
49             cv2.circle(img, (x, y), 5, (128, 0, 0), -1, cv2.LINE_AA)
50
51     cv2.imshow('knn', img)
52
53
54     NUM = 30
55     rn = np.zeros((NUM, 2), np.int32)
56
57     cv2.randn(rn, 0, 50)
58     for j in range(NUM):
59         addPoint(rn[j, 0] + 150, rn[j, 1] + 150, 0)
60
```

❖ KNearest 클래스 사용하기 (9/13)

코드 15-1 kNN 알고리즘을 이용한 2차원 점 분류 (knnplane.py)

```
61  cv2.randn(rn, 0, 50)
62  for j in range(NUM):
63      addPoint(rn[j, 0] + 350, rn[j, 1] + 150, 1)
64
65  cv2.randn(rn, 0, 70)
66  for j in range(NUM):
67      addPoint(rn[j, 0] + 250, rn[j, 1] + 400, 2)
68
69  img = np.zeros((500, 500, 3), np.uint8)
70  knn = cv2.ml.KNearest_create()
71
72  cv2.namedWindow('knn')
73  cv2.createTrackbar('k_value', 'knn', k_value, 5, on_k_changed)
74
75  trainAndDisplay()
76
77  cv2.imshow('knn', img)
78  cv2.waitKey()
79  cv2.destroyAllWindows()
```

❖ KNearest 클래스 사용하기 (10/13)

- knnplane.py 소스 코드 설명

- 72~73행 kNN 이름의 창에 트랙바를 부착하고, 트랙바가 움직이면 `on_k_changed()` 함수가 실행되도록 합니다.
- 57~67행 (150, 150) 좌표를 중심으로 하는 0번 클래스 점, (350, 150) 좌표를 중심으로 하는 1번 클래스 점, (250, 400) 좌표를 중심으로 하는 2번 클래스 점을 각각 30개씩 생성하여 학습 데이터에 추가합니다. 0번과 1번 클래스 점은 각각의 중심을 기준으로 표준 편차 50에 해당하는 가우시안 분포를 따르고, 2번 클래스 점은 중심을 기준으로 표준 편차 70에 해당하는 가우시안 분포를 따릅니다.
- 75행 프로그램이 처음 실행되자 kNN 알고리즘으로 분류된 결과를 보여 주도록 `trainAndDisplay()` 함수를 호출합니다.

❖ KNearest 클래스 사용하기 (11/13)

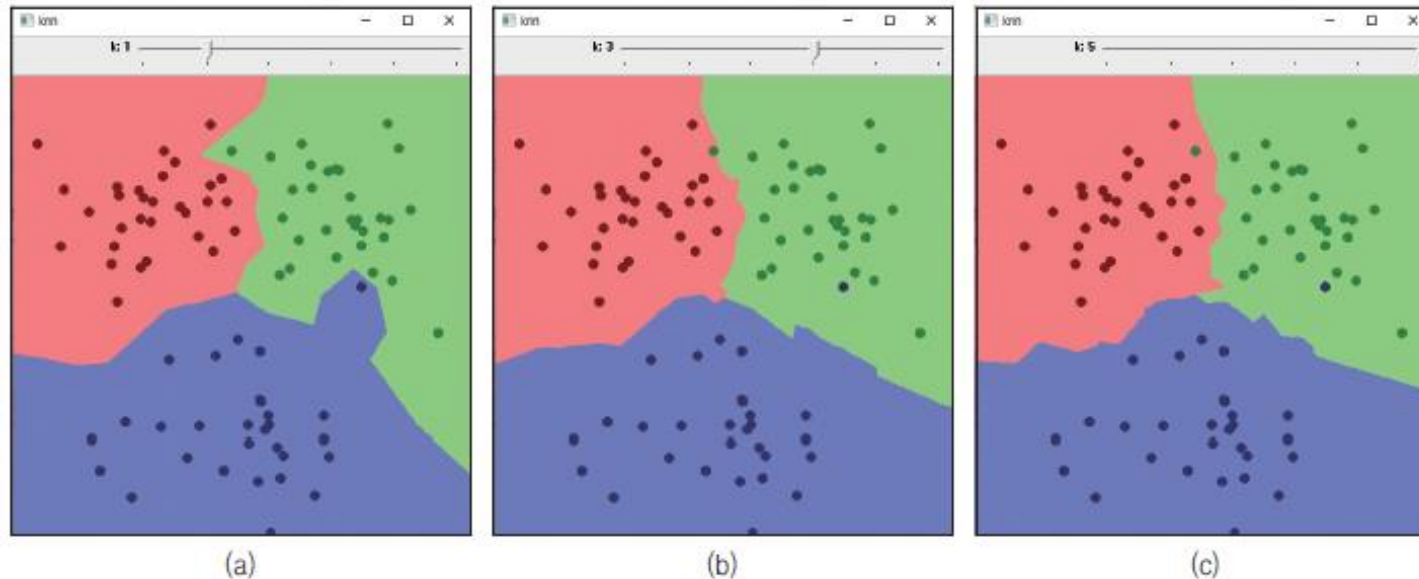
- knnplane.py 소스 코드 설명

- 8~15행 트랙바를 움직여서 k 값이 바뀌면 다시 kNN 알고리즘을 학습시키고 그 결과를 화면에 나타냅니다.
- 17~19행 addPoint() 함수는 좌표 점과 레이블 정보를 리스트에 추가합니다.
- 24행 train() 메서드를 이용하여 kNN 알고리즘을 학습합니다.
- 30~38행 img 영상 전체 좌표에 대해 kNN 분류기 응답을 조사하여 빨간색, 녹색, 파란색으로 표시합니다.
- 40~49행 54~67행에서 추가한 학습 데이터 점 좌표에 반지름 5인 원을 각각 빨간색, 녹색, 파란색으로 표시합니다.

❖ KNearest 클래스 사용하기 (12/13)

- 그림 15-5는 knnplane 프로그램을 실행하여 트랙바의 k 값을 각각 1, 3, 5로 지정했을 때의 분류 결과임
- 그림 15-5(a)는 k가 1인 경우의 분류 결과이고, 그림 15-5(b)는 k가 3인 경우이며, 그림 15-5(c)는 k가 5인 경우임

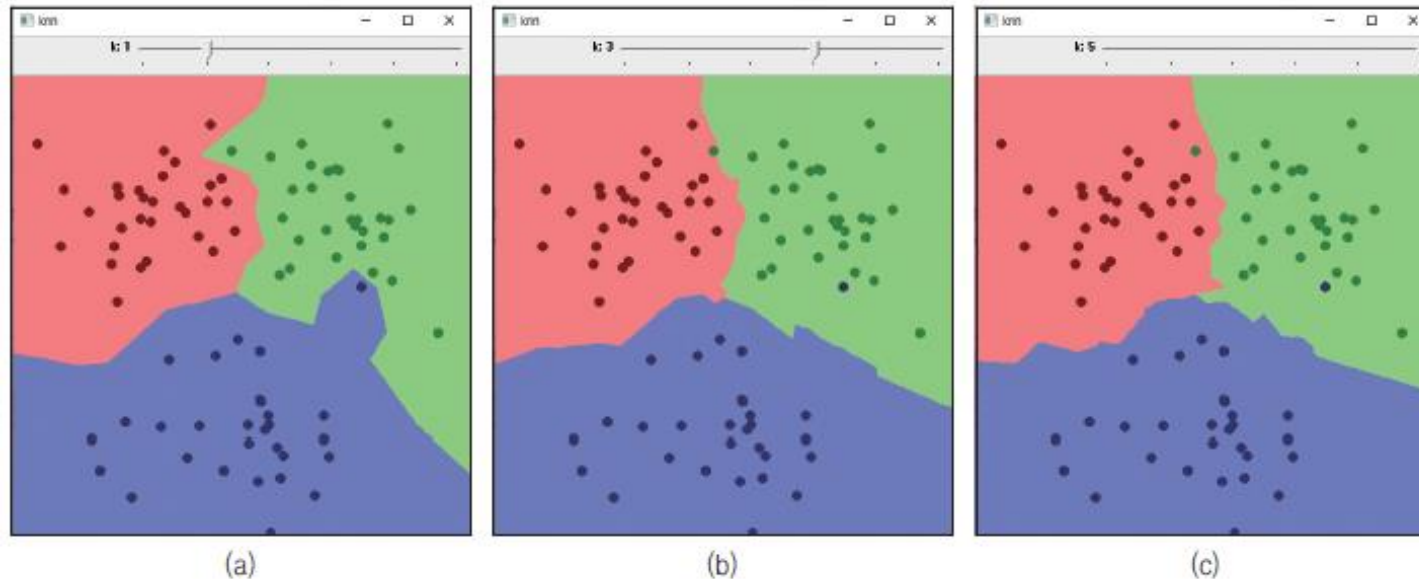
▼ 그림 15-5 kNN 알고리즘을 이용한 2차원 점 분류



❖ KNearest 클래스 사용하기 (13/13)

- k 값이 1인 경우에는 붉은색 영역에 가깝게 위치한 녹색 점과 녹색 영역에 가깝게 위치한 파란 점 때문에 클래스 경계면이 유난히 불룩하게 튀어나온 부분이 발생함
- 반면에 k 값을 3 또는 5로 바꾸면 클래스 경계면이 다소 완만한 형태로 바뀌게 됨
- 이는 k 값이 증가함에 따라 잡음 또는 이상치에 해당하는 학습 데이터 영향이 줄어드는 것으로 생각할 수 있음

▼ 그림 15-5 kNN 알고리즘을 이용한 2차원 점 분류

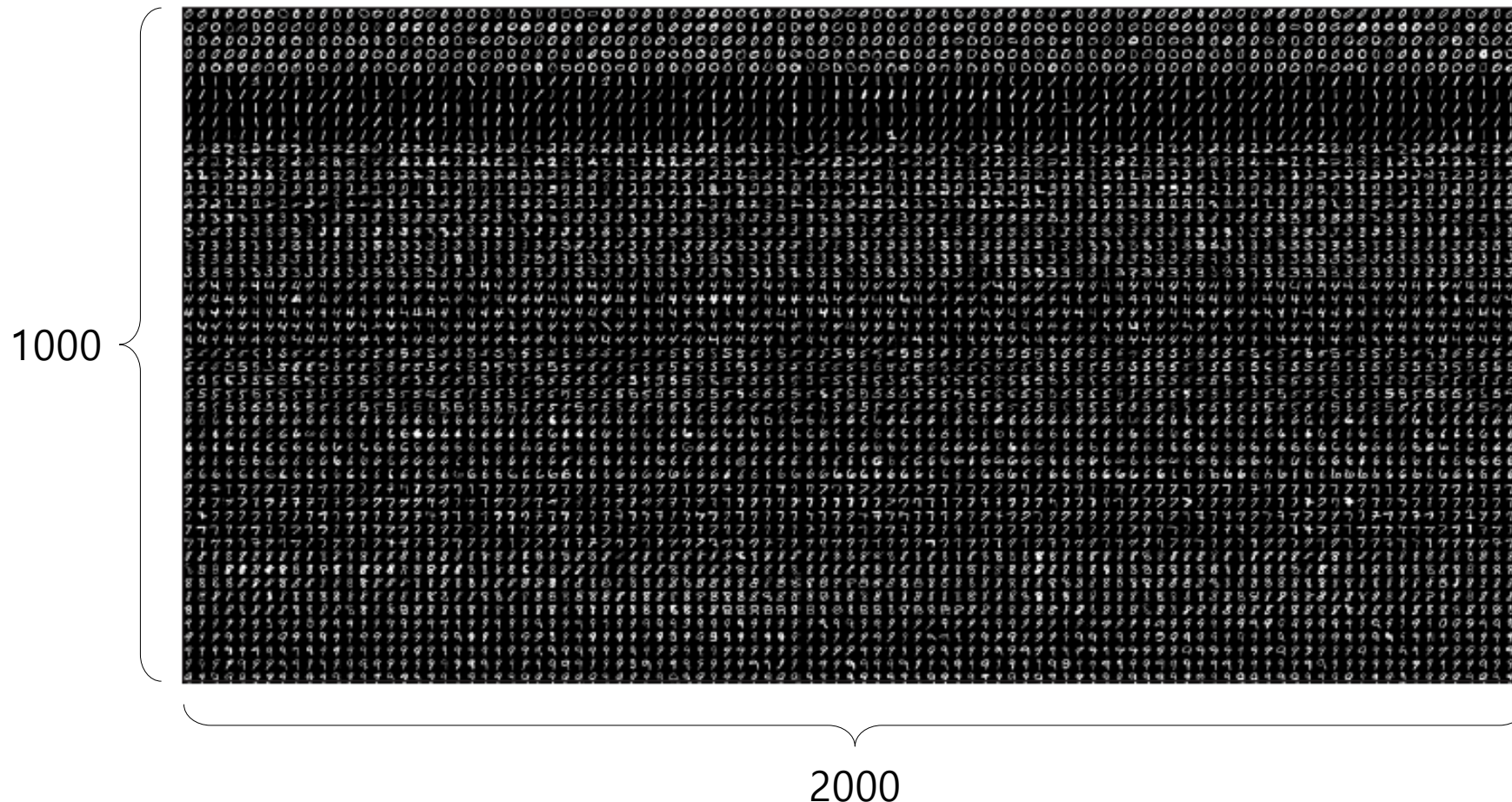


❖ kNN을 이용한 필기체 숫자 인식 (1/19)

- 머신 러닝으로 특정 문제를 해결하려면 많은 양의 학습 데이터가 필요함
- 머신 러닝으로 필기체 숫자 인식을 수행하려면 충분히 많은 필기체 숫자 영상을 학습 데이터로 사용해야 함
- digits.png 파일은 0부터 9까지의 필기체 숫자가 5000개 적혀 있는 영상임

❖ kNN을 이용한 필기체 숫자 인식 (2/19)

▼ 그림 15-6 OpenCV에서 제공하는 필기체 숫자 영상(digits.png)



❖ kNN을 이용한 필기체 숫자 인식 (3/19)

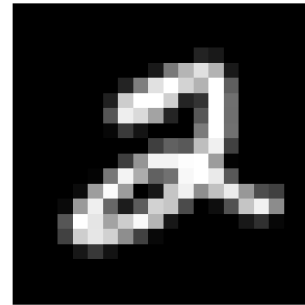
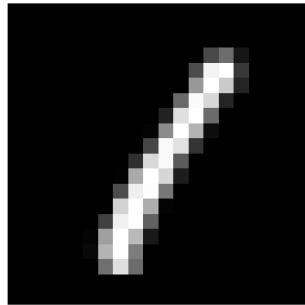
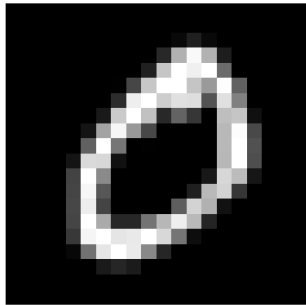
- digits.png 숫자 영상에는 0부터 9까지의 숫자가 각각 가로로 100개, 세로로 다섯 개씩 적혀 있음
- 각각의 숫자는 20×20 픽셀 크기로 적혀 있으며, digits.png 숫자 영상의 전체 크기는 1000×2000임
- 이 영상을 이용하여 머신 러닝 알고리즘을 학습시키려면 **각각의 숫자 영상을 부분 영상으로 추출하여 학습 데이터를 생성해야 함**

❖ kNN을 이용한 필기체 숫자 인식 (4/19)

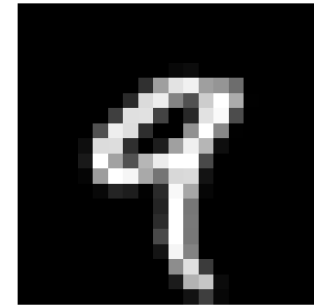
- 보통 머신 러닝으로 영상을 인식 또는 분류할 경우, 영상으로부터 인식 목적에 적합한 **특징 벡터를 추출하여 머신 러닝 입력으로 사용함**
- 이 절에서는 단순히 **20×20 숫자 영상 픽셀 값 자체를 kNN 알고리즘 입력으로 사용함**

❖ kNN을 이용한 필기체 숫자 인식 (5/19)

- digits.png 숫자 영상에 적혀 있는 필기체 숫자들이 대체로 20×20 부분 영상 정중앙에 위치해 있음
- 숫자 크기도 거의 일정하기 때문에 픽셀 값을 그대로 이용해도 충분한 인식 결과를 얻을 수 있음
- 5000개의 숫자 영상 데이터를 학습 및 시험 데이터로 구분하지 않고, 5000개 전체를 학습에 사용함



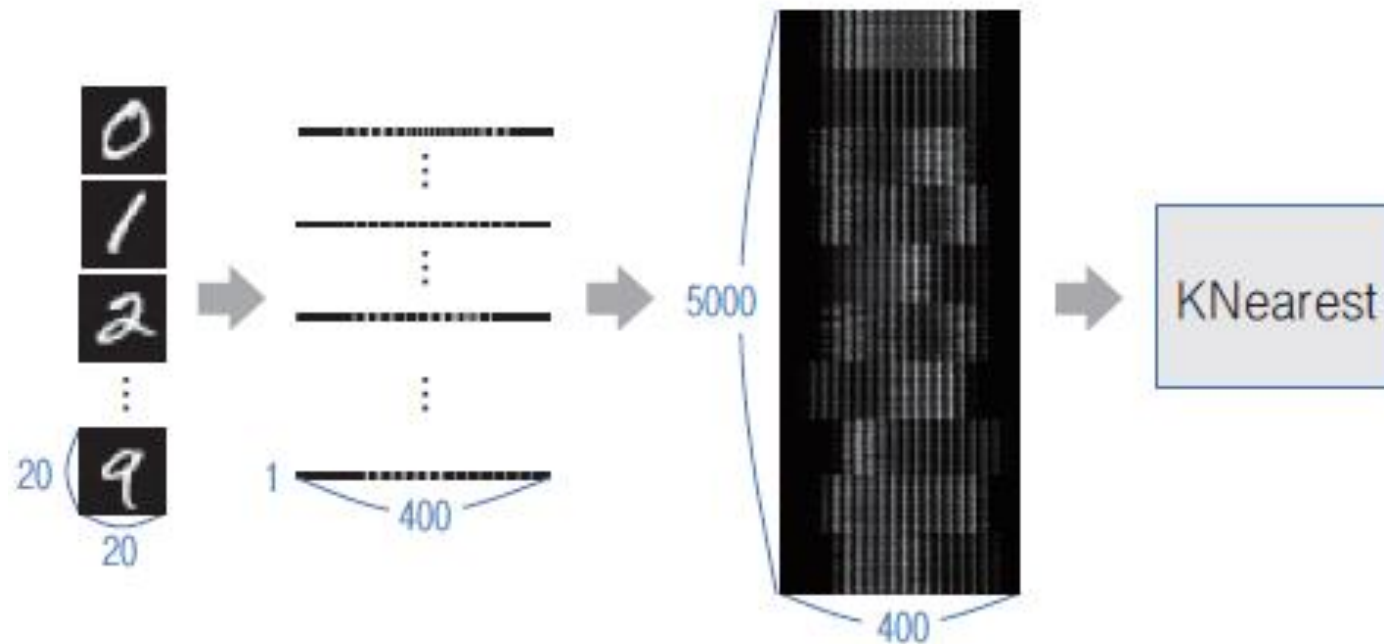
...



❖ kNN을 이용한 필기체 숫자 인식 (6/19)

- 숫자 영상 픽셀 값 자체를 이용하여 KNearest 학습 데이터 행렬을 만드는 과정을 그림 15-7에 나타냄

▼ 그림 15-7 픽셀 값 자체를 이용한 필기체 숫자 학습



❖ kNN을 이용한 필기체 숫자 인식 (7/19)

- 한 장의 숫자 영상은 20×20 픽셀 크기임
- 이 픽셀 값을 모두 일렬로 늘어놓으면 1×400 크기의 행렬로 변환할 수 있음
- 필기체 숫자 학습 데이터 하나는 400개의 숫자 값으로 표현됨
- 이는 400차원 공간에서의 한 점과 같음
- digits.png 영상에 있는 각각의 숫자 영상을 1×400 행렬로 바꿈
- 이 행렬을 모두 세로로 쌓으면 전체 숫자 영상 데이터를 표현하는 5000×400 크기의 행렬을 만들 수 있음
- 이 행렬을 KNearest 클래스의 학습 데이터로 전달함

❖ kNN을 이용한 필기체 숫자 인식 (8/19)

- kNN 알고리즘으로 필기체 숫자 영상을 학습시키려면 각 필기체 숫자 영상이 나타내는 숫자 값을 레이블 행렬로 함께 전달해야 함
- 이 레이블 행렬의 행 크기는 학습 데이터 영상 개수와 같고, 열 크기는 1임
- 그림 15-7에 나타난 학습 데이터 행렬에서 처음 500개 행은 숫자 0에 대한 데이터임
- 그다음 500개 행은 숫자 1에 대한 데이터임
- 레이블 행렬도 처음 500개 행 원소는 0으로 설정하고, 그다음 500개 행은 1로 설정함
- 이와 같은 방식으로 5000×1 행렬 원소를 모두 설정한 후, KNearest 클래스의 레이블 데이터로 전달함

❖ kNN을 이용한 필기체 숫자 인식 (9/19)

- digits.png 영상으로부터 학습 데이터 행렬과 레이블 데이터 행렬을 만듦
- KNearest 객체를 학습시키는 내용의 소스 코드를 코드 15-2에 나타냄

코드 15-2 kNN 알고리즘을 이용한 필기체 숫자 학습 (knndigits.py)

```
1  import sys
2  import numpy as np
3  import cv2
4
5  # Generate training samples and labels
6  digits = cv2.imread('digits.png', cv2.IMREAD_GRAYSCALE)
7
8  if digits is None:
9      print('Image load failed!')
10     sys.exit()
11
```


❖ kNN을 이용한 필기체 숫자 인식 (10/19)

코드 15-2 kNN 알고리즘을 이용한 필기체 숫자 학습 (knndigits.py)

```
12 h, w = digits.shape[:2] # h=1000, w=2000
13
14 cells = []
15 for row in np.vsplit(digits, h//20):      # row.shape = (20, 2000)
16     for col in np.hsplit(row, w//20):      # col.shape = (20, 20)
17         cells.append(col)
18
19 cells = np.array(cells)
20 train_images = cells.reshape(-1, 400).astype(np.float32)
21 train_labels = np.repeat(np.arange(10), len(train_images)/10)
22
23 # Training KNN
24 knn = cv2.ml.KNearest_create()
25 knn.train(train_images, cv2.ml.ROW_SAMPLE, train_labels)
```

// 연산자: 나누기 연산 후 정수 부분의 수만 취함

❖ kNN을 이용한 필기체 숫자 인식 (11/19)

● knndigits.py 소스 코드 설명

- 6행 digits.png 영상을 불러와 digits에 저장합니다.
- 15~16행 digits.png 영상에 가로 100개, 세로 50개의 필기체 숫자가 적혀 있으므로 for 반복문 범위를 동일하게 설정합니다.
- 17행 가로 row번째, 세로 col 번째 필기체 숫자 영상을 cells에 저장합니다.
- 20행 (5000, 20, 20) cells 영상을 (5000, 400) 크기의 영상으로 변환하고, 자료형을 float32로 변환하여 train_images에 저장합니다.
- 21행 5000장의 필기체 숫자 영상의 정답(레이블)을 train_labels에 저장합니다.
- 24~25행 KNearest 객체 knn을 생성하고, kNN 학습을 수행합니다.
train() 메서드에 인자로 사용되는 train_images 행렬 크기는 (5000, 400)이고,
train_labels 행렬 크기는 (5000,)입니다.

❖ kNN을 이용한 필기체 숫자 인식 (12/19)

- knndigits 예제 프로그램은 필기체 숫자 인식 성능을 확인하기 위해 사용자가 직접 마우스로 영상 위에 글씨를 씀
- 학습된 kNN 모델로 확인하는 인터페이스를 제공함
- knndigits 프로그램은 **마우스 이벤트를 처리하여 영상에 그림을 그리는 함수를 만들어 사용**할 것임
- 코드 15-3에 나타난 `on_mouse()` 함수는 마우스 왼쪽 버튼을 누른 상태에서 마우스를 움직이면 해당 위치에 두께 40픽셀로 흰색 글씨를 쓸 수 있음

❖ kNN을 이용한 필기체 숫자 인식 (13/19)

코드 15-3 마우스로 숫자 그리기 (knndigits.py)

```
1  import cv2
2
3  oldx, oldy = -1, -1
4
5  def on_mouse(event, x, y, flags, _):
6      global oldx, oldy
7
8      if event == cv2.EVENT_LBUTTONDOWN:
9          oldx, oldy = x, y
10
11     elif event == cv2.EVENT_LBUTTONUP:
12         oldx, oldy = -1, -1
13
14     elif event == cv2.EVENT_MOUSEMOVE:
15         if flags & cv2.EVENT_FLAG_LBUTTON:
16             cv2.line(img, (oldx, oldy), (x, y), (255, 255, 255), 40, cv2.LINE_AA)
17             oldx, oldy = x, y
18             cv2.imshow('img', img)
```

❖ kNN을 이용한 필기체 숫자 인식 (14/19)

- knndigits.py 소스 코드 설명

- 8~9행 마우스 왼쪽 버튼을 누른 위치를 `oldx`, `oldy`에 저장합니다.
- 11~12행 마우스 왼쪽 버튼을 떼면 `oldx`, `oldy` 좌표를 $(-1, -1)$ 로 초기화합니다.
- 14~17행 마우스 왼쪽 버튼을 누른 상태로 마우스가 움직이면 `(oldx, oldy)` 좌표부터 `(x, y)` 좌표까지 직선을 그립니다.
그리고 `(x, y)` 좌표를 `(oldx, oldy)`로 변경합니다.

❖ kNN을 이용한 필기체 숫자 인식 (15/19)

- 코드 15-4에서는 사용자가 영상 위에 글씨를 쓸 수 있게끔 검정색 화면을 만들어 띄움
- 그리고 마우스 콜백 함수를 등록하는 setMouseCallback() 함수를 호출함
- 키보드 입력을 확인하여 사용자가 [Space] 키를 누를 때마다 사용자가 그린 글씨를 인식하여 실행 결과 창에 출력함

코드 15-4 KNearest 클래스를 이용한 필기체 숫자 인식 (knndigits.py)

```
1 import numpy as np
2 import cv2
3
4 img = np.zeros((400, 400), np.uint8)
5
6 cv2.imshow('img', img)
7 cv2.setMouseCallback('img', on_mouse)
8
```

❖ kNN을 이용한 필기체 숫자 인식 (16/19)

코드 15-4 KNearest 클래스를 이용한 필기체 숫자 인식 (knndigits.py)

```
9  while True:
10     c = cv2.waitKey()
11
12     if c == 27: # ESC Key
13         break
14     elif c == ord(' '): # Space Key, ord(' ') 대신에 32라고 적어도 됨
15         img_resize = cv2.resize(img, (20, 20), interpolation=cv2.INTER_AREA)
16         img_flatten = img_resize.reshape(-1, 400).astype(np.float32)
17
18         _ret, res, _, _ = knn.findNearest(img_flatten, 3)
19         print(int(res[0, 0]))
20
21         img.fill(0)
22         cv2.imshow('img', img)
23
24     cv2.destroyAllWindows()
```

**ord(): 하나의 문자를 입력 받고,
해당 문자에 대응하는 유니코드 정수를 반환**

❖ kNN을 이용한 필기체 숫자 인식 (17/19)

● knndigits.py 소스 코드 설명

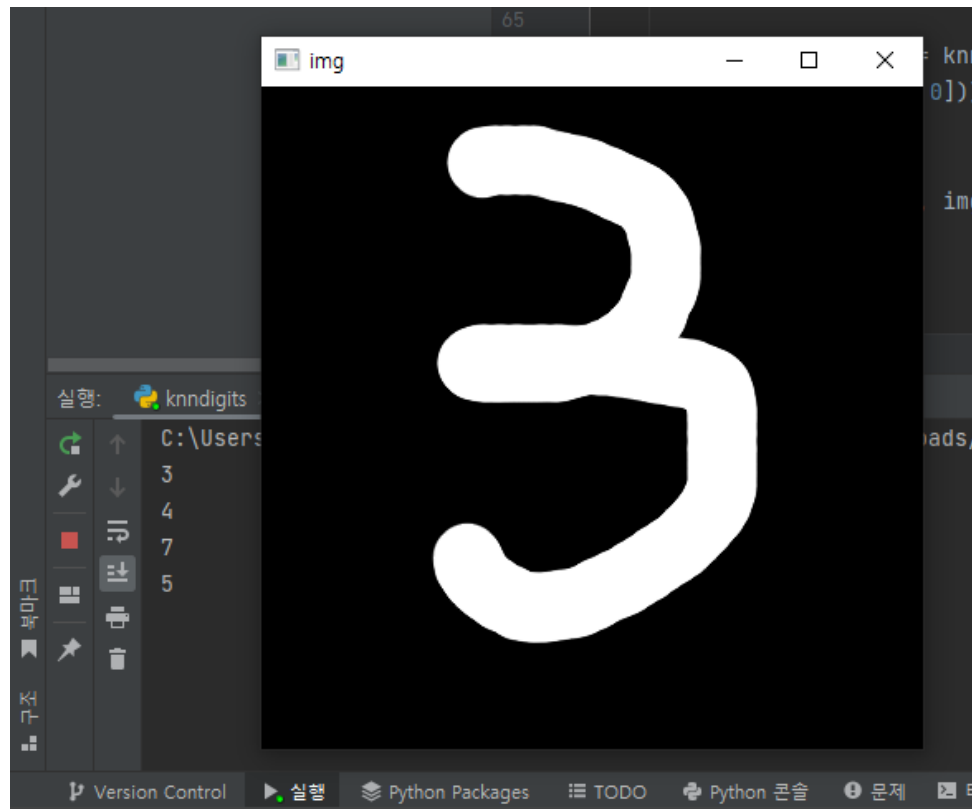
- 4행 (400, 400) 크기의 영상 `img`를 생성합니다.
`img` 영상에 마우스로 글씨를 쓰고 숫자를 인식할 것입니다.
- 12~13행 키보드에서 [ESC] Key를 누르면 프로그램을 종료합니다.
- 14행 키보드에서 [Space] Key를 누르면 필기체 숫자 인식을 수행합니다.
- 15행 숫자가 쓰여진 `img` 영상을 (20, 20) 크기로 변환하여 `img_resize`에 저장합니다.
- 16행 (20, 20) 크기의 `img_resize` 영상을 (400,) 크기의 영상으로 변환하고, 자료형을 `float`로 변환하여 `img_flatten`에 저장합니다.
- 18~19행 kNN 알고리즘으로 분류한 결과를 실행 결과 창에 출력합니다.
- 21~22행 `img` 영상을 검은색으로 초기화한 후 화면에 나타냅니다.

❖ kNN을 이용한 필기체 숫자 인식 (18/19)

- knndigits 프로그램을 실행하면 검은색으로 초기화된 img 영상이 화면에 나타남
- 이 위에서 마우스를 이용하여 숫자를 그릴 수 있음
- 키보드에서 [Space] 키를 누르면 인식된 숫자가 실행 결과 창에 나타남
- 사용자가 필기체 숫자를 img 창 중앙에 적당한 크기로 입력하면 **대체로 정확하게 숫자를 인식하는 것을 확인**할 수 있음
- 다만 글씨 크기를 너무 크거나 작게 입력하면, 또는 중앙이 아닌 위치에 글씨를 입력할 경우에는 인식 결과가 잘못될 수 있음

❖ kNN을 이용한 필기체 숫자 인식 (19/19)

▼ 그림 15-8 KNearest 클래스를 이용한 필기체 숫자 인식 실행 화면



15.3 서포트 벡터 머신

15.1 머신 러닝과 OpenCV

15.3 서포트 벡터 머신

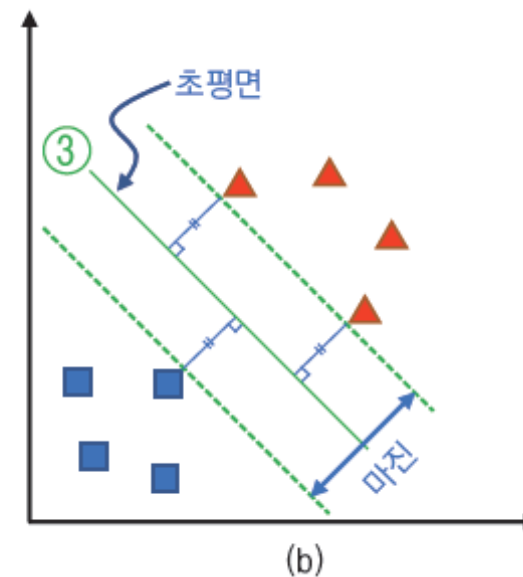
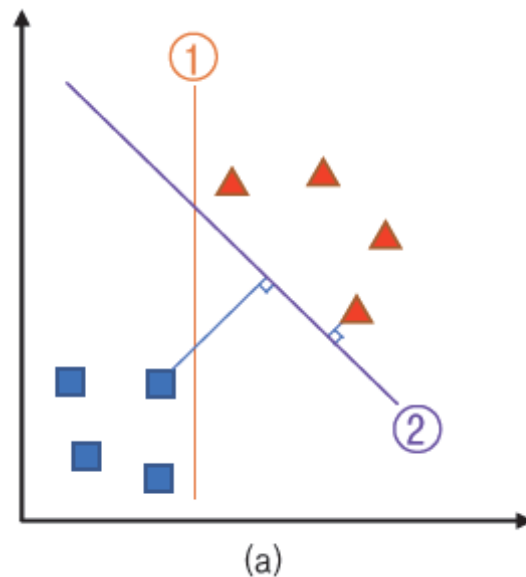
❖ 서포트 벡터 머신 알고리즘 (1/4)

- **서포트 벡터 머신(SVM, Support Vector Machine)**은 기본적으로 두 개의 클래스로 구성된 데이터를 가장 여유 있게 분리하는 초평면(hyperplane)을 찾는 머신 러닝 알고리즘임
- 초평면이란 두 클래스의 데이터를 분리하는 N차원 공간상의 평면을 의미함
- 예를 들어 2차원 공간상의 점들을 분리하는 초평면은 단순한 직선 형태로 정의됨
- 3차원 공간상의 점들을 분리하는 초평면은 3차원 공간에서의 평면의 방정식으로 표현할 수 있음
- SVM 알고리즘은 지도 학습의 일종이며, 분류와 회귀에 사용될 수 있음

❖ 서포트 벡터 머신 알고리즘 (2/4)

- 그림 15-9는 파란색 사각형과 빨간색 삼각형으로 표시된 두 클래스 점들의 분포를 보여 줌
- 이 두 클래스 점들을 구분하기 위한 직선은 매우 다양한 형태로 만들 수 있음
- 그림 15-9(a)에 나타난 1번과 2번 직선은 모두 두 종류의 점들을 잘 분리함

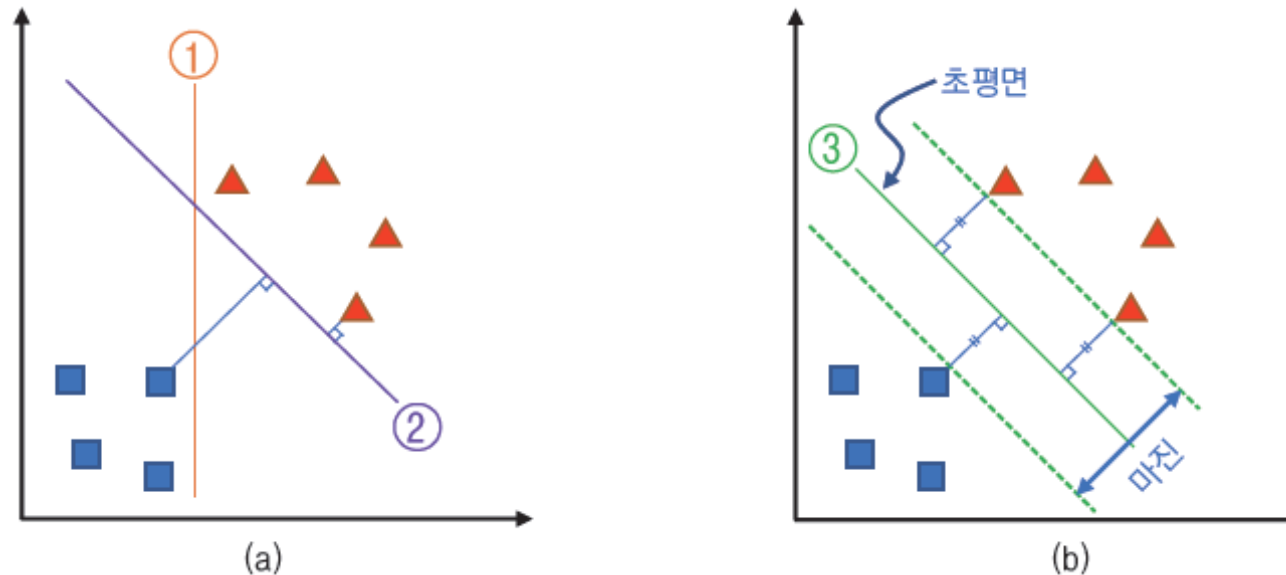
▼ 그림 15-9 SVM 알고리즘으로 두 클래스의 점 분할



❖ 서포트 벡터 머신 알고리즘 (3/4)

- ①번 직선은 조금만 왼쪽 또는 오른쪽으로 이동하면 분리에 실패할 수 있음
- ②번 직선은 왼쪽으로 조금 이동하는 것은 무난하지만, 오른쪽으로 조금만 이동하면 분리에 실패하게 됨
- 이러한 현상이 나타나는 ①번과 ②번 직선이 모두 입력 점 데이터에 너무 가까이 위치하고 있기 때문임

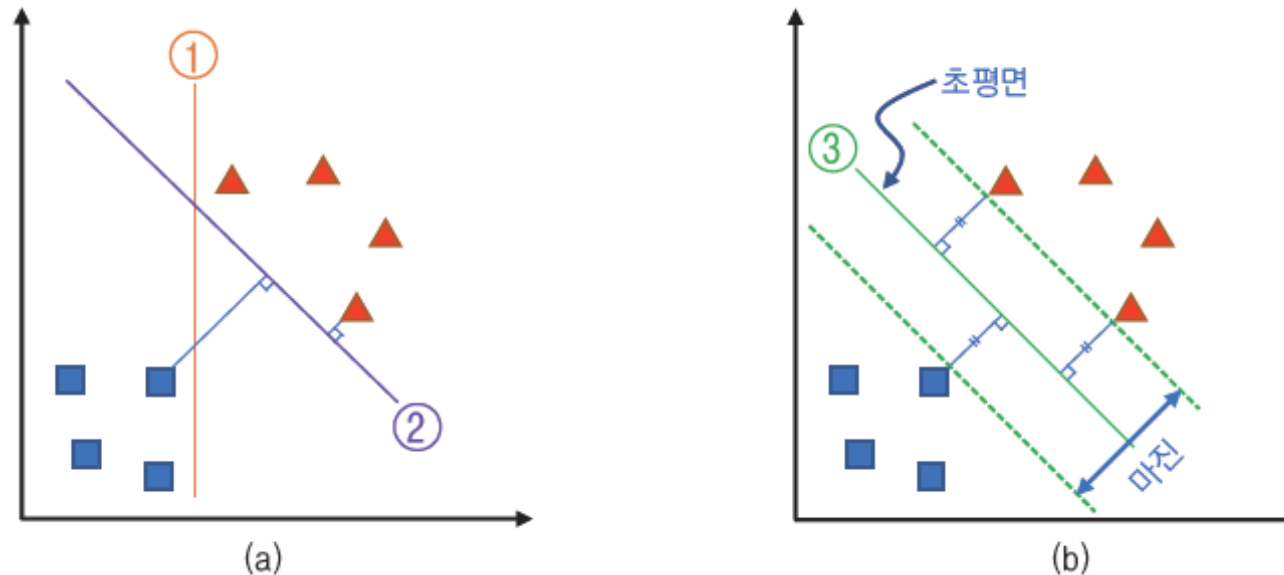
▼ 그림 15-9 SVM 알고리즘으로 두 클래스의 점 분할



❖ 서포트 벡터 머신 알고리즘 (4/4)

- 반면에 그림15-9(b)에서 ③번 직선은 두 클래스 점들 사이를 충분히 여유 있게 분할하고 있음
- 이때 ③번 직선에 해당하는 초평면과 가장 가까이 있는 빨간색 또는 파란색 점과의 거리를 **마진(margin)**이라고 함
- **SVM은 이 마진을 최대로 만드는 초평면을 구하는 알고리즘임**

▼ 그림 15-9 SVM 알고리즘으로 두 클래스의 점 분할



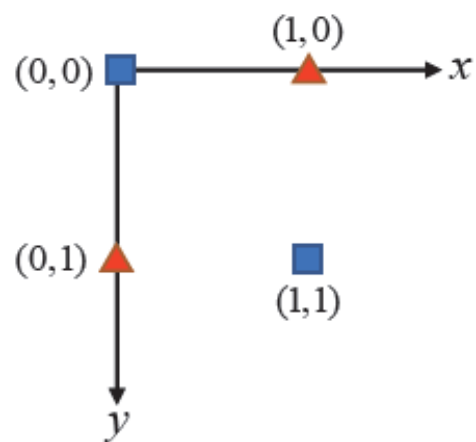
❖ 커널 트릭 (1/4)

- SVM 알고리즘은 기본적으로 선형으로 분리 가능한 데이터에 적용할 수 있음
- 실생활에서 사용하는 데이터는 선형으로 분리되지 않는 경우가 많음
- 이러한 경우에도 SVM 알고리즘을 적용하기 위하여 SVM에서는
커널 트릭(kernel trick)이라는 기법을 사용함
- 커널 트릭이란 적절한 커널 함수를 이용하여 입력 데이터 특징 공간 차원을 늘리는 방식임
- 원본 데이터 차원에서는 선형으로 분리할 수 없었던 데이터를 커널 트릭으로
고차원 특징 공간으로 이동하면 선형으로 분리 가능한 형태로 바뀔 수 있음

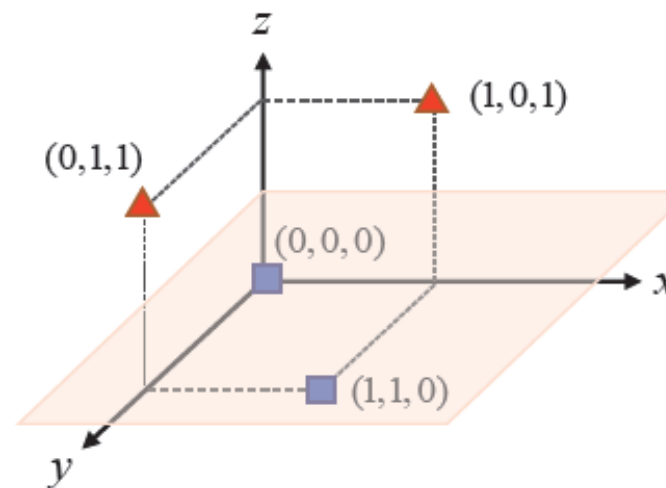
❖ 커널 트릭 (2/4)

- 데이터 특징 공간 차원을 증가시켜서 데이터를 선형 분리하는 예를 살펴보자
- 2차원 좌표 평면상의 점 집합 $X = \{(0,0), (1,1)\}$ 와 $Y = \{(1,0), (0,1)\}$ 가 있다고 가정함
- 이 두 클래스 점들을 그림 15-10(a)에 각각 파란색 사각형과 빨간색 삼각형으로 나타냄
- 2차원 평면상에서 X 와 Y 두 클래스 점들을 분리할 수 있는 직선은 존재하지 않음

▼ 그림 15-10 비선형 데이터에 커널 트릭 적용하기



(a)

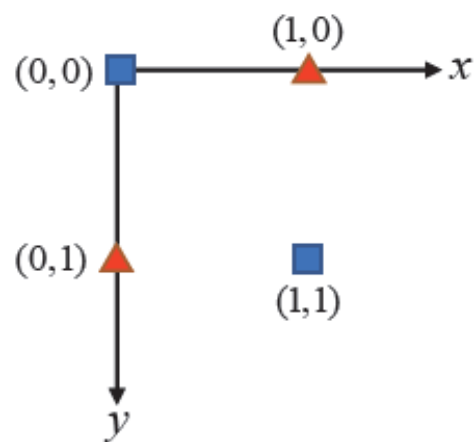


(b)

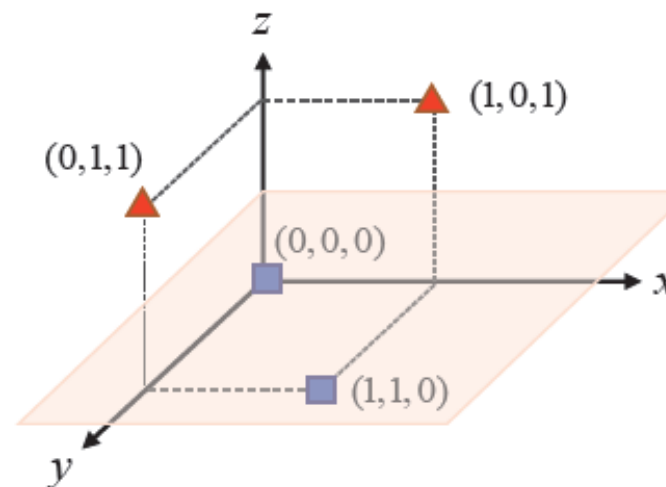
❖ 커널 트릭 (3/4)

- 입력 점들의 좌표에 가상의 z 축 좌표를 $z_i = |x_i - y_i|$ 형태로 추가할 경우, $X = \{(0,0,0), (1,1,0)\}$ 와 $Y = \{(1,0,1), (0,1,1)\}$ 형태로 3차원 공간상에서의 점 집합으로 바뀌게 됨
- 이렇게 3차원 공간으로 변경된 X 와 Y 점들을 그림 15-10(b)에 나타냄
- 이 두 클래스 점들은 $z = 0.5$ 평면의 방정식을 이용하여 효과적으로 분리할 수 있음
- 2차원 평면에서 선형 분리할 수 없었던 X 와 Y 데이터 집합이 가상의 차원이 추가됨으로써 선형으로 분리할 수 있게 됨

▼ 그림 15-10 비선형 데이터에 커널 트릭 적용하기



(a)



(b)

❖ 커널 트릭 (4/4)

- 표 15-3에서 가장 널리 사용하는 커널은 **방사 기저 함수(RBF, radial basis function)**임
- 이 커널을 사용할 때에는 γ 인자 값을 적절하게 설정해야 함
- 만약 입력 데이터가 선형으로 분리가 가능하다면 선형 커널을 사용하는 것이 가장 빠르게 동작함

▼ 표 15-3 다양한 SVM 커널

SVM 커널	커널 함수
선형(linear)	$K(x_i, x_j) = x_i^T x_j$
다항식(polynomial)	$K(x_i, x_j) = (\gamma x_i^T x_j + c_0)^{\text{degree}}, \gamma > 0$
방사 기저 함수(radial basis function)	$K(x_i, x_j) = \exp(-\gamma \ x_i - x_j\ ^2), \gamma > 0$
시그모이드(sigmoid)	$K(x_i, x_j) = \tanh(\gamma x_i^T x_j + c_0)$
지수 카이 제곱(exponential chi-square)	$K(x_i, x_j) = \exp\left(-\gamma \frac{(x_i - x_j)^2}{(x_i + x_j)}\right), \gamma > 0$
히스토그램 교차(histogram intersection)	$K(x_i, x_j) = \min(x_i, x_j)$

❖ SVM 클래스 사용하기 (1/15)

- OpenCV에서 SVM 알고리즘은 같은 이름의 SVM 클래스에 구현되어 있음
- SVM 클래스는 ml 모듈에 포함되어 있음
- OpenCV에 구현된 SVM 클래스는 유명한 오픈 소스 라이브러리인 LIBSVM을 기반으로 만들어짐
- SVM 클래스를 이용하려면 먼저 SVM 객체를 생성해야 함
- SVM 객체는 SVM_create() 메서드를 사용하여 생성할 수 있음

```
svm = cv2.SVM_create()
```

svm	SVM 객체
-----	--------

❖ SVM 클래스 사용하기 (2/15)

- SVM 클래스 객체를 생성한 후, 학습 데이터를 학습하기 전에 먼저 SVM 알고리즘 속성을 설정해야 함
- 대표적으로 설정해야 할 SVM 클래스 속성에는 타입과 커널 함수 선택이 있음
- 먼저 SVM 타입을 설정하는 `svm.setType()` 함수 원형은 다음과 같음

```
svm.setType(val)
```

val	SVM 타입. 열거형 상수 중 하나를 지정합니다.
-----	-----------------------------

❖ SVM 클래스 사용하기 (3/15)

- SVM 클래스는 기본적으로 `cv2.ml.SVM_C_SVC` 타입을 사용하도록 초기화됨
- `cv2.ml.SVM_C_SVC` 타입은 일반적인 N-클래스 분류 문제에서 사용되는 방식임
- `cv2.ml.SVM_C_SVC`가 아닌 다른 타입을 사용하려면 `setType()` 함수를 이용하여 타입을 변경해야 함
- `setType()` 함수의 `val` 인자에는 열거형 상수 중 하나를 지정할 수 있음

▼ 표 15-4 SVM 타입 열거형 상수

SVM::Types 열거형 상수	설명	파라미터
C_SVC	C-서포트 벡터 분류. 일반적인 n-클래스 분류 문제에서 사용됩니다.	C
NU_SVC	v-서포트 벡터 분류. C_SVC와 비슷하지만 Nu 값 범위가 0~1 사이로 정규화되어 있습니다.	Nu
ONE_CLASS	1-분류 서포트 벡터 머신. 데이터 분포 측정에 사용됩니다.	C, Nu
EPS_SVR	ϵ -서포트 벡터 회귀	P, C
NU_SVR	v-서포트 벡터 회귀	Nu, C

❖ SVM 클래스 사용하기 (4/15)

- `cv2.ml.SVM_C_SVC` 타입을 사용하는 경우, SVM 알고리즘 내부에서 사용하는 C 파라미터 값을 적절하게 설정해야 함
- C 값을 작게 설정하면 학습 데이터 중에 잘못 분류되는 데이터가 있어도 최대 마진을 선택함
- C 값을 크게 설정하면 마진이 작아지더라도 잘못 분류되는 데이터가 적어지도록 분류함
- 만약 학습 샘플 데이터에 잡음 또는 이상치 데이터가 많이 포함되어 있는 경우에는 C 파라미터 값을 조금 크게 설정하는 것이 유리함

❖ SVM 클래스 사용하기 (5/15)

- SVM 타입을 설정하였으면, SVM 알고리즘에서 사용할 커널 함수를 지정해야 함

```
svm.setKernel(kernelType)
```

kernelType	커널 함수 종류. 열거형 상수 중 하나를 지정합니다.
------------	-------------------------------

❖ SVM 클래스 사용하기 (6/15)

- 참고로 SVM 클래스는 기본적으로 `cv2.ml.SVM_RBF` 커널을 사용하도록 초기화됨

▼ 15-5 주요 SVM 커널 열거형 상수

SVM::KernelTypes 열거형 상수	설명	파라미터
LINEAR	선형 커널	
POLY	다항식 커널	Degree, Gamma, Coef0
RBF	방사 기저 함수 커널	Gamma
SIGMOID	시그모이드 커널	Gamma, Coef0
CHI2	지수 카이 제곱 커널	Gamma
INTER	히스토그램 교차 커널	

❖ SVM 클래스 사용하기 (7/15)

- SVM 알고리즘 타입과 커널 함수 종류를 설정한 후에는 각각의 타입과 커널 함수 정의에 필요한 파라미터를 설정해야 함
- SVM 클래스에서 설정할 수 있는 파라미터는 C, Nu, P, Degree, Gamma, Coef0 등이 있음
- 이들 파라미터는 차례대로 1, 0, 0, 0, 1, 0으로 초기화됨
- 각각의 파라미터는 파라미터 이름에 해당하는 setXXX()와 getXXX() 함수를 이용하여 값을 설정하거나 읽어 올 수 있음
- 예를 들어 cv2.ml.SVM_C_SVC 타입을 사용하고 cv2.ml.SVM_RBF 커널을 사용할 경우, svm.setC() 메서드와 svm.setGamma() 메서드를 사용하여 적절한 C와 Gamma 파라미터 값을 설정해야 함

❖ SVM 클래스 사용하기 (8/15)

- SVM 객체를 생성하고 타입과 커널 함수 선택, 그리고 파라미터를 설정함
- `svm.train()` 메서드를 이용하여 학습시킬 수 있음
- SVM에서 사용하는 파라미터를 적절하게 설정하지 않으면 학습이 제대로 되지 않는 경우가 발생함
- 사실 대부분의 학습 데이터는 다차원간에서 다양한 분포와 형태를 갖기 때문에 SVM 파라미터 값을 **어떻게 설정해야 학습이 잘 될 것인지를 직관적으로 알기 어려움**
- OpenCV의 SVM 클래스는 각각의 파라미터에 대해 설정 가능한 값을 적용해 봄
- 그중 가장 성능이 좋은 파라미터를 자동으로 찾아 학습하는 `svm.trainAuto()` 메서드를 제공함

`svm.trainAuto(samples, layout, responses)`

<code>samples</code>	학습 데이터 행렬
<code>layout</code>	학습 데이터 배치 방법. ROW_SAMPLE 또는 COL_SAMPLE를 지정합니다.
<code>responses</code>	각 학습 데이터에 대응되는 응답 벡터

❖ SVM 클래스 사용하기 (9/15)

- 2차원 평면에서 두 개의 클래스로 구성된 점들을 SVM 알고리즘으로 분류함
- 그 경계면을 화면에 표시하는 예제 프로그램 소스 코드를 코드 15-5에 나타냄
- SVM 알고리즘을 이용하여 이 두 점들을 효과적으로 분리하는 초평면을 구하여 화면에 나타냄

코드 15-5 SVM 알고리즘을 이용한 2차원 점 분류 (svmplane.py)

```
1  import numpy as np
2  import cv2
3
4  train = np.array([[150, 200], [200, 250],
5                    [100, 250], [150, 300],
6                    [350, 100], [400, 200],
7                    [400, 300], [350, 400]]).astype(np.float32)
8
9  label = np.array([0, 0, 0, 0, 1, 1, 1, 1])
10
11  svm = cv2.ml.SVM_create()
12  svm.setType(cv2.ml.SVM_C_SVC)
13  svm.setKernel(cv2.ml.SVM_RBF)
14  svm.trainAuto(train, cv2.ml.ROW_SAMPLE, label)
15
```

❖ SVM 클래스 사용하기 (10/15)

코드 15-5 SVM 알고리즘을 이용한 2차원 점 분류 (svmplane.py)

```
16  img = np.zeros((500, 500, 3), np.uint8)
17
18  for j in range(img.shape[0]):
19      for k in range(img.shape[1]):
20          test = np.array([[k, j]], dtype=np.float32)
21          _, res = svm.predict(test)
22
23          if res == 0:
24              img[j, k] = (128, 128, 255)
25          elif res == 1:
26              img[j, k] = (128, 255, 128)
27
28  color = [(0, 0, 128), (0, 128, 0)]
29
30  for k in range(train.shape[0]):
31      x = int(train[k, 0])
32      y = int(train[k, 1])
33      l = label[k]
34
35      cv2.circle(img, (x, y), 5, color[l], -1, cv2.LINE_AA)
36
37  cv2.imshow('svm', img)
38  cv2.waitKey()
39  cv2.destroyAllWindows()
```

❖ SVM 클래스 사용하기 (11/15)

- svmplane.py 소스 코드 설명

- 4~7행 여덟 개 점 좌표를 포함하는 `train` 행렬을 생성합니다.
- 9행 학습 데이터 점들의 클래스를 정의한 `label` 행렬을 생성합니다.
처음 네 개의 점의 클래스는 0이고, 나머지 네 개 점의 클래스는 1입니다.
- 11행 SVM 객체를 생성하여 `svm`에 저장합니다.
- 12행 SVM 타입을 `C_SVC`로 설정합니다.
- 13행 SVM 커널 함수를 `RBF`로 설정합니다.
- 14행 `trainAuto()` 메서드를 사용하여 최적의 파라미터 `C`와 `Gamma`를 자동으로 찾아 학습합니다.
- 16행 SVM 분류 결과를 나타낼 `img` 영상을 생성합니다.

❖ SVM 클래스 사용하기 (12/15)

- svmplane.py 소스 코드 설명

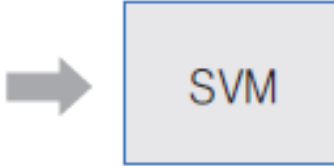
- 18~26행 `img` 영상의 모든 픽셀 좌표에 대해 SVM 응답을 구하여 빨간색 또는 녹색으로 표현합니다.
- 30~35행 `train` 행렬에 저장된 학습 데이터 점을 반지름 5인 원으로 표시합니다.
0번 클래스 점은 빨간색 원으로, 1번 클래스 점은 녹색 원으로 그립니다.

❖ SVM 클래스 사용하기 (13/15)

- 코드 15-5에서는 SVM 타입은 C_SVC로 설정하였고, 커널 함수는 방사 기저 함수를 사용함
- SVM 학습을 위한 입력으로는 train 행렬과 label 행렬을 지정하였고, 이 두 행렬의 모양과 원소 값을 그림 15-11에 나타냄

▼ 그림 15-11 SVM 학습을 위한 train과 label 행렬

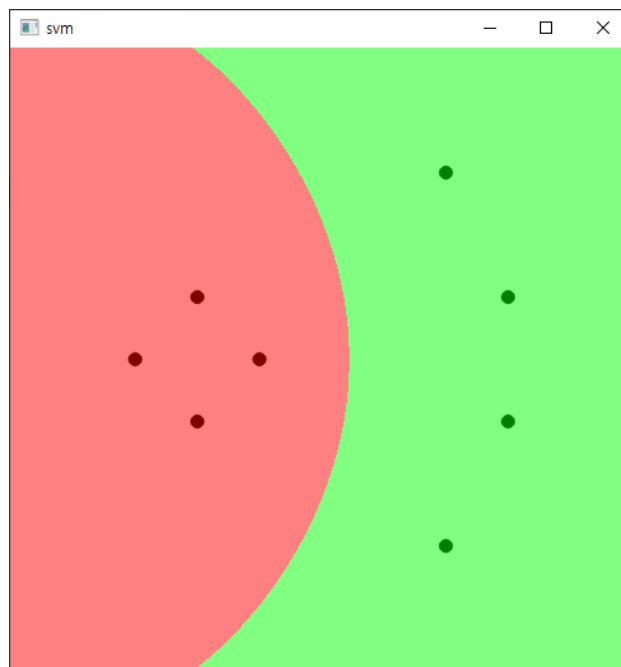
train		label
150	200	0
200	250	0
100	250	0
150	300	0
350	100	1
400	200	1
400	300	1
350	400	1



❖ SVM 클래스 사용하기 (14/15)

- 빨간색 원으로 표시된 점들이 0번 클래스이고, 녹색 점으로 표시된 점들은 1번 클래스임
- img 영상의 전체 픽셀 좌표를 학습된 SVM 분류기의 시험 데이터로 사용하여 그 결과를 빨간색 또는 녹색으로 나타냄
- RBF를 SVM 커널로 사용하였기 때문에 두 클래스 경계면이 곡선의 형태로 나타나고 있음

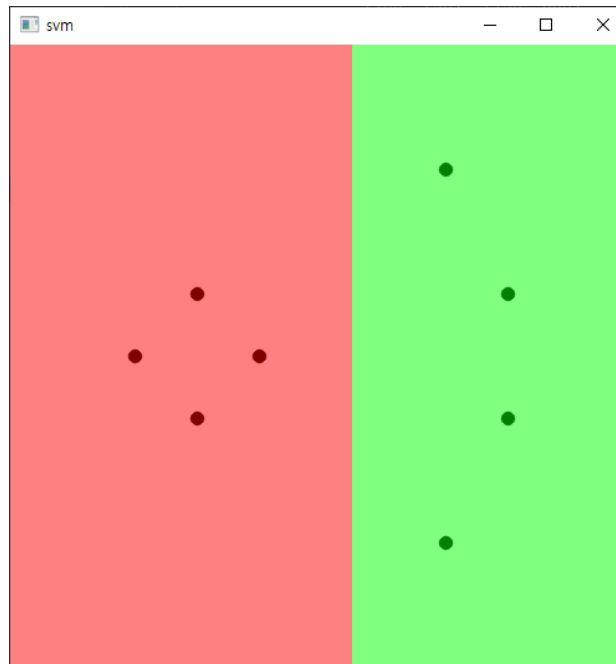
▼ 그림 15-12 SVM 알고리즘을 이용한 2차원 점 분류 실행 결과



❖ SVM 클래스 사용하기 (15/15)

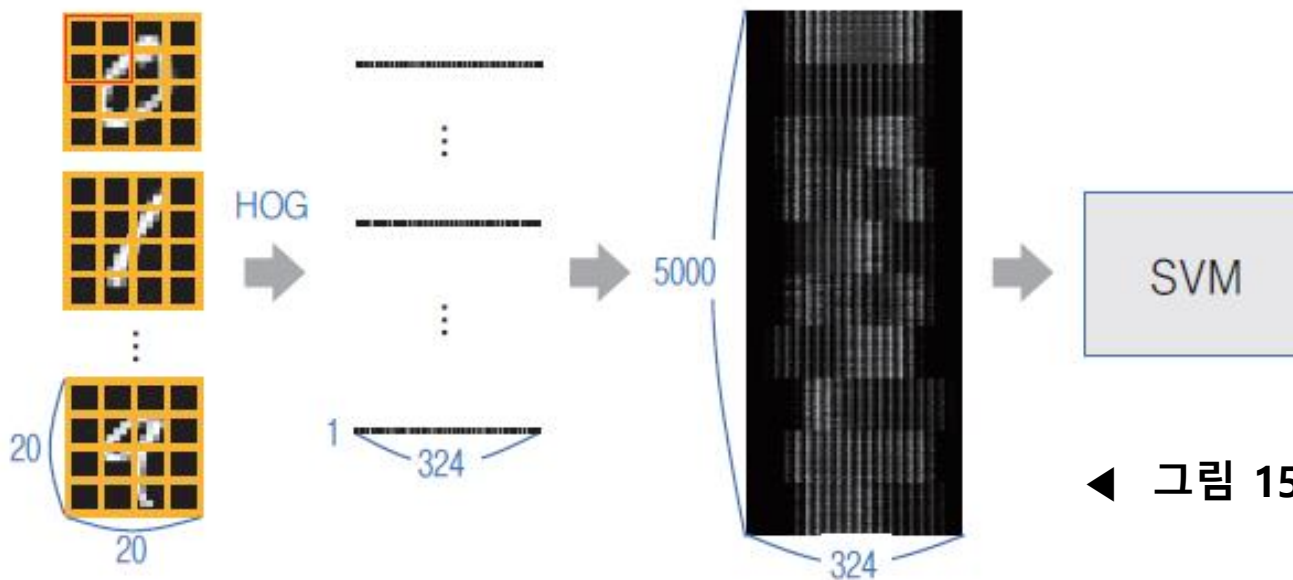
- 코드 15-5에서 커널 함수를 `cv2.ml.SVM_LINEAR`로 변경할 경우, 아래 그림에서와 같이 두 점들을 세로로 양분하는 형태의 경계면이 만들어짐

커널 함수: `cv2.ml.SVM_LINEAR`



❖ HOG & SVM 필기체 숫자 인식 (1/14)

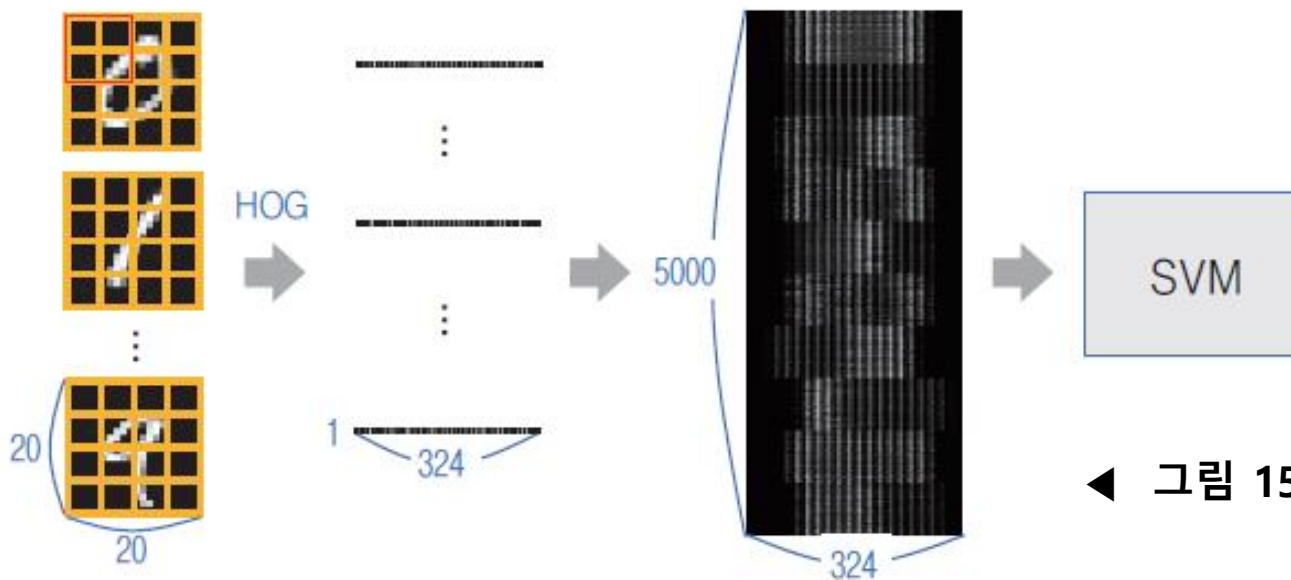
- HOG는 입력 영상을 일정 크기의 셀(cell)로 나누고, 2×2 셀을 합쳐 하나의 블록(block)으로 설정함
- 필기체 숫자 영상 하나의 크기는 20×20 이므로 여기서는 셀 하나의 크기를 5×5 로 지정함
- 블록 하나의 크기는 10×10 크기로 설정함
- 셀 하나에서 그래디언트 방향 히스토그램은 아홉 개의 bin으로 구성함



◀ 그림 15-13 HOG 특징 벡터를 이용한 필기체 숫자 학습

❖ HOG & SVM 필기체 숫자 인식 (2/14)

- 블록 하나에서는 $9 \times 4 = 36$ 개의 bin으로 구성된 히스토그램 정보가 추출됨
- 블록은 보통 하나의 셀 단위로 이동하므로 가로로 세 개, 세로로 세 개 만들 수 있음
- 필기체 숫자 영상 하나에서 만들어지는 HOG 특징 벡터의 차원 크기는 $36 \times 9 = 324$ 로 결정됨
- 20×20 숫자 영상 하나에서 1×324 특징 벡터 행렬이 만들어짐
- 이 행렬을 모두 세로로 쌓으면 5000×324 크기의 HOG 특징 벡터 행렬을 만들 수 있음
- 이 행렬을 SVM 클래스의 학습 데이터로 전달함



◀ 그림 15-13 HOG 특징 벡터를 이용한 필기체 숫자 학습

❖ HOG & SVM 필기체 숫자 인식 (3/14)

- 필기체 숫자 영상에서 HOG 특징 벡터를 추출하려면 HOGDescriptor 클래스를 사용함
- HOGDescriptor 클래스는 임의의 영상에서 HOG 기술자를 추출하는 기능도 제공함
- 임의의 영상에서 HOG 기술자를 추출하려면 HOGDescriptor 객체를 먼저 생성해야 함
- 이때 HOGDescriptor 클래스 기본 생성자를 사용하면 보행자 검출 시 사용한 설정을 기반으로 HOGDescriptor 객체가 생성됨
- 20×20 영상에서 5×5 셀과 10×10 블록을 사용하는 HOG 기술자를 생성하려면 다음의 생성자를 사용해야 함

```
cv2.HOGDescriptor(_winSize, _blockSize, _blockStride, _cellSize, _nbins)
```

_winSize	검출 윈도우 크기
_blockSize	블록 크기
_blockStride	블록 이동 크기
_cellSize	셀 크기
_nbins	히스토그램 빈 개수

❖ HOG & SVM 필기체 숫자 인식 (4/14)

- 이 생성자는 매우 많은 인자를 받도록 되어 있지만, 뒷부분의 많은 인자는 기본값이 설정되어 있음
- `_winSize`, `_blockSize`, `_blockStride`, `_cellSize`, `_nbins` 인자만 적절하게 지정하여 사용할 수 있음
- 20×20 영상에서 5×5 셀과 10×10 크기의 블록을 사용함
- 각 셀마다 아홉 개의 그래디언트 방향 히스토그램을 구하도록 설정하려면 다음과 같이 `HOGDescriptor` 객체를 생성함

```
cv2.HOGDescriptor((20, 20), (10, 10), (5, 5), (5, 5), 9)
```

❖ HOG & SVM 필기체 숫자 인식 (5/14)

- 일단 HOGDescriptor 객체를 생성한 후에는 compute() 메서드를 이용하여 HOG 기술자를 계산할 수 있음

```
descriptors = hog.compute(img)
```

img	입력 영상
descriptors	출력 HOG 기술자

❖ HOG & SVM 필기체 숫자 인식 (6/14)

- digits.png 영상으로부터 HOG 특징 벡터를 추출하여 SVM 알고리즘으로 학습시킴
- svmdigits 예제 프로그램은 앞서 설명한 knndigits 예제 프로그램과 동작 방식이 거의 동일함
- 다만 내부적으로 사용하는 머신 러닝 알고리즘과 머신 러닝에서 학습에 사용하는 특징 벡터 종류만 다름

코드 15-6 SVM 알고리즘을 이용한 필기체 숫자 인식 (svmdigits.py)

```
1  import sys
2  import numpy as np
3  import cv2
4
5
6  oldx, oldy = -1, -1
7
8
9  def on_mouse(event, x, y, flags, _):
10     global oldx, oldy
11
12     if event == cv2.EVENT_LBUTTONDOWN:
13         oldx, oldy = x, y
14
```


❖ HOG & SVM 필기체 숫자 인식 (7/14)

코드 15-6 SVM 알고리즘을 이용한 필기체 숫자 인식 (svmdigits.py)

```
15 elif event == cv2.EVENT_MOUSEMOVE:
16     if flags & cv2.EVENT_FLAG_LBUTTON:
17         cv2.line(img, (oldx, oldy), (x, y), (255, 255, 255), 40, cv2.LINE_AA)
18         oldx, oldy = x, y
19         cv2.imshow('img', img)
20
21 # Generate training samples and labels
22 digits = cv2.imread('digits.png', cv2.IMREAD_GRAYSCALE)
23 if digits is None:
24     print('Image load failed!')
25     sys.exit()
26
27 h, w = digits.shape[:2]
28 hog = cv2.HOGDescriptor((20, 20), (10, 10), (5, 5), (5, 5), 9)
29
30 cells = []
31 for row in np.vsplit(digits, h//20): # row.shape = (20, 2000)
32     for col in np.hsplit(row, w//20): # cell.shape = (20, 20)
33         cells.append(col)
34
35 cells = np.array(cells) # (5000, 20, 20)
36
```

❖ HOG & SVM 필기체 숫자 인식 (8/14)

코드 15-6 SVM 알고리즘을 이용한 필기체 숫자 인식 (svmdigits.py)

```
37 desc = []
38 for img in cells:
39     dd = hog.compute(img)
40     desc.append(dd)
41
42 train_desc = np.array(desc).squeeze().astype(np.float32)
43 train_labels = np.repeat(np.arange(10), len(train_desc)/10)
44
45 # Training SVM
46 svm = cv2.ml.SVM_create()
47 svm.setType(cv2.ml.SVM_C_SVC)
48 svm.setKernel(cv2.ml.SVM_RBF)
49 svm.setC(2.5)
50 svm.setGamma(0.50625)
51 svm.train(train_desc, cv2.ml.ROW_SAMPLE, train_labels)
52
53 # Tests
54 img = np.zeros((400, 400), np.uint8)
55
56 cv2.imshow('img', img)
57 cv2.setMouseCallback('img', on_mouse)
58
```

❖ HOG & SVM 필기체 숫자 인식 (9/14)

코드 15-6 SVM 알고리즘을 이용한 필기체 숫자 인식 (svmdigits.py)

```
59 while True:
60     c = cv2.waitKey()
61
62     if c == 27:    # [ESC] Key
63         break
64     elif c == ord(' '):    # [Space] Key
65         img_resize = cv2.resize(img, (20, 20), interpolation=cv2.INTER_AREA)
66
67         desc = hog.compute(img_resize)
68         test_desc = np.array(desc).astype(np.float32)
69         test_desc = test_desc.reshape(1, len(test_desc))
70         _, res = svm.predict(test_desc)
71         print(int(res[0, 0]))
72
73         img.fill(0)
74         cv2.imshow('img', img)
75
76 cv2.destroyAllWindows()
```

❖ HOG & SVM 필기체 숫자 인식 (10/14)

- svmdigits.py 소스 코드 설명

- 28행 HOGDescriptor 객체 hog를 생성합니다.
- 37~43행 digits.png에 포함된 5000개의 필기체 숫자 부분 영상으로부터 각각 HOG 특징 벡터를 추출하여 (5000, 324) 크기의 train_desc 행렬과 (5000,) 크기의 train_labels 행렬을 생성합니다.
- 46행 SVM 객체를 생성합니다.
- 47~48행 SVM 타입은 C_SVC로 설정하고, 커널 함수는 RBF로 설정합니다.
- 49~50행 파라미터 C와 Gamma 값을 각각 2.5, 0.50625로 설정합니다.
- 51행 SVM 학습을 진행합니다.
- 64~71행 img 창에서 [Space] 키를 누르면 img 영상을 (20, 20) 크기로 변환한 후 HOG 특징 벡터를 계산합니다. 계산된 HOG 특징 벡터를 (1, 324) 크기의 행렬로 변환하여 SVM 결과를 예측하고, 그 결과를 실행 결과 창에 출력합니다.

❖ HOG & SVM 필기체 숫자 인식 (11/14)

- 코드 15-6의 49~50행에서 설정한 C와 Gamma 파라미터 값은 사실 svm.trainAuto() 메서드를 이용하여 구한 값임
- 51행 svm.train() 메서드 호출 코드 대신 다음 코드를 실행하고, 실행 결과 창에 출력되는 C와 Gamma 값을 코드 15-6의 49~50행에서 사용한 것임

```
svm.trainAuto(train_desc, cv2.ml.ROW_SAMPLE, train_labels)
print(svm.getC())
print(svm.getGamma())
```

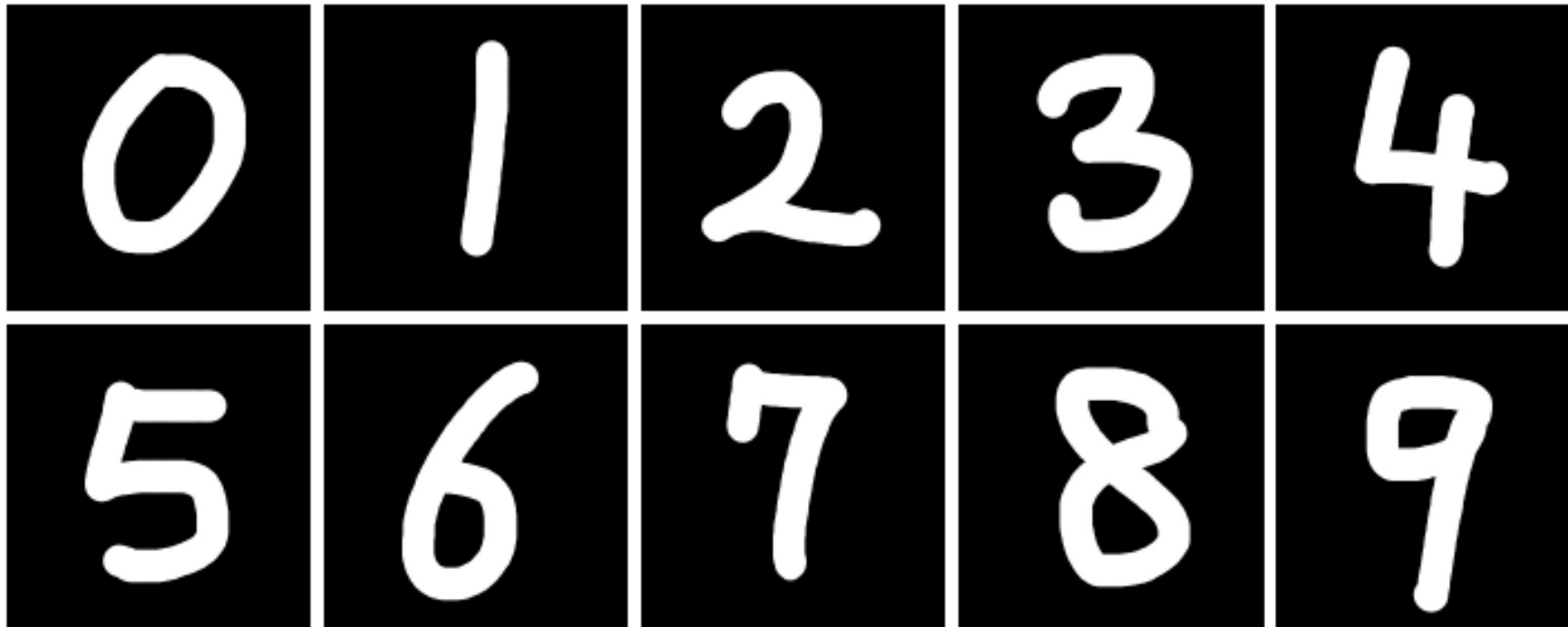
❖ HOG & SVM 필기체 숫자 인식 (12/14)

- 코드 15-6에서 `svm.train()` 메서드 대신 `svm.trainAuto()` 메서드를 호출하면 상당히 오랜 시간 동안 학습이 진행됨
- 최적의 파라미터를 알아낸 후에는 `svm.trainAuto()` 메서드 대신 `svm.train()` 메서드를 사용하는 것이 효율적임
- 코드 15-6의 `svmdigits` 예제 프로그램을 실행하면 검은색으로 초기화된 `img` 창이 나타남
- `img` 창에 마우스로 숫자를 쓰고 키보드의 [Space] 키를 누르면 인식된 결과 숫자가 실행 결과 창에 나타남

❖ HOG & SVM 필기체 숫자 인식 (13/14)

- svmdigits 프로그램에서 제대로 숫자가 인식된 영상의 예를 그림 15-14에 나타남
- 그림 15-14에 나타낸 영상은 img 창에 실제로 마우스로 쓴 숫자 영상이며, 각각의 숫자는 모두 정상적으로 인식됨

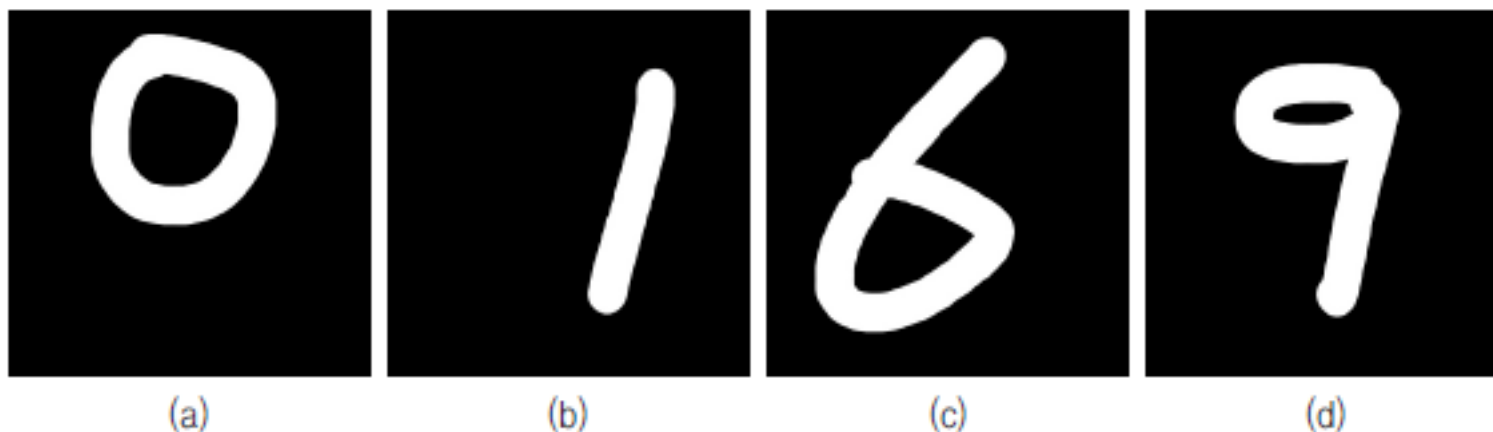
▼ 그림 15-14 svmdigits 예제 프로그램에서 인식에 성공한 영상의 예



❖ HOG & SVM 필기체 숫자 인식 (14/14)

- 반면에 정상적으로 인식되지 않은 필기체 숫자의 예를 그림 15-15에 나타냄
- 그림 15-15(a)는 숫자 0을 적었지만 2로 인식되었고, 그림 15-15(b)는 숫자 1을 적었지만 4로 인식됨
- 그림 15-15(c)는 숫자 6을 적었지만 5로 인식되었고, 그림 15-15(d)는 숫자 9를 적었지만 7로 인식됨
- 숫자를 중앙이 아닌 한편으로 치우치게 적거나, 다른 숫자와 비슷한 형태로 쓴 경우에 잘못 인식하는 경우가 많이 발생함

▼ 그림 15-15 svmdigits 예제 프로그램에서 인식에 실패한 영상의 예



THANK YOU!

Q & A

- Name: 권범
- Office: 동양미래대학교 2호관 704호 (02-2610-5238)
- E-mail: bkwon@dongyang.ac.kr
- Homepage: <https://sites.google.com/view/beomkwon/home>