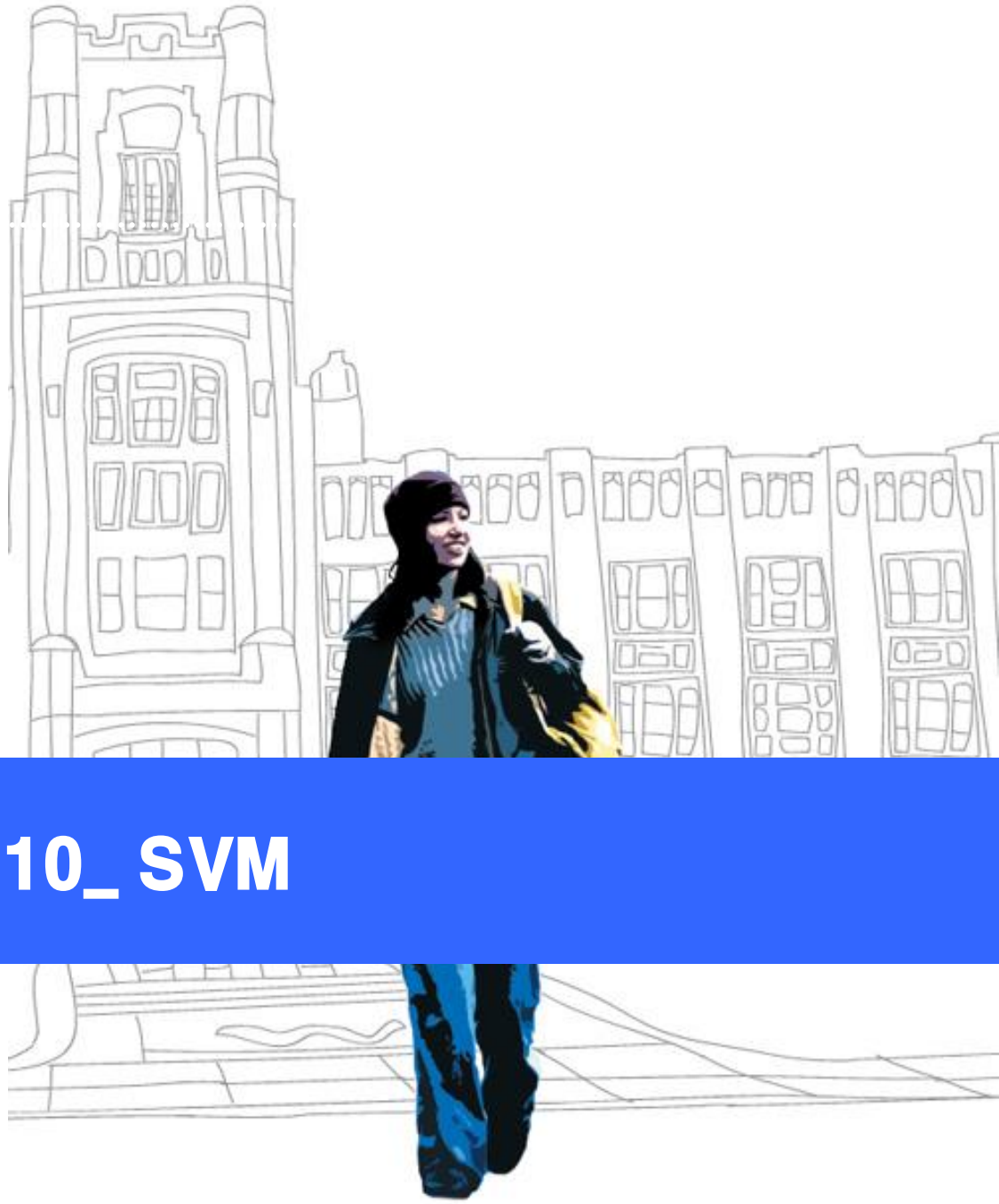
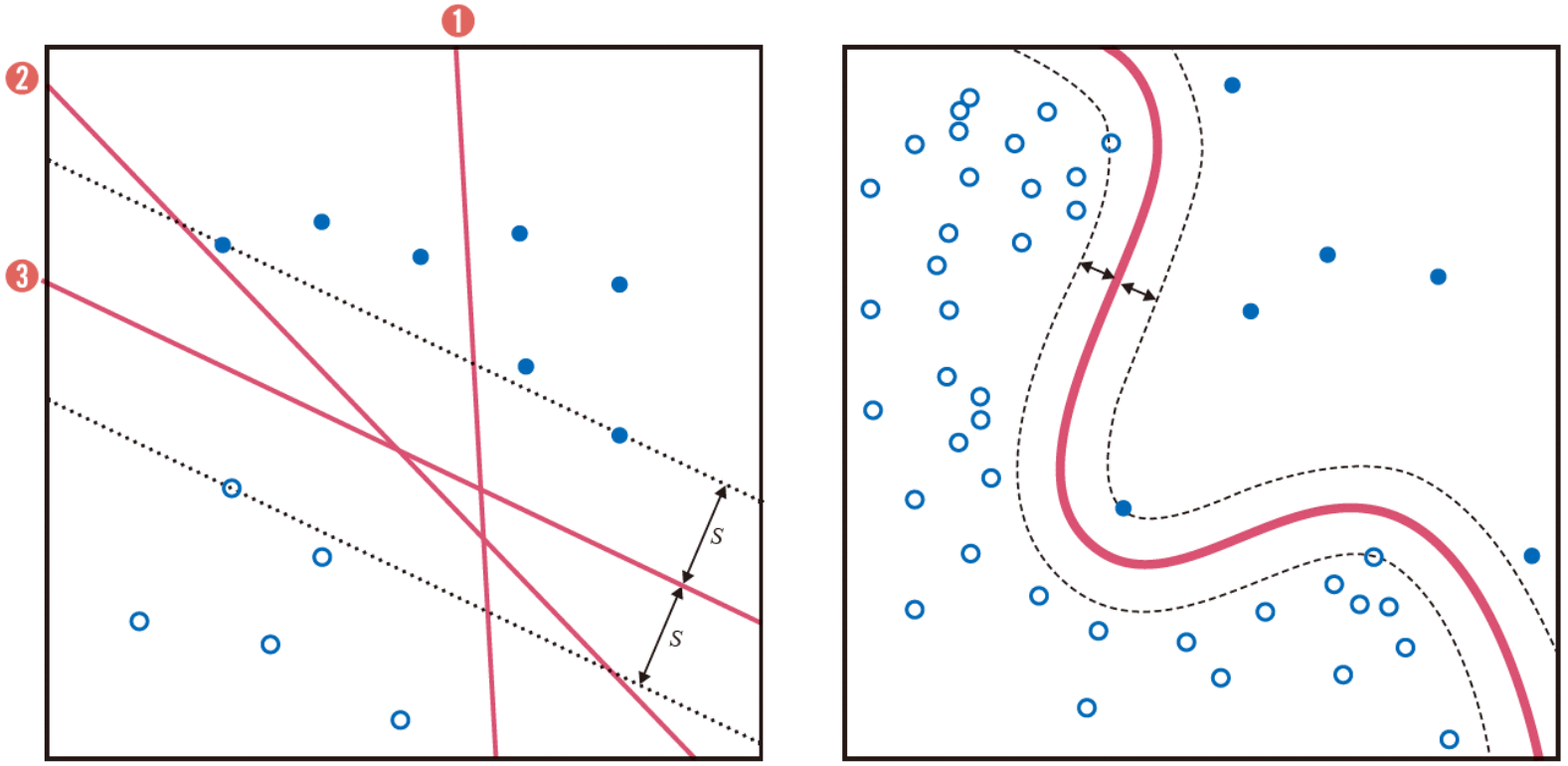


10_ SVM

3.7.2 SVM의 원리

SVM의 동기

- 100% 정확률인 두 분류기 ②와 ③은 같은가?



(a) 선형 SVM (b) 비선형 SVM

그림 3-17 여백을 최대화하여 일반화 능력을 극대화하는 SVM

3.7.2 SVM의 원리

C라는 하이퍼 매개변수

- 지금까지 모든 샘플을 옳게 분류하는 경우를 다룸. 실제로는 오류를 허용하는 수밖에 없음
- C를 크게 하면, 잘못 분류한 훈련 집합의 샘플을 적은데 여백이 작아짐(훈련 집합에 대한 정확률은 높지만 일반화 능력 떨어짐)
- C를 작게 하면, 여백은 큰데 잘못 분류한 샘플이 많아짐(훈련 집합에 대한 정확률은 낮지만 일반화 능력 높아짐)

[프로그램 3-4]의 07행

- 커널 함수로 기본값 rbf를 사용. gamma는 rbf 관련한 매개변수
- C=10 사용

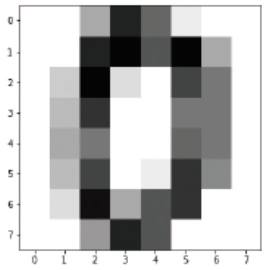
```
07    s=svm.SVC(gamma=0.1,C=10)
```

3.5.1 화소 값을 특징으로 사용

화소 각각을 특징으로 간주

- sklearn의 필기 숫자는 8*8 맵으로 표현되므로 64차원 특징 벡터
- 2차원 구조를 1차원 구조로 변환
- 예) [프로그램 3-3(a)]의 샘플

```
x=(0,0,5,13,9,1,0,0,0,0,13,15,10,15,5,0,0,3,15,2,0,11,8,0,0,4,12,0,0,8,8,
0,0,5,8,0,0,9,8,0,0,4,11,0,1,12,7,0,0,2,14,5,10,12,0,0,0,0,6,13,10,0,0,0)
```



```
[ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15. 10. 15.  5.  0.  0.  3.
 15.  2.  0. 11.  8.  0.  0.  4. 12.  0.  0.  8.  8.  0.  0.  5.  8.  0.
  0.  9.  8.  0.  0.  4. 11.  0.  1. 12.  7.  0.  0.  2. 14.  5. 10. 12.
  0.  0.  0.  0.  6. 13. 10.  0.  0.  0.]

이 숫자는 0입니다.
```

영상 데이터 사례: 필기 숫자

[프로그램 3-3.py)]

- matplotlib 라이브러리를 이용한 샘플 디스플레이와 샘플 내용(화소값) 출력

프로그램 3-3(a) 필기 숫자 데이터

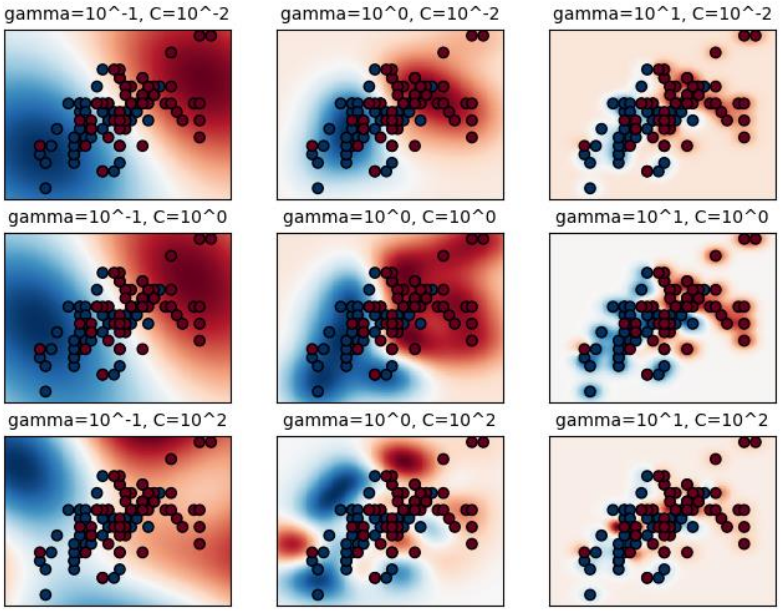
```
01  from sklearn import datasets
02  import matplotlib.pyplot as plt
03
04  digit=datasets.load_digits()
05
06  plt.figure(figsize=(5,5))
07  plt.imshow(digit.images[0],cmap=plt.cm.gray_r,interpolation='nearest')
                                                    # 0번 샘플을 그림
08  plt.show()
09  print(digit.data[0])                            # 0번 샘플의 화솟값을 출력
10  print("이 숫자는 ",digit.target[0],"입니다.")
```

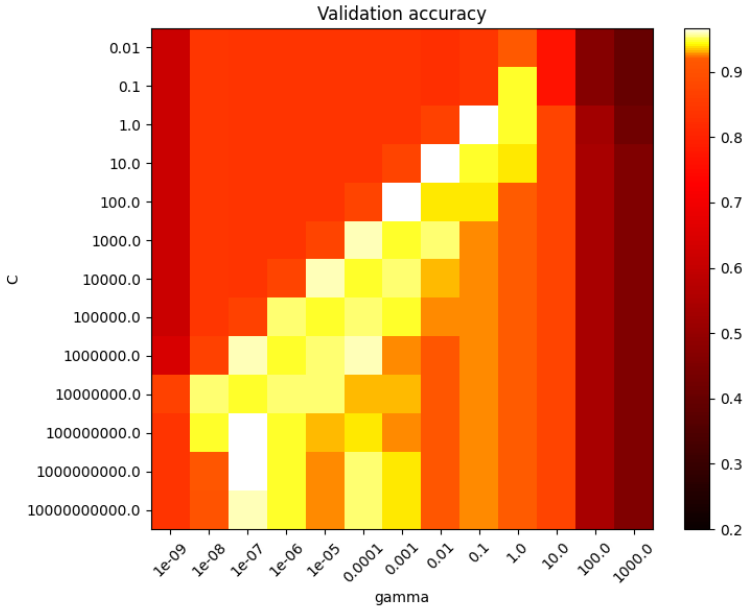
❖ SVM

(https://scikit-learn.org/stable/modules/svm.html)

- Kernel functions
  - linear: $\langle x, x' \rangle$.
  - polynomial: $(\gamma \langle x, x' \rangle + r)^d$, where d is specified by parameter `degree`, r by `coef0`.
  - rbf: $\exp(-\gamma \|x - x'\|^2)$, where γ is specified by parameter `gamma`, must be greater than 0.
  - sigmoid $\tanh(\gamma \langle x, x' \rangle + r)$, where r is specified by `coef0`.

This example illustrates the effect of the parameters **gamma** and **C** of the Radial Basis Function (RBF) kernel SVM.





https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html#sphx-glr-auto-examples-svm-plot-rbf-parameters-py

sklearn.svm.SVC

Support Vector Classification

(https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC)

class **sklearn.svm.SVC**(*C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*)

C : *float, default=1.0*

Kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'*

gamma*{'scale', 'auto'} or float, default='scale'*
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
if gamma='scale' (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
if 'auto', uses 1 / n_features
if float, must be non-negative.

sklearn.svm.SVC

Support Vector Classification

(https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC)

fit(*X*) :Fit the SVM model according to the given training data.

predict(*X*): Perform classification on samples in X.

predict_log_proba(*X*): Compute log probabilities of possible outcomes for samples in X.

score(*X*,*y*): Return the mean accuracy on the given test data(X) and labels(y: true labels of X).

필기 숫자 인식

- 필기 숫자 데이터셋을 가지고 프로그래밍 연습
  - 특징 추출을 위한 코드 작성
  - sklearn이 제공하는 fit 함수로 모델링(학습)
  - predict 함수로 예측

화소 값을 특징으로 사용

[프로그램 3-4.py]

- 07~08행: SVC로 학습 수행(특징 벡터 digit.data, 레이블 digit.target 사용)
- 11~14행: 맨 앞의 세 개 샘플을 테스트 집합으로 간주하고 예측을 해봄
- 17~20행: 훈련 집합을 테스트 집합으로 간주하고 정확률을 측정

프로그램 3-4 필기 숫자 인식 – 각 화소를 특징으로 간주하여 64차원 특징 벡터 사용

```
01  from sklearn import datasets
02  from sklearn import svm
03
04  digit=datasets.load_digits()
05
06  # svm의 분류기 모델 SC를 학습
07  s=svm.SVC(gamma=0.1,C=10)
08  s.fit(digit.data,digit.target) # digit 데이터로 모델링
09
10  # 훈련 집합의 앞에 있는 샘플 3개를 새로운 샘플로 간주하고 인식해봄
11  new_d=[digit.data[0],digit.data[1],digit.data[2]]
12  res=s.predict(new_d)
13  print("예측값은", res)
14  print("참값은", digit.target[0],digit.target[1],digit.target[2])
15
16  # 훈련 집합을 테스트 집합으로 간주하여 인식해보고 정확률을 측정
17  res=s.predict(digit.data)
18  correct=[i for i in range(len(res)) if res[i]==digit.target[i]]
19  accuracy=len(correct)/len(res)
20  print("화소 특징을 사용했을 때 정확률=",accuracy*100, "%")
```

```
예측값은 [0 1 2]
참값은 0 1 2

화소 특징을 사용했을 때 정확률=100.0%
```

3.6 성능 측정

객관적인 성능 측정의 중요성

- 모델 선택할 때 중요
- 현장 설치 여부 결정할 때 중요

일반화generalization 능력

- 학습에 사용하지 않았던 새로운 데이터에 대한 성능
- 가장 확실한 방법은 실제 현장에 설치하고 성능 측정 → 비용 때문에 실제 적용 어려움
- 주어진 데이터를 분할하여 사용하는 지혜 필요

3.6.1 혼동 행렬과 성능 측정 기준

혼동 행렬confusion matrix

- 부류 별로 옳은 분류와 틀린 분류의 개수를 기록한 행렬
  - n_{ij}는 모델이 i 라고 예측했는데 실제 부류는 j 인 샘플의 개수

		참값(그라운드 트루스)					
		부류 1	부류 2	…	부류 j	…	부류 c
예측한 부류	부류 1	n_{11}	n_{12}		n_{1j}		n_{1c}
	부류 2	n_{21}	n_{22}		n_{2j}		n_{2c}
	…						
	부류 i	n_{i1}	n_{i2}		n_{ij}		n_{ic}
	…						
	부류 c	n_{c1}	n_{c2}		n_{cj}		n_{cc}

(a) 부류가 c개인 경우

		그라운드 트루스	
		긍정	부정
예측값	긍정	TP	FP
	부정	FN	TN

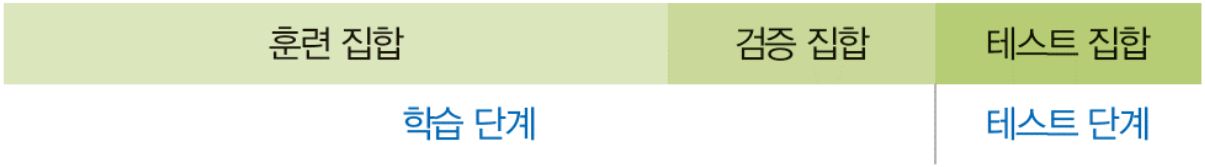
(b) 부류가 2개인 경우

그림 3-10 혼동 행렬

- 이진 분류에서 긍정positive과 부정negative
  - 검출하고자 하는 것이 긍정(환자가 긍정이고 정상인이 부정, 불량품이 긍정이고 정상이 부정)

3.6.2 훈련/검증/테스트 집합으로 쪼개기

- **주어진 데이터를 적절한 비율로 훈련, 검증, 테스트 집합으로 나누어 씀**
  - **모델 선택 포함: 훈련/검증/테스트 집합으로 나눔**
  - **모델 선택 제외: 훈련/테스트 집합으로 나눔**



(a) 모델 선택 포함

(b) 모델 선택 제외

그림 3-11 훈련/검증/테스트 집합으로 쪼개기

3.6.2 훈련/검증/테스트 집합으로 쪼개기

- **[프로그램 3-5.py]는 모델 선택 제외**
  - 08행: train_test_split **함수로 훈련** 60%, **테스트** 40%**로 랜덤 분할**
  - 12행: **훈련 집합** x_train, y_train**을** fit **함수에 주어 학습 수행**
  - 14행: **테스트 집합의 특징 벡터** x_test**를** predict **함수에 주어 예측 수행**
  - 17~20행: **테스트 집합의 레이블** y_test**를 가지고 혼동 행렬 계산**

프로그램 3-5 필기 숫자 인식 – 훈련 집합으로 학습하고 테스트 집합으로 성능 측정

```
01  from sklearn import datasets
02  from sklearn import svm
03  from sklearn.model_selection import train_test_split
04  import numpy as np
05
06  # 데이터셋을 읽고 훈련 집합과 테스트 집합으로 분할
07  digit=datasets.load_digits()
08  x_train,x_test,y_train,y_test=train_test_split(digit.data,digit.target,train_size=0.6)
09
```

3.6.2 훈련/검증/테스트 집합으로 쪼개기

```
10  # svm의 분류 모델 SVC를 학습
11  s=svm.SVC(gamma=0.001)
12  s.fit(x_train,y_train)
13
14  res=s.predict(x_test)
15
16  # 혼동 행렬 구함
17  conf=np.zeros((10,10))
18  for i in range(len(res)):
19      conf[res[i]][y_test[i]]+=1
20  print(conf)
21
22  # 정확률 측정하고 출력
23  no_correct=0
24  for i in range(10):
25      no_correct+=conf[i][i]
26  accuracy=no_correct/len(res)
27  print("테스트 집합에 대한 정확률은", accuracy*100, "%입니다.")
```

감마값 변화에 따른 결과 확인

예) 부류 3에 속하는 75개 샘플 중 73개를 3, 1개를 2, 1개를 7로 인식

```
[[76.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 78.  0.  0.  0.  0.  0.  0.  3.  0.]
 [ 0.  0. 66.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 73.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 63.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 70.  0.  0.  0.  2.]
 [ 0.  0.  0.  0.  0.  0. 77.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0. 77.  0.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. 74.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0. 56.]]

테스트 집합에 대한 정확률은 98.74826147426981%입니다.
```

3.6.2 훈련/검증/테스트 집합으로 쪼개기

NOTE 난수를 사용하기 때문에 실행할 때마다 다른 결과가 나오는 프로그램

[프로그램 3-5]는 실행할 때마다 출력이 다르게 나온다. 08행의 train_test_split 함수가 난수를 사용해 데이터를 분할하기 때문이다. 앞으로 등장하는 프로그램에서도 난수를 사용하는 경우가 있는데 마찬가지로 실행할 때마다 다른 결과를 얻게 된다. 동일한 실행 결과를 얻으려면 08행 이전에 np.random.seed(0)을 추가하면 된다. 매개변수 0은 다른 값을 사용해도 된다. 어떤 값이든 고정시키면 매번 같은 난수 열이 생성된다.

3.6.3 교차 검증

- **훈련/테스트 집합 나누기의 한계**
  - 우연히 높은 정확률 또는 우연히 낮은 정확률 발생 가능성
- **k-겹 교차 검증**

k-fold cross validation

- 훈련 집합을 k개의 부분집합으로 나누어 사용. 한 개를 남겨두고 k-1개로 학습한 다음 남겨둔 것으로 성능 측정. k개의 성능을 평균하여 신뢰도 높임

(a) 모델 선택 포함

(b) 모델 선택 제외

그림 3-12 k-겹 교차 검증(k=5인 경우)

3.6.3 교차 검증

- **[프로그램 3-6.py]은 digit 데이터에 교차 검증 적용(모델 선택 제외)**
  - Cross_val_score 함수가 교차 검증 수행해줌(cv=5는 5-겹 교차 검증하라는 뜻)

프로그램 3-6 필기 숫자 인식 – 교차 검증으로 성능 측정

```
01  from sklearn import datasets
02  from sklearn import svm
03  from sklearn.model_selection import cross_val_score
04  import numpy as np
05
06  digit=datasets.load_digits()
07  s=svm.SVC(gamma=0.001)
08  accuracies=cross_val_score(s,digit.data,digit.target,cv=5) # 5-겹 교차 검증
09
10  print(accuracies)
11  print("정확률(평균)=%0.3f, 표준편차=%0.3f"%(accuracies.mean()*100,accuracies.std()))
```

감마값 변화에 따른 결과 확인

K 값(CV): 3, 4, 6도 실행하여 결과변화 확인

```
[0.97527473 0.95027624 0.98328691 0.99159664 0.95774648]
정확률(평균)=97.164, 표준편차=0.015
```

  - 실행 결과 정확률이 들쭉날쭉. 한번만 시도하는 [프로그램 3-5]의 위험성을 잘 보여줌
  - k를 크게 하면 신뢰도 높아지지만 실행 시간이 더 걸림