빅데이터분석 실습

모델 평가와 성능 향상

데이터 사이언스 전공
담당교수: 곽철완

강의 내용

- 교차 검증
- 그리드 서치

▪ 개요

- 지도 학습 모델 만들기
  - 데이터 세트를 학습용 데이터 세트와 평가용 데이터 세트로 구분
    train_test_split( ) 함수
  - 모델을 만들기 위해 학습용 데이터 세트에 fit 메서드 적용
  - 모델 평가를 위해 평가용 데이터 세트에 score 메서드 적용
    - 분류에서 score 메서드는 정확히 분류된 샘플의 비율을 계산하는 역할 담당

- 사례
  - 데이터를 학습용과 평가용으로 나누어 새로운 데이터에 모델이 얼마나 일반화 되는지 평가
  - 인위적인 데이터 세트(make_blobs 함수 이용)를 만들어
  - 로지스틱 회귀분석 실행 → 평가용 데이터 세트를 이용해 모델 평가

```
from sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X, y = make_blobs(random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
logreg = LogisticRegression().fit(X_train, y_train)

print("평가용 데이터 세트 점수: {:.2f}".format(logreg.score(X_test, y_test)))
```

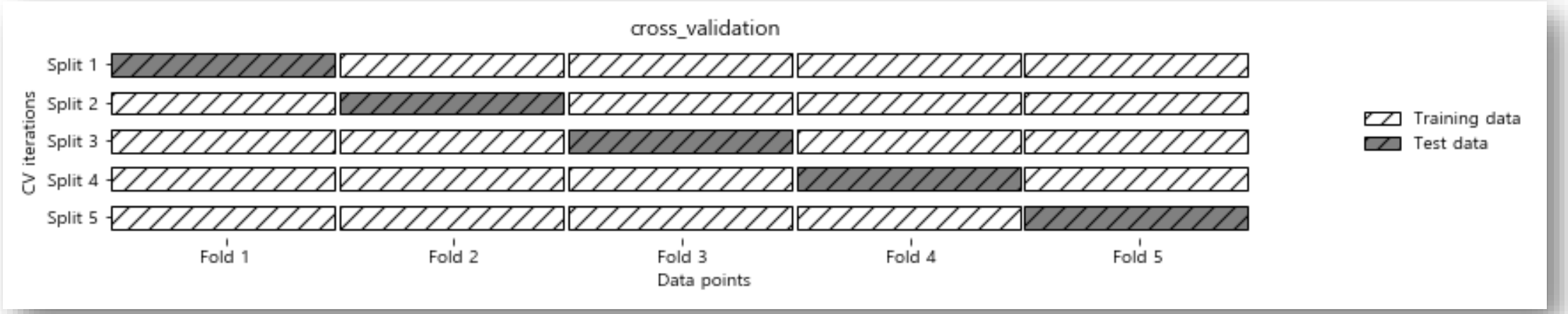
```
평가용 데이터 세트 점수: 0.88
```

1. 교차 검증

▪ 개요

- 교차검증은 학습용 및 평가용으로 나누는 것보다 더 안정적이고 뛰어난 통계적 평가 방법
- 데이터를 여러 번 반복해서 나누고 여러 모델을 학습한다
- k-겹 교차 검증(k-fold cross-validation)(k=5 혹은 10)
  - 데이터를 폴드(fold)라 하는 비슷한 크기의 부분 집합 5개로 나눔
  - 다음에 일련의 모델 작성. 첫 번째 모델은 폴드 1을 평가용 세트로 사용하고 나머지를 학습용 모델로 사용하여 학습한다
  - 두번째 모델은 폴드 2를 평가용 모델로 사용하고 나머지 폴드를 학습용 모델로 사용하여 계속 반복
  - 5번의 분할마다 정확도를 측정하여 최종적인 값을 얻는다

```
mglearn.plots.plot_cross_validation()
```



- 위의 그림과 같이 5회 반복하여 5개의 정확도 점수를 얻는다

scikit-learn의 교차 검증

- model_selection 모듈의 cross_val_score 함수에 구현되어 있다
- cross_val_score 함수의 매개변수는 평가하려는 모델, 훈련 데이터, 타깃 레이블이다
- iris 데이터 세트를 이용한 로지스틱 회귀 모델을 평가해보자

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

iris = load_iris()
logreg = LogisticRegression()

scores = cross_val_score(logreg, iris.data, iris.target)
print("교차 검증 점수:", scores)
```

```
교차 검증 점수: [0.96078431 0.92156863 0.95833333]
```

- 폴드 수 변경
  - cv=5 사용

```
scores = cross_val_score(logreg, iris.data, iris.target, cv=5)
print("교차 검증 점수:", scores)
```

```
교차 검증 점수: [1.         0.96666667 0.93333333 0.9        1.        ]
```

  - 보통 교차 검증의 정확도를 간단하게 나타내기 위해 평균을 사용

```
print("교차 검증 평균 점수: {:.2f}".format(scores.mean()))
```

```
교차 검증 평균 점수: 0.96
```

  - 이 모델의 정확도가 약 96%으로 기대
    - 90% ~ 100%

◦ 교차 검증에 cross_validate 함수를 사용할 수도 있다

```
from sklearn.model_selection import cross_validate
res = cross_validate(logreg, iris.data, iris.target, cv=5, return_train_score=True)
display(res)
```

```
{'fit_time': array([0.00299883, 0.00100088, 0.00099921, 0.00099897, 0.00099945]),
 'score_time': array([0.00100946, 0.00099874, 0.        , 0.        , 0.        ]),
 'test_score': array([1.        , 0.96666667, 0.93333333, 0.9       , 1.        ]),
 'train_score': array([0.95      , 0.96666667, 0.96666667, 0.975     , 0.95833333])}
```

◦ 학습과 평가에 걸린 시간을 보여준다

◦ pandas 를 이용하여 결과값 출력하고 평균 계산

```
res_df = pd.DataFrame(res)
display(res_df)
print("평균 시간과 점수:\n", res_df.mean())
```

	fit_time	score_time	test_score	train_score
0	0.002999	0.001009	1.000000	0.950000
1	0.001001	0.000999	0.966667	0.966667
2	0.000999	0.000000	0.933333	0.966667
3	0.000999	0.000000	0.900000	0.975000
4	0.000999	0.000000	1.000000	0.958333

```
평균 시간과 점수:
 fit_time       0.001399
score_time      0.000402
test_score      0.960000
train_score     0.963333
dtype: float64
```

▪ 교차 검증의 장점과 단점

- 머신러닝 모델을 평가하기 위해 학습용 및 평가용 데이터세트로 구분하면서 사용하는 train_test_split( )는 데이터를 무작위로 나눈다
  - 하지만 경우에 따라 평가 결과가 높게 혹은 낮게 나올 수 있다
- 교차 검증은 평가용 데이터 세트에 각각 샘플이 정확하게 한번씩 들어간다
- 분할을 한번 했을 때 보다 데이터를 더 효과적으로 사용할 수 있다
  - train_test_split는 75 : 25 로 나누지만
  - 5-겹 교차 검증을 사용하면 반복해서 4/5의 데이터(80%)를 모델 학습에 사용한다
- 단점으로 연산 비용이 증가하고, 모델을 k 개 만들어 k 배 느리다

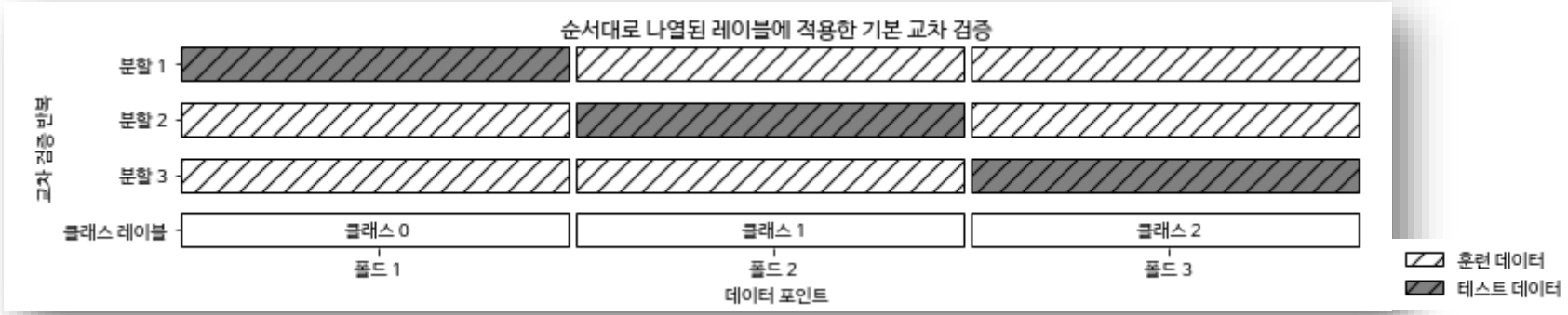
■ 계층별 k-겹 교차 검증과 기타

◦ iris 데이터 세트 예

◦ 나열 순서대로 k 개의 폴드로 나누는 것이 항상 좋지 않다

```
from sklearn.datasets import load_iris
iris = load_iris()
print("Iris 레이블:\n", iris.target)
```

```
Iris 레이블:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

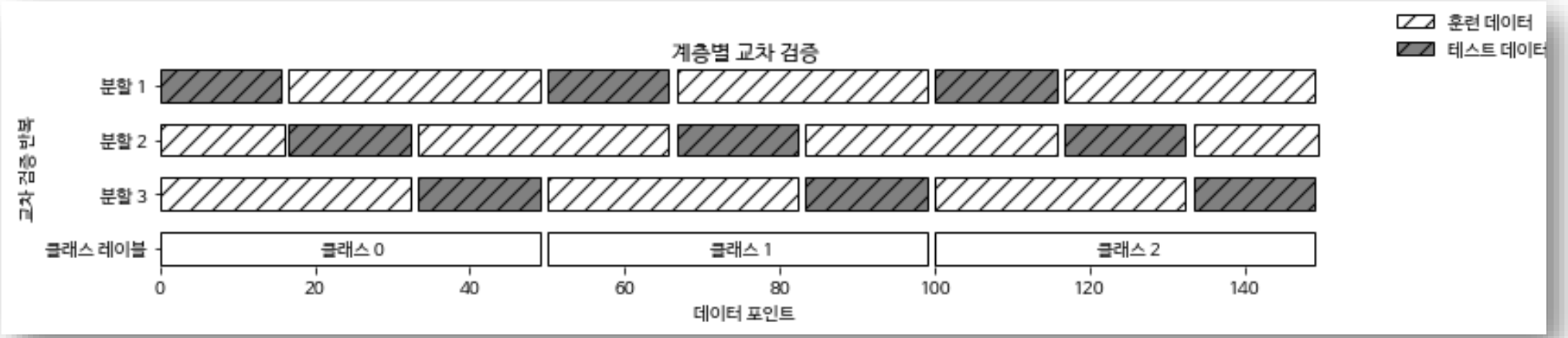


◦ iris 데이터 세트는 3개의 클래스 0, 1, 2로 구성되어 있는데, k 폴드(3개)로 나눈다면 폴드 1은 클래스 0 으로 학습용은 0, 평가용은 1, 2만 가진다

◦ 폴드 2는 학습용 클래스만 1, 나머지는 0, 2이다 그러면 교차 검증의 정확도는 0이 된다

- 계층별 k-겹 교차 검증은 폴드 안의 클래스 비율이 전체 데이터세트의 클래스 비율과 같도록 데이터를 나눈다(예를 들면 클래스 A 60%, 클래스 B 40%의 비율)

```
mglearn.plots.plot_stratified_cross_validation()
```



- 교차 검증 상세 옵션
  - scikit-learn에서는 cv 매개변수에 교차 검증 분할기 cross-validation splitter를 전달하여 더 세밀한 제어가 가능하다
  - 하지만, 다른 사람이 조사한 결과를 재현하기 위해 k-겹 교차 검증을 사용할 때는 다른 전략이 필요하다
  - 예, iris 데이터 세트
  - 분석을 위해 Kfold 분할기를 import 한다

```
from sklearn.model_selection import KFold
kfold = KFold(n_splits=5)
```

```
print("교차 검증 점수:\n",
    cross_val_score(logreg, iris.data, iris.target, cv=kfold))
```

```
교차 검증 점수:
 [1.         0.93333333 0.43333333 0.96666667 0.43333333]
```

  - kfold 객체를 cross_val_score의 cv 매개변수로 전달한다

◦ iris 데이터 세트에 3겹 교차 검증을 적용한다

```
kfold = KFold(n_splits=3)
print("교차 검증 점수:\n",
      cross_val_score(logreg, iris.data, iris.target, cv=kfold))
```

```
교차 검증 점수:
 [0.  0.  0.]
```

◦ 결과는 0, 0, 0 이다
◦ 각 폴드는 iris 데이터 세트 중 하나에 대응하므로 아무것도 학습할 수 없다 → 계층별 대신 섞어주는 방법을 적용

◦ kfold의 shuffle 매개변수에 True를 지정하면 뒤죽박죽 섞인다

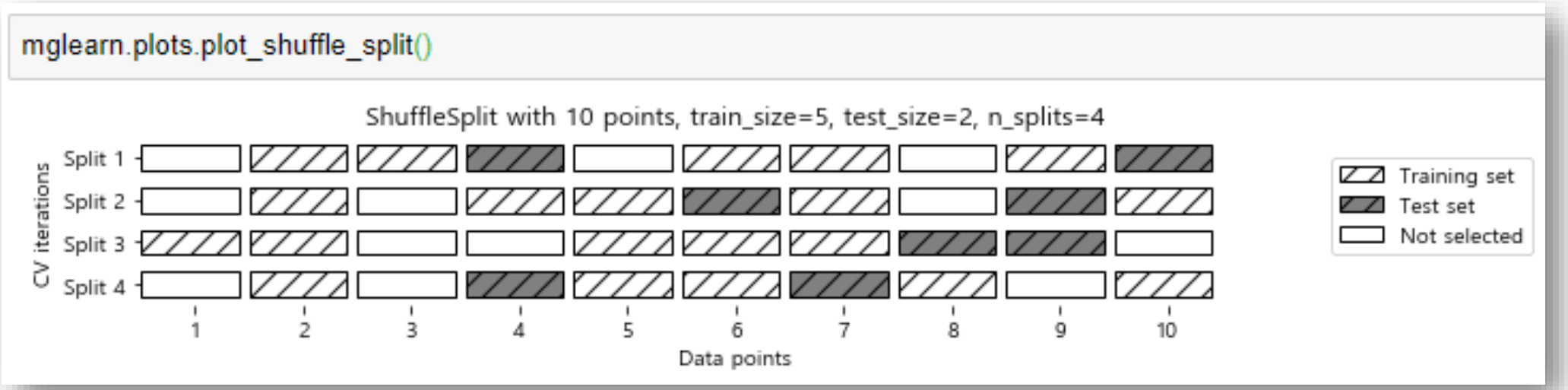
```
kfold = KFold(n_splits=3, shuffle=True, random_state=0)
print("교차 검증 점수:\n",
    cross_val_score(logreg, iris.data, iris.target, cv=kfold))
```

```
교차 검증 점수:
 [0.9  0.96 0.96]
```

◦ 검증 점수가 바뀌었다

- 임의 분할 교차 검증
  - train_size 만큼의 포인트로 학습용 데이터 세트를 만들고 test_size 만큼의 포인트로 평가용 데이터 세트를 만들도록 분할한다
  - 그리고 n_splits 횟수만큼 반복한다

```
mglearn.plots.plot_shuffle_split()
```



◦ 사례, 데이터 세트의 50%를 학습용 데이터 세트로, 50%를 평가용 데이터 세트로 10회 반복 분할한다

```
from sklearn.model_selection import ShuffleSplit
shuffle_split = ShuffleSplit(test_size=.5, train_size=.5, n_splits=10)
scores = cross_val_score(logreg, iris.data, iris.target, cv=shuffle_split)
print("교차 검증 점수:\n", scores)
```

```
교차 검증 점수:
 [0.92        0.96        0.88        0.92        0.90666667 0.90666667
 0.93333333 0.88        0.93333333 0.96       ]
```

◦ 임의 분할 교차 검증은 반복 횟수를 학습용 세트나 평가용 세트의 크기와 독립적으로 조절해야 할 때 유리하다

- 그룹별 교차 검증
  - (예) 얼굴 구분을 위한 시스템 구축
    - 모델을 구축을 위해 사진 100장 수집(동일 인물의 여러 사진 포함)
    - 계층별 교차 검증은 동일 인물이 학습용과 평가용에 모두 나타날 수 있다
    - 새 사진을 더 잘 구별할 수 있는지 평가하려면, 학습용과 평가용에 다른 사진이 포함되어야 한다(과적합 감소)

- 인물사진분류시스템
  - 100명의 사진 수집→한 사람을 찍은 여러 장이 사진이 각기 다른 표정을 담고 있다
  - 시스템 구축 목적 → 새로운 인물 사진을 구분할 수 있는 분류기를 만드는 것이다
  - 분류기 성능을 측정하기 위해 계층별 교차 검증을 사용할 수 있지만, 같은 사람의 사진이 학습용 세트와 평가용 세트에 모두 나타날 수 있다
    - 새 얼굴에 대한 일반화 성능을 더 정확하게 평가하기 위해서 학습용 세트와 평가용 세트에 서로 다른 사람의 사진이 포함되어야 한다
    - 그룹별 교차 검증이 이 문제를 해결한다

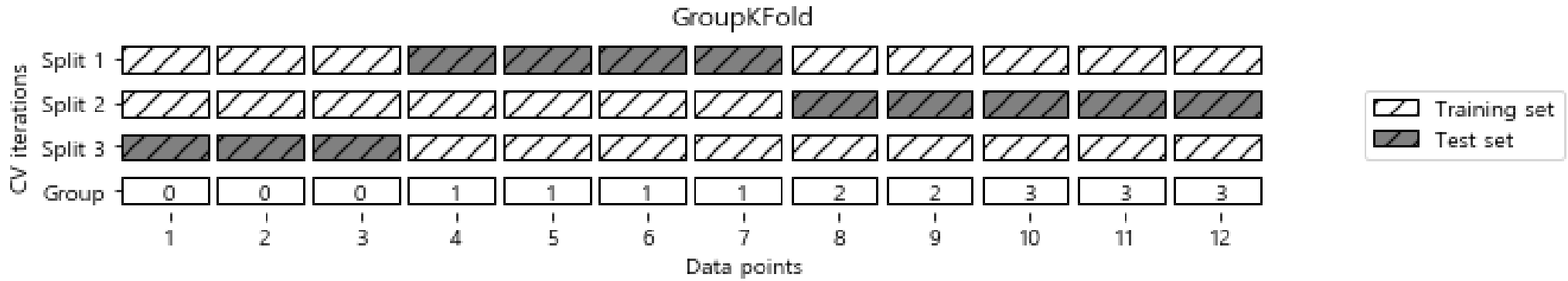
◦ 그룹별 교차 검증에는 GroupKFold 함수를 사용한다

```
from sklearn.model_selection import GroupKFold
# 인위적 데이터셋 생성
X, y = make_blobs(n_samples=12, random_state=0)
# 처음 세 개의 샘플은 같은 그룹에 속하고
# 다음은 네 개의 샘플이 같습니다.
groups = [0, 0, 0, 1, 1, 1, 1, 2, 2, 3, 3, 3]
scores = cross_val_score(logreg, X, y, groups, cv=GroupKFold(n_splits=3))
print("교차 검증 점수:\n", scores)
```

```
교차 검증 점수:
 [0.75       0.8        0.66666667]
```

- GroupKFold는 그룹별로 학습용 혹은 평가용 데이터세트에 나뉘어져 있다

```
mglearn.plots.plot_group_kfold()
```



2. 그리드 서치

- 기본 개념
  - 매개 변수 튜닝을 통한 모델의 일반화 성능 개선
  - 그리드 서치는 매개 변수들을 대상으로 가능한 모든 조합을 시도하여 적합한 매개 변수를 제시
- 사례
  - RBF(Radial Basis Function) 커널 SVM의 경우를 보면, gamma와 규제 매개 변수 C 가 있는데 이 두 매개 변수의 조합을 그리드 서치를 통해 얻을 수 있다

▪ 간단한 그리드 서치

```
# 간단한 그리드 서치 구현
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
                                                    random_state=0)
print("훈련 세트의 크기: {}   테스트 세트의 크기: {}".format(
    X_train.shape[0], X_test.shape[0]))

best_score = 0
```

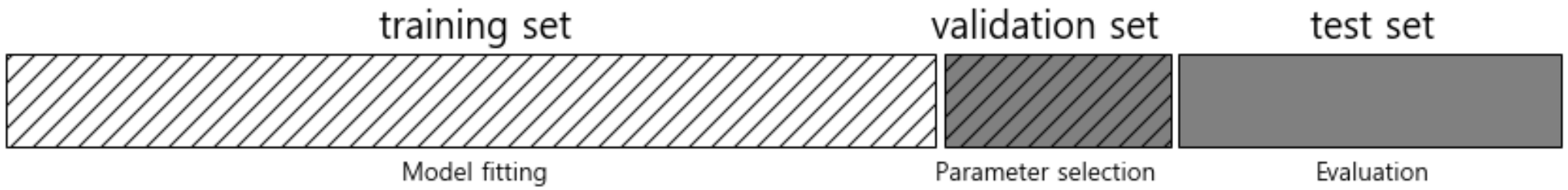
```
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # 매개변수의 각 조합에 대해 SVC를 훈련시킵니다
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # 테스트 세트로 SVC를 평가합니다
        score = svm.score(X_test, y_test)
        # 점수가 더 높으면 매개변수와 함께 기록합니다
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

print("최고 점수: {:.2f}".format(best_score))
print("최적 파라미터:", best_parameters)
```

```
훈련 세트의 크기: 112   테스트 세트의 크기: 38
최고 점수: 0.97
최적 파라미터: {'C': 100, 'gamma': 0.001}
```

▪ 매개변수 과대적합과 검증 세트

- 앞의 모델 정확도가 97%로 보고된다
- 하지만, 새로운 데이터에 적합하지 않을 수 있다 → 매개변수 조정을 위해 평가용 데이터 세트를 이미 사용했기 때문에, 평가를 위해 모델을 만들 때 사용하지 않은 데이터가 필요
- 학습용 세트 → 모델 개발
- 검증용 세트 → 모델의 매개변수 선택
- 평가용 세트 → 매개변수 성능 평가

mglearn.plots.plot_threefold_split()



- 검증 세트를 이용해 최적의 매개변수를 선택
- 그 매개변수에서 학습용 세트와 검증 세트 데이터 모두를 이용해 모델을 다시 만든다 ← 데이터를 많이 수집하기 위해

```
from sklearn.svm import SVC
# 데이터를 훈련+검증 세트 그리고 테스트 세트로 분할
X_trainval, X_test, y_trainval, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
# 훈련+검증 세트를 훈련 세트와 검증 세트로 분할
X_train, X_valid, y_train, y_valid = train_test_split(
    X_trainval, y_trainval, random_state=1)
print("훈련 세트의 크기: {}  검증 세트의 크기: {}  테스트 세트의 크기:"
      " {}\n".format(X_train.shape[0], X_valid.shape[0], X_test.shape[0]))
```

```
best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # 매개변수의 각 조합에 대해 SVC를 훈련시킵니다
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # 검증 세트로 SVC를 평가합니다
        score = svm.score(X_valid, y_valid)
        # 점수가 더 높으면 매개변수와 함께 기록합니다
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

# 훈련 세트와 검증 세트를 합쳐 모델을 다시 만든 후
# 테스트 세트를 사용해 평가합니다
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
test_score = svm.score(X_test, y_test)
print("검증 세트에서 최고 점수: {:.2f}".format(best_score))
print("최적 파라미터: ", best_parameters)
print("최적 파라미터에서 테스트 세트 점수: {:.2f}".format(test_score))
```

```
훈련 세트의 크기: 84   검증 세트의 크기: 28   테스트 세트의 크기: 38

검증 세트에서 최고 점수: 0.96
최적 파라미터:  {'C': 10, 'gamma': 0.001}
최적 파라미터에서 테스트 세트 점수: 0.92
```

- 최고 점수: 96%
  - 학습용 데이터 세트가 줄었기 때문에 감소했다
- 평가용 데이터 세트 점수: 92%
  - 새로운 데이터 92%만 정확하게 분류한다
- 정확도가 감소한 이유는 동일한 데이터를 사용했기 때문

```
best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # 매개변수의 각 조합에 대해 SVC를 훈련시킵니다
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # 검증 세트로 SVC를 평가합니다
        score = svm.score(X_valid, y_valid)
        # 점수가 더 높으면 매개변수와 함께 기록합니다
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

# 훈련 세트와 검증 세트를 합쳐 모델을 다시 만든 후
# 테스트 세트를 사용해 평가합니다
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
test_score = svm.score(X_test, y_test)
print("검증 세트에서 최고 점수: {:.2f}".format(best_score))
print("최적 파라미터: ", best_parameters)
print("최적 파라미터에서 테스트 세트 점수: {:.2f}".format(test_score))
```

```
훈련 세트의 크기: 84   검증 세트의 크기: 28   테스트 세트의 크기: 38

검증 세트에서 최고 점수: 0.96
최적 파라미터:  {'C': 10, 'gamma': 0.001}
최적 파라미터에서 테스트 세트 점수: 0.92
```

▪ 교차 검증을 사용한 그리드 서치

- 앞의 결과는 매개변수 C가 100에서 10으로 바뀌었다
- 일반화 성능을 잘 평가하기 위해 교차 검증을 사용할 수 있다

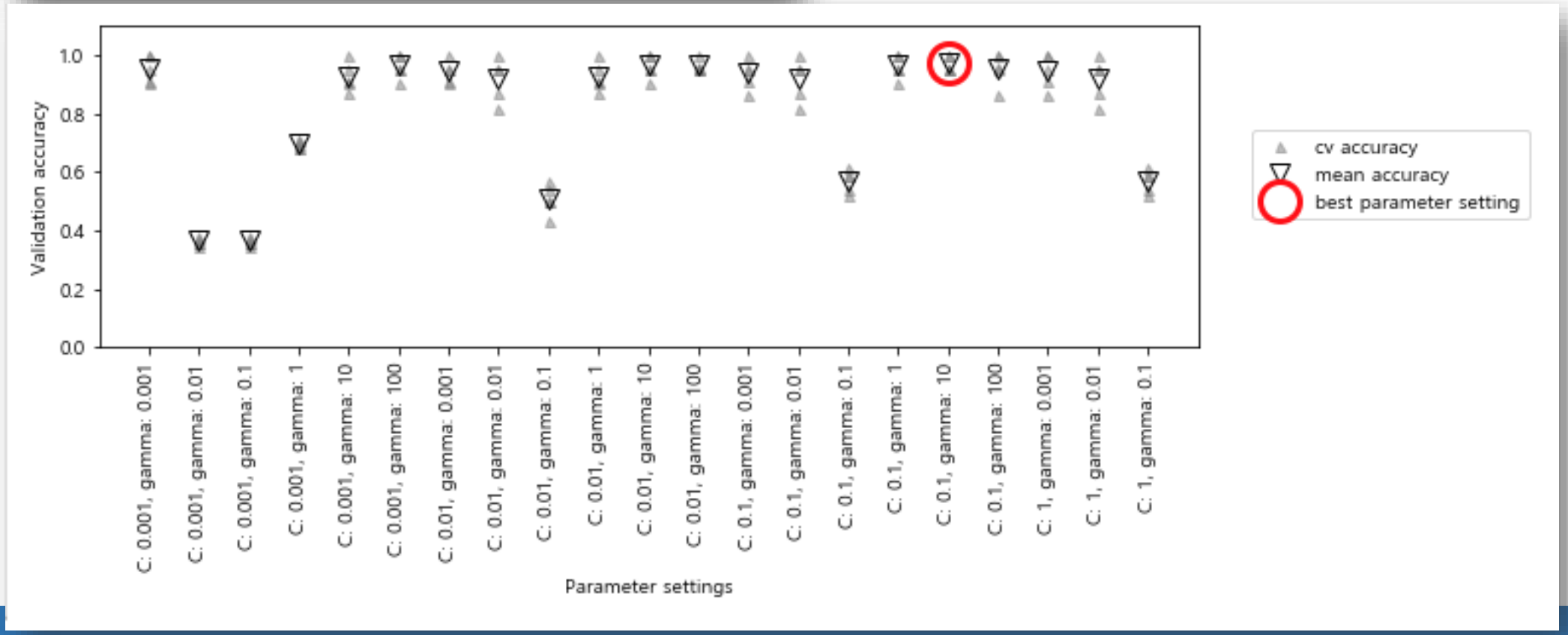
```
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # 매개변수의 각 조합에 대해 SVC를 훈련시킵니다
        svm = SVC(gamma=gamma, C=C)
        # 교차 검증을 적용합니다
        scores = cross_val_score(svm, X_trainval, y_trainval, cv=5)
        # 교차 검증 정확도의 평균을 계산합니다
        score = np.mean(scores)
        # 점수가 더 높으면 매개변수와 함께 기록합니다
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
```

```
# 훈련 세트와 검증 세트를 합쳐 모델을 다시 만듭니다
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
```

```
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

◦ 앞의 코드에서 최적의 매개 변수 선택 방법

```
mglearn.plots.plot_cross_val_selection()
```



- GridSearchCV
  - 먼저 딕셔너리 형태로 검색 대상 매개 변수를 지정해야 한다

```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
print("매개변수 그리드:\n", param_grid)
```

```
매개변수 그리드:
 {'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
```

- 다음으로, 모델(여기서는 SVC), 검색 대상 매개 변수 그리드(param_grid), 원하는 교차 검증(5-겹 계층별 교차 검증)으로 GridSearchCV 객체 생성

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
grid_search = GridSearchCV(SVC(), param_grid, cv=5, return_train_score=True)
```

◦ 과대적합을 피하기 위해 학습용 및 평가용 데이터 세트로 구분한다

```
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, random_state=0)
```

◦ grid_search 객체는 fit, predict, score 메서드를 제공한다
  ◦ fit 메서드는 param_grid에 설정된 매개 변수 조합에 대한 교차 검증을 수행한다

```
grid_search.fit(X_train, y_train)
```

- GridSearchCV 객체의 fit 메서드 역할
  - 최적의 매개 변수를 찾는다
  - 교차 검증 성능이 좋은 매개 변수로 전체 학습용 데이터 세트에 대해 새로운 모델을 자동으로 만든다
    - → fit 메서드는 29쪽의 코드의 결과와 동일하다
  - 선택한 매개 변수: best_params_
  - 최상의 교차 검증 정확도: best_score_

```
print("최적 매개변수:", grid_search.best_params_)
print("최고 교차 검증 점수: {:.2f}".format(grid_search.best_score_))
```

```
최적 매개변수: {'C': 10, 'gamma': 0.1}
최고 교차 검증 점수: 0.97
```

- 교차 검증 결과 분석
  - 교차 검증 결과를 시각화 하여 검색 대상 매개 변수가 모델의 일반화에 영향을 미치는지 파악한다
    - 적은 수의 그리드로 시작하는 것이 좋다

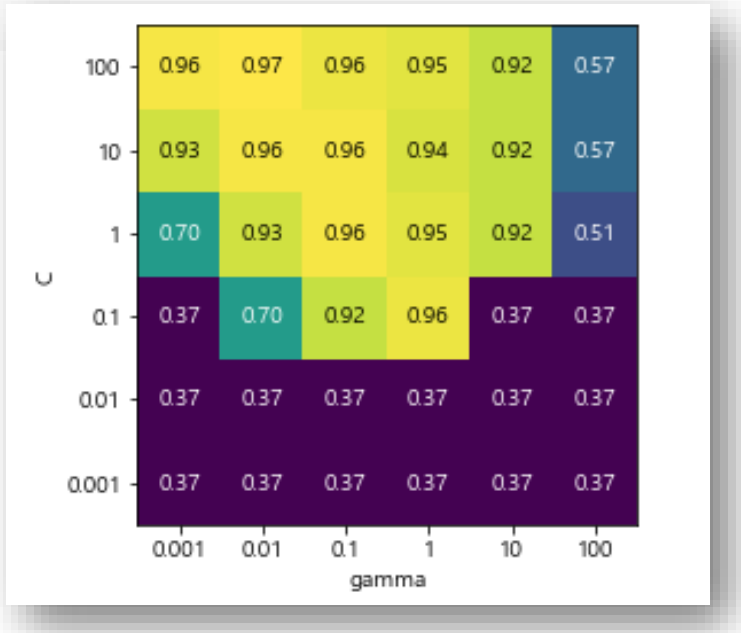
```
import pandas as pd
pd.set_option('display.max_columns', None)
# DataFrame으로 변환합니다
results = pd.DataFrame(grid_search.cv_results_)
# 처음 다섯 개 행을 출력합니다
display(np.transpose(results.head()))
```

	0	1	2	3	4
mean_fit_time	0.000400066	0.00059967	0.000599575	0.000599432	0.000399828
std_fit_time	0.000489979	0.000489629	0.000489551	0.000489434	0.000489687
mean_score_time	0.000599766	0.000399828	0.00019989	0.000399971	0.000599527
std_score_time	0.000489707	0.000489687	0.00039978	0.000489862	0.000489512
param_C	0.001	0.001	0.001	0.001	0.001
param_gamma	0.001	0.01	0.1	1	10
params	{'C': 0.001, 'gamma': 0.001}	{'C': 0.001, 'gamma': 0.01}	{'C': 0.001, 'gamma': 0.1}	{'C': 0.001, 'gamma': 1}	{'C': 0.001, 'gamma': 10}
split0_test_score	0.375	0.375	0.375	0.375	0.375
split1_test_score	0.347826	0.347826	0.347826	0.347826	0.347826
split2_test_score	0.363636	0.363636	0.363636	0.363636	0.363636
split3_test_score	0.363636	0.363636	0.363636	0.363636	0.363636
split4_test_score	0.380952	0.380952	0.380952	0.380952	0.380952
mean_test_score	0.366071	0.366071	0.366071	0.366071	0.366071
std_test_score	0.0113708	0.0113708	0.0113708	0.0113708	0.0113708
rank_test_score	22	22	22	22	22
split0_train_score	0.363636	0.363636	0.363636	0.363636	0.363636
split1_train_score	0.370787	0.370787	0.370787	0.370787	0.370787
split2_train_score	0.366667	0.366667	0.366667	0.366667	0.366667
split3_train_score	0.366667	0.366667	0.366667	0.366667	0.366667
split4_train_score	0.362637	0.362637	0.362637	0.362637	0.362637
mean_train_score	0.366079	0.366079	0.366079	0.366079	0.366079
std_train_score	0.00285176	0.00285176	0.00285176	0.00285176	0.00285176

◦ 검색 대상 매개 변수 그리드가 2차원이므로 히트맵으로 시각화한다

```
scores = np.array(results.mean_test_score).reshape(6, 6)

# 교차 검증 평균 점수 히트맵 그래프
mglearn.tools.heatmap(scores, xlabel='gamma', xticklabels=param_grid['gamma'],
                      ylabel='C', yticklabels=param_grid['C'], cmap="viridis")
```

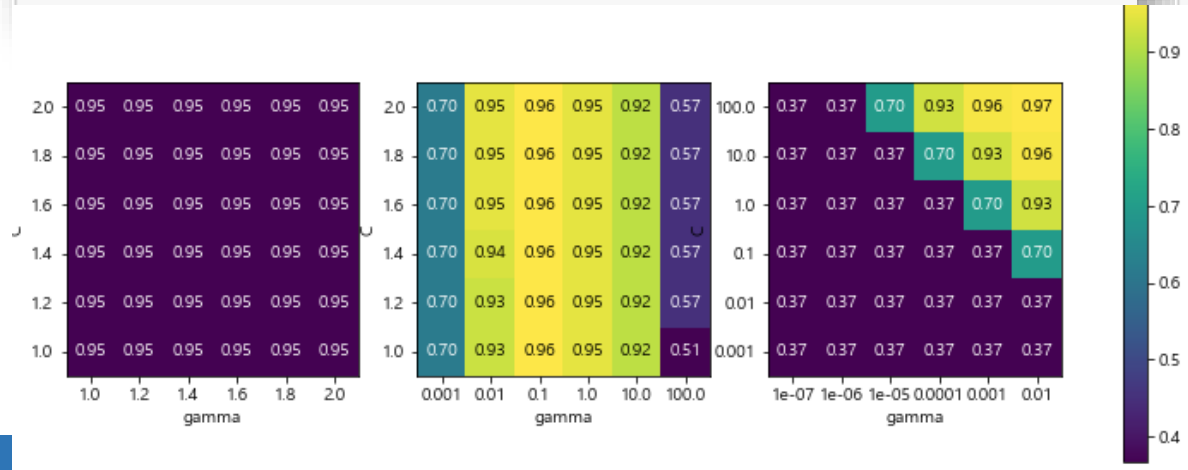
◦ 정확도가 높으면 밝은 색, 낮으면 어두운 색



◦ 검색 범위를 잘못 선택한 결과

```
fig, axes = plt.subplots(1, 3, figsize=(13, 5))

param_grid_linear = {'C': np.linspace(1, 2, 6),
                     'gamma':  np.linspace(1, 2, 6)}

param_grid_one_log = {'C': np.linspace(1, 2, 6),
                      'gamma':  np.logspace(-3, 2, 6)}

param_grid_range = {'C': np.logspace(-3, 2, 6),
                    'gamma':  np.logspace(-7, -2, 6)}
```

```
for param_grid, ax in zip([param_grid_linear, param_grid_one_log,
                           param_grid_range], axes):
    grid_search = GridSearchCV(SVC(), param_grid, cv=5)
    grid_search.fit(X_train, y_train)
    scores = grid_search.cv_results_['mean_test_score'].reshape(6, 6)

    # 교차 검증 평균 점수의 히트맵 그래프
    scores_image = mglearn.tools.heatmap(
        scores, xlabel='gamma', ylabel='C', xticklabels=param_grid['gamma'],
        yticklabels=param_grid['C'], cmap="viridis", ax=ax)

plt.colorbar(scores_image, ax=axes.tolist())
```



요약

- 교차 검증
- 그리드 서치