

Introduction to Machine Learning with Python

5. Model Evaluation and Improvement(1)

Honedae Machine Learning Study Epoch #2

Contacts

Haesun Park

Email : haesunrpark@gmail.com

Meetup: <https://www.meetup.com/Hongdae-Machine-Learning-Study/>

Facebook : <https://facebook.com/haesunrpark>

Blog : <https://tensorflow.blog>

Book

파이썬 라이브러리를 활용한 머신러닝, 박해선.

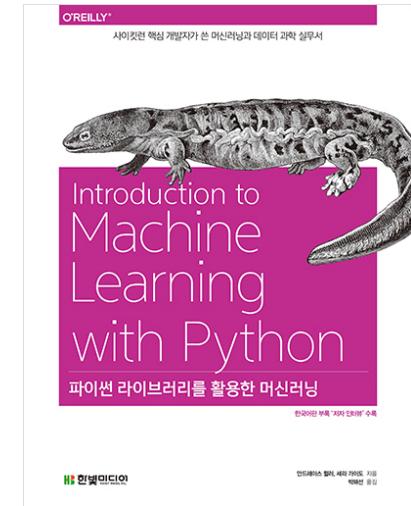
(Introduction to Machine Learning with Python, Andreas Muller & Sarah Guido의 번역서입니다.)

번역서의 1장과 2장은 블로그에서 무료로 읽을 수 있습니다.

원서에 대한 프리뷰를 온라인에서 볼 수 있습니다.

Github:

https://github.com/rickiepark/introduction_to_ml_with_python/



교차 검증

모델 평가

여기서는 비지도 학습의 평가는 정성적이므로 지도 학습의 문제에 집중합니다.

train_test_split → fit → score

```
In [3]: from sklearn.datasets import make_blobs
        from sklearn.linear_model import LogisticRegression
        from sklearn.model_selection import train_test_split

        # 인위적인 데이터셋을 만듭니다
        X, y = make_blobs(random_state=0)
        # 데이터와 타깃 레이블을 훈련 세트와 테스트 세트로 나눕니다
        X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
        # 모델 객체를 만들고 훈련 세트로 학습시킵니다
        logreg = LogisticRegression().fit(X_train, y_train)
        # 모델을 테스트 세트로 평가합니다
        print("테스트 세트 점수: {:.2f}".format(logreg.score(X_test, y_test)))
```

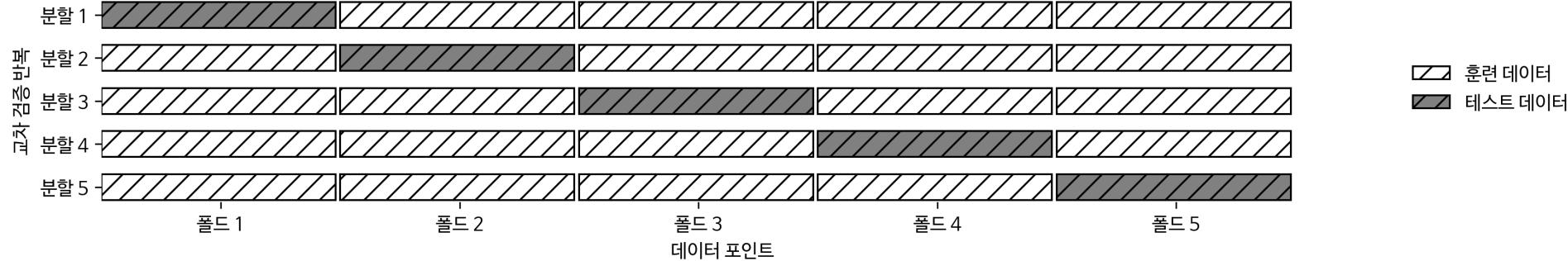
테스트 세트 점수: 0.88

일반화 성능 측정(정확도, R^2)

교차 검증 cross-validation

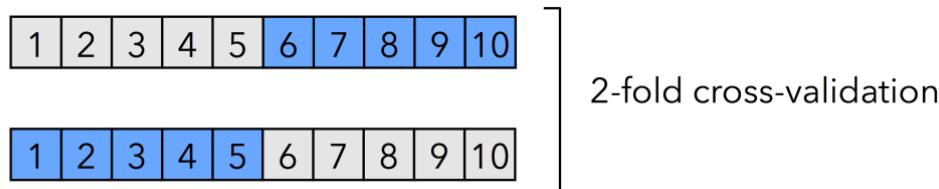
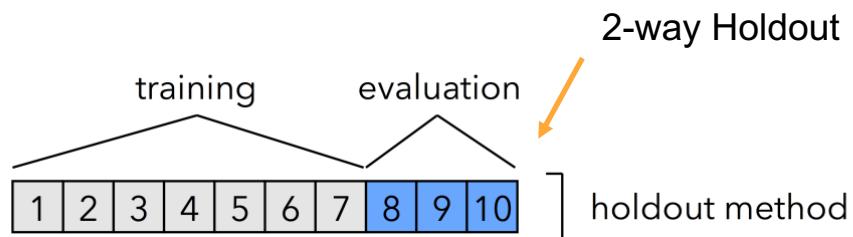
k-겹 교차 검증 k-fold cross-validation ($k=5$ or $k=10$)

1. 훈련 데이터를 k 개의 부분 집합(폴드)으로 나눕니다.
2. 첫 번째 폴드를 테스트 세트로하고 나머지 폴드로 모델을 훈련 시킵니다.
3. 테스트 폴드를 바꾸어 가며 모든 폴드가 사용될 때까지 반복합니다.



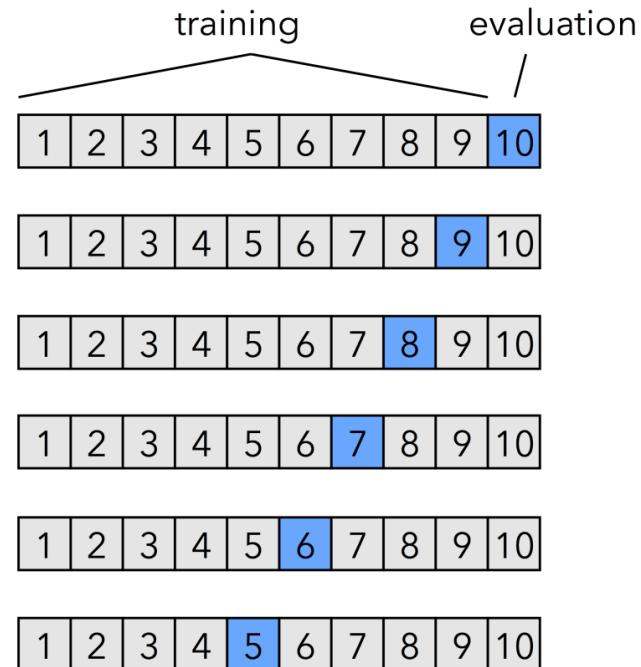
Others

홀드아웃 Holdout



k=n인 교차검증(CV)

LOOCV leave-one-out cross-validation



cross_val_score

```
In [5]: 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: {}".format(scores))
```

교차 검증 점수: [0.961 0.922 0.958] ← 기본 폴드 수 : 3

모델, 데이터, 타깃

```
In [6]: scores = cross_val_score(logreg, iris.data, iris.target, cv=5)  
print("교차 검증 점수: {}".format(scores))
```

교차 검증 점수: [1. 0.967 0.933 0.9 1.]

폴드 수 변경

```
In [7]: print("교차 검증 평균 점수: {:.2f}".format(scores.mean()))
```

교차 검증 평균 점수: 0.96

교차 검증의 장점

train_test_split는 무작위로 데이터를 나누기 때문에 우연히 평가가 좋게 혹은 나쁘게 나올 수 있습니다.

→교차검증(cross_val_score)은 모든 데이터가 테스트 대상이 되기 때문에 공평합니다.

train_test_split는 보통 70%~80%를 훈련에 사용합니다.

→예를 들어, 10겹 교차 검증은 90%를 훈련에 사용하기 때문에 조금 더 정확한 평가를 얻습니다.

모델이 훈련 데이터에 얼마나 민감한지 가늠할 수 있습니다.

[1, 0.967, 0.933, 0.9, 1] → 90~100% Accuracy

[단점]: 데이터를 한 번 나누었을 때보다 k개의 모델을 만드므로 k배 느립니다.

[주의]: cross_val_score는 교차 검증 동안 만든 모델을 반환하지 않습니다!
(cross_val_predict로 예측 점수를 만들 수 있습니다)

교차 검증의 함정

iris 데이터셋에 3-겹 교차 검증을 적용할 경우 테스트 세트에는 한 종류의 붓꽃만 포함됩니다.

```
In [8]: from sklearn.datasets import load_iris  
iris = load_iris()  
print("Iris 레이블:\n{}".format(iris.target))
```

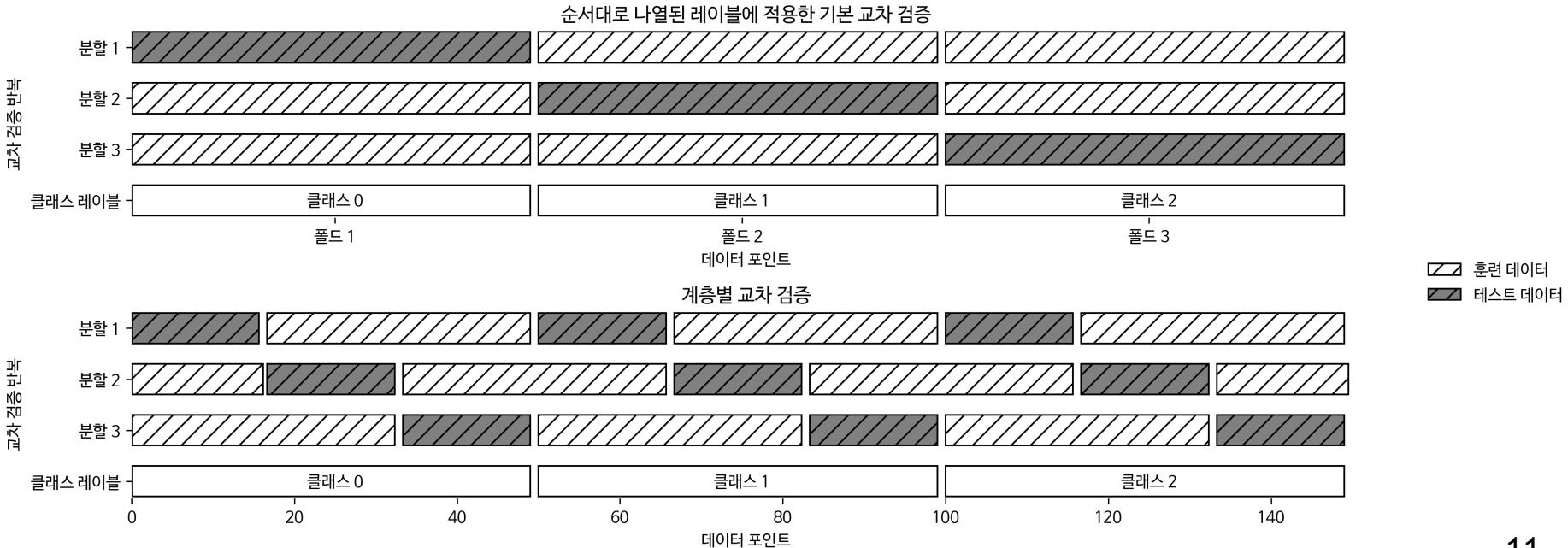
Iris 레이블:

KFold(n_splits=3)

→ 데이터를 3등분하면 훈련 세트에 레이블 2가 포함되어 있지 않으므로 테스트 세트에 대한 점수가 0이 됩니다.

계층별 k-겹 교차 검증 Stratified k-fold CV

분류 문제일 경우 `cross_val_score`는 기본적으로 `StratifiedKFold()`를 사용합니다.
회귀에서는 `KFold()`를 사용합니다.



교차 검증 분할기

분류에 KStratifiedFold() 대신 기본 KFold()를 적용하는 등 기본 분할기 대신 다른 전략(ShuffleSplit, GroupKFold 등)을 사용하려면 cv 매개변수를 사용합니다.

회귀 문제일 경우 보통 데이터를 섞지 않지만 필요할 경우 Kfold(shuffle=True)를 사용합니다.

```
In [10]: from sklearn.model_selection import KFold  
kfold = KFold(n_splits=5)
```

```
In [11]: print("교차 검증 점수:\n{}".format(  
    cross_val_score(logreg, iris.data, iris.target, cv=kfold)))
```

교차 검증 점수:
[1. 0.933 0.433 0.967 0.433]

분류에 기본 KFold() 사용하기(실패 사례)

Iris 레이블:

```
In [12]: kfold = KFold(n_splits=3)
print("교차 검증 점수:\n{}".format(
    cross_val_score(logreg, iris.data, iris.target, cv=kfold)))
```

교차 검증 점수:

[0. 0. 0.]

데이터 섭기

```
In [13]: kfold = KFold(n_splits=3, shuffle=True, random_state=0)
print("교차 검증 점수: \n{}".format(
    cross_val_score(logreg, iris.data, iris.target, cv=kfold)))
```

교차 검증 점수:

[-0.9 0.96 0.96]

StratifiedKFold(n_splits=3): [0.961, 0.922, 0.958]

LOOCV\leave-one-out cross-validation

k=n 인 k-겹 교차 검증

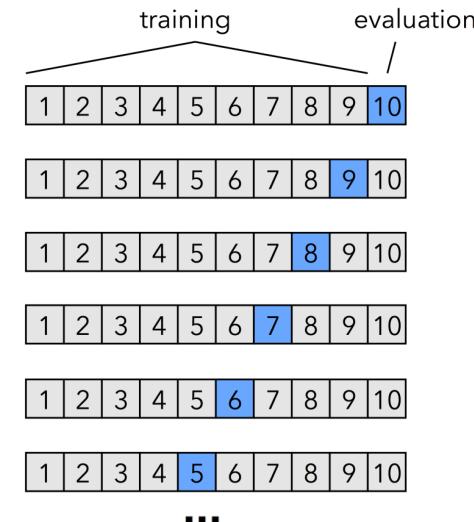
데이터셋이 클 때는 시간이 오래 걸리지만, 작은 데이터셋에서는 이따금 좋은 결과를 만듭니다.

In [14]:

```
from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()
scores = cross_val_score(logreg, iris.data, iris.target, cv=loo)
print("교차 검증 분할 횟수: ", len(scores))
print("평균 정확도: {:.2f}".format(scores.mean()))
```

교차 검증 분할 횟수: 150

평균 정확도: 0.95



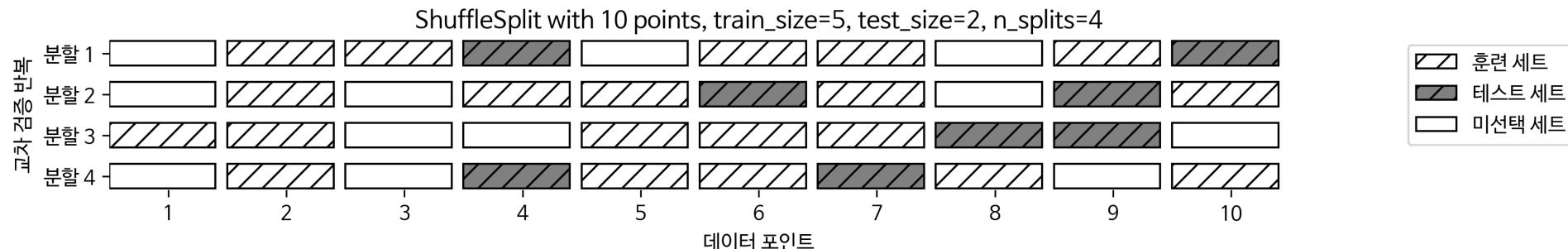
임의 분할 교차 검증 shuffle-split CV

train_size/test_size: 훈련/테스트 세트 크기(정수는 절대 개수, 실수는 비율을 의미)

n_splits: 폴드 수

랜덤 추출이기 때문에 한 샘플이 여러 폴드에 포함될 수 있습니다.

ShuffleSplit(n_splits=4, train_size=5, test_size=2, random_state=42)



ShuffleSplit

test size, train size에 비율을 입력할 수 있습니다.

test size + train size < 1 일 경우 부분 샘플링 subsampling이 됩니다.

분류에 사용할 수 있는 StratifiedShuffleSplit()도 있습니다.

In [16]: `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{}".format(scores))`

절반씩 나누어 훈련/테스트
10번 반복

그룹별 교차 검증

타깃에 따라 폴드를 나누지 않고 입력 특성에 따라 폴드를 나누어야 할 경우

예를 들어 100장의 사진 데이터로 사람의 표정을 분류하는 문제에서 한 사람이 훈련 세트와 테스트 세트에 모두 나타날 경우 분류기의 성능을 정확히 측정할 수 없습니다.(의료 정보나 음성 인식 등에서도)

입력 데이터의 그룹 정보를 받을 수 있는 GroupKFold()를 사용합니다.

```
cross_val_score(model, X, y, groups, cv=GroupKFold(n_splits=3))
{...
cv.split(X, y, groups)
...}
```

GroupKFold

```
In [17]: 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{}".format(scores))
```

교차 검증 점수:

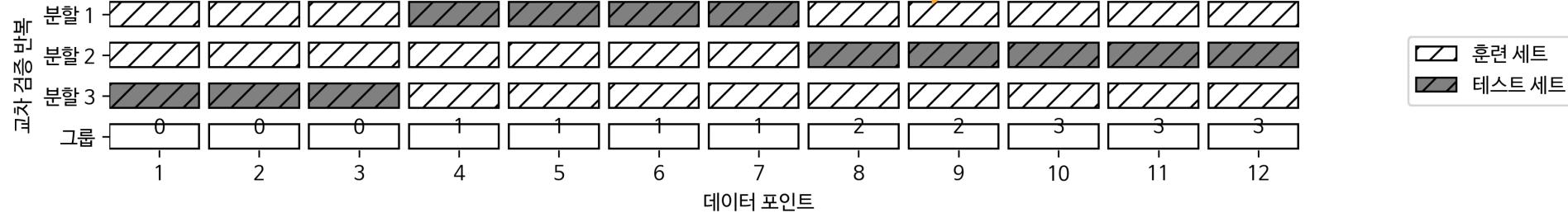
[0.75 0.8 0.667]

4개의 그룹

3개의 폴드

한 개의 그룹이 훈련/테스트 세트로
나누어지지 않음

GroupKFold



반복 교차 검증

scikit-learn 0.19 버전에서 추가: RepeatedKFold(), RepeatedStratifiedKFold()

안정된 교차 검증의 결과를 얻기 위해 지정한 횟수만큼 반복해서 실행합니다.

반복할 때 무작위성 때문에 분할기에 shuffle=True 옵션이 자동으로 적용됩니다.

자세한 내용은 블로그 참조: <https://goo.gl/ufQHQZ>

5겹 교차 검증이 5번 반복

```
from sklearn.model_selection import RepeatedKFold  
rkfold = RepeatedKFold(n_splits=5, n_repeats=5, random_state=42)  
scores = cross_val_score(logreg, iris.data, iris.target, cv=rkfold)  
scores, scores.mean()
```

교차 검증 결과는 25개

```
(array([ 1.          ,  0.93333333,  0.9        ,  0.96666667,  0.96666667,  
       0.96666667,  0.93333333,  1.          ,  1.          ,  0.83333333,  
       0.93333333,  0.9        ,  0.96666667,  0.9        ,  0.93333333,  
       0.96666667,  1.          ,  0.96666667,  0.93333333,  0.93333333,  
       0.96666667,  0.9        ,  1.          ,  0.93333333,  0.93333333],  
      0.9466666666666677)
```

그리드 서치|Grid Search

매개변수 튜닝

모델의 일반화 성능을 최대로 만드는 매개변수를 찾는 작업입니다(필수적인 과정).

GridSearchCV와 RandomizedSearchCV를 사용하여 손쉽게 처리할 수 있습니다.

예) RBF 커널 SVM의 여러가지 매개변수 조합을 테스트합니다.

	C = 0.001	C = 0.01	...	C = 10
gamma=0.001	SVC(C=0.001, gamma=0.001)	SVC(C=0.01, gamma=0.001)	...	SVC(C=10, gamma=0.001)
gamma=0.01	SVC(C=0.001, gamma=0.01)	SVC(C=0.01, gamma=0.01)	...	SVC(C=10, gamma=0.01)
...
gamma=100	SVC(C=0.001, gamma=100)	SVC(C=0.01, gamma=100)	...	SVC(C=10, gamma=100)

그리드 서치 직접 구현

```
In [19]: # 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]))
```

gamma와 C에 대한
반복 루프

```
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: {}".format(best_parameters))
```

훈련 세트의 크기: 112 테스트 세트의 크기: 38

최고 점수: 0.97

최적 파라미터: {'gamma': 0.001, 'C': 100}

36개의 모델이 만들어짐

가장 좋은 테스트 점수와
매개변수를 기록

검증 세트 validation set

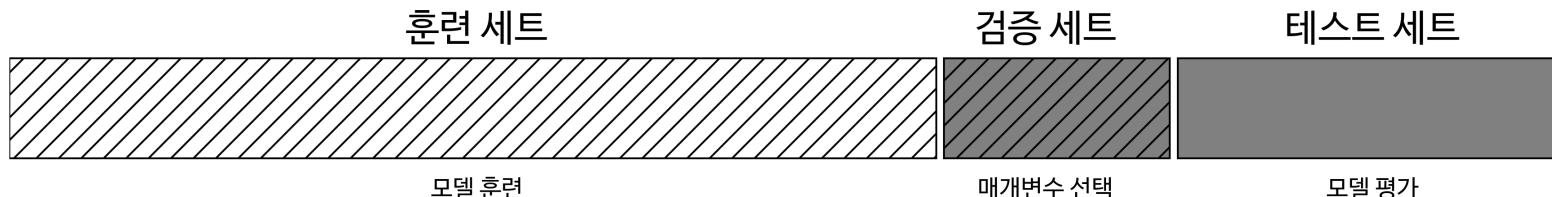
최고 점수: 0.97

최적 파라미터: {'gamma': 0.001, 'C': 100}

그러나 테스트 세트로 여러가지 매개변수 조합에 대해 평가했다면 이 모델이 새로운 데이터도 동일한 성능을 낸다고 생각하는 것은 매우 낙관적인 추정입니다.

최종 평가를 위해서는 독립된 데이터 세트가 필요합니다.

이를 검증 세트 validation set 혹은 개발 세트 dev set라고 합니다.



검증 세트 구현

```
In [21]: 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 데이터 → 훈련검증세트, 테스트 세트
훈련검증세트 → 훈련 세트, 검증 세트

검증세트로 모델을 고릅니다.

두 구현의 결과 비교

훈련/테스트 분할

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

훈련/검증/테스트 분할

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

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

사용하지 않은 테스트 세트로
최종 모델을 평가

절대로 모델 시각화, 탐색적 분석, 모델 선택, 매개변수 튜닝 등에 테스트 세트를 사용하면 안됩니다.(6장 정보 누설 참고)

검증 세트 대신 cross_val_score

In [22]:

```
svm.fit(X_train, y_train)
# 검증 세트로 SVC를 평가합니다
score = svm.score(X_valid, y_valid)
```

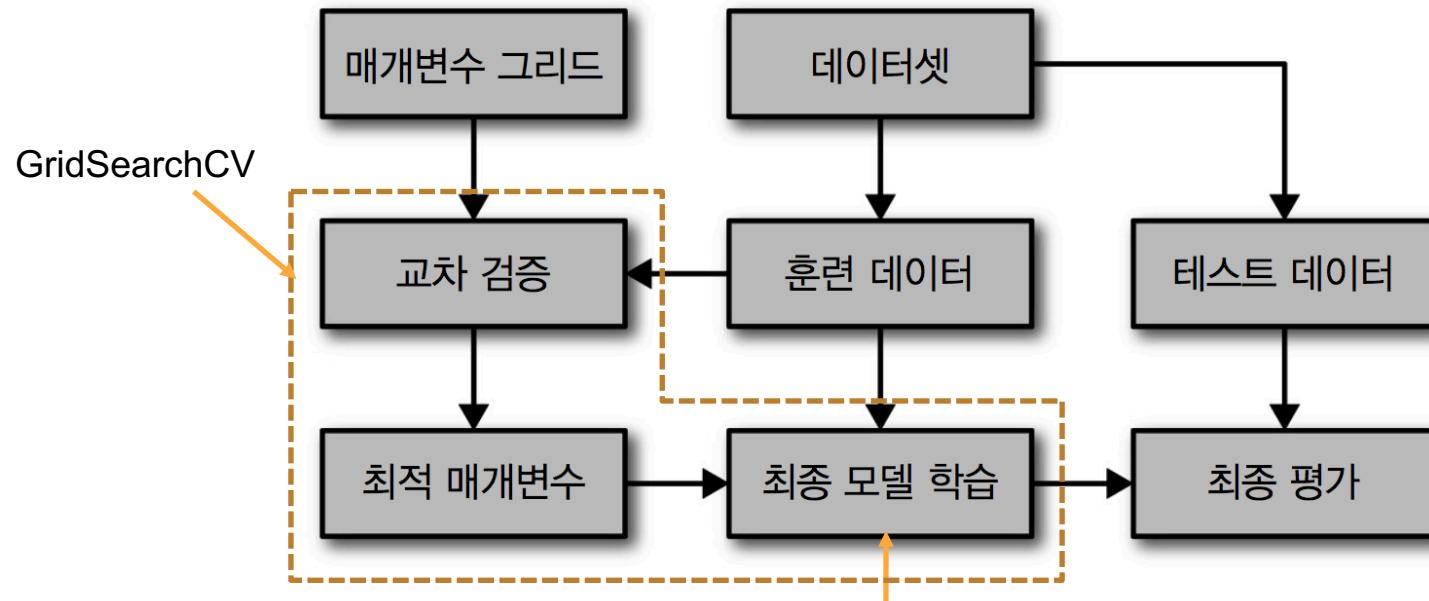
검증세트로 직접 점수를 계산하는 대신에 훈련+검증 세트에 대해 교차검증을 적용합니다.

```
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)
```

6x6x5 개의 SVM 모델이 만들어짐

그리드 서치를 사용한 매개변수 탐색의 전체 과정



종종 교차검증을 교차검증을
사용한 그리드 서치를
의미하는 것으로 사용합니다.

최종 모델에서는 훈련/검증 세트를 모두 활용합니다.

GridSearchCV

교차 검증을 사용한 그리드 서치(for loop + cross_val_score)

검색하려는 매개변수를 키로 하는 딕셔너리를 사용함

```
In [25]: 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{}".format(param_grid))
```

매개변수 그리드:

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

```
In [26]: from sklearn.model_selection import GridSearchCV  
from sklearn.svm import SVC  
grid_search = GridSearchCV(SVC(), param_grid, cv=5)
```

훈련 점수를 보려면
return_train_score=True
(v0.21에서 기본값이 False가
됨을 알리는 경고 발생하므로)

모델, 매개변수그리드, 폴드수
회귀:KFold, 분류:StratifiedKFold
cross_val_score처럼 분할기 주입 가능

```
In [27]: X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,  
                                                       random_state=0)
```

```
In [28]: grid_search.fit(X_train, y_train)
```

fit, predict, score,
predict_proba, decision_function 제공

6*6*5 개의 모델 생성 + 최적매개변수 탐색 + 최종 모델 학습

최종 모델 테스트

GridSearchCV에 사용하지 않은 테스트 세트로 평가

```
In [29]: print("테스트 세트 점수: {:.2f}".format(grid_search.score(X_test, y_test)))  
테스트 세트 점수: 0.97
```

```
In [30]: print("최적 매개변수: {}".format(grid_search.best_params_))  
print("최고 교차 검증 점수: {:.2f}".format(grid_search.best_score_))  
최적 매개변수: {'gamma': 0.01, 'C': 100}  
최고 교차 검증 점수: 0.97
```

훈련 데이터에 대한 교차검증 점수의
평균 최대값

```
In [31]: print("최고 성능 모델:\n{}".format(grid_search.best_estimator_))  
최고 성능 모델:  
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)
```

최종 학습 모델

결과 분석

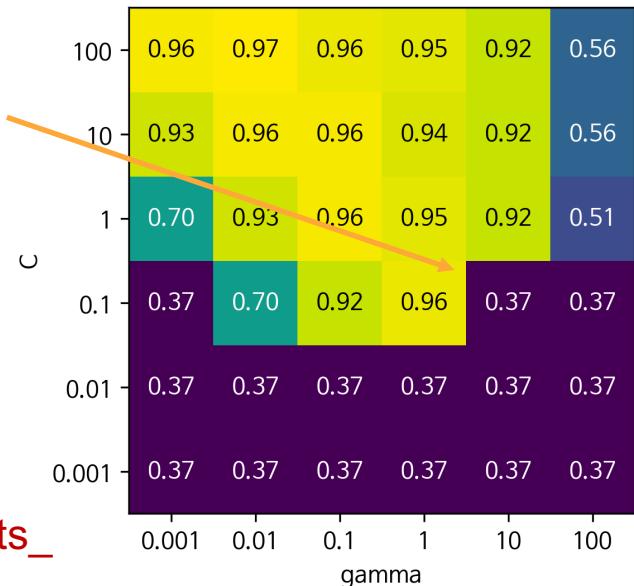
간격을 넓게 하여 그리드 서치를 시작하고 결과를 분석해 검색 영역을 조정합니다.

RandomizedSearchCV는 매개변수 조합이 매우 많거나 매개변수 C와 같이 연속형 값을 조정할 때 많이 사용됩니다.

	param_C	param_gamma	params	mean_test_score
0	0.001	0.001	{'C': 0.001, 'gamma': 0.001}	0.366
1	0.001	0.01	{'C': 0.001, 'gamma': 0.01}	0.366
2	0.001	0.1	{'C': 0.001, 'gamma': 0.1}	0.366
3	0.001	1	{'C': 0.001, 'gamma': 1}	0.366
4	0.001	10	{'C': 0.001, 'gamma': 10}	0.366
	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	22	0.375	0.347	0.363
1	22	0.375	0.347	0.363
2	22	0.375	0.347	0.363
3	22	0.375	0.347	0.363
4	22	0.375	0.347	0.363
	split3_test_score	split4_test_score	std_test_score	
0	0.363	0.380	0.011	
1	0.363	0.380	0.011	
2	0.363	0.380	0.011	
3	0.363	0.380	0.011	
4	0.363	0.380	0.011	

매개변수에 민감한
모델을 보여줍니다.

pandas.DataFrame
search.cv_result



GridSearchCV + RepeatedKFold

그리드서치에 기본 분할기인 KFold, StratifiedKFold 대신에 반복 교차 검증을 적용할 수 있습니다.(0.19 버전 추가 내용)

5폴드 분할을 5번 반복

```
from sklearn.model_selection import RepeatedStratifiedKFold
rskfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=5, random_state=42)

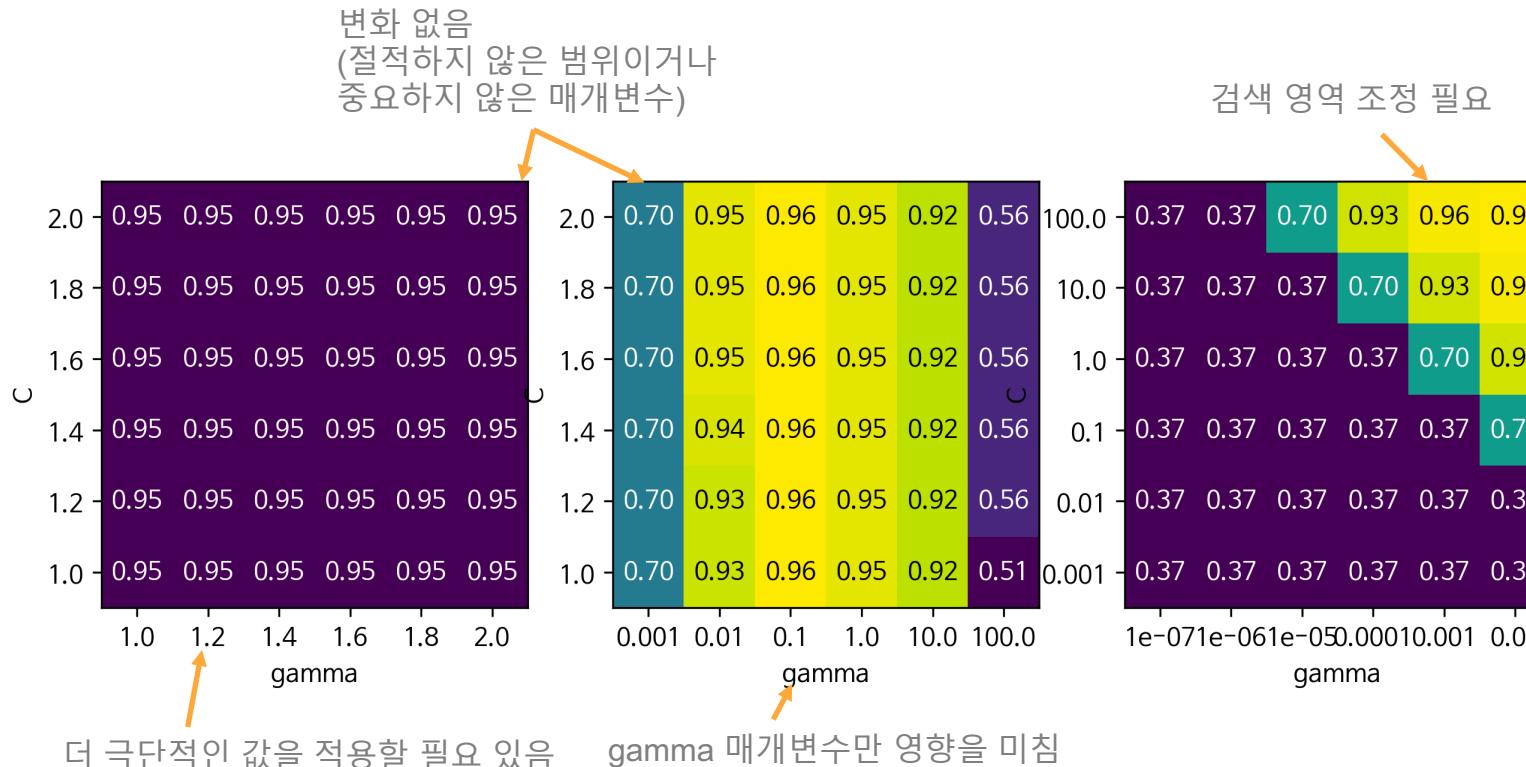
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_search = GridSearchCV(logreg, param_grid, cv=rskfold, return_train_score=True)
grid_search.fit(X_train, y_train)
```

탐색할 매개변수 그리드 7개

split21_test_score [0.34782609 0.65217391 0.82608696 0.91304348 0.86956522 0.91304348
0.86956522]
split6_test_score [0.34782609 0.65217391 0.7826087 0.91304348 0.95652174 0.95652174
0.95652174]
split24_train_score [0.34065934 0.64835165 0.83516484 0.94505495 0.96703297 0.96703297
0.96703297]
split3_train_score [0.34444444 0.65555556 0.82222222 0.95555556 0.96666667 0.96666667
0.96666667]
split2_test_score [0.34782609 0.65217391 0.86956522 0.91304348 0.91304348 0.91304348
0.91304348]
split16_train_score [0.33707865 0.65168539 0.80898876 0.95505618 0.96629213 0.96629213
0.96629213]
split10_test_score [0.34782609 0.65217391 0.82608696 0.91304348 0.95652174 1. 1.]

7*5*5=175개 모델 생성

부적절한 그리드



비대칭 그리드 서치

매개변수에 따라 선택적으로 다른 매개변수의 조합을 적용할 수 있습니다.

매개변수 그리드를 딕셔너리의 리스트로 만듭니다.

```
In [35]: param_grid = [{"kernel": ["rbf"],  
                      "C": [0.001, 0.01, 0.1, 1, 10, 100],  
                      "gamma": [0.001, 0.01, 0.1, 1, 10, 100]},  
                     {"kernel": ["linear"],  
                      "C": [0.001, 0.01, 0.1, 1, 10, 100]}]  
print("그리드 목록:\n{}".format(param_grid))
```

linear 커널일 경우에는 gamma 매개변수를 지정하지 않습니다.

비대칭 그리드 서치 결과

	0	1	2	3	...	38	39	40	41
param_C	0.001	0.001	0.001	0.001	...	0.1	1	10	100
param_gamma	0.001	0.01	0.1	1	...	NaN	NaN	NaN	NaN
param_kernel	rbf	rbf	rbf	rbf	...	linear	linear	linear	linear
params	{'C': 0.001, 'kernel': 'rbf', 'gamma': 0.001}	{'C': 0.001, 'kernel': 'rbf', 'gamma': 0.01}	{'C': 0.001, 'kernel': 'rbf', 'gamma': 0.1}	{'C': 0.001, 'kernel': 'rbf', 'gamma': 1}	...	{'C': 0.1, 'kernel': 'rbf', 'gamma': 0.01}	{'C': 1, 'kernel': 'rbf', 'gamma': 0.01}	{'C': 10, 'kernel': 'rbf', 'gamma': 0.01}	{'C': 100, 'kernel': 'rbf', 'gamma': 0.01}
mean_test_score	0.37	0.37	0.37	0.37					
rank_test_score	27	27	27	27					
split0_test_score	0.38	0.38	0.38	0.38					
split1_test_score	0.35	0.35	0.35	0.35					
split2_test_score	0.36	0.36	0.36	0.36	...	1	1	1	1
split3_test_score	0.36	0.36	0.36	0.36	...	0.91	0.95	0.91	0.91
split4_test_score	0.38	0.38	0.38	0.38	...	0.95	0.95	0.95	0.95
std_test_score	0.011	0.011	0.011	0.011	...	0.033	0.022	0.034	0.034

```
In [36]: grid_search = GridSearchCV(SVC(), param_grid, cv=5)
grid_search.fit(X_train, y_train)
print("최적 파라미터: {}".format(grid_search.best_params_))
print("최고 교차 검증 점수: {:.2f}".format(grid_search.best_score_))

최적 파라미터: {'kernel': 'rbf', 'gamma': 0.01, 'C': 100}
최고 교차 검증 점수: 0.97
```

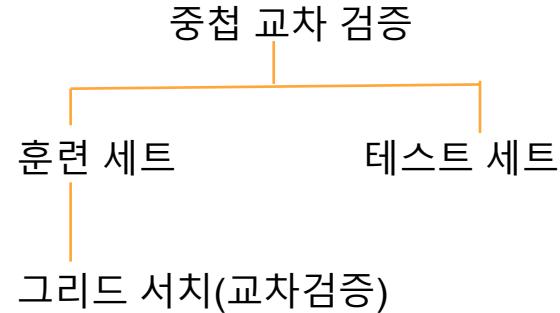
kernel이 Linear일 때 gamma는 적용되지 않음

scikit-learn 0.19
return_train_score=True

중첩 교차 검증nested CV

중첩 교차 검증(cross_val_score)은
그리드서치(GridSearchCV)의 교차 검증입니다.

주어진 모델이 얼마나 잘 일반화 되는지 평가합니다.



```
In [38]: param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],  
                   'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}  
scores = cross_val_score(GridSearchCV(SVC(), param_grid, cv=5),  
                         iris.data, iris.target, cv=5)  
print("교차 검증 점수: ", scores)  
print("교차 검증 평균 점수: ", scores.mean())  
print(param_grid)
```

교차 검증 점수: [0.967 1. 0.967 0.967 1.]

교차 검증 평균 점수: 0.98

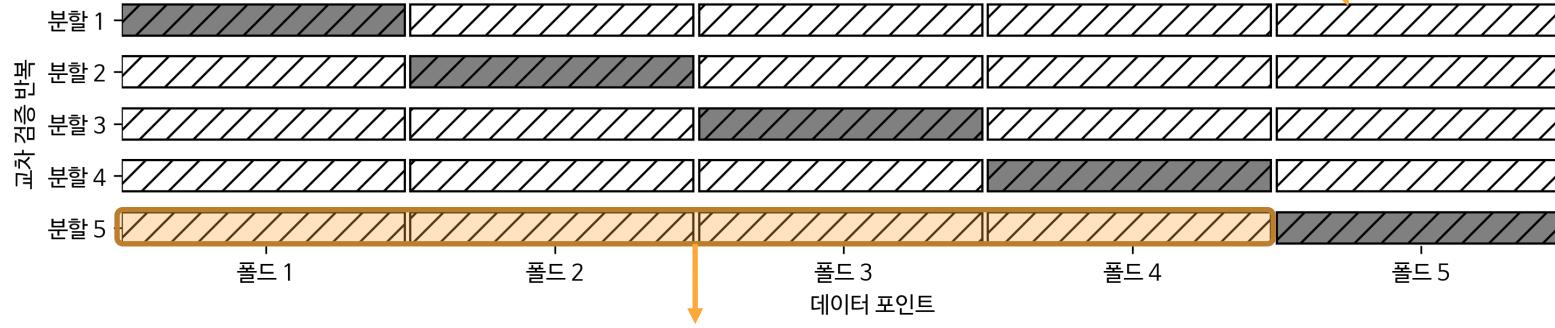
{'gamma': [0.001, 0.01, 0.1, 1, 10, 100], 'C': [0.001, 0.01, 0.1, 1, 10, 100]}

5개의 점수

cross_val_score + GridSearchCV

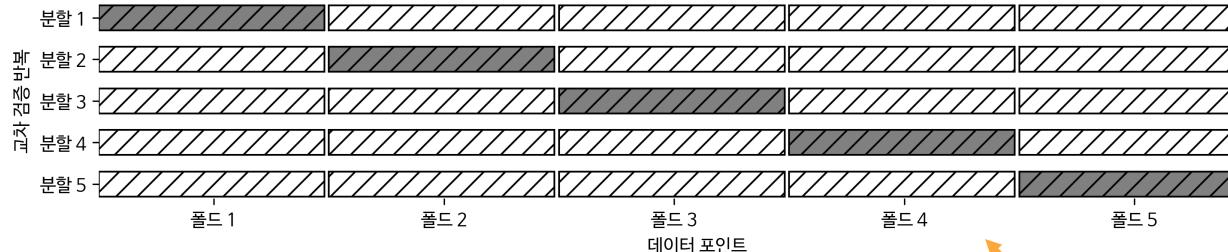
```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],  
             'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
```

36개의 조합



cross_val_score

데이터 포인트



36x5x5=900개의
모델이 만들어짐

GridSearchCV(cross_val_score)

병렬화

그리드 서치는 연산에 비용이 많이 들지만 매개변수 조합마다 쉽게 병렬화 가능합니다.

GridSearchCV와 cross_val_score의 n_jobs에 사용할 CPU 코어수를 지정할 수 있습니다(기본 1, 최대 -1).

하지만 GridSearchCV와 RandomForestClassifier 같은 모델이 동시에 n_jobs 옵션을 사용할 수 없습니다(이중 fork로 데몬 프로세스가 되는 것을 방지).

마찬가지로 cross_val_score와 GridSearchCV도 동시에 n_jobs 옵션을 사용할 수 없습니다.

대안: ipyparallel(<https://goo.gl/4A7u6Z>), dask-searchcv(<https://goo.gl/h5fzSx>), spark-sklearn(<https://goo.gl/9qY98a>)

감사합니다.

-질문-