# 모델 평가와 성능 향상

5.1~5.2

1770119 황서현

# INDEX

# 모델평가

- Train_test_split 을 쓰는 이유

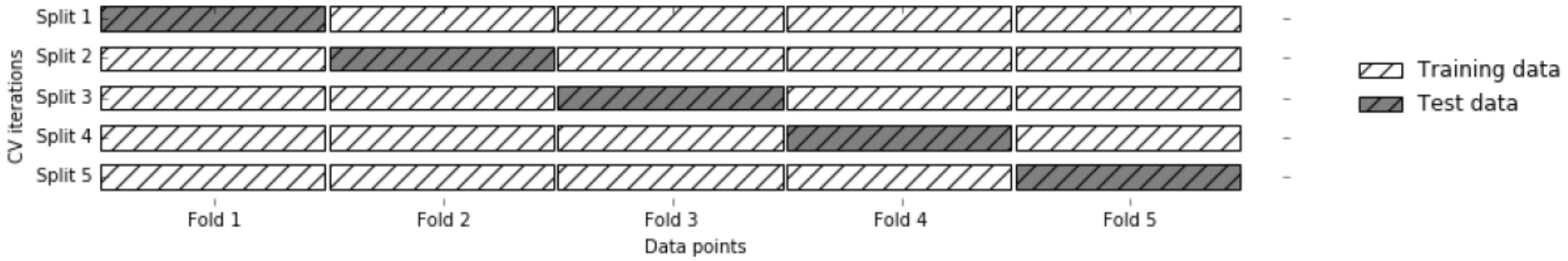
  새로운 데이터를 모델이 얼마나 잘 일반화했는지 측정하기 위해

- 앞으로의 목표

  평가 방법을 확장시켜 보자 -> 교차검증

# 교차 검증

# 교차 검증(cross-validation)




종류

- K겹 교차 검증 (K-fold cross-validation)
- 계층별 K겹 교차 검증

# Scikit-learn CV 코드

In [2]:
```
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,cv=5)
print("교차검증 점수: ",scores)
print("교차검증 평균점수: {:.2f}".format(scores.mean()))
```

```
교차검증 점수:  [1.         0.96666667 0.93333333 0.9        1.        ]
교차검증 평균점수: 0.96
```

# 교차 검증 장단점

## 장점

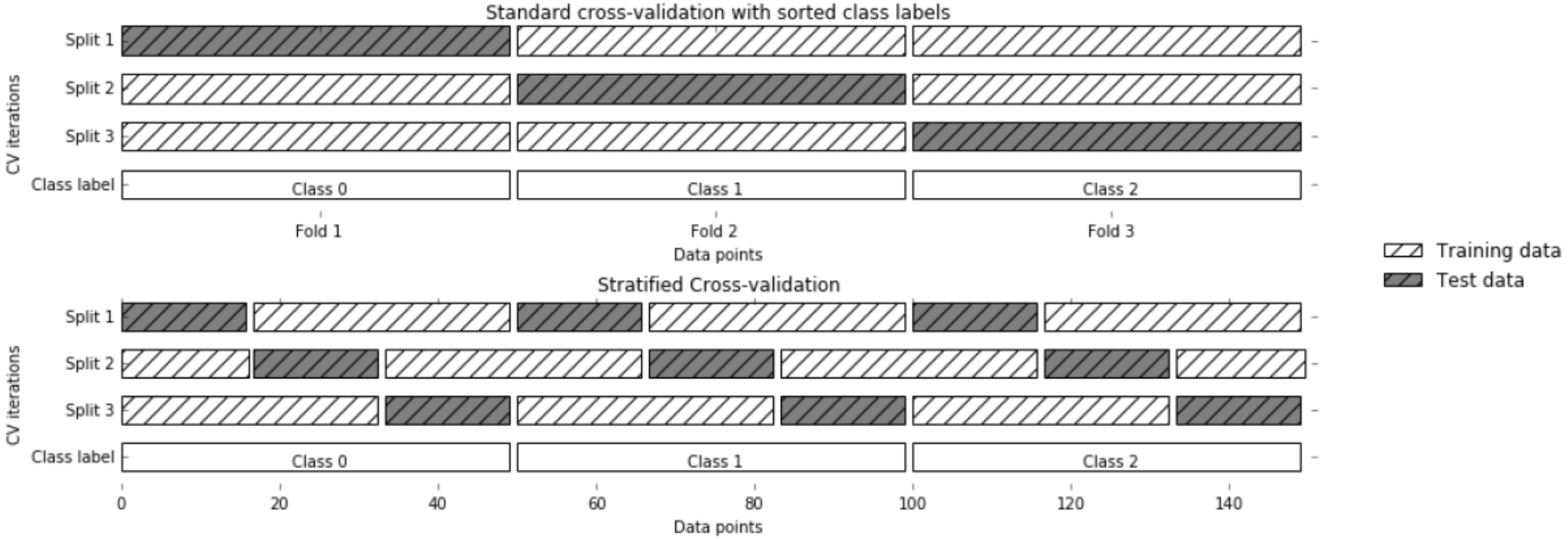
- 랜덤 split의 단점 보완 (ex 분류가 어려운 샘플이 다 훈련데이터가 됨)
- 훈련 데이터가 모델에 얼마나 민감한지 (ex 훈련 데이터 정확도를 토대로 예측)
- 훈련 데이터량 증가(ex train_test_split[75%] < 10-fold cross_validation[90%]

## 단점

- 모델을 k번 더 만드므로 k배 더 느림

# 계층별 K-겹 교차 검증과 그 외 전략

- 데이터 순서대로 K를 나누면 안된다
- 클래스의 비율이 전체 데이터셋의 클래스 비율과 같도록!
- 보통 회귀는 기본 K겹 CV, 분류는 계층별 K겹 CV 이용함



# 계층별 K-겹 교차 검증과 그 외 전략

## LOOCV(import LeaveOneOut)

Fold 하나에 샘플 하나

In [3]:
```
from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()
scores = cross_val_score(logreg,iris.data,iris.target,cv=loo)
print("교차검증 분할횟수: ",len(scores))
print("교차검증 평균점수: {:.2f}".format(scores.mean()))
```

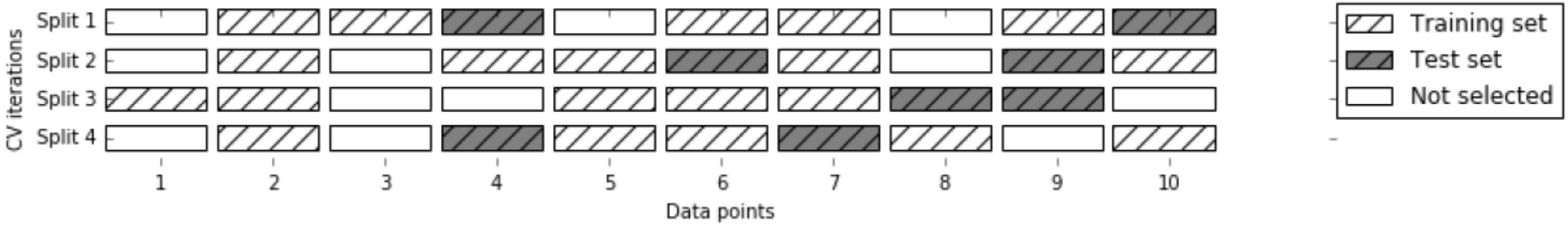
```
교차검증 분할횟수:  150
교차검증 평균점수: 0.95
```

# 계층별 K-겹 교차 검증과 그 외 전략

## 임의 분할 교차 검증(import ShuffleSplit)

Train과 test 세트 임의로 분할 가능

부분 샘플링에 많이 쓰임



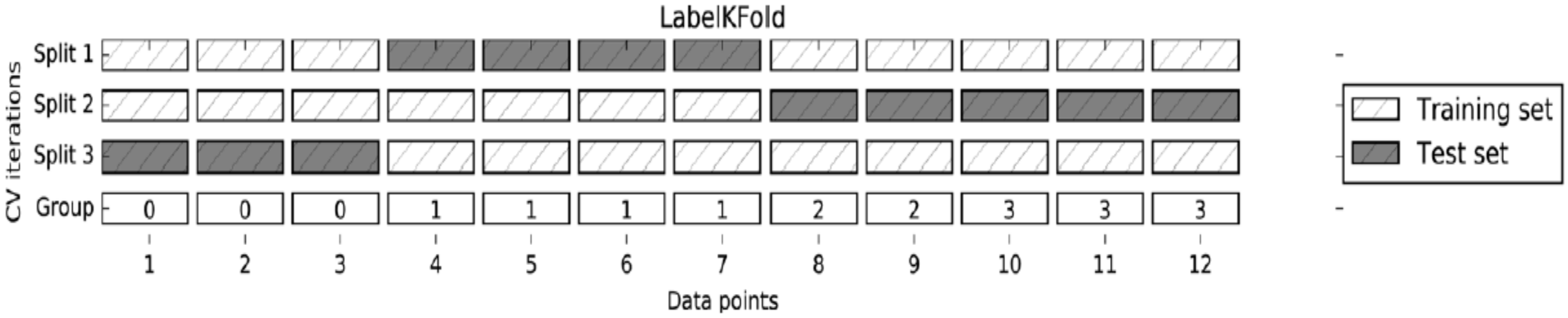

Train_size = 5  test_size = 2  n_split=4를 적용한 ShuffleSplit

# 계층별 K-겹 교차 검증과 그 외 전략

## 그룹별 교차 검증(import GroupKFold)

Train과 test에 같은 group이 들어가면 안됨

Ex) 의료 데이터 / 표정 구분용 사진 데이터 / 음성 인식 등



# 계층별 K-겹 교차 검증과 그 외 전략

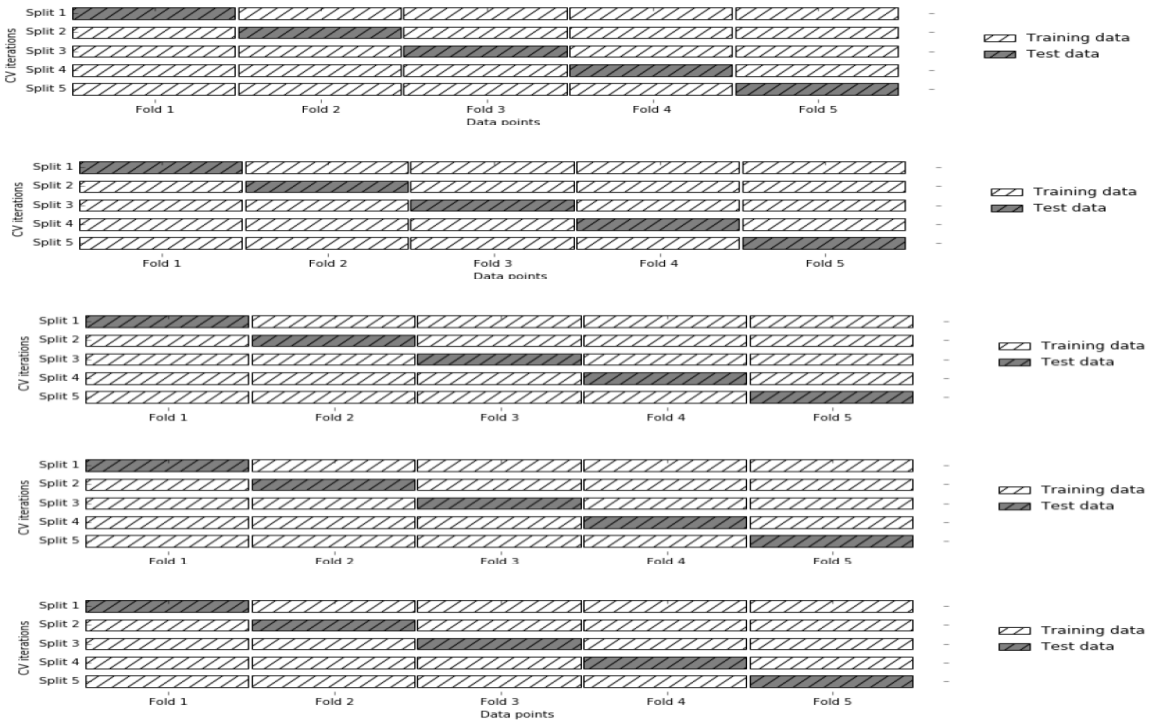
## 반복 교차 검증(import RepeatedStratifiedKFold)

CV전체를 다시 반복하여 검증함

N_splits = 5 (fold값)

N_repeats = 10 (반복값)

총 50번 반복



# 그리드 서치

# 그리드 서치

- 매개변수 튜닝
- 관심있는 매개변수를 대상으로 가능한 모든 조합을 시도

| | 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) |

# 간단한 그리드 서치 코드

In [7]:

```
##간단 그리드 서치 과정
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(iris.data,iris.target,random_state=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]:
        #매개변수 조합에 대해 svc 훈련
        svm = SVC(gamma=gamma,C=C)
        svm.fit(X_train,y_train)
        #test세트로 svc 평가
        score = svm.score(X_test,y_test)
        #점수가 높으면 best_score에 저장
        if score>best_score:
            best_score=score
            best_parameters = {"C":C,"gamma":gamma}
print("최고점수: {:.2f}".format(best_score))
print("최적 매개변수: ",best_parameters)
```

```
최고점수: 0.97
최적 매개변수:  {'gamma': 0.001, 'C': 100}
```

# 매개변수 과대적합(overfit)과 검증세트

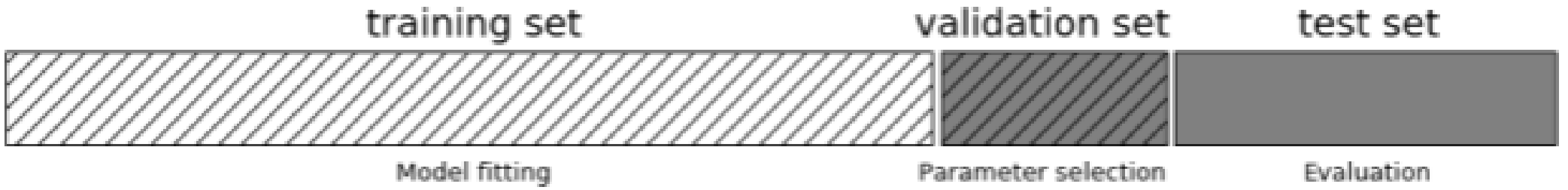
매개변수를 선택하기 위해 이미 테스트 세트를 이용해 버렸다?!

== 처음 train vs test 나눈 게 의미 없어짐

→ 세 개의 세트로 나눈다( 훈련, 검증, 테스트 세트)

**In[19]:**

```
mglearn.plots.plot_threefold_split()
```



# 매개변수 과대적합(overfit)과 검증세트

코드

In [7]:

```
X_trainval,X_test,y_trainval,y_test=train_test_split(iris.data,iris.target,random_state=0)
X_train,X_valid,y_train,y_valid = train_test_split(iris.data,iris.target,random_state=0)
print("훈련세트 크기 = {} 검증세트 크기 = {} 테스트세트 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]:
        #매개변수 조합에 대해 svc 훈련
        svm = SVC(gamma=gamma,C=C)
        svm.fit(X_train,y_train) #######train으로 바꿈
        #test세트로 svc 평가
        score = svm.score(X_valid,y_valid) #######valid로 바꿈
        #점수가 높으면 best_score에 저장
        if score>best_score:
            best_score=score
            best_parameters = {"C":C,"gamma":gamma}
#훈련 세트와 검증 세트를 합쳐 모델을 만들고 테스트 세트로 평가
svm = SVC(**best_parameters)
svm.fit(X_trainval,y_trainval) ######trainval로 바꿈
test_score = svm.score(X_test,y_test) ######test로 바꿈

print("검증 최고점수: {:.2f}".format(best_score))
print("최적 매개변수: ",best_parameters)
print("최적 매개변수에서 테스트 세트 점수: {:.2f}".format(test_score))
```

```
훈련세트 크기 = 112 검증세트 크기 = 38 테스트세트 크기 = 38

검증 최고점수: 0.97
최적 매개변수:  {'C': 100, 'gamma': 0.001}
최적 매개변수에서 테스트 세트 점수: 0.97
```

# 교차 검증(CV)을 이용한 그리드 서치

- 훈련+검증 데

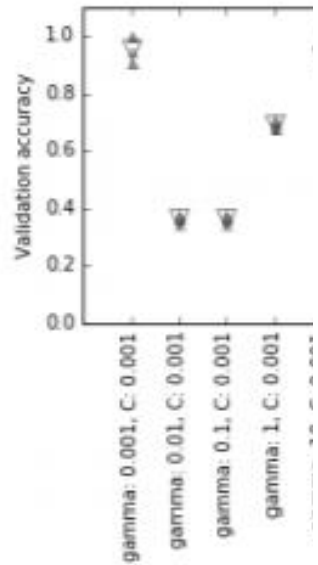
In [11]:
```
##교차검증을 이용한 그리드 서치
import numpy as np

for gamma in [0.001,0.01,0.1,1,10,100]:
    for C in [0.001,0.01,0.1,1,10,100]:
        #매개변수 조합에 대해 svc 훈련
        svm = SVC(gamma=gamma,C=C)
        #교차검증을 적용
        scores = cross_val_score(svm,X_trainval,y_trainval,cv=5)
        #교차 검증 정확도의 평균
        score = np.mean(scores)
        #점수가 높으면 best_score에 저장
        if score>best_score:
            best_score=score
            best_parameters = {"C":C,"gamma":gamma}
svm = SVC(**best_parameters)
svm.fit(X_trainval,y_trainval)
```

Out[11]:
```
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

**In[22]:**

```
mglearn.plots.
```



In [17]:

```
##교차검증을 이용한 그리드 == GridSerachCV
#매개변수를 딕셔너리 형태로 지정해야
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)

from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

#gridSearchCV는 3세트로 나누고 cv도 해줌
grid_search = GridSearchCV(SVC(),param_grid,cv=5,return_train_score=True)

#테스트 세트는 따로 떼어두기
X_train,X_test,y_train,y_test=train_test_split(iris.data,iris.target,random_state=0)
grid_search.fit(X_train,y_train)

print("\n 테스트 세트 점수: {:.2f}\n".format(grid_search.score(X_test,y_test)))
print("최적 매개변수: ",grid_search.best_params_)
print("\n 최고 교차 검증 점수 : {:.2f}n".format(grid_search.best_score_))
```

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

 테스트 세트 점수: 0.97

최적 매개변수:  {'gamma': 0.01, 'C': 100}

 최고 교차 검증 점수 : 0.97n
```

# 교차 검증(CV) 결과 분석

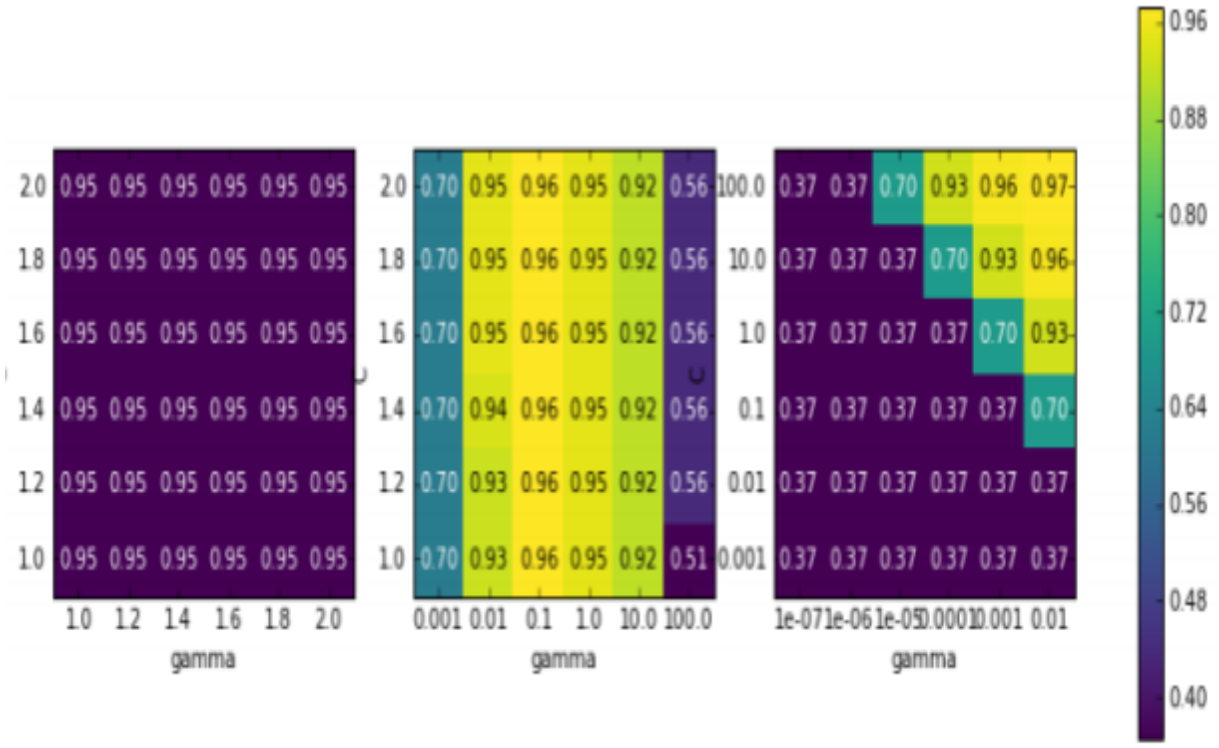
- CV_results 에는 관련 정보가 다 들어있다.

**Out[31]:**

| | 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 |



# 비대칭 매개변수 그리드 탐색

In [22]:
```
##비대칭 매개변수 그리드 탐색
#매개변수를 딕셔너리 형태로 지정해야
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("매개변수 그리드: \n",param_grid)

#gridSearchCV는 3세트로 나누고 cv도 해줌
grid_search = GridSearchCV(SVC(),param_grid,cv=5,return_train_score=True)
grid_search.fit(X_train,y_train)

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

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

최적 매개변수:  {'kernel': 'rbf', 'gamma': 0.01, 'C': 100}

 최고 교차 검증 점수 : 0.97
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 |

# 중첩(nested) 교차 검증(CV)

- 3개 세트로 나눈 것도 CV + (훈련+검증 세트)도 CV 하기 == 중첩

In [24]:
```
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:  [0.96666667 1.         0.96666667 0.96666667 1.        ]
교차 검증 평균 점수:  0.9800000000000001
{'gamma': [0.001, 0.01, 0.1, 1, 10, 100], 'C': [0.001, 0.01, 0.1, 1, 10, 100]}
```

# 교차 검증과 그리드 서치 병렬화

- 데이터 용량이 큼
- 하지만 병렬화가 쉬움
- 왜?? 각 CV 모델을 만드는 게 동시에 일어날 수 있기 때문에
- 따라서, 여러 CPU에서 한 번에 돌리는 것이 가능하다.

감사합니다.