# 모델 최적화 - 그리드 서치

## 학습 내용

- 최적의 매개변수를 찾는 그리드 서치에 대해 알아본다.
- 실제 모델 의사결정트리에서 최적의 파라미터를 찾는 실습을 수행해 본다.
- 실제 모델 서포트 벡터 머신에서 최적의 파라미터를 찾는 실습을 수행해 본다.
- 그리드 서치의 다양한 결과에 대해 어떻게 파라미터를 변경해 갈지에 대해 알아본다.

## 목차



## 1-1-1 그리드 서치란 무엇인가?

목차로 이동하기

### 먼저 하이퍼 파라미터는 무엇인가?

- 하이퍼 파라미터는 **사용자가 조정할 수 있는 매개변수**이다.
- 랜덤 포레스트의 모델의 경우, 트리를 몇개로 할지, 트리의 깊이는 얼마로 할지.
- 예를 들어, 딥러닝에서는 layer의 개수, 에폭(학습 횟수) 등이 하이퍼 파라미터가 된다.

### 하이퍼 파라미터(Hyperparameter) 튜닝을 위한 다양한 방법

- Manual Search - 사용자가 직접 조절
- GridSearch - 사용자가 정한 후보군 중에 찾기
- RandomSearch - 적절한 구역에 랜덤한 수로 search
- HyperOpt - 프로그램을 이용한 하이퍼 파라미터 찾기

### 그리드 서치는 무엇인가?

- 모델에 가장 적합한 하이퍼 파리머터를 찾기
- 사용자가 지정해 준 몇가지 잠재적 Parameter들의 후보군 중에 가장 최적의 조합을 찾는다.

### 그리드 서치(Grid Search)를 하는 이유

- 가장 우수한 성능을 보이는 모델의 하이퍼 파라미터를 찾기 위해
- Grid Search로서 관심 있는 매개변수들을 대상으로 가능한 모든 조합을 시도해보는 것을 통해 모델에서 중요한 일반화 성능이 높여주는 매개변수를 찾는다.

### 그리드 서치의 단점

- 모든 경우의 수를 확인해 보기에 시간이 많이 걸린다.

## 1-1-2 간단한 Grid Search

목차로 이동하기

### 라이브러리 불러오기

In [1]:

```
from sklearn.model_selection import train_test_split
import mglearn
import sklearn
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
```

In [2]:

```
print(mglearn.__version__)
print(sklearn.__version__)
```

```
0.1.9
0.23.2
```

In [3]:

```
iris = load_iris()
X = iris.data
y = iris.target

# 데이터 셋 나누기
X_train, X_test, y_train, y_test = train_test_split(X,  y, random_state=0)

print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(112, 4) (112,) (38, 4) (38,)
```

### 모델 선택 및 학습, 그리고 평가

In [5]:

```
model = DecisionTreeClassifier(max_depth=2, min_samples_split=3)
model.fit(X_train, y_train)
print(model.score(X_train, y_train), model.score(X_test, y_test))
```

0.9642857142857143 0.8947368421052632

## 의사결정트리의 max_depth, min_samples_split 등의 매개변수를 변경해 보면 성능을 확인하기

- 모델 : DecisionTreeClassifier
- 매개변수 max_depth와 min_samples_split 매개변수

## for문을 이용한 확인

- iris 데이터 셋을 활용.
- 정확도가 높으면 해당 파라미터를 저장

In [6]:

```
# 최적의 score 변수
best_score = 0

for depth in [2,3,4,5,6]:
    for min_samples in [5,10,30,50,100]:
        # 매개변수의 각 조합에 대해 SVC를 훈련
        tree = DecisionTreeClassifier(max_depth=depth, min_samples_split=min_samples)
        tree.fit(X_train, y_train)

        # 테스트 세트로 의사결정트리를 평가
        score = tree.score(X_test, y_test)

        # 점수가 더 높으면 매개변수와 함께 기록
        if score > best_score:
            best_score = score
            best_parameters = {'max_depth':depth, 'min_samples_split':min_samples}

            print(best_score)
            print(best_parameters)

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

```
0.8947368421052632
{'max_depth': 2, 'min_samples_split': 5}
0.9736842105263158
{'max_depth': 3, 'min_samples_split': 5}
최고 점수 : 0.97
최적 매개변수 : {'max_depth': 3, 'min_samples_split': 5}
```

- 여러가지 매개변수로 많이 시도해보고 테스트 세트 정확도가 가장 높은 조합을 선택. 하지만 이 정확도는 새로운 데이터까지 이어지지 않을 수 있다. score를 높은 점수를 얻기 위한 **너무 많은 시도는 또한 과적합**이 일

어날 수 있다.

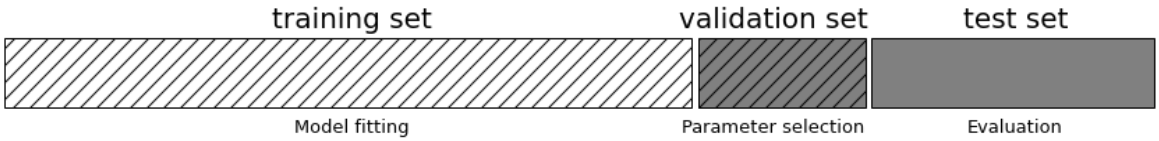
### 그렇다면 과도한 하이퍼 파라미터 튜닝에 이를 좀 더 일반화 시키기 위한 방법은 어떤 것이 있을까?

### 좀더 일반화된 모델을 만들기 위해 학습용 데이터셋을 나눠보자.

- 학습용 -> 학습 + 파라미터 튜닝 평가용
- 테스트는 그대로 마지막 평가용으로

In [7]:

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




- (설명)
  - 학습용 데이터 세트로 모델을 만들고,
  - 검증 세트로는 모델의 매개변수를 선택하고,
  - 테스트 세트로는 선택된 매개변수의 성능을 평가

## 1-1-3 Grid Search 실습 두번째

목차로 이동하기

- 이번 Grid Search에서는 먼저 학습용과 테스트용으로 분리
  - 이제 학습용 데이터 셋을 (1) 실제 학습, 학습 후, (2) 파라미터 튜닝용으로 변경

In [8]:

```
from sklearn.tree import DecisionTreeClassifier

# 학습(학습+검증), 테스트로 분할
X_trainval, X_test, y_trainval, y_test = train_test_split(iris.data,
                                                          iris.target,
                                                          random_state=0)
```

In [9]:

```
# 학습+검증(X_trainvalid, y_trainvalid)을 활용하여
# 학습, 검증로 분할
X_train, X_valid, y_train, y_valid = train_test_split(X_trainval,
                                                      y_trainval,
                                                      random_state=0)
```

- 최종적으로 데이터 셋은 학습, 검증(파라미터 튜닝용), 테스트(최종 자체 평가)으로 변경.

In [10]:

```
print("훈련 세트: {}, 검증 세트: {}, 테스트 세트: {}".format(X_train.shape[0], X_valid.shape[0], X_t
```

훈련 세트: 84, 검증 세트: 28, 테스트 세트: 38

### 데이터 셋을 3개 데이터 셋을 분리 후, 그리드 서치

In [11]:

```
# 최적의 score 변수
best_score = 0

for depth in [2,3,4,5,6]:
    for min_samples in [5,10,30,50,100]:
        # 매개변수의 각 조합에 대해 트리를 훈련
        tree = DecisionTreeClassifier(max_depth=depth, min_samples_split=min_samples)
        tree.fit(X_train, y_train)

        # 테스트 세트로 트리를 평가
        score = tree.score(X_test, y_test)

        # 점수가 더 높으면 매개변수와 함께 기록
        if score > best_score:
            best_score = score
            best_parameters = {'max_depth':depth, 'min_samples_split':min_samples}

            print(best_score)
            print(best_parameters)

# 학습용 세트와 검증 세트를 합쳐 모델을 다시 만든 후,
# 테스트 세트를 사용해 평가합니다.
tree = DecisionTreeClassifier(**best_parameters)
tree.fit(X_trainval, y_trainval)
test_score = tree.score(X_test, y_test)

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

```
0.8947368421052632
{'max_depth': 2, 'min_samples_split': 5}
0.9736842105263158
{'max_depth': 3, 'min_samples_split': 5}
검증 세트에서 최고 점수 : 0.97
최적 매개변수 :  {'max_depth': 3, 'min_samples_split': 5}
최적 매개변수에서 테스트 세트 점수 : 0.97
```

## 1-1-4 교차 검증을 사용한 그리드 서치

목차로 이동하기

In [12]:

```
import numpy as np
```

In [13]:

```
for depth in [2,3,4,5,6]:
    for min_samples in [5,10,30,50,100]:
        # 매개변수의 각 조합에 대해 트리를 훈련
        tree = DecisionTreeClassifier(max_depth=depth, min_samples_split=min_samples)

        # 교차 검증을 적용합니다.
        scores =  cross_val_score(tree, X_trainval, y_trainval, cv=5)

        # 교차 검증 정확도의 평균을 계산.
        score = np.mean(scores)

        # 점수가 더 높으면 매개변수와 함께 기록
        if score > best_score:
            best_score = score
            best_parameters = {'max_depth':depth, 'min_samples_split':min_samples}
```

In [18]:

```
# 학습용 세트와 검증 세트를 합쳐 모델을 다시 만든 후,
# 테스트 세트를 사용해 평가합니다.
tree = DecisionTreeClassifier(**best_parameters)
tree.fit(X_trainval, y_trainval)
test_score = tree.score(X_test, y_test)

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

```
최적 매개변수 :  {'max_depth': 3, 'min_samples_split': 5}
최적 매개변수에서 테스트 세트 점수 : 0.97
```

- 위의 코드와 같이 수행할 경우, 하나의 단점이라면 모델들(180개)을 학습 시키는데 걸리는 시간.

In [19]:

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

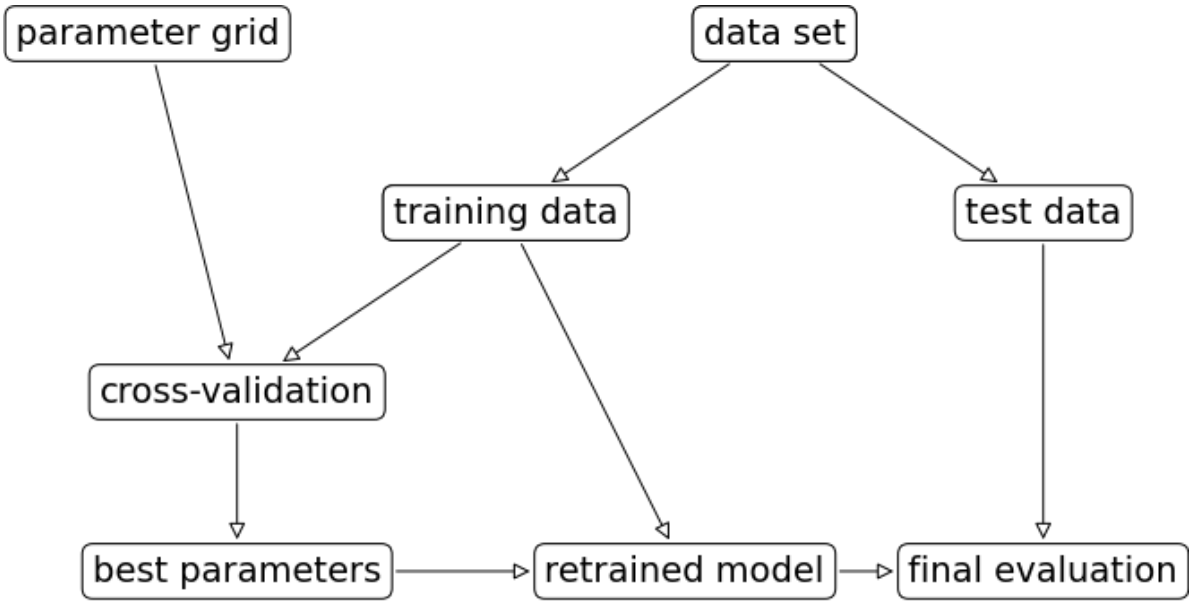
## 1-1-5 교차 검증을 사용한 그리드 서치

목차로 이동하기

- 데이터를 선택한 이후에 학습용과 테스트용으로 분류
- 학습용을 이용하여 교차검증을 수행하고, 적절한 파라미터를 선택.
- 최적의 파라미터를 지정하여 모델을 재 학습한다.
- 최종적으로 학습된 모델을 활용하여 최종 평가를 수행. 이때 test data를 사용.

In [20]:

```
mglearn.plots.plot_grid_search_overview()
```




## 1-1-6 GridSearchCV를 사용한 최적의 매개변수 찾기

목차로 이동하기

- GridSearchCV를 사용하려면 먼저 딕셔너리 형태로 검색 대상 매개변수를 지정해야 한다.
- GridSearchCV는 모든 모델을 학습 시킨다.

In [21]:

```
# 매개변수
param_grid = {'max_depth':[2,3,4,5,6],
              'min_samples_split':[5,10,30,50,100] }
print("매개변수 그리드 :₩n", param_grid)
```

```
매개변수 그리드 :
 {'max_depth': [2, 3, 4, 5, 6], 'min_samples_split': [5, 10, 30, 50, 100]}
```

### GridSearchCV를 이용한 최적의 매개변수 찾기

- GridSearchCV([모델], param_grid, cv=개수, return_train_score=True)
  - param_grid : 딕셔너리 형태의 파라미터
  - return_train_score : 기본값(False)이면 training scores에 포함되지 않는다.

In [22]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

grid_search = GridSearchCV(DecisionTreeClassifier(),
                           param_grid, cv=5, return_train_score=True)
grid_search
```

Out[22]:

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [2, 3, 4, 5, 6],
                         'min_samples_split': [5, 10, 30, 50, 100]},
             return_train_score=True)
```

In [23]:

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

In [24]:

```
# 찾기 학습
grid_search.fit(X_train, y_train)
```

Out[24]:

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [2, 3, 4, 5, 6],
                         'min_samples_split': [5, 10, 30, 50, 100]},
             return_train_score=True)
```

In [25]:

```
print("최적 매개변수 :", grid_search.best_params_)
print("최고 교차 검증 점수 : {:.4f}".format(grid_search.best_score_))
```

```
최적 매개변수 : {'max_depth': 5, 'min_samples_split': 5}
최고 교차 검증 점수 : 0.9731
```

In [26]:

```
# 훈련 후, 테스트로 적용 후, 점수
grid_search.score(X_test, y_test)
```

Out[26]:

```
0.9736842105263158
```

## 매개변수를 선택하는데 테스트 세트를 사용하지 않음!!!

## 선택한 매개변수 및 교차 검증의 정확도 확인

- 위에서의 테스트 세트로 적용한 점수와 아래의 코드의 best_score_(학습용세트)는 다른 점수이다.
- 선택한 매개변수는 best_params_ 속성에 담겨져 있다.

In [27]:

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

### 교차 검증 중, 최고 성능 모델 확인
print("최고 성능 모델 :₩n", grid_search.best_estimator_)
```

```
최적 매개변수 : {'max_depth': 5, 'min_samples_split': 5}
최고 교차 검증 점수 : 0.97
최고 성능 모델 :
 DecisionTreeClassifier(max_depth=5, min_samples_split=5)
```

## 1-1-7 교차 검증 결과 분석

목차로 이동하기

- 그리드 서치는 **연산 비용이 매우 크다.** 비교적 **간격을 넓게 하여 적은 수의 그리드로 시작**하는 것이 좋다.
- 교차 검증된 그리드 서치의 결과를 분석하여 검색 확장이 가능하다.
- 그리드 서치의 결과는 검색과 관련한 여러 정보가 함께 저장되어 있는 딕셔너리인 cv_results_속성에 담겨 있다.

In [28]:

```
import pandas as pd
pd.set_option('display.max_columns', None)

# DataFrame으로 변환
results = pd.DataFrame(grid_search.cv_results_)
results.shape
```

Out[28]:

```
(25, 22)
```

In [30]:

```
# 전체 행을 출력
display(np.transpose(results.head(25) ))
```

| | 0 | 1 | 2 | |
|---|---|---|---|---|
| **mean_fit_time** | 0 | 0.000197697 | 0 | 0.0003 |
| **std_fit_time** | 0 | 0.000395393 | 0 | 0.0004 |
| **mean_score_time** | 0.000594616 | 0.000602818 | 0.00100026 | |
| **std_score_time** | 0.000485556 | 0.000492211 | 5.80998e-06 | |
| **param_max_depth** | 2 | 2 | 2 | |
| **param_min_samples_split** | 5 | 10 | 30 | |
| **params** | {'max_depth': 2, 'min_samples_split': 5} | {'max_depth': 2, 'min_samples_split': 10} | {'max_depth': 2, 'min_samples_split': 30} | {'max_de 'min_samples |
| **split0_test_score** | 0.956522 | 0.956522 | 0.956522 | 0.9 |
| **split1_test_score** | 0.913043 | 0.913043 | 0.913043 | 0.9 |
| **split2_test_score** | 1 | 1 | 1 | |
| **split3_test_score** | 0.909091 | 0.909091 | 0.909091 | 0.9 |
| **split4_test_score** | 0.954545 | 0.954545 | 0.954545 | 0.9 |
| **mean_test_score** | 0.94664 | 0.94664 | 0.94664 | 0. |
| **std_test_score** | 0.0333049 | 0.0333049 | 0.0333049 | 0.03 |
| **rank_test_score** | 9 | 9 | 9 | |
| **split0_train_score** | 0.966292 | 0.966292 | 0.966292 | 0.9 |
| **split1_train_score** | 0.966292 | 0.966292 | 0.966292 | 0.9 |
| **split2_train_score** | 0.955556 | 0.955556 | 0.955556 | 0.9 |
| **split3_train_score** | 0.977778 | 0.977778 | 0.977778 | 0.9 |
| **split4_train_score** | 0.955556 | 0.955556 | 0.955556 | 0.9 |
| **mean_train_score** | 0.964295 | 0.964295 | 0.964295 | 0.9 |
| **std_train_score** | 0.00827669 | 0.00827669 | 0.00827669 | 0.008 |

- 각 열은 테스트를 위한 정해진 매개변수를 가진다. 각 열은 하나를 변경하여 테스트한 결과를 보여준다.
- 각 설정에 대한 교차 검증의 모든 분할의 평균값, 표준편차를 포함한 결과가 기록되어 있다.
- 매개변수 그리드가 2차원이므로 C와 gamma는 히트맵으로 시각화하기가 좋다.
- 테스트 점수의 표준편차가 작은 매개변수 조합이 더 좋다고 판단할 수 있다.

## 기타 방법

- GridSearchCV이외에 RandomizedSearchCV가 있다.
  - 이방법은 GridSearchCV보다 크게 뒤지지 않고 검색이 빠르다.

- 매개변수의 조합이 매우 많거나 규제 매개변수와 같이 연속형 값을 조정해야 할 때 널리 사용.
- RandomizedSearchCV의 매개변수 샘플링 기본 횟수는 10개이며 n_iter 옵션에서 조절 가능

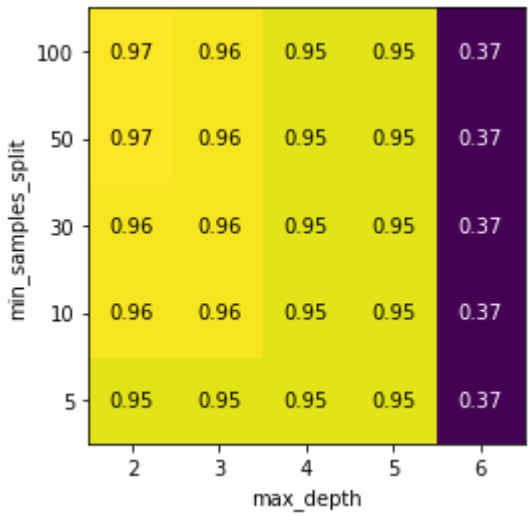
### 히트맵

In [31]:

```
scores = np.array(results.mean_test_score).reshape(5,5)

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

Out[31]:

<matplotlib.collections.PolyCollection at 0x2afcf7b32b0>




### 시각화(히트맵)를 통한 이해

- SVC는 매개변수 설정에 민감.
- 많은 매개변수는 37%부근의 낮은 정확도를 보인다.
- 어떤 설정에서는 97%이상의 만들었다.
- 높은 성능을 얻으려면 매개 변수 조정이 매우 중요하다.
- 어떻게 조정하는가에 따라 정확도가 37%에서 97%까지 차이가 난다.

## 1-1-8 매개변수의 검색 범위

목차로 이동하기

In [32]:

```
import matplotlib.pyplot as plt
from sklearn.svm import SVC
```

In [33]:

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

grid_search = GridSearchCV(SVC(),
                           param_grid, cv=5, return_train_score=True)
grid_search
```

Out[33]:

```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100],
                         'gamma': [0.001, 0.01, 0.1, 1, 10, 100]},
             return_train_score=True)
```

In [34]:

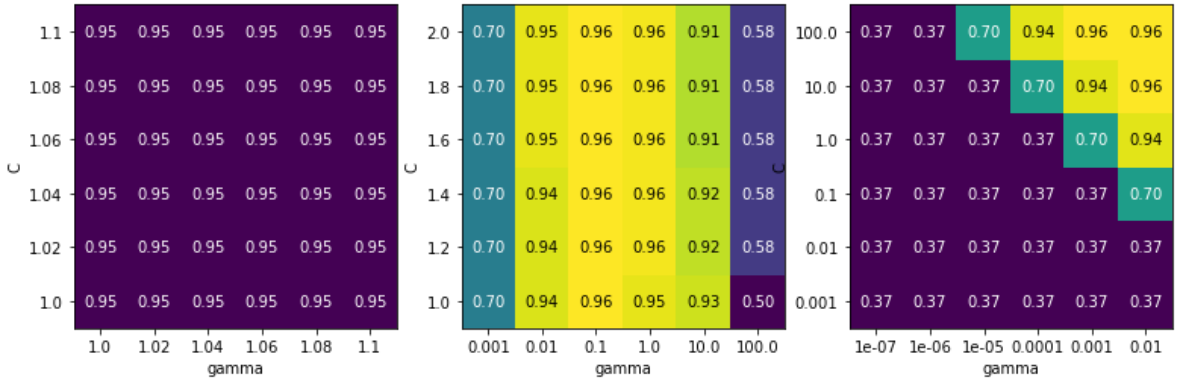
```
# 데이터 셋 나누기
X_train, X_test, y_train, y_test = train_test_split(iris.data,
                                                    iris.target,
                                                    random_state=0)

# 찾기 학습
grid_search.fit(X_train, y_train)
```

Out[34]:

```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100],
                         'gamma': [0.001, 0.01, 0.1, 1, 10, 100]},
             return_train_score=True)
```

In [35]:

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

param_grid_linear = {'C':np.linspace(1.0, 1.1, 6),
                     'gamma': np.linspace(1.0, 1.1, 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 점수 히트맵 그래프
    scores_image = mglearn.tools.heatmap(scores,
                      xlabel='gamma', xticklabels=param_grid['gamma'],
                      ylabel='C', yticklabels=param_grid['C'],
                      cmap='viridis', ax=ax
                      )
```




- 첫번째 그래프는 점수 변화가 없어 전체 매개변수 그리드가 같은 색이다.
- 두번째 그래프는 세로 띠 형태. 이는 gamma매개변수만 정확도에 영향을 준다.
  - 즉 gamma 매개변수는 적절한 범위를 탐색하고 있지만, C매개변수는 그렇지 못하든지, 아니면 중요한 매개변수가 아닐 수도 있습니다.
- 세번째 그래프는 C와 gamma 둘 모두에 따라 값이 변했다. 하지만 그래프 왼쪽 아래 영역에서는 아무런 변화가 없다.

## 적절한 방법

- 처음에는 간격을 넓게하고, 매우 극단적인 값을 적용 후, 매개변수를 바꿔가며 정확도가 변하는지 살펴보기

- history
  - 2022/08 최종 업데이트 v12