In [1]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

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

```
테스트 세트 점수: 0.88
```

In [3]:

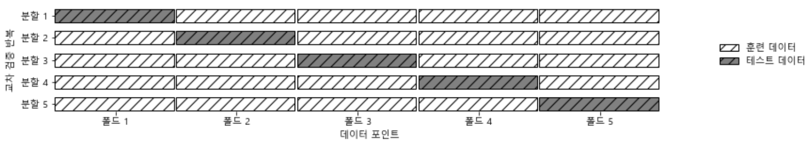
```
import mglearn
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import font_manager, rc

font_name = font_manager.FontProperties(fname="C:/Windows/Fonts/malgun.ttf").get_name()
rc('font', family=font_name)

plt.rcParams['axes.unicode_minus'] = False
```

```
C:₩Anaconda₩lib₩site-packages₩sklearn₩externals₩six.py:31: DeprecationWarning: The
module is deprecated in version 0.21 and will be removed in version 0.23 since w
e've dropped support for Python 2.7. Please rely on the official version of six (h
ttps://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
C:₩Anaconda₩lib₩site-packages₩sklearn₩externals₩joblib₩__init__.py:15: Deprecation
Warning: sklearn.externals.joblib is deprecated in 0.21 and will be removed in 0.2
3. Please import this functionality directly from joblib, which can be installed w
ith: pip install joblib. If this warning is raised when loading pickled models, yo
u may need to re-serialize those models with scikit-learn 0.21+.
  warnings.warn(msg, category=DeprecationWarning)
```

교차 검증(Cross Validation)

In [4]:

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



scikit-learn의 교차 검증

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: [0.96078431 0.92156863 0.95833333]
```

In [6]:

```
scores = cross_val_score(logreg, iris.data, iris.target, cv=5)

print("교차 검증 점수: {}".format(scores))
```

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

In [7]:

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

```
교차 검증 평균 점수: 0.96
```

계층별(Stratified) k-겹 교차 검증과 그외 전략들

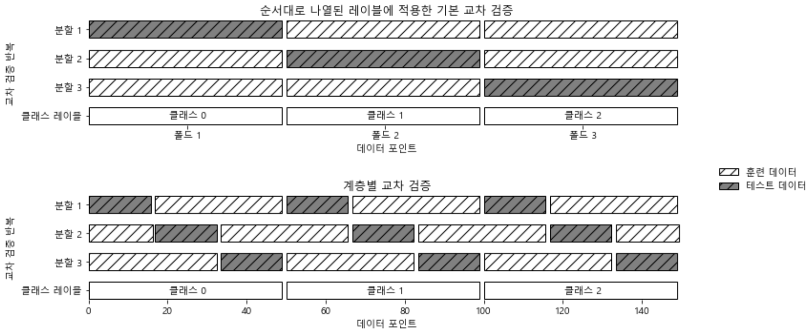
In [8]:

```
from sklearn.datasets import load_iris

iris = load_iris()

print("Iris 레이블:₩n{}".format(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 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
 2 2]
```

In [9]:

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



교차 검증 상세 옵션

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.         0.93333333 0.43333333 0.96666667 0.43333333]
```

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.]
```

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]
```

LOOCV(Leave-One-Out cross-validation)

In [14]:

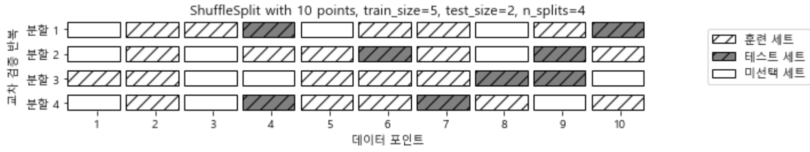
```
from sklearn.model_selection import LeaveOneOut

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

In [15]:

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



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))
```

```
교차 검증 점수:
[0.94666667 0.97333333 0.84       0.92       0.88       0.96
 0.96       0.92       0.94666667 0.96      ]
```

그룹별 교차 검증

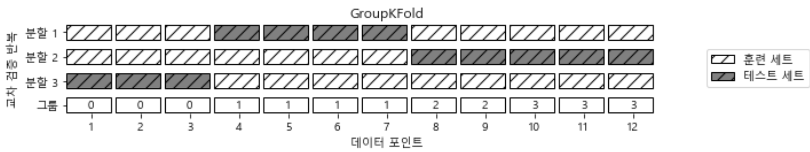
In [17]:

```
from sklearn.model_selection import GroupKFold

X, y = make_blobs(n_samples=12, random_state=0)
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        0.66666667]
```

In [18]:

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



간단한 그리드 서치

In [19]:

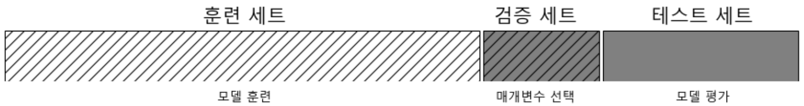
```
from sklearn.svm import SVC

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]:
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        score = svm.score(X_test, y_test)

        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: {'C': 100, 'gamma': 0.001}
```

매개변수 과대적합과 검증 세트

In [20]:

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



In [21]:

```
from sklearn.svm import SVC

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(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]:
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        score = svm.score(X_valid, y_valid)

        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

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

In [22]:

```
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        svm = SVC(gamma=gamma, C=C)
        scores = cross_val_score(svm, X_trainval, y_trainval, cv=5)
        score = np.mean(scores)

        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
```

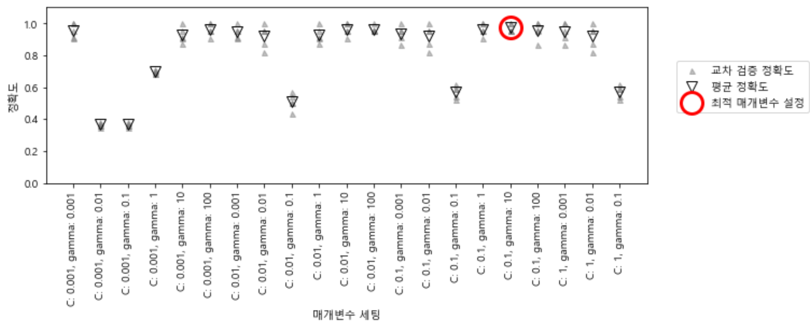
Out[22]:

```
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)
```

In [23]:

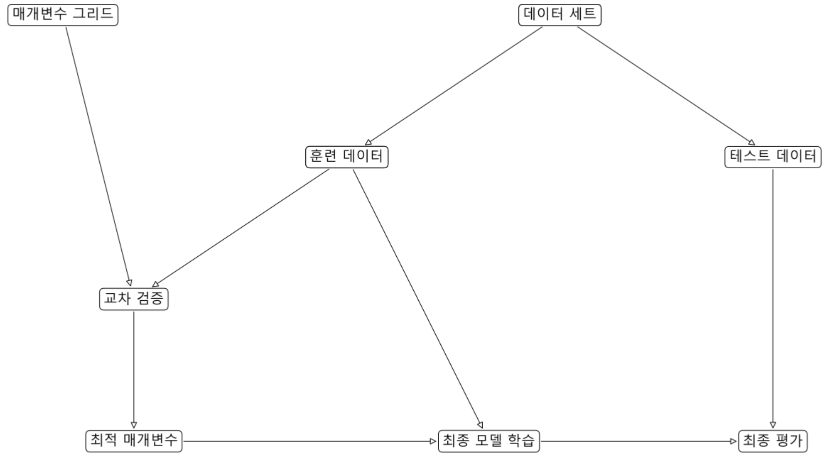
```
mglearn.plots.plot_cross_val_selection()
```

```
C:\Anaconda\lib\site-packages\sklearn\model_selection\_search.py:813: Deprecation\
arning: The default of the `iid` parameter will change from True to False in versi
on 0.22 and will be removed in 0.24. This will change numeric results when test-se
t sizes are unequal.
  DeprecationWarning)
```



In [25]:

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



In [26]:

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

In [27]:

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

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

In [28]:

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

In [29]:

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

```
C:WAnacondaWlibWsite-packagesWsklearnWmodel_selectionW_search.py:813: DeprecationW
arning: The default of the `iid` parameter will change from True to False in versi
on 0.22 and will be removed in 0.24. This will change numeric results when test-se
t sizes are unequal.
  DeprecationWarning)
```

Out[29]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='auto_deprecated', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='warn', n_jobs=None,
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100],
                         'gamma': [0.001, 0.01, 0.1, 1, 10, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [30]:

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

```
테스트 세트 점수: 0.97
```

In [31]:

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

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

In [32]:

```
print("최고 성능 모델:Wn{}".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)
```

교차 검증 결과 분석

In [33]:

```
import pandas as pd

pd.set_option('display.max_columns', None)
results = pd.DataFrame(grid_search.cv_results_)
display(np.transpose(results.head()))
```

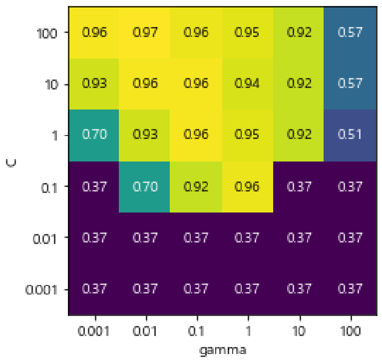
	0	1	2	3	4
mean_fit_time	0.000400209	0.000625515	0.000585175	0.000598335	0.000600529
std_fit_time	0.000490154	0.000513277	0.000478642	0.000488553	0.00049033
mean_score_time	0.000202084	0.000200176	0.000398397	0	0.000200081
std_score_time	0.000404167	0.000400352	0.000490709	0	0.000400162
param_C	0.001	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

In [34]:

```
scores = np.array(results.mean_test_score).reshape(6, 6)
mglearn.tools.heatmap(scores, xlabel='gamma', xticklabels=param_grid['gamma'], ylabel='C', ytick
labels=param_grid['C'],
                      cmap="viridis")
```

Out[34]:

```
<matplotlib.collections.PolyCollection at 0x1fef1751d30>
```



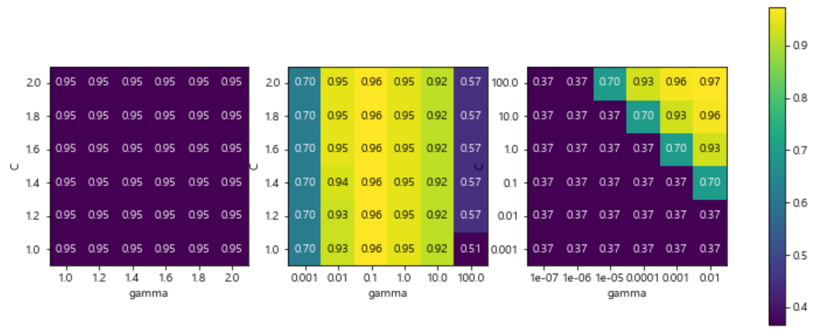
In [35]:

```
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)
    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())
```

```
C:\Anaconda\lib\site-packages\sklearn\model_selection\_search.py:813: DeprecationW
arning: The default of the `iid` parameter will change from True to False in versi
on 0.22 and will be removed in 0.24. This will change numeric results when test-se
t sizes are unequal.
  DeprecationWarning)
C:\Anaconda\lib\site-packages\sklearn\model_selection\_search.py:813: DeprecationW
arning: The default of the `iid` parameter will change from True to False in versi
on 0.22 and will be removed in 0.24. This will change numeric results when test-se
t sizes are unequal.
  DeprecationWarning)
C:\Anaconda\lib\site-packages\sklearn\model_selection\_search.py:813: DeprecationW
arning: The default of the `iid` parameter will change from True to False in versi
on 0.22 and will be removed in 0.24. This will change numeric results when test-se
t sizes are unequal.
  DeprecationWarning)
```

Out[35]:

```
<matplotlib.colorbar.Colorbar at 0x1fef20e4630>
```



비대칭 매개변수 그리드 탐색

In [36]:

```
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))
```

```
그리드 목록:
[{'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]}]
```

In [37]:

```
grid_search = GridSearchCV(SVC(), param_grid, cv=5, return_train_score=True)
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: {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
최고 교차 검증 점수: 0.97
```

```
C:₩Anaconda₩lib₩site-packages₩sklearn₩model_selection₩_search.py:813: DeprecationW
arning: The default of the `iid` parameter will change from True to False in versi
on 0.22 and will be removed in 0.24. This will change numeric results when test-se
t sizes are unequal.
  DeprecationWarning)
```

In [38]:

```
results = pd.DataFrame(grid_search.cv_results_)
display(results.T)
```

	0	1	2	3	4	
mean_fit_time	0.000400019	0.000598526	0.000400257	0.000598621	0.000598621	0.00060
std_fit_time	0.000489921	0.000488708	0.000490213	0.000488785	0.000488785	0.00049
mean_score_time	0.000600672	0	0.000202036	0.000202036	0.000200176	0.00020
std_score_time	0.000490447	0	0.000404072	0.000404072	0.000400352	0.00040
param_C	0.001	0.001	0.001	0.001	0.001	0
param_gamma	0.001	0.01	0.1	1	10	
param_kernel	rbf	rbf	rbf	rbf	rbf	
params	{'C': 0.001, 'gamma': 0.001, 'kernel': 'rbf'}	{'C': 0.001, 'gamma': 0.01, 'kernel': 'rbf'}	{'C': 0.001, 'gamma': 0.1, 'kernel': 'rbf'}	{'C': 0.001, 'gamma': 1, 'kernel': 'rbf'}	{'C': 0.001, 'gamma': 10, 'kernel': 'rbf'}	{'C': 0 'gam 100, 'ke
split0_test_score	0.375	0.375	0.375	0.375	0.375	0
split1_test_score	0.347826	0.347826	0.347826	0.347826	0.347826	0.34
split2_test_score	0.363636	0.363636	0.363636	0.363636	0.363636	0.36
split3_test_score	0.363636	0.363636	0.363636	0.363636	0.363636	0.36
split4_test_score	0.380952	0.380952	0.380952	0.380952	0.380952	0.38
mean_test_score	0.366071	0.366071	0.366071	0.366071	0.366071	0.36
std_test_score	0.0113708	0.0113708	0.0113708	0.0113708	0.0113708	0.011
rank_test_score	27	27	27	27	27	
split0_train_score	0.363636	0.363636	0.363636	0.363636	0.363636	0.36
split1_train_score	0.370787	0.370787	0.370787	0.370787	0.370787	0.37
split2_train_score	0.366667	0.366667	0.366667	0.366667	0.366667	0.36
split3_train_score	0.366667	0.366667	0.366667	0.366667	0.366667	0.36
split4_train_score	0.362637	0.362637	0.362637	0.362637	0.362637	0.36
mean_train_score	0.366079	0.366079	0.366079	0.366079	0.366079	0.36
std_train_score	0.00285176	0.00285176	0.00285176	0.00285176	0.00285176	0.0028

중첩 교차 검증

In [39]:

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

In [40]:

```
def nested_cv(X, y, inner_cv, outer_cv, Classifier, parameter_grid):
    outer_scores = []

    for training_samples, test_samples in outer_cv.split(X, y):
        best_parms = {}
        best_score = -np.inf

        for parameters in parameter_grid:
            cv_scores = []

            for inner_train, inner_test in inner_cv.split(X[training_samples], y[training_sampl
es]):
                clf = Classifier(**parameters)
                clf.fit(X[inner_train], y[inner_train])
                score = clf.score(X[inner_test], y[inner_test])
                cv_scores.append(score)
            mean_score = np.mean(cv_scores)

            if mean_score > best_score:
                best_score = mean_score
                best_params = parameters

        clf = Classifier(**best_params)
        clf.fit(X[training_samples], y[training_samples])
        outer_scores.append(clf.score(X[test_samples], y[test_samples]))

    return np.array(outer_scores)
```

In [41]:

```
from sklearn.model_selection import ParameterGrid, StratifiedKFold

scores = nested_cv(iris.data, iris.target, StratifiedKFold(5), StratifiedKFold(5), SVC, Paramete
rGrid(param_grid))

print("교차 검증 점수: {}".format(scores))
```

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

불균형 데이터셋

In [42]:

```
from sklearn.datasets import load_digits

digits = load_digits()
y = digits.target == 9
X_train, X_test, y_train, y_test = train_test_split(digits.data, y, random_state=0)
```

In [43]:

```
from sklearn.dummy import DummyClassifier

dummy_majority = DummyClassifier(strategy='most_frequent').fit(X_train, y_train)
pred_most_frequent = dummy_majority.predict(X_test)

print("예측된 레이블의 고유값: {}".format(np.unique(pred_most_frequent)))
print("테스트 점수: {:.2f}".format(dummy_majority.score(X_test, y_test)))
```

```
예측된 레이블의 고유값: [False]
테스트 점수: 0.90
```

In [44]:

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(max_depth=2).fit(X_train, y_train)
pred_tree = tree.predict(X_test)

print("테스트 점수: {:.2f}".format(tree.score(X_test, y_test)))
```

```
테스트 점수: 0.92
```

In [45]:

```
from sklearn.linear_model import LogisticRegression

dummy = DummyClassifier().fit(X_train, y_train)
pred_dummy = dummy.predict(X_test)

print("dummy 점수: {:.2f}".format(dummy.score(X_test, y_test)))

logreg = LogisticRegression(C=0.1).fit(X_train, y_train)
pred_logreg = logreg.predict(X_test)

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

```
dummy 점수: 0.82
logreg 점수: 0.98
```

오차 행렬(Confusion matrices)

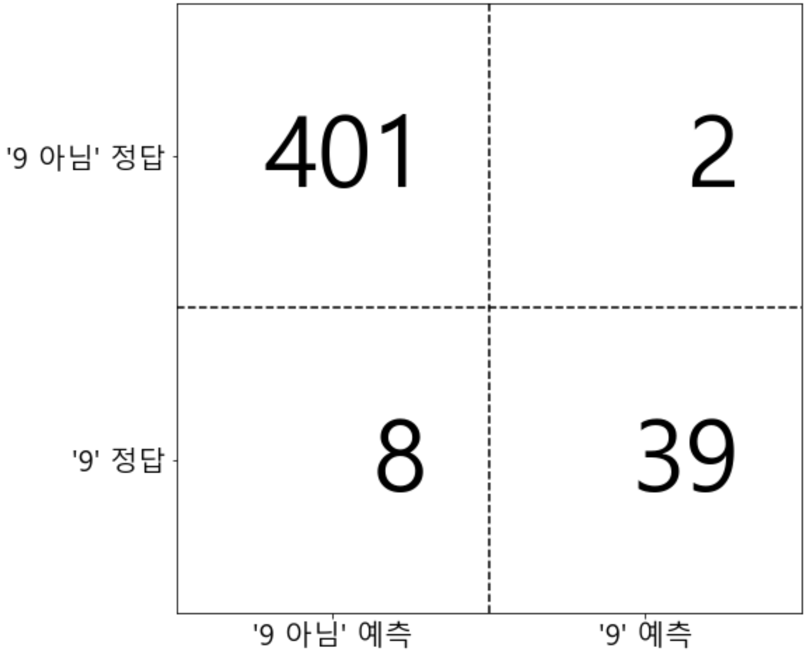
In [46]:

```
from sklearn.metrics import confusion_matrix

confusion = confusion_matrix(y_test, pred_logreg)

print("오차 행렬:\n{}".format(confusion))
```

```
오차 행렬:
[[401   2]
 [  8  39]]
```

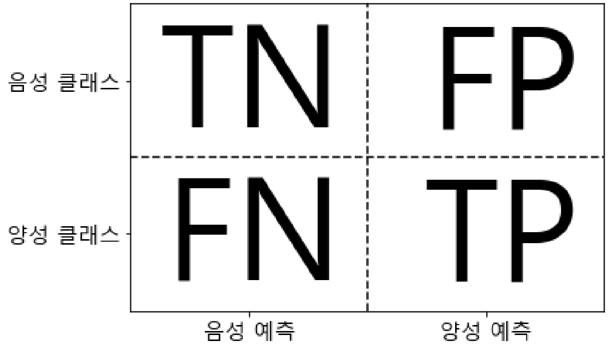
In [47]:

```
mglearn.plots.plot_confusion_matrix_illustration()
```



In [48]:

```
mglearn.plots.plot_binary_confusion_matrix()
```



In [49]:

```
print("빈도 기반 더미 모델:")
print(confusion_matrix(y_test, pred_most_frequent))
print("\n무작위 더미 모델:")
print(confusion_matrix(y_test, pred_dummy))
print("\n결정 트리:")
print(confusion_matrix(y_test, pred_tree))
print("\n로지스틱 회귀")
print(confusion_matrix(y_test, pred_logreg))
```

```
빈도 기반 더미 모델:
[[403   0]
 [ 47   0]]

무작위 더미 모델:
[[369  34]
 [ 43   4]]

결정 트리:
[[390  13]
 [ 24  23]]

로지스틱 회귀
[[401   2]
 [  8  39]]
```

In [50]:

```
from sklearn.metrics import f1_score

print("빈도 기반 더미 모델의 f1 score: {:.2f}".format(f1_score(y_test, pred_most_frequent)))
print("무작위 더미 모델의 f1 score: {:.2f}".format(f1_score(y_test, pred_dummy)))
print("트리 모델의 f1 score: {:.2f}".format(f1_score(y_test, pred_tree)))
print("로지스틱 회귀 모델의 f1 score: {:.2f}".format(f1_score(y_test, pred_logreg)))
```

```
빈도 기반 더미 모델의 f1 score: 0.00
무작위 더미 모델의 f1 score: 0.09
트리 모델의 f1 score: 0.55
로지스틱 회귀 모델의 f1 score: 0.89
```

In [51]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_most_frequent, target_names=["9 아님", "9"]))
```

```
              precision    recall  f1-score   support

        9 아님       0.90      1.00      0.94       403
           9       0.00      0.00      0.00        47

    accuracy                           0.90       450
   macro avg       0.45      0.50      0.47       450
weighted avg       0.80      0.90      0.85       450
```

In [52]:

```
print(classification_report(y_test, pred_dummy, target_names=["9 아님", "9"]))
```

```
              precision    recall  f1-score   support

        9 아님       0.90      0.92      0.91       403
           9       0.11      0.09      0.09        47

    accuracy                           0.83       450
   macro avg       0.50      0.50      0.50       450
weighted avg       0.81      0.83      0.82       450
```

In [53]:

```
print(classification_report(y_test, pred_logreg, target_names=["9 아님", "9"]))
```

```
              precision    recall  f1-score   support

        9 아님       0.98      1.00      0.99       403
           9       0.95      0.83      0.89        47

    accuracy                           0.98       450
   macro avg       0.97      0.91      0.94       450
weighted avg       0.98      0.98      0.98       450
```

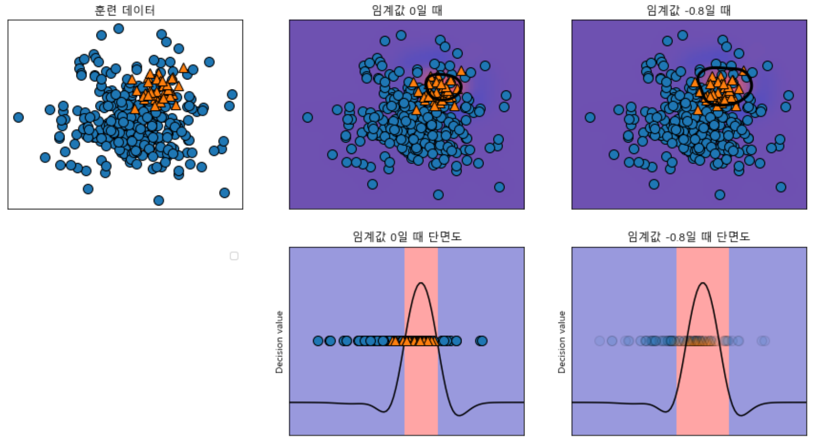
불확실성 고려

In [54]:

```
from mglearn.datasets import make_blobs

X, y = make_blobs(n_samples=(400, 50), centers=2, cluster_std=[7.0, 2], random_state=22)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
svc = SVC(gamma=.05).fit(X_train, y_train)
```

In [55]:

```
mglearn.plots.plot_decision_threshold()
```



In [56]:

```
print(classification_report(y_test, svc.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.97      0.89      0.93       104
           1       0.35      0.67      0.46         9

    accuracy                           0.88       113
   macro avg       0.66      0.78      0.70       113
weighted avg       0.92      0.88      0.89       113
```

In [57]:

```
y_pred_lower_threshold = svc.decision_function(X_test) > -.8
```

In [58]:

```
print(classification_report(y_test, y_pred_lower_threshold))
```

```
              precision    recall  f1-score   support

           0       1.00      0.82      0.90       104
           1       0.32      1.00      0.49         9

    accuracy                           0.83       113
   macro avg       0.66      0.91      0.69       113
weighted avg       0.95      0.83      0.87       113
```

정밀도-재현율 곡선과 ROC 곡선

In [59]:

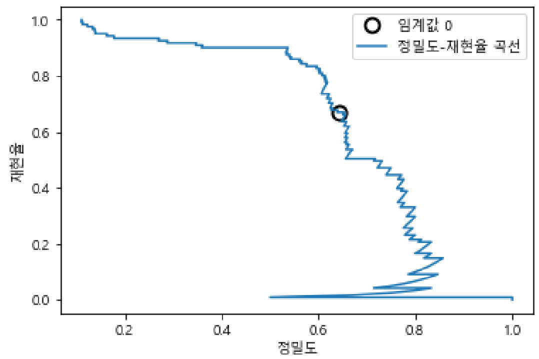
```
from sklearn.metrics import precision_recall_curve

precision, recall, thresholds = precision_recall_curve(y_test, svc.decision_function(X_test))
```

In [60]:

```
X, y = make_blobs(n_samples=(4000, 500), centers=2, cluster_std=[7.0, 2], random_state=22)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
svc = SVC(gamma=.05).fit(X_train, y_train)
precision, recall, thresholds = precision_recall_curve( y_test, svc.decision_function(X_test))
close_zero = np.argmin(np.abs(thresholds))

plt.plot(precision[close_zero], recall[close_zero], 'o', markersize=10, label="임계값 0", fillst
yle="none", c='k', mew=2)
plt.plot(precision, recall, label="정밀도-재현율 곡선")
plt.xlabel("정밀도")
plt.ylabel("재현율")
plt.legend(loc="best")
```

Out[60]:

<matplotlib.legend.Legend at 0x1fef15a6780>

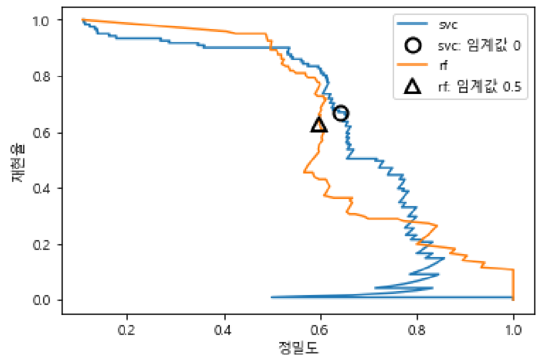


In [61]:

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=0, max_features=2)
rf.fit(X_train, y_train)
precision_rf, recall_rf, thresholds_rf = precision_recall_curve(y_test, rf.predict_proba(X_test)
[:, 1])

plt.plot(precision, recall, label="svc")
plt.plot(precision[close_zero], recall[close_zero], 'o', markersize=10, label="svc: 임계값 0", f
illstyle="none", c='k', mew=2)
plt.plot(precision_rf, recall_rf, label="rf")

close_default_rf = np.argmin(np.abs(thresholds_rf - 0.5))

plt.plot(precision_rf[close_default_rf], recall_rf[close_default_rf], '^', c='k', markersize=10,
label="rf: 임계값 0.5",
         fillstyle="none", mew=2)
plt.xlabel("정밀도")
plt.ylabel("재현율")
plt.legend(loc="best")
```

Out[61]:

<matplotlib.legend.Legend at 0x1fef1b46940>



In [62]:

```
print("랜덤 포레스트의 f1_score: {:.3f}".format(f1_score(y_test, rf.predict(X_test))))
print("svc의 f1_score: {:.3f}".format(f1_score(y_test, svc.predict(X_test))))
```

```
랜덤 포레스트의 f1_score: 0.610
svc의 f1_score: 0.656
```

In [63]:

```
from sklearn.metrics import average_precision_score

ap_rf = average_precision_score(y_test, rf.predict_proba(X_test)[:, 1])
ap_svc = average_precision_score(y_test, svc.decision_function(X_test))

print("랜덤 포레스트의 평균 정밀도: {:.3f}".format(ap_rf))
print("svc의 평균 정밀도: {:.3f}".format(ap_svc))
```

```
랜덤 포레스트의 평균 정밀도: 0.660
svc의 평균 정밀도: 0.666
```

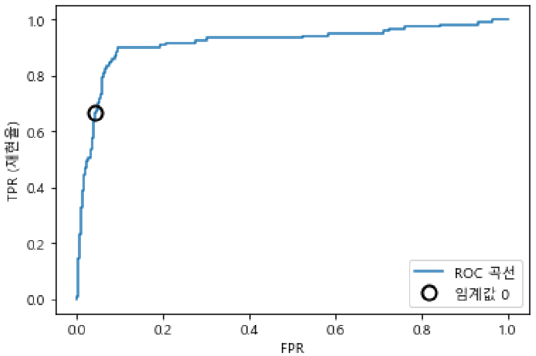
ROC 와 AUC

In [64]:

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, svc.decision_function(X_test))

plt.plot(fpr, tpr, label="ROC 곡선")
plt.xlabel("FPR")
plt.ylabel("TPR (재현율)")

close_zero = np.argmin(np.abs(thresholds))

plt.plot(fpr[close_zero], tpr[close_zero], 'o', markersize=10, label="임계값 0", fillstyle="none", c='k', mew=2)
plt.legend(loc=4)
```

Out[64]:

```
<matplotlib.legend.Legend at 0x1fef1b9dc50>
```

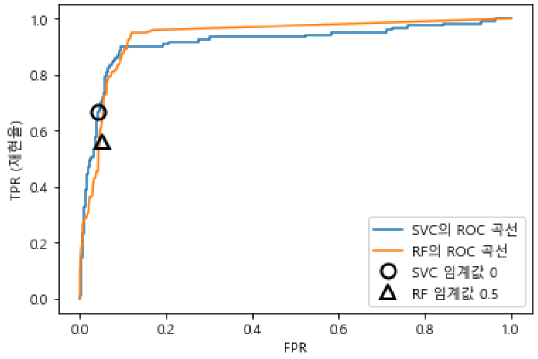


In [65]:

```
from sklearn.metrics import roc_curve

fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, rf.predict_proba(X_test)[:, 1])

plt.plot(fpr, tpr, label="SVC의 ROC 곡선")
plt.plot(fpr_rf, tpr_rf, label="RF의 ROC 곡선")
plt.xlabel("FPR")
plt.ylabel("TPR (재현율)")
plt.plot(fpr[close_zero], tpr[close_zero], 'o', markersize=10, label="SVC 임계값 0", fillstyle="none", c='k', mew=2)

close_default_rf = np.argmin(np.abs(thresholds_rf - 0.5))

plt.plot(fpr_rf[close_default_rf], tpr[close_default_rf], '^', markersize=10, label="RF 임계값 0.5", fillstyle="none", c='k', mew=2)
plt.legend(loc=4)
```

Out[65]:

```
<matplotlib.legend.Legend at 0x1fef1c365f8>
```



In [66]:

```
from sklearn.metrics import roc_auc_score

rf_auc = roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1])
svc_auc = roc_auc_score(y_test, svc.decision_function(X_test))

print("랜덤 포레스트의 AUC: {:.3f}".format(rf_auc))
print("SVC의 AUC: {:.3f}".format(svc_auc))
```

```
랜덤 포레스트의 AUC: 0.937
SVC의 AUC: 0.916
```

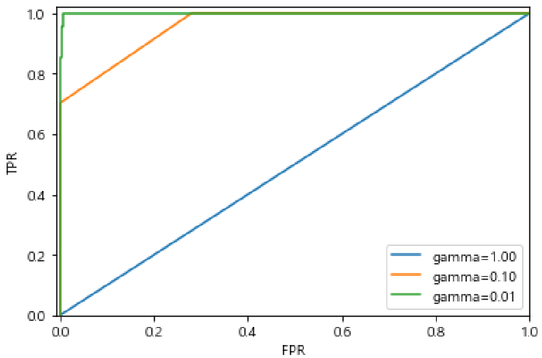
In [67]:

```
y = digits.target == 9
X_train, X_test, y_train, y_test = train_test_split(digits.data, y, random_state=0)
plt.figure()

for gamma in [1, 0.1, 0.01]:
    svc = SVC(gamma=gamma).fit(X_train, y_train)
    accuracy = svc.score(X_test, y_test)
    auc = roc_auc_score(y_test, svc.decision_function(X_test))
    fpr, tpr, _ = roc_curve(y_test , svc.decision_function(X_test))

    print("gamma = {:.2f}  정확도 = {:.2f}  AUC = {:.2f}".format(gamma, accuracy, auc))
    plt.plot(fpr, tpr, label="gamma={:.2f}".format(gamma))

plt.xlabel("FPR")
plt.ylabel("TPR")
plt.xlim(-0.01, 1)
plt.ylim(0, 1.02)
plt.legend(loc="best")
```

```
gamma = 1.00  정확도 = 0.90  AUC = 0.50
gamma = 0.10  정확도 = 0.90  AUC = 0.96
gamma = 0.01  정확도 = 0.90  AUC = 1.00
```

Out[67]:

<matplotlib.legend.Legend at 0x1fef1a54438>



다중 분류의 평가 지표

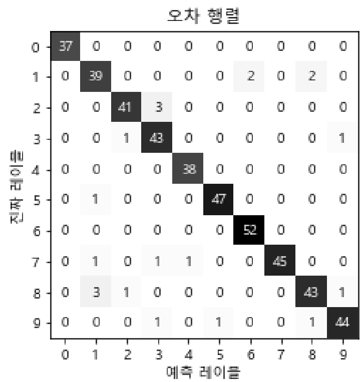
In [68]:

```
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, random_state=0)
lr = LogisticRegression().fit(X_train, y_train)
pred = lr.predict(X_test)

print("정확도: {:.3f}".format(accuracy_score(y_test, pred)))
print("오차 행렬:₩n{}".format(confusion_matrix(y_test, pred)))
```

```
정확도: 0.953
오차 행렬:
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 39  0  0  0  0  2  0  2  0]
 [ 0  0 41  3  0  0  0  0  0  0]
 [ 0  0  1 43  0  0  0  0  0  1]
 [ 0  0  0  0 38  0  0  0  0  0]
 [ 0  1  0  0  0 47  0  0  0  0]
 [ 0  0  0  0  0  0 52  0  0  0]
 [ 0  1  0  1  1  0  0 45  0  0]
 [ 0  3  1  0  0  0  0  0 43  1]
 [ 0  0  0  1  0  1  0  0  1 44]]
```

In [69]:

```
scores_image = mglearn.tools.heatmap(confusion_matrix(y_test, pred), xlabel='예측 레이블', ylabe
l='진짜 레이블',
                                     xticklabels=digits.target_names, yticklabels=digits.target_
names,
                                     cmap=plt.cm.gray_r, fmt="%d")

plt.title("오차 행렬")
plt.gca().invert_yaxis()
```



In [70]:

```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        37
           1       0.89      0.91      0.90        43
           2       0.95      0.93      0.94        44
           3       0.90      0.96      0.92        45
           4       0.97      1.00      0.99        38
           5       0.98      0.98      0.98        48
           6       0.96      1.00      0.98        52
           7       1.00      0.94      0.97        48
           8       0.93      0.90      0.91        48
           9       0.96      0.94      0.95        47

    accuracy                           0.95       450
   macro avg       0.95      0.95      0.95       450
weighted avg       0.95      0.95      0.95       450
```

In [71]:

```
print("micro 평균 f1 점수: {:.3f}".format(f1_score(y_test, pred, average="micro")))
print("macro 평균 f1 점수: {:.3f}".format(f1_score(y_test, pred, average="macro")))
```

```
micro 평균 f1 점수: 0.953
macro 평균 f1 점수: 0.954
```

모델 선택에서 평가 지표 사용하기

In [72]:

```
print("기본 평가 지표: {}".format(cross_val_score(SVC(), digits.data, digits.target == 9)))

explicit_accuracy =  cross_val_score(SVC(), digits.data, digits.target == 9, scoring="accuracy")

print("정확도 지표: {}".format(explicit_accuracy))

roc_auc =  cross_val_score(SVC(), digits.data, digits.target == 9, scoring="roc_auc")

print("AUC 지표: {}".format(roc_auc))
```

```
기본 평가 지표: [0.89983306 0.89983306 0.89983306]
정확도 지표: [0.89983306 0.89983306 0.89983306]
AUC 지표: [0.99372294 0.98957947 0.99594929]
```

In [73]:

```
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target == 9, random_stat
e=0)

param_grid = {'gamma': [0.0001, 0.01, 0.1, 1, 10]}
grid = GridSearchCV(SVC(), param_grid=param_grid)
grid.fit(X_train, y_train)

print("정확도 지표를 사용한 그리드 서치")
print("최적의 파라미터:", grid.best_params_)
print("최상의 교차 검증 점수 (정확도)): {:.3f}".format(grid.best_score_))
print("테스트 세트 AUC: {:.3f}".format(roc_auc_score(y_test, grid.decision_function(X_test))))
print("테스트 세트 accuracy: {:.3f}".format(grid.score(X_test, y_test)))
```

```
정확도 지표를 사용한 그리드 서치
최적의 파라미터: {'gamma': 0.0001}
최상의 교차 검증 점수 (정확도)): 0.970
테스트 세트 AUC: 0.992
테스트 세트 accuracy: 0.973
```

In [74]:

```
grid = GridSearchCV(SVC(), param_grid=param_grid, scoring="roc_auc")
grid.fit(X_train, y_train)

print("AUC 지표를 사용한 그리드 서치")
print("최적의 파라미터:", grid.best_params_)
print("최상의 교차 검증 점수 (AUC): {:.3f}".format(grid.best_score_))
print("테스트 세트 AUC: {:.3f}".format(grid.score(X_test, y_test)))
```

```
AUC 지표를 사용한 그리드 서치
최적의 파라미터: {'gamma': 0.01}
최상의 교차 검증 점수 (AUC): 0.997
테스트 세트 AUC: 1.000
```

In [75]:

```
from sklearn.metrics.scorer import SCORERS

print("가능한 평가 방식:\n{}".format(sorted(SCORERS.keys())))
```

```
가능한 평가 방식:
['accuracy', 'adjusted_mutual_info_score', 'adjusted_rand_score', 'average_precisi
on', 'balanced_accuracy', 'brier_score_loss', 'completeness_score', 'explained_var
iance', 'f1', 'f1_macro', 'f1_micro', 'f1_samples', 'f1_weighted', 'fowlkes_mallow
s_score', 'homogeneity_score', 'jaccard', 'jaccard_macro', 'jaccard_micro', 'jacca
rd_samples', 'jaccard_weighted', 'max_error', 'mutual_info_score', 'neg_log_loss',
'neg_mean_absolute_error', 'neg_mean_squared_error', 'neg_mean_squared_log_error',
'neg_median_absolute_error', 'normalized_mutual_info_score', 'precision', 'precisi
on_macro', 'precision_micro', 'precision_samples', 'precision_weighted', 'r2', 're
call', 'recall_macro', 'recall_micro', 'recall_samples', 'recall_weighted', 'roc_a
uc', 'v_measure_score']
```

In [ ]:

```
```