

평가 지표 및 측정

1.1.1 최종 목표를 기억하라

1.1.2 이진 분류의 평가지표

1.1.3 다중 분류의 평가지표

1.1.4 회귀의 평가 지표

학습 내용

- 불균형 데이터 셋을 만들고 모델에 따른 정확도를 확인해 본다.
 - 학습을 시키지 않아도, 학습을 시켜도 불균형 데이터는 기본적으로 성능이 좋을 수 있다.

In [1]:

```
from IPython.display import display, Image
```

In [2]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
```

불균형 데이터 셋

- 손글씨 데이터
- data : 1797장, 64개의 pixel 데이터
 - images : 1797, 8, 8
- target : 0~9까지의 손글씨 값

In [6]:



```
from sklearn.datasets import load_digits

digits = load_digits()
print(digits.data.shape)
print(digits.keys(), digits.target)
print(np.unique( digits.target ) )
sns.countplot(digits.target)
```

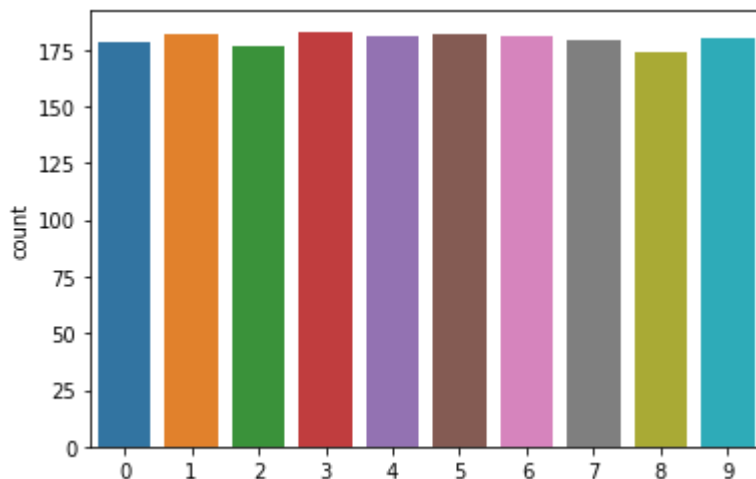
```
(1797, 64)
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR']) [0 1 2 ... 8 9 8]
[0 1 2 3 4 5 6 7 8 9]
```

C:\Users\Wfront\Anaconda3\lib\site-packages\seaborn\decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```

Out[6]:

```
<AxesSubplot:ylabel='count'>
```



타깃이 9:1의 비율을 갖도록 하기

- 9이면 True
- 9가 아니면 False

In [8]:



```
X = digits.data
y = digits.target == 9

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

In [9]:



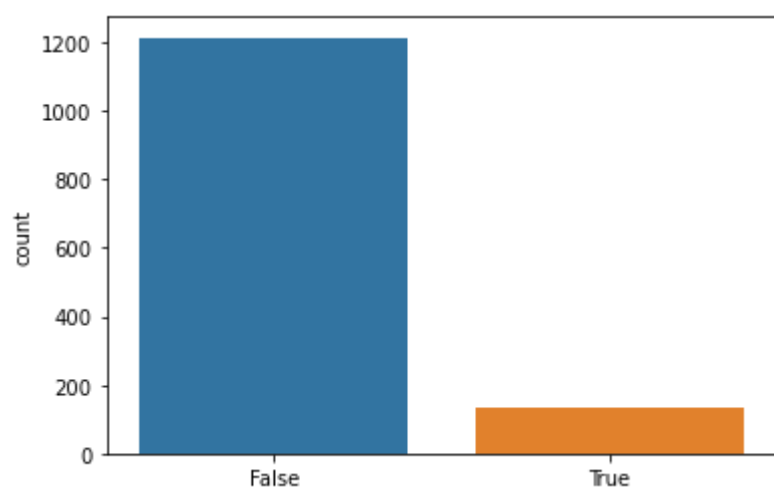
```
sns.countplot(y_train)
```

C:\Users\Wfront\Anaconda3\lib\site-packages\seaborn\decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```

Out[9]:

<AxesSubplot:ylabel='count'>



In [10]:

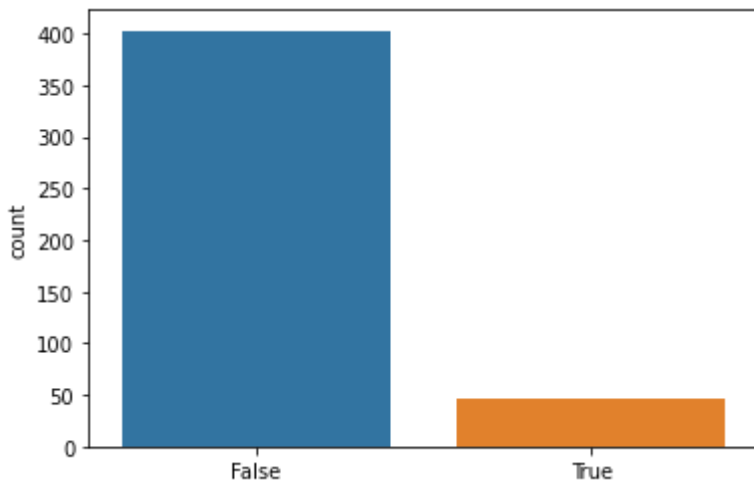
```
sns.countplot(y_test)
```

C:\Users\Wfront\Anaconda3\lib\site-packages\seaborn\decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```

Out[10]:

<AxesSubplot:ylabel='count'>



기본 모델 DummyClassifier를 사용한 정확도 계산

- 간단한 룰을 사용하여 예측을 수행한다.
- 실제 문제에서는 사용하지 않으며, 간단한 베이스라인 모델로서 사용된다.
- DummyClassifier(strategy='most_frequent') : 학습용 세트에서 가장 빈번한 라벨을 예측한다.
 - stratified : 클래스 분포를 존중하여 예측을 생성
 - uniform : 무작위로 균일하게 예측을 생성, 기타 : prior, constant
- 아래 모델은 가장 많은 레이블을 가진 False만 예측하게 된다.

In [12]:

```
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

기본 stratified : 클래스 분포를 가만하여 예측

- 클래스의 9:1 분포를 가만하여 예측

In [25]:

```
dummy = DummyClassifier().fit(X_train, y_train)
pred_dummy = dummy.predict(X_test)
print("예측된 레이블의 고유값: {}".format(np.unique(pred_dummy)))
print("dummy 점수: {:.2f}".format(dummy.score(X_test, y_test)))
```

예측된 레이블의 고유값: [False True]

dummy 점수: 0.81

실제 분류기를 사용해보기 - DecisionTreeClassifier

In [26]:

```
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

- 실제 분류기와 기본적인 모델 dummy 분류기와 성능차이가 거의 없다.

LogisticRegression

In [27]:

```
from sklearn.linear_model import LogisticRegression

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)))
```

logreg 점수: 0.98

C:\Users\Wfront\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

n_iter_i = _check_optimize_result(

하나만 예측하는 기본 모델도 90% 이상의 정확도를 갖는다.

- 정확도는 때로는 평가지표로 사용하기에 부족한 부분이 있다.

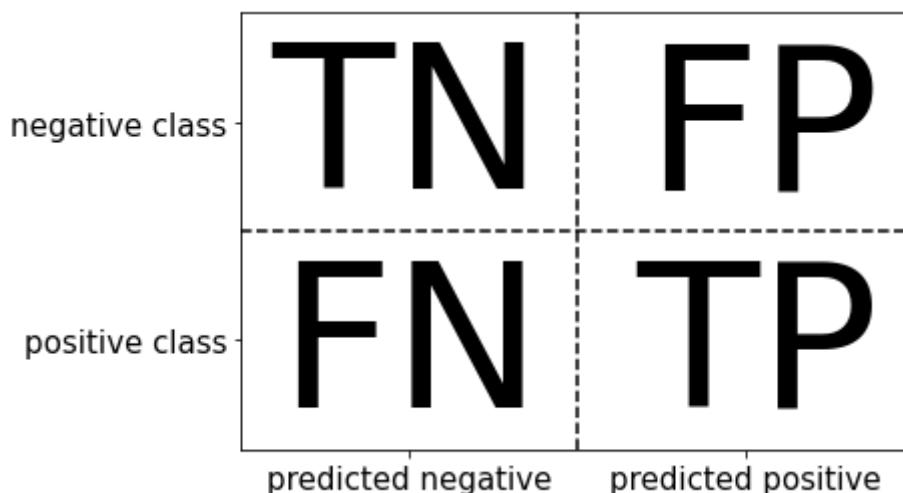
오차행렬(confusion matrix)을 이용하기

In [28]:

```
import mglearn
```

In [29]:

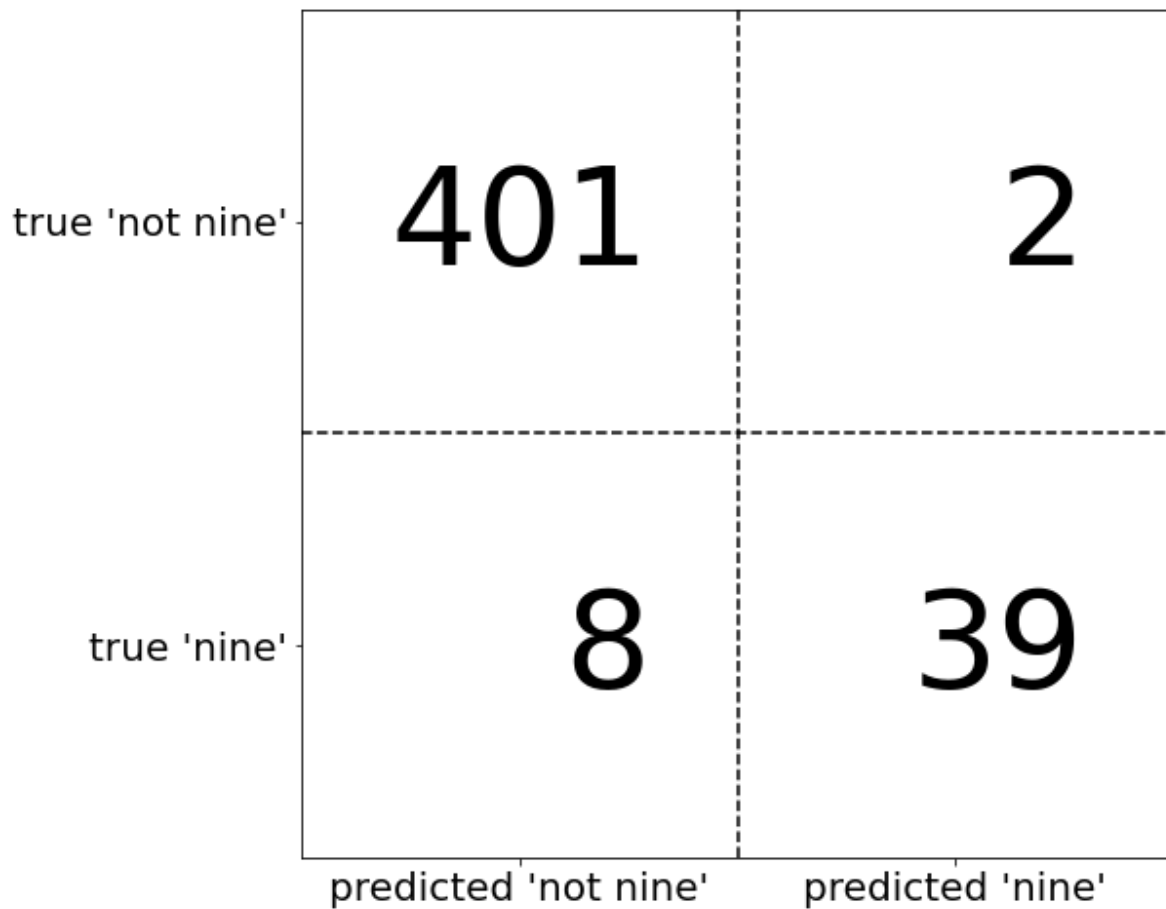
```
mglearn.plots.plot_binary_confusion_matrix()
```



In [30]:



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



In [31]:



```
from sklearn.metrics import confusion_matrix  
  
confusion = confusion_matrix(y_test, pred_logreg)  
print("오차 행렬:\n{}".format(confusion))
```

오차 행렬:

```
[[402  1]  
 [ 6 41]]
```

각각의 예측값에 대한 오차행렬을 확인해보기

In [33]:

```

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:

```

[[359 44]
 [ 41  6]]

```

결정 트리:

```

[[390 13]
 [ 24 23]]

```

로지스틱 회귀

```

[[402  1]
 [  6 41]]

```

분류의 평가지표를 살펴보자.

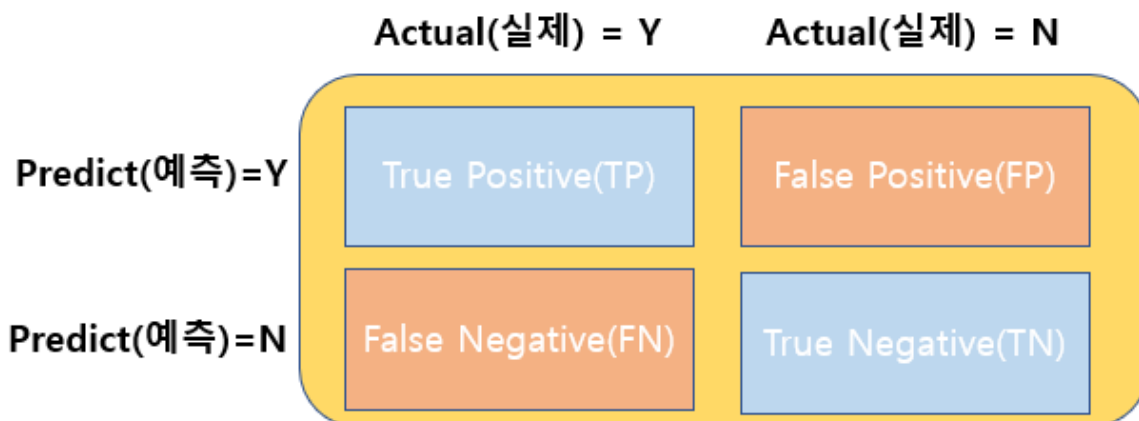
- 정확도(accuracy) : 정확하게 예측/전체 예측수

In [34]:

```

## 머신러닝 작업 flow
display(Image(filename='img/model_validation01.png'))

```



분류의 평가지표를 살펴보자.

정확도(accuracy) : 정확하게 예측/전체 예측수

$$\text{accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

정밀도(precision) : 예측을 양성(Positive)으로 한것 전체-TP+FP중에 맞는 것 (T*)

$$\text{정밀도(precision)} = \frac{\text{잘 예측(TP)}}{\text{예측을 양성으로 한 것 전체(TP+FP)}}$$

- 언제 사용하는가? : 거짓 양성(FP)의 수를 줄일 때 사용
 - 약이 효과 있다고 예측(positive), 하지만 예측 결과가 틀림
 - 암이 아닌데 암이라고 예측

민감도(sensitivity), 재현율(recall, TPRate)

- 실제 데이터의 양성 데이터(TP + FN)중에 얼마나 많은 샘플을 양성으로 잘 분류했나?(TP)
- $TP / (TP + FN)$

$$\text{민감도(recall, 재현율)} = \frac{\text{잘 예측(TP)}}{\text{실제 값이 양성인것 전체(TP+FN)}}$$

- 재현율이 높아지면 FP는 상대적으로 낮아짐.
- 언제 사용? FN(음성 예측. 잘못 예측함.)을 줄일 때, 성능 지표로 사용합니다.
- (암인데 음성(아니라고-Negative)예측하여, 실수. 엄청난 실수 **실제 암인데(Positive)인데,
 - 이를 병원에서 암이 아니다(Negative)**로 예측하면 얼마나 큰일인가?
- 다른 말로 **민감도(sensitivity), 적중률(hit rate), 진짜 양성 비율(TPR)**이라고 합니다.
- 따라서 병원의 암 예측 같은 경우는 FN를 최소화시켜 재현율을 줄이면, 상대적으로 정밀도를 최대화된다.

특이도

- 실제 데이터의 음성 데이터(FP + TN)중에 얼마나 많은 샘플을 잘 분류했나?(TN)
- $TN / (FP + TN)$

$$\text{특이도} = \frac{\text{잘 예측(TN)}}{\text{실제 값이 음성인것 전체(FP + TN)}}$$

FPRate

- 실제 데이터의 음성 데이터(FP + TN)중에 잘 분류하지 못한 것?(FP)
- $FP / (FP + TN)$

$$\text{FPRate} = \frac{\text{틀린 예측(FP)}}{\text{실제 값이 음성인것 전체(FP + TN)}}$$

다양한 분류 측정 방법

- https://en.wikipedia.org/wiki/Sensitivity_and_specificity
(https://en.wikipedia.org/wiki/Sensitivity_and_specificity).
- 이진 분류에서는 정밀도와 재현율을 가장 많이 사용.
 - 분야마다 다른 지표를 사용할 수 있다.

F-score

- 정밀도와 민감도(recall, 재현율)을 하나만 가지고 측정이 안된다. 정밀도(precision)와 재현율(recall)의 조화 평균인 f-점수 또는 f-측정은 이 둘을 하나로 요약해 줍니다.

$$F = 2 \times (\text{정밀도} \times \text{재현율}) / (\text{정밀도} + \text{재현율})$$

- 다른 말로 F1-score라고 한다.

각각의 모델 예측값을 f1-score로 예측

In [35]:



```
from sklearn.metrics import f1_score

# Dummy분류
print("무작위 더미 모델의 f1 score: {:.2f}".format(f1_score(y_test, pred_dummy)))

# 의사결정트리
print("트리 모델의 f1 score: {:.2f}".format(f1_score(y_test, pred_tree)))

# 로지스틱
print("로지스틱 회귀 모델의 f1 score: {:.2f}".format(
    f1_score(y_test, pred_logreg)))
```

무작위 더미 모델의 f1 score: 0.12
 트리 모델의 f1 score: 0.55
 로지스틱 회귀 모델의 f1 score: 0.92

f1-score를 요약해서 보여주기

- `classification_report()`: 정밀도, 재현율, f1-score를 모두 한번에 계산
- `support`는 단순히 샘플의 수

In [40]:



```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_most_frequent))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| False | 0.90 | 1.00 | 0.94 | 403 |
| True | 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 |

dummyClassifier 모델

In [41]:



```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_dummy,
                           target_names=["not 9", "is 9"]))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| not 9 | 0.90 | 0.89 | 0.89 | 403 |
| is 9 | 0.12 | 0.13 | 0.12 | 47 |
| accuracy | | | 0.81 | 450 |
| macro avg | 0.51 | 0.51 | 0.51 | 450 |
| weighted avg | 0.82 | 0.81 | 0.81 | 450 |

로지스틱 회귀

In [42]:



```
from sklearn.metrics import classification_report
print(classification_report(y_test,
                           pred_logreg,
                           target_names=["not 9", "is 9"]))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| not 9 | 0.99 | 1.00 | 0.99 | 403 |
| is 9 | 0.98 | 0.87 | 0.92 | 47 |
| accuracy | | | 0.98 | 450 |
| macro avg | 0.98 | 0.93 | 0.96 | 450 |
| weighted avg | 0.98 | 0.98 | 0.98 | 450 |

이진 분류 - 예측을 0,1로 하는 것이 아니라 확률로 해보기

- 400개(음성), 50개(양성) 으로 이루어진 불균형 데이터
- 사용 함수 : `decision_function()`, `predict_proba()`
 - `decision_function`을 0으로, `predict_proba`를 0.5의 임계값으로 사용

In [48]:

```
from mglearn.datasets import make_blobs
X, y = make_blobs(n_samples=(400, 50),
                  centers=2, cluster_std=[7.0, 2],
                  random_state=22)

print(X.shape, y.shape)
```

(450, 2) (450,)

C:\Users\Wfront\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:86: FutureWarning: Function make_blobs is deprecated; Please import make_blobs directly from sci-kit-learn
 warnings.warn(msg, category=FutureWarning)

In [44]:

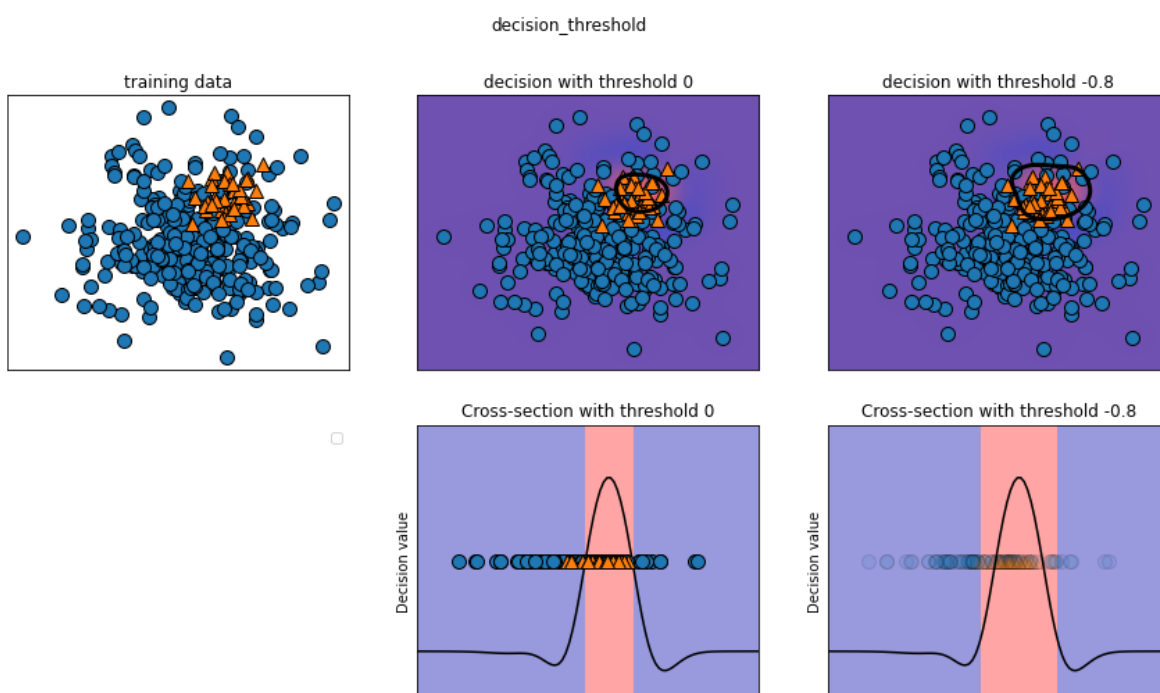
```
from sklearn.svm import SVC
```

In [45]:

```
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 [46]:

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



- 클래스 1에 대해 상당한 작은 정밀도를 얻었음. 재현율은 절반

- 클래스 0의 샘플이 매우 많으므로 분류기는 소수인 클래스 (양성)1보다 클래스 (음성)0에 초점.

In [49]:



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

| | 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 |

앞에서 이야기를 했듯 재현율(암인데 잘못 예측할 확률을 낮추어야 한다.)

- 정밀도(precision)을 높이기
 - TP/(TP + NP) : 양성 예측한 것중에 얼마나 잘 예측했나?
- 재현율(recall) : 실제 양성 데이터(TP + FN)중에서 얼마나 많은 샘플이 양성 클래스(TP)로 분류했을까?
- TP/(TP + FN) : 실제 데이터의 양성으로 맞추는 비율
 - 다른 말로 민감도(sensitivity), 적중률(hit rate), 진짜 양성 비율(TPR)이라고 합니다.

기본 0에서 -0.8로 낮추기

재현율(recall)을 높이기

- TP(양성 예측 잘함)을 늘리기, FP(양성 예측)

In [50]:



```
# 0으로 분류
decision_0 = svc.decision_function(X_test) > 0
decision_m08 = svc.decision_function(X_test) > -.8

# TP - 잘 맞추는 것을 늘린다.
print("양성을 양성으로 예측(TP) 개수 :", decision_0.sum() )
print("양성을 양성으로 예측(TP) 개수 :", decision_m08.sum() )

# (FP-양성을 잘못 예측하는 것은 줄어듬) 즉 NP는 올라가고 정밀도가 낮아짐
print("양성을 음성예측의(FP) 개수 :", len(decision_0) - decision_0.sum() )
print("양성을 음성예측의(FP) 개수 :", len(decision_m08) - decision_m08.sum() )
```

```
양성을 양성으로 예측(TP) 개수 : 17
양성을 양성으로 예측(TP) 개수 : 28
양성을 음성예측의(FP) 개수 : 96
양성을 음성예측의(FP) 개수 : 85
```

임계값을 낮추기(True)을 키우기

In [51]:



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

- 정밀도(precision) 0.35에서 0.32로 낮아지고
- 재현율(recall)-sensitivity(민감도)는 0.67에서 1로 올라감.

In [52]:



```
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 |

실습

- 임계값을 0보다 큰 값으로 조정해 보고 recall를 확인해 보기
- 임계값을 0으로 하고 recall과 기타 평가지표를 확인해 보기

기타 방법

- predict_proba()메서드는 출력이 0에서 1 사이로 고정
 - 보통은 0.5를 임계값-이는 양성(양성)과 음성(음성)이 50%분류이다.
 - 임계값을 높이는 것은 양성이 분류될 확률이 많이 나올 때, 수행