모델 평가와 성능 향상

모델 훈련과 모델 평가

- 전체 데이터를 훈련 세트와 데이터 세트로 분할
- 훈련 데이터로 학습 (fit)
- 테스트 데이터로 평가 (score)
  - 분류에서 score는 정확히 분류된 샘플의 비율 (정확도)
  - 회귀에서는 R^2 값

```
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) # 인위적인 데이터셋 생성

# 데이터와 타깃 레이블을 훈련 세트와 테스트 세트로 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

logreg = LogisticRegression(solver='lbfgs', multi_class="auto") #모델 생성
logreg = logreg.fit(X_train, y_train) # 훈련 세트로 학습

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

```
테스트 세트 점수: 0.88
```

모델 훈련과 모델 평가

- 훈련 데이터와 테스트 데이터로 분할 → 모델이 지금까지 본 적 없는 데이터에 얼마나 잘 일반화되는지 측정하기 위한 것

- 모델이 훈련 세트에 잘 맞는 것보다 학습 과정에 없었던 데이터에 대해 예측을 잘 하는 것이 중요

교차 검증

교차 검증

- [훈련 / 테스트 데이터 분할 → 훈련 데이터로 학습 → 테스트 데이터로 평가] 작업 과정을 여러 번 반복 수행해서 안정적이고 정확한 평가 가능

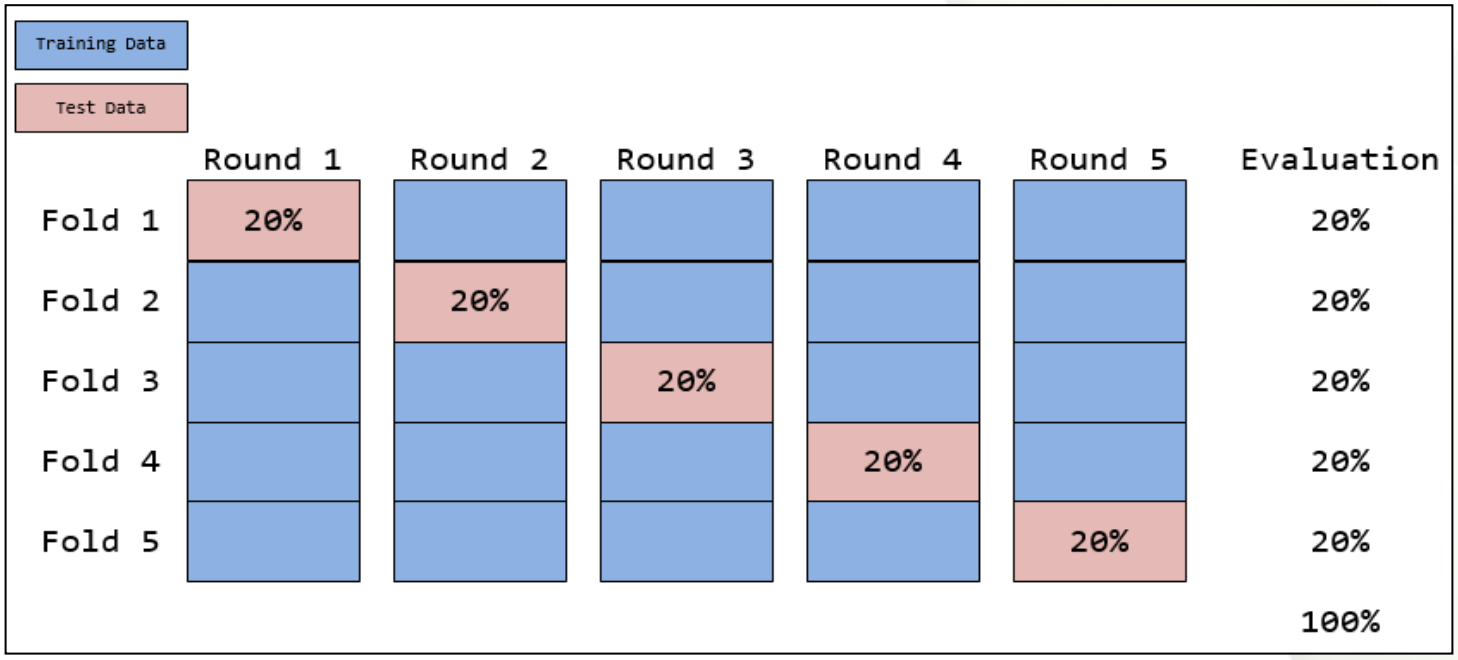
- 장점
  - 여러 번 반복하는 과정을 통해 훈련 데이터 세트와 테스트 데이터 세트가 우연하게 비정상적인 분포를 갖는 문제 해결
  - 데이터를 효과적으로 사용
    - » 테스트 데이터로 (1 / 전체 반복회수) 만큼 사용하기 때문에 반복회수가 많아지면 훈련 데이터로 사용할 수 있는 데이터 양 증가

- 단점
  - 연산 비용 증가
    - » 모델을 반복 회수만큼 만들기 때문에 반복 회수만큼 느려짐

k-fold cross validation

- 가장 널리 사용되는 교차 검증 기법

- 절차
  - 전체 데이터를 비슷한 규모를 갖는 k개의 데이터셋으로 구분
  - (k-1)개의 데이터셋으로 학습하고 한 개의 데이터셋으로 평가
  - 이 작업을 k번 반복 (매번 테스트 데이터셋을 바꾸어서 실행)



k-fold cross validation 구현

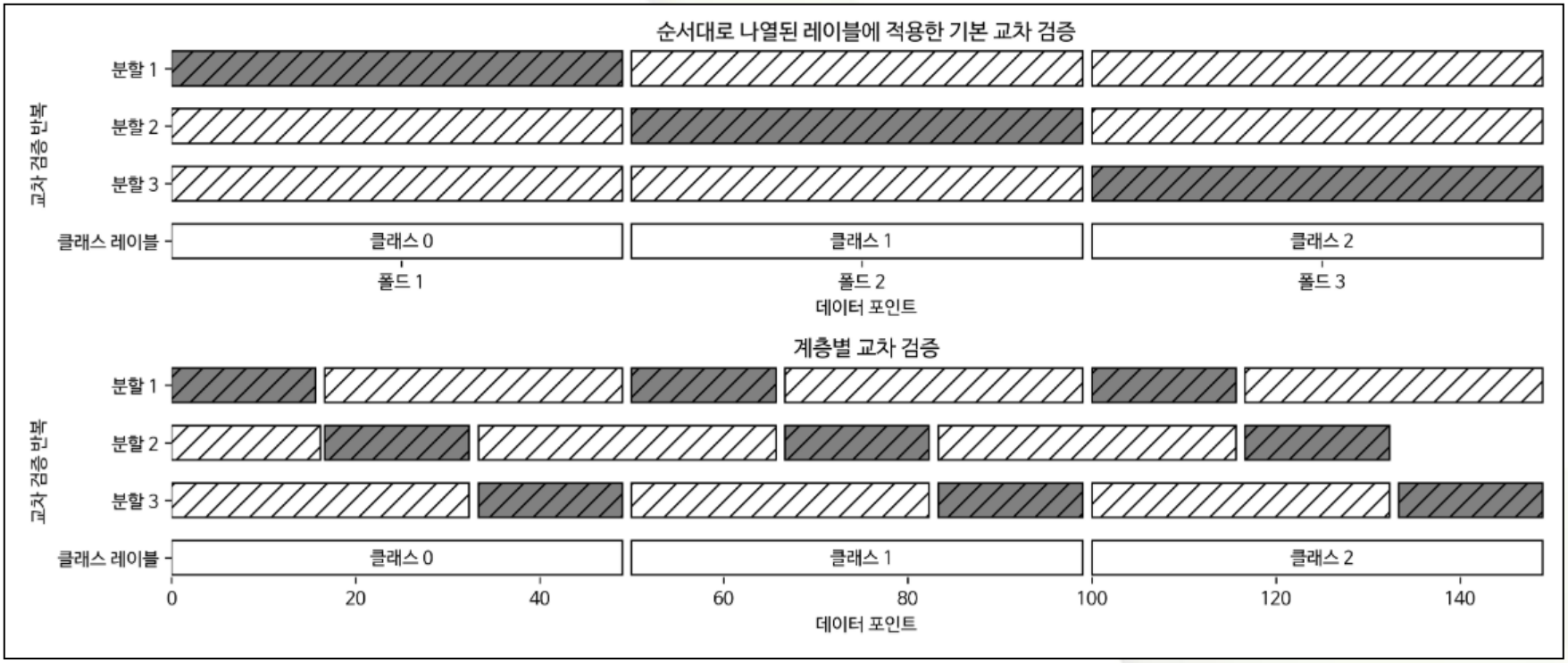
- scikit-learn 교차 검증 구현 (단순 교차 검증)
  - cross_val_score 함수 사용

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

```
교차 검증 점수: [1.    0.967 0.933 0.9   1.   ]
교차 검증 평균 점수: 0.96
```

k-fold cross validation 구현

- 데이터에 따라 각 분할 데이터셋에 포함된 데이터 구성 비율을 전체 데이터의 데이터 구성 비율과 비슷하게 유지하는 것이 필요한 경우가 있음 (예 : iris 데이터셋)



- 일반적으로 회귀에는 k-겹 교차 검증, 분류에는 계층별 k-겹 교차 검증 기본값이 잘 동작

k-fold cross validation 구현

- scikit-learn 교차 검증 구현 (속성 제어)

```
from sklearn.model_selection import KFold
kfold = KFold(n_splits=5)
print("교차 검증 점수:{}".format(
      cross_val_score(logreg, iris.data, iris.target, cv=kfold)))
```

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

폴드와 분류가 겹치면 학습을 정상적으로 수행할 수 없음

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

```
교차 검증 점수:[0. 0. 0.]
```

```
kfold = KFold(n_splits=3, shuffle=True, random_state=0)
print("교차 검증 점수:{}".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)

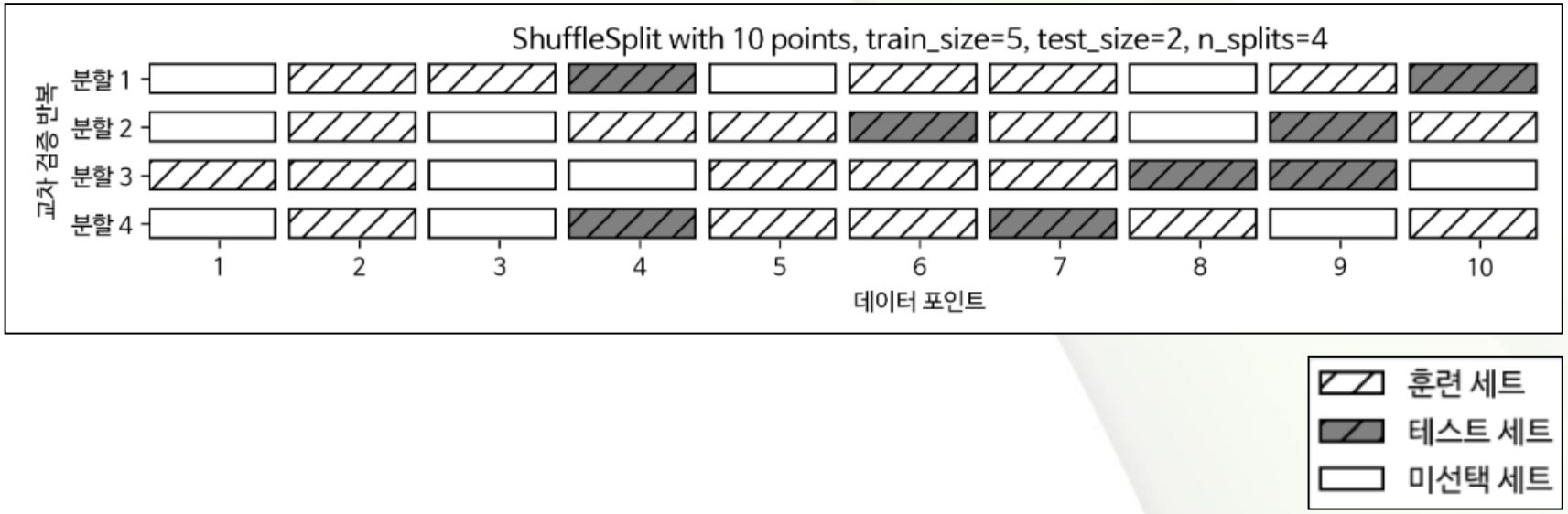
- 분할 하나에 샘플 하나만 포함된 k-fold cross validation
- 각 반복에서 하나의 데이터 포인트를 선택해서 테스트 세트로 사용
- 보통 작은 데이터셋에서 향상된 결과 도출

```
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
```

shuffle-split cross-validation

- 훈련 세트의 크기, 테스트 세트의 크기, 반복 횟수를 지정해서 데이터 분할
  - 크기가 정수인 경우 데이터 개수 / 실수인 경우 비율
  - 반복 횟수를 훈련 또는 테스트 세트의 크기와 독립적으로 조절해야 하는 경우에 유용
  - 전체 데이터의 일부만 사용할 수 있기 때문에 대규모 데이터 세트로 작업할 때 유용



shuffle-split cross-validation

- 훈련 데이터 세트 50%, 테스트 데이터 세트 50 %, 반복 10회

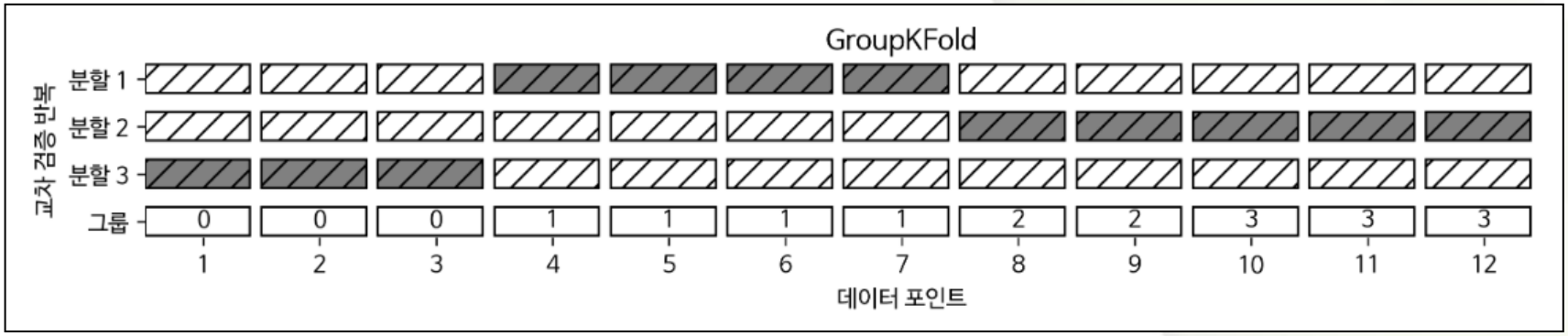
```
from sklearn.model_selection import ShuffleSplit

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:
[0.96  0.92  0.96  0.907 0.987 0.92  0.92  0.92  0.947 0.947]
```

그룹별 교차 검증

- 각 데이터에 그룹을 지정해서 훈련 세트와 테스트 세트에 같은 그룹으로 지정된 데이터가 포함되지 않도록 분할

- 사례
  - 사진에서 표정을 인식하는 시스템을 만들기 위해 100명에 대한 n개의 사진을 모은 경우 → 각 사람을 그룹으로 지정해서 같은 사람의 사진이 훈련 세트와 테스트 세트에 동시에 포함되지 않도록 분할
  - 100명의 환자로부터 n개의 데이터가 수집된 경우 → 각 환자를 그룹으로 지정해서 같은 환자가 동시에 훈련 세트와 테스트 세트에 포함되지 않도록 분할



그룹별 교차 검증

```
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]
```

그리드 서치 (매개변수 튜닝)

그리드 서치 (Grid Search)

- 일반화 성능을 최대로 높여주는 모델의 매개변수를 찾는 과정이 필요한데 가장 널리 사용하는 매개변수 탐색 방법 중 하나가 Grid Search

- 방법
  - 관심 있는 매개변수들을 대상으로 가능한 모든 조합을 시도해서 결과 비교
  - 사례 : SVM의 gamma와 C 매개변수 설정 값에 대한 테스트 조합

	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 (Grid Search)

- 구현

```
from sklearn.svm import SVC
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]:
        # 매개변수의 각 조합에 대해 SVC를 훈련시킵니다
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        score = svm.score(X_test, y_test) # 테스트 세트로 평가

        if score > best_score: # 점수가 더 높으면 매개변수와 함께 기록
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

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

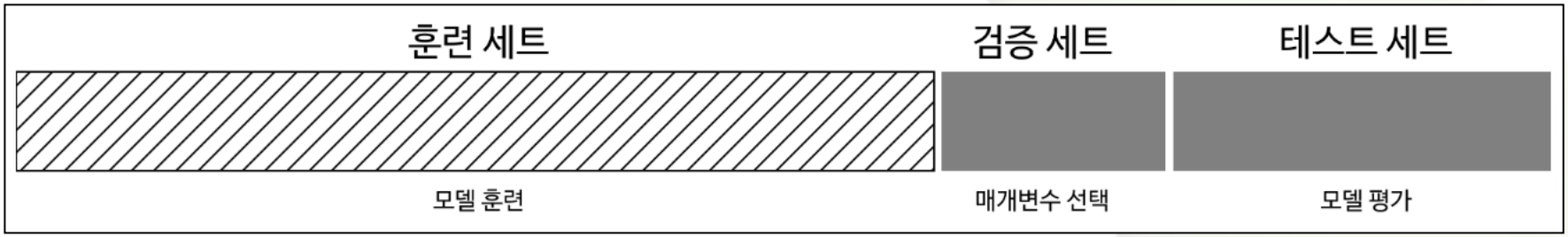
```
최고 점수: 0.97
최적 파라미터: {'C': 100, 'gamma': 0.001}
```

매개변수 과대적합 검증 세트

- Grid Search와 같은 방법으로 매개변수를 최적화하는 과정에서 테스트 데이터를 사용하기 때문에 최적화된 매개변수를 적용한 최종 모델을 테스트할 데이터가 없음

- 이를 위해 모델을 만들고 매개변수를 최적화 할 때 사용하지 않은 독립된 데이터 세트 필요

- 훈련 세트로는 모델을 만들고 검증 (또는 개발) 세트로는 모델의 매개변수를 선택하고 테스트 세트로 성능을 평가하는 방법 사용



매개변수 과대적합 검증 세트

- 구현

```
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]))
```

매개변수 과대적합 검증 세트

- 구현 (계속)

```
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)

        score = svm.score(X_valid, y_valid) # 검증 세트로 SVC를 평가합니다

        if score > best_score: # 점수가 더 높으면 매개변수와 함께 기록합니다
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

# 훈련 세트와 검증 세트를 합쳐 모델을 다시 만든 후 테스트 세트를 사용해 평가
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))
```

매개변수 과대적합 검증 세트

- 구현 (계속)

```
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

```
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)
```

```
최적 파라미터에서 테스트 세트 점수: 0.97
```

교차 검증을 사용한 그리드 서치

- scikit-learn 지원 GirdSearchCV 사용

```
# 딕셔너리 형식의 매개변수 그리드 만들기
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))

# GridSearchCV 객체 만들기
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
grid_search = \
GridSearchCV(SVC(), param_grid, cv=5, return_train_score=True, iid=None)

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

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

테스트 세트 점수: 0.97
```

교차 검증을 사용한 그리드 서치

- scikit-learn 지원 GirdSearchCV 사용

```
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
```

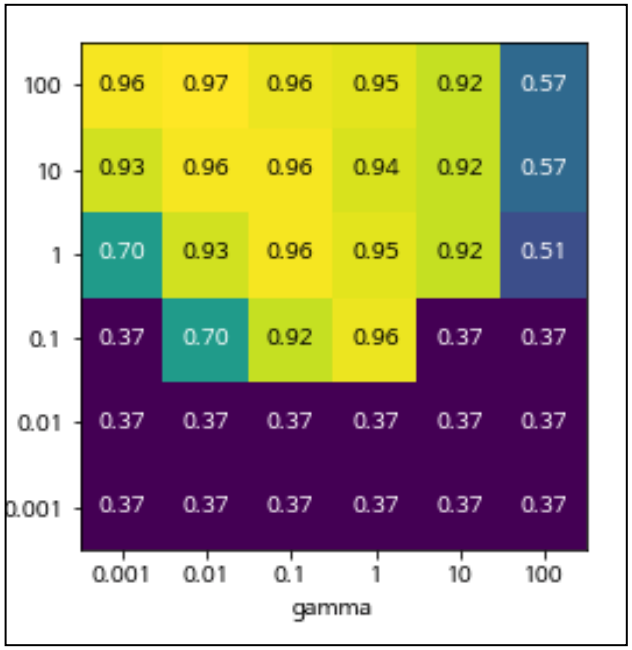
주의
score 함수의 결과는 전체 훈련 세트에서 학습한 모델을 테스트 세트로 평가한 결과.
grid_search.best_score_는 교차 검증의 평균 정확도

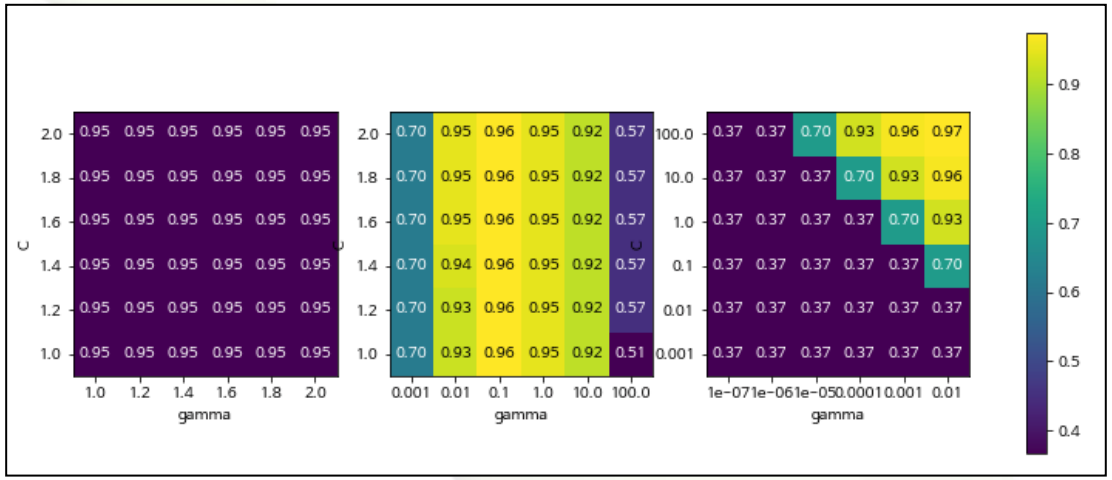
```
print("최고 성능 모델:\n{}".format(grid_search.best_estimator_))
```

```
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)
```

교차 검증 결과 분석

- Pandas DataFrame, HeatMap 등을 사용해서 결과 표시
- 매개변수의 최적값이 그래프 끝에 놓이지 않도록 매개변수 조정 필요





- 교차 검증 점수를 기반으로 매개변수 그리드를 튜닝하는 것은 안전한 방법이며 매개변수의 중요도를 확인할 때에도 유용함

비대칭 매개변수 그리드 탐색

- 사용할 모델에 따라 모든 매개 변수의 조합을 조사하는 것은 적절하지 않음
  - SVC는 kernel 매개변수에 따라 C 매개변수만 사용하거나 gamma 매개변수를 함께 사용할 수 있음

- GridSearchCV에 전달할 param_grid를 딕셔너리의 리스트로 제공해서 해결

```
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]}]

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

평가 지표와 측정

정확도

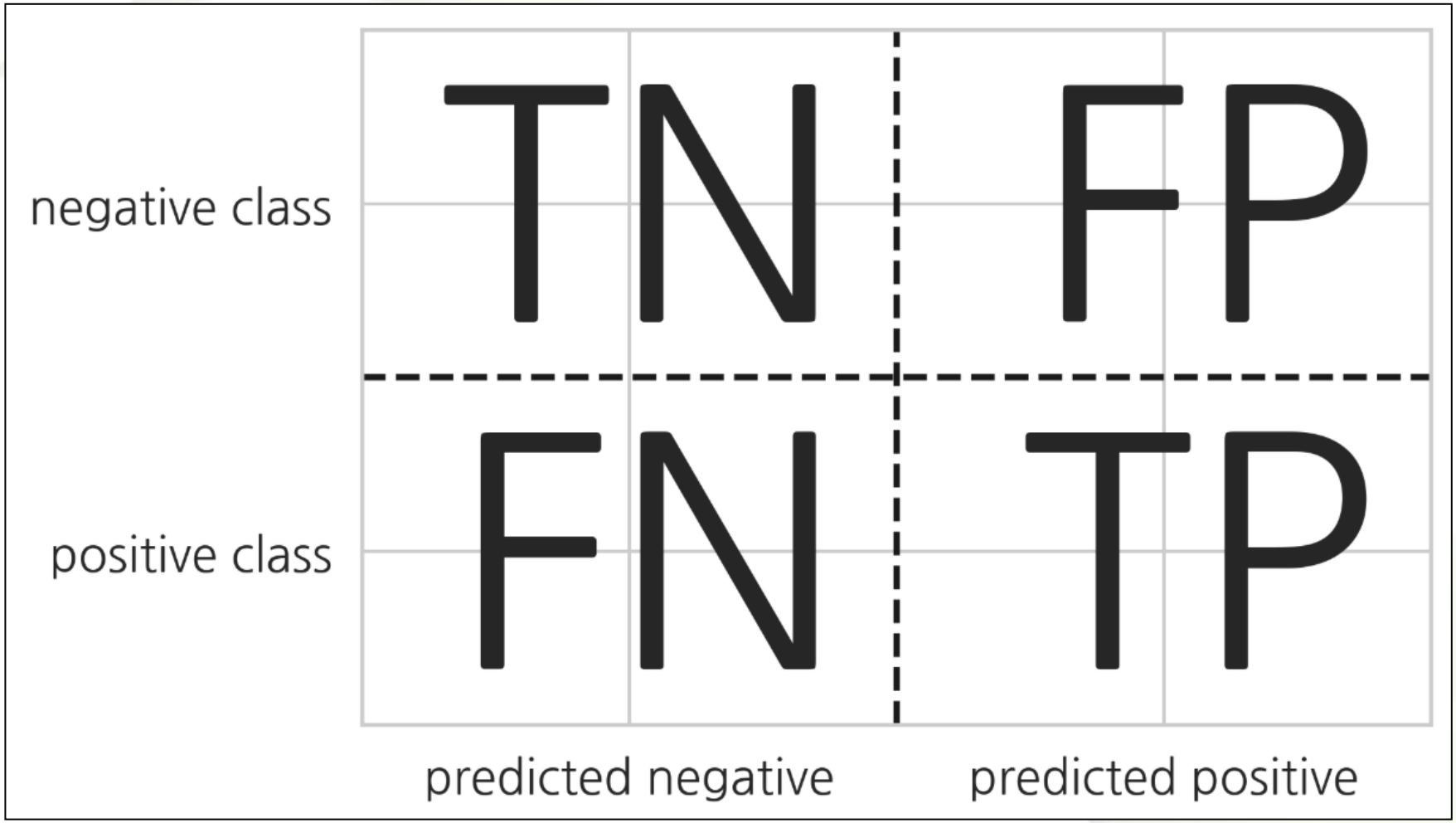
- 전체 데이터에서 예측 데이터가 실제 데이터와 일치하는 비율

- 정확도만으로 예측 성능을 측정할 때의 문제
  - 거짓 양성 (양성으로 판단했지만 실제 음성인 경우)과 거짓 음성 (음성으로 판단했지만 양성인 경우)의 중요도가 같지 않은 경우
    - » 사례 → 암이 아닌데 암으로 판단 vs 암인데 암이 아닌 것으로 판단

  - 불균형 데이터 세트
    - » 90%가 양성인 데이터 세트에 대해 모두 양성으로 예측하면 정확도 90%가 도출됨

오차 행렬

- 이진 분류 평가 결과를 표시할 때 가장 많이 사용하는 방법



오차 행렬

- 비슷한 정확도를 보이는 모델을 오차 행렬로 표현

```
모두 음성으로 판단한 모델:
[[403   0]
 [ 47   0]]

무작위 더미 모델:
[[357  46]
 [ 43   4]]

결정 트리:
[[390  13]
 [ 24  23]]

로지스틱 회귀
[[401   2]
 [  8  39]]
```

정확도 : 90%

정확도 : 82%

정확도 : 92%

정확도 : 98%

정확도, 정밀도, 재현율, F-Score

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

$$\text{정밀도} = \frac{TP}{TP + FP}$$

$$\text{재현율} = \frac{TP}{TP + FN}$$

$$F = 2 \cdot \frac{\text{정밀도} \cdot \text{재현율}}{\text{정밀도} + \text{재현율}}$$

- 정밀도 또는 재현율 하나 만으로는 전체 그림을 볼 수 없기 때문에 이 둘을 함께 반영하는 F1-Score 사용

오차 행렬

- 비슷한 정확도를 보이는 모델을 오차 행렬로 표현

```
모두 음성으로 판단한 모델:
[[403   0]
 [ 47   0]]

무작위 더미 모델:
[[357  46]
 [ 43   4]]

결정 트리:
[[390  13]
 [ 24  23]]

로지스틱 회귀
[[401   2]
 [  8  39]]
```

정확도 : 90%

F-Score : 0.00

정확도 : 82%

F-Score : 0.08

정확도 : 92%

F-Score : 0.55

정확도 : 98%

F-Score : 0.89

- `classification_report` 함수를 사용해서 요약된 정보 표시

정밀도와 재현율

- 재현율이 중요한 경우 → 실제 양성 데이터를 음성 데이터로 잘못 판단했을 때 상대적으로 더 큰 문제가 되는 경우
  - 사례 : 암 진단에서 암 환자(양성)을 정상(음성)으로 판단한 경우

- 정밀도가 중요한 경우 → 실제 음성 데이터를 양성 데이터로 잘못 판단했을 때 상대적으로 더 큰 문제가 되는 경우
  - 사례 : 스팸 메일 필터링에서 중요한 메일(음성)을 스팸 메일(양성)으로 판단한 경우

- 일반적으로 재현율이 정밀도보다 중요한 업무가 상대적으로 많음

- 두 지표는 한 쪽을 높이면 다른 쪽은 낮아지는 상충 관계

불확실성 고려

- 대부분의 분류기는 예측의 확신 정도를 보여주는 decision_function 또는 predict_proba 함수 제공
  - decision_function의 임계값은 0
  - predict_proba의 임계값은 0.5

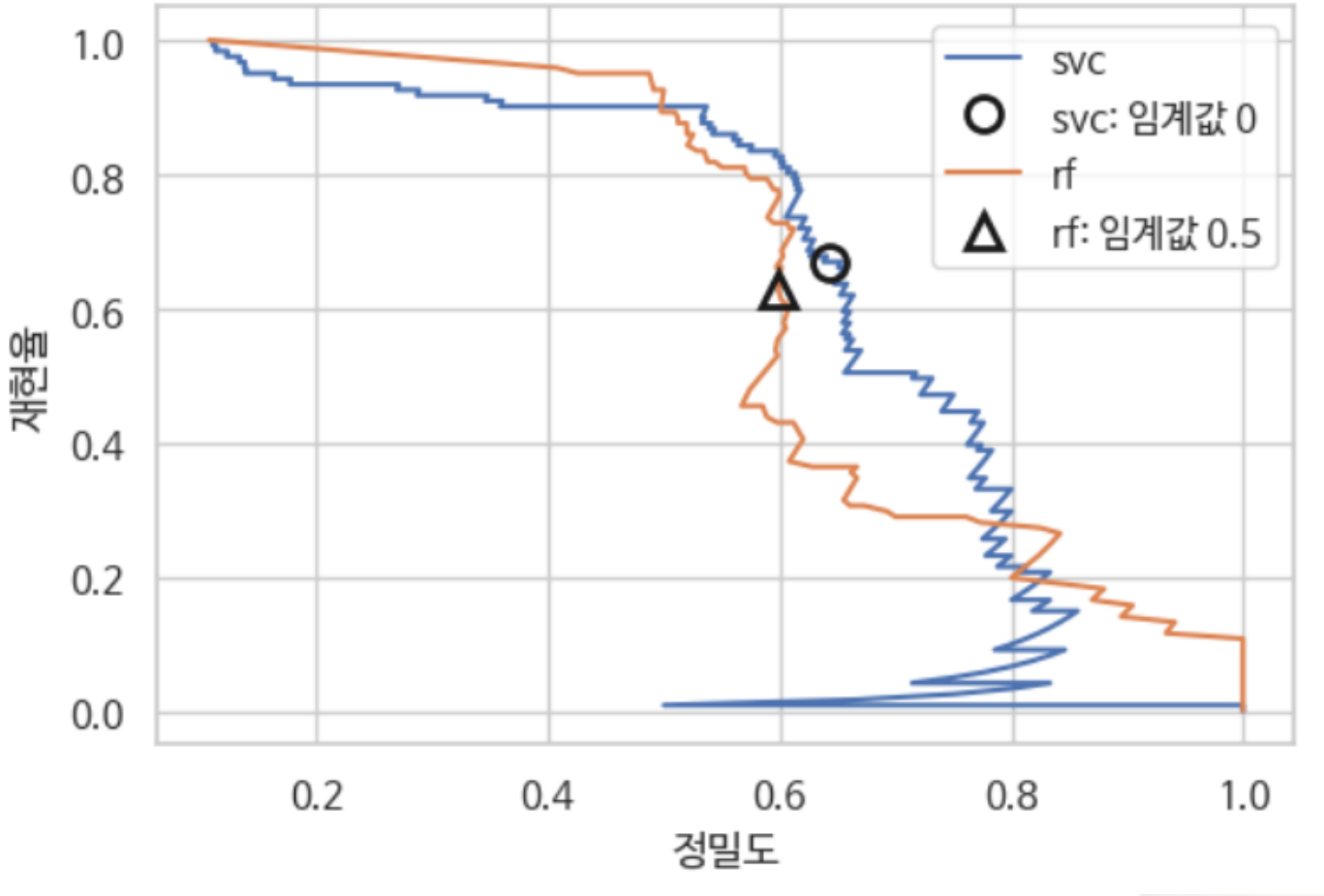
- 임계점을 조정해서 재현율에 더 가중치를 두거나 혹은 정밀도에 더 가중치를 두도록 구현할 수 있음

정밀도-재현율 곡선

- 모델의 분류 작업을 결정하는 임계값을 변경하는 것은 해당 모델의 정밀도와 재현율의 상충 관계를 조정하는 것
- 적절한 임계값 설정을 위해 정밀도와 재현율을 종합적으로 살펴보아야 하며 이를 위해 정밀도-재현율 곡선 사용
- scikit-learn의 precision_recall_curve 함수 사용해서 정밀도-재현율 곡선 표시
- 특정 임계값이나 운영 포인트에 국한하지 않고 전체 곡선에 담긴 정보를 요약하기 위해 정밀도-재현율 곡선의 아래 부분 면적을 계산
  - 평균 정밀도
  - average_precision_score 함수 사용

정밀도 - 재현율 곡선

- 정밀도 – 재현율 곡선 사례



ROC 곡선

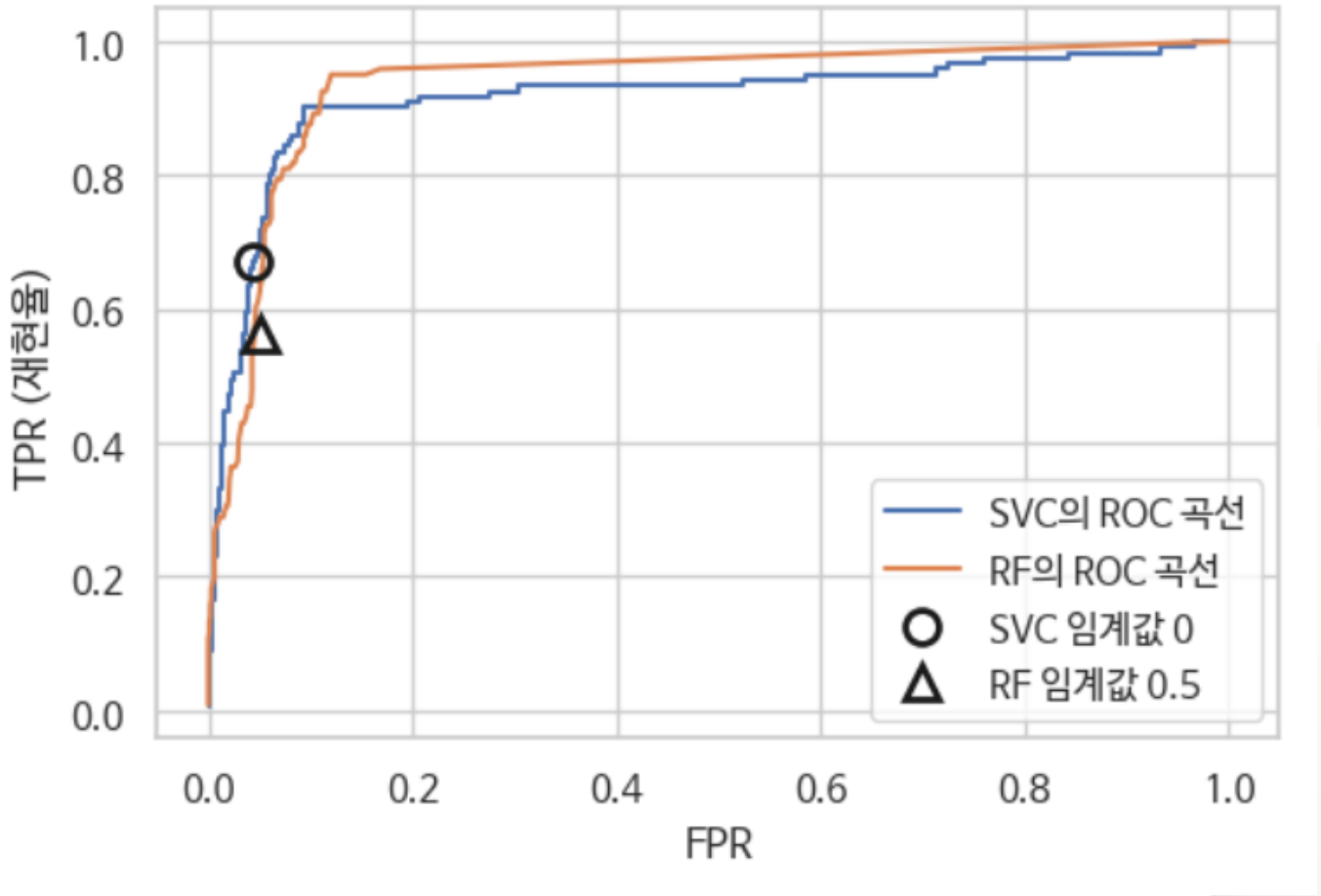
- 여러 임계값에서 분류기의 특성을 분석하는데 널리 사용되는 도구

- 정밀도 – 재현율 대신 진짜 양성 비율에 대한 거짓 양성 비율을 표현
  - 진짜 양성비율은 재현율 (TPR)
  - 거짓 양성 비율은 전체 음성 샘플 중 양성으로 잘못 분류한 비율 (FPR)

- roc_curve 함수를 사용해서 표시
  - 좌측 상단에 가까울수록 이상적인 모델

ROC 곡선

- ROC 곡선 사례



AUC

- ROC 곡선 아래쪽의 면적 값으로 ROC 곡선을 요약할 때 사용

- roc_auc_score 함수로 구현