교차 검증

교차 검증

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

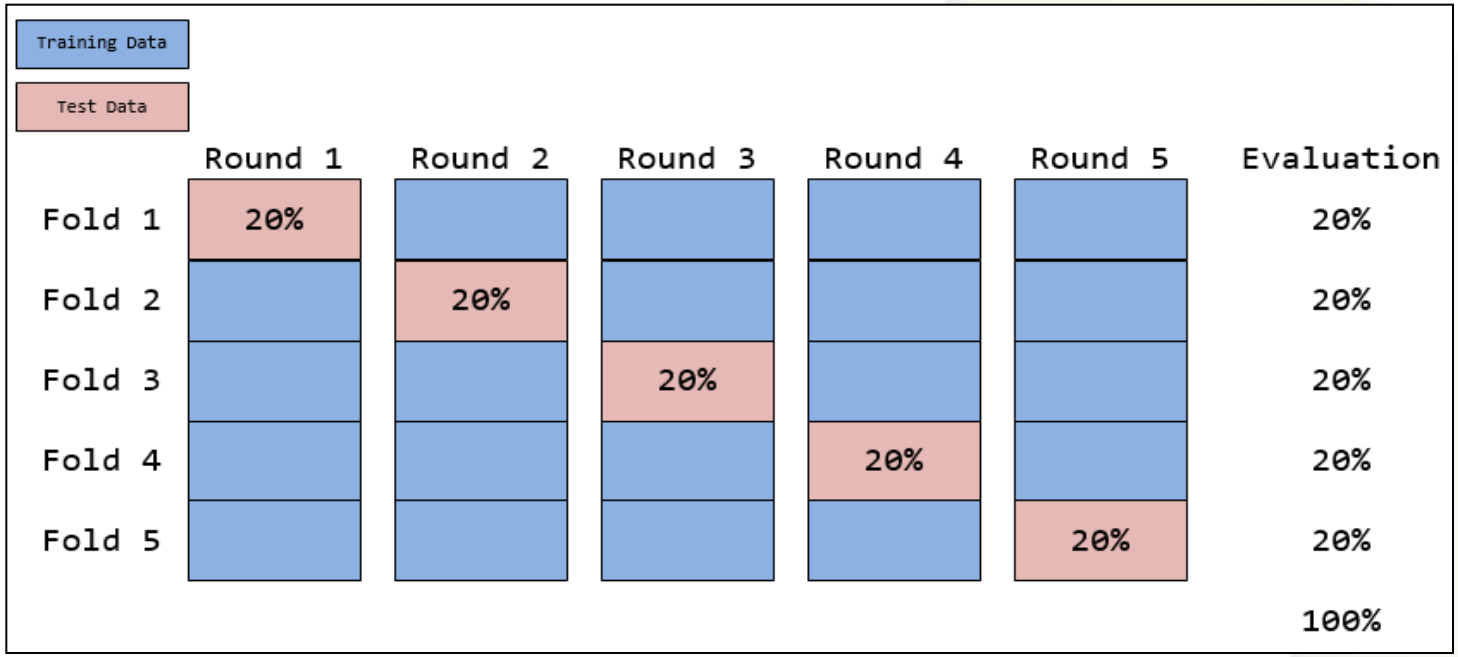
- 장점
  - 여러 번 반복하는 과정을 통해 훈련 데이터 세트와 테스트 데이터 세트가 우연하게 비정상적인 분포를 갖는 문제 해결
  - 데이터를 효과적으로 사용
    - » 테스트 데이터로 (1 / 전체 반복회수) 만큼 사용하기 때문에 반복회수가 많아지면 훈련 데이터로 사용할 수 있는 데이터 양 증가

- 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

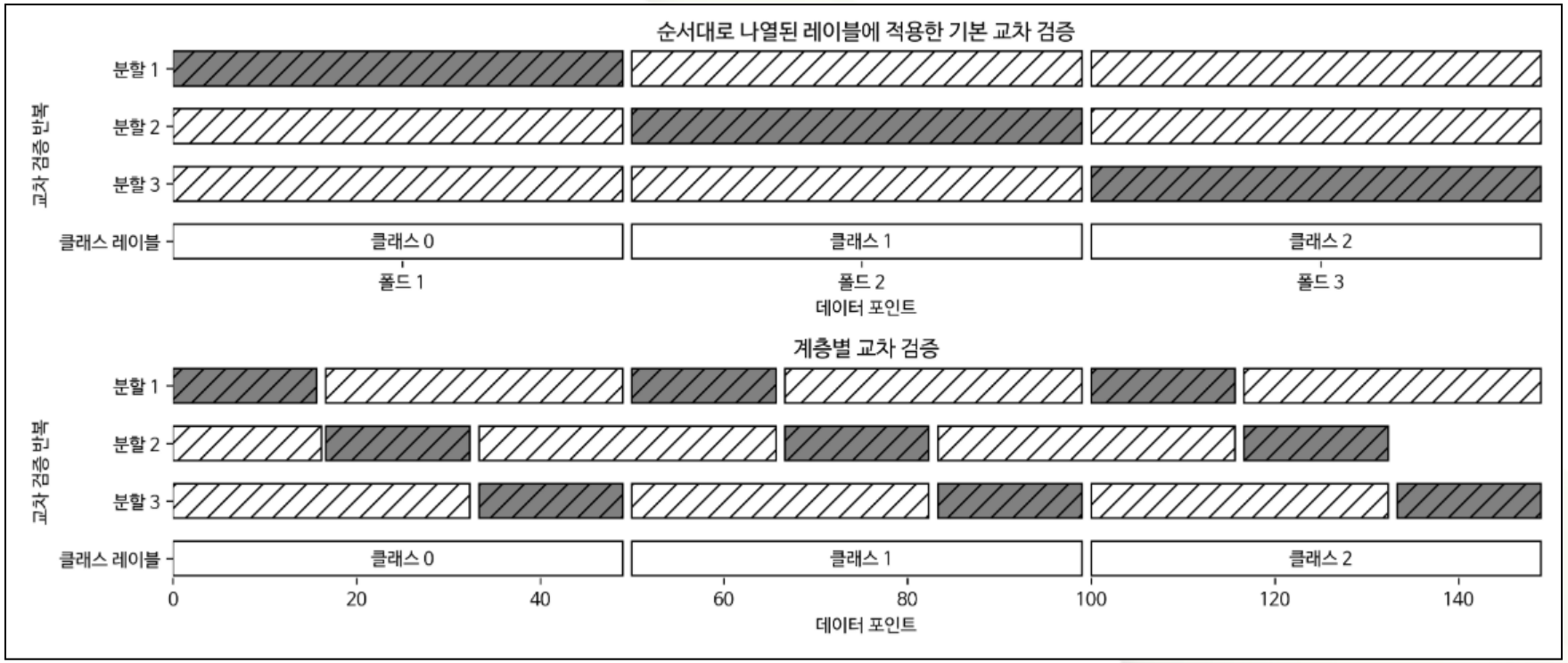
- scikit-learn 교차 검증 구현 (단순 교차 검증)
  - cross_val_score 함수 사용

```
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

LOOCV (leave-one-out cross-validation)

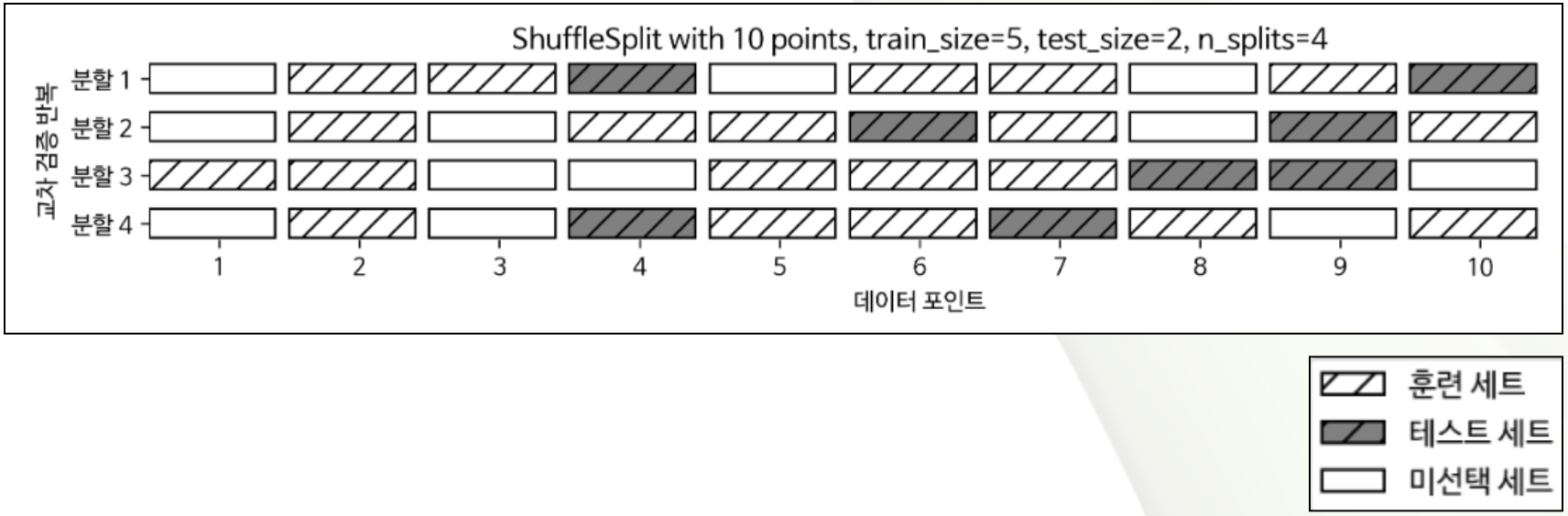
- 분할 하나에 샘플 하나만 포함된 k-fold cross validation
- 각 반복에서 하나의 데이터 포인트를 선택해서 테스트 세트로 사용
- 보통 작은 데이터셋에서 향상된 결과 도출

```
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

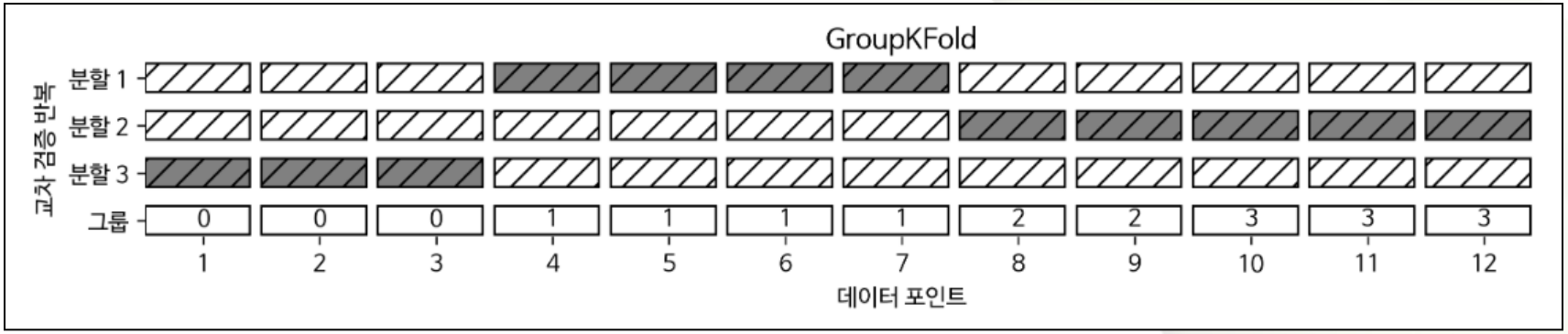
- 훈련 세트의 크기, 테스트 세트의 크기, 반복 횟수를 지정해서 데이터 분할
  - 크기가 정수인 경우 데이터 개수 / 실수인 경우 비율
  - 반복 횟수를 훈련 또는 테스트 세트의 크기와 독립적으로 조절해야 하는 경우에 유용
  - 전체 데이터의 일부만 사용할 수 있기 때문에 대규모 데이터 세트로 작업할 때 유용



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



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

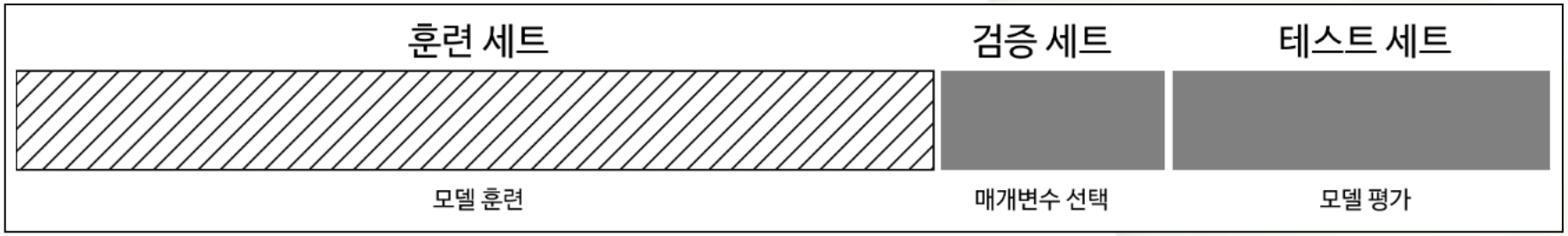
	C = 0.001	C = 0.01	···	C = 10
gamma=0.001	SVC(C=0.001, gamma=0.001)	SVC(C=0.01, gamma=0.001)	···	SVC(C=10, gamma=0.001)
gamma=0.01	SVC(C=0.001, gamma=0.01)	SVC(C=0.01, gamma=0.01)	···	SVC(C=10, gamma=0.01)
···	···	···	···	···
gamma=100	SVC(C=0.001, gamma=100)	SVC(C=0.01, gamma=100)	···	SVC(C=10, gamma=100)

매개변수 과대적합 검증 세트

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

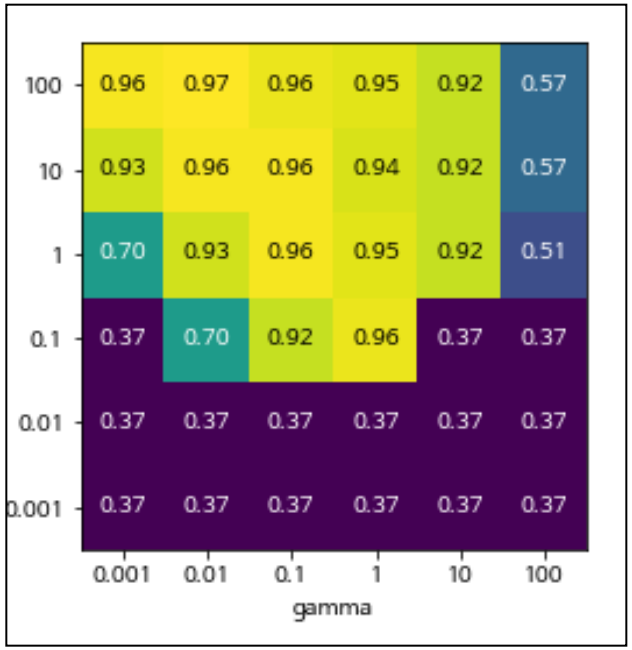
- 이를 위해 모델을 만들고 매개변수를 최적화 할 때 사용하지 않은 독립된 데이터 세트 필요

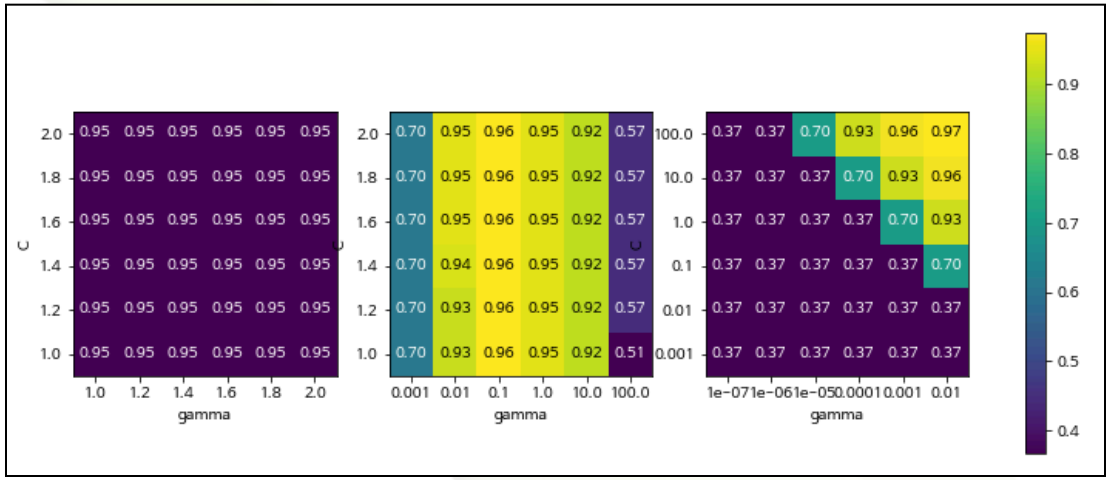
- 훈련 세트로는 모델을 만들고 검증 (또는 개발) 세트로는 모델의 매개변수를 선택하고 테스트 세트로 성능을 평가하는 방법 사용



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

알고리즘 체인과 파이프라인

파이프라인

- 대부분의 머신러닝 애플리케이션은 여러 단계의 처리 과정과 머신러닝 모델이 연결되어 구성됨

- 데이터 변환 과정과 머신러닝 모델을 쉽게 연결할 수 있는 Pipeline 파이썬 클래스 지원

- 전처리, 분류에 국한하지 않고 어떤 추정기와도 연결 가능

- 인터페이스
  - 파이프라인에 들어갈 추정기는 마지막 단계를 제외하고 모두 transform 메서드를 구현해야 함 → 다음 단계를 위한 새로운 데이터 표현 반환
  - 예측 단계에서는 predict 메서드 호출
  - [fit → transform] → [fit → transform] → … → [predict]

파이프라인 만들기

- Pipeline 클래스를 사용해서 파이프라인 생성
  - 각 단계의 이름을 명시해서 생성

- make_pipeline 함수 사용해서 파이프라인 생성
  - 단계 이름 자동 생성
  - 단계 이름은 파이썬 클래스 이름의 소문자 버전
  - 같은 클래스가 여러 개 사용되면 번호로 식별