

BIG_PY
LEVEL.2
—————
SESSION

Chapter.07

**양상을 학습과
랜덤 포레스트**

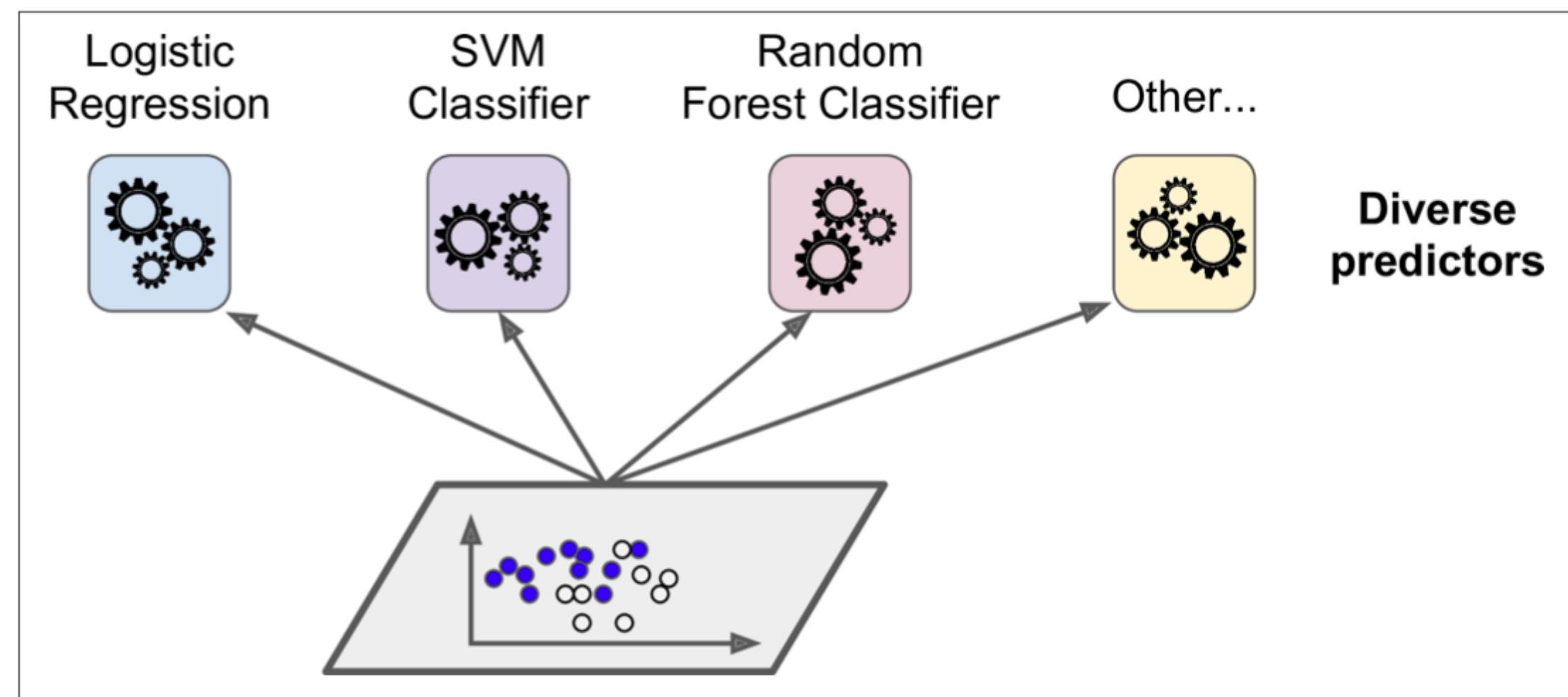
**세션장
나마로**

- wisdom of the crowd
- 일련의 예측기(즉, 분류나 회귀 모델)로부터 예측을 수집하면 가장 좋은 모델 하나보다 더 좋은 예측을 얻을 수 있음
 - ✓ 이때 일련의 예측기를 양상블이라고 부르기 때문에 이를 양상블 학습(ensemble learning)이라고 함
 - ✓ 양상블 학습 알고리즘을 양상블 방법(ensemble method)이라고 함
- 예를 들어 결정 트리의 양상블을 랜덤 포레스트(random forest)라고 부름
- 배깅(bagging), 부스팅(boosting), 스태킹(stacking) 등 다양한 양상블 방법이 있음



투표 기반 분류기

- 각 분류기의 예측을 모아서 가장 많이 선택된 클래스를 예측하는 것
- 직접 투표(hard voting) 분류기: 다수결 투표로 정해지는 분류기
- 간접 투표(soft voting) 분류기: 개별 분류기의 예측을 평균 내어 확률이 가장 높은 클래스를 예측



- 각 분류기의 예측을 모아서 가장 많이 선택된 클래스를 예측하는 것
- 직접 투표(hard voting) 분류기: 다수결 투표로 정해지는 분류기
- 간접 투표(soft voting) 분류기: 개별 분류기의 예측을 평균 내어 확률이 가장 높은 클래스를 예측

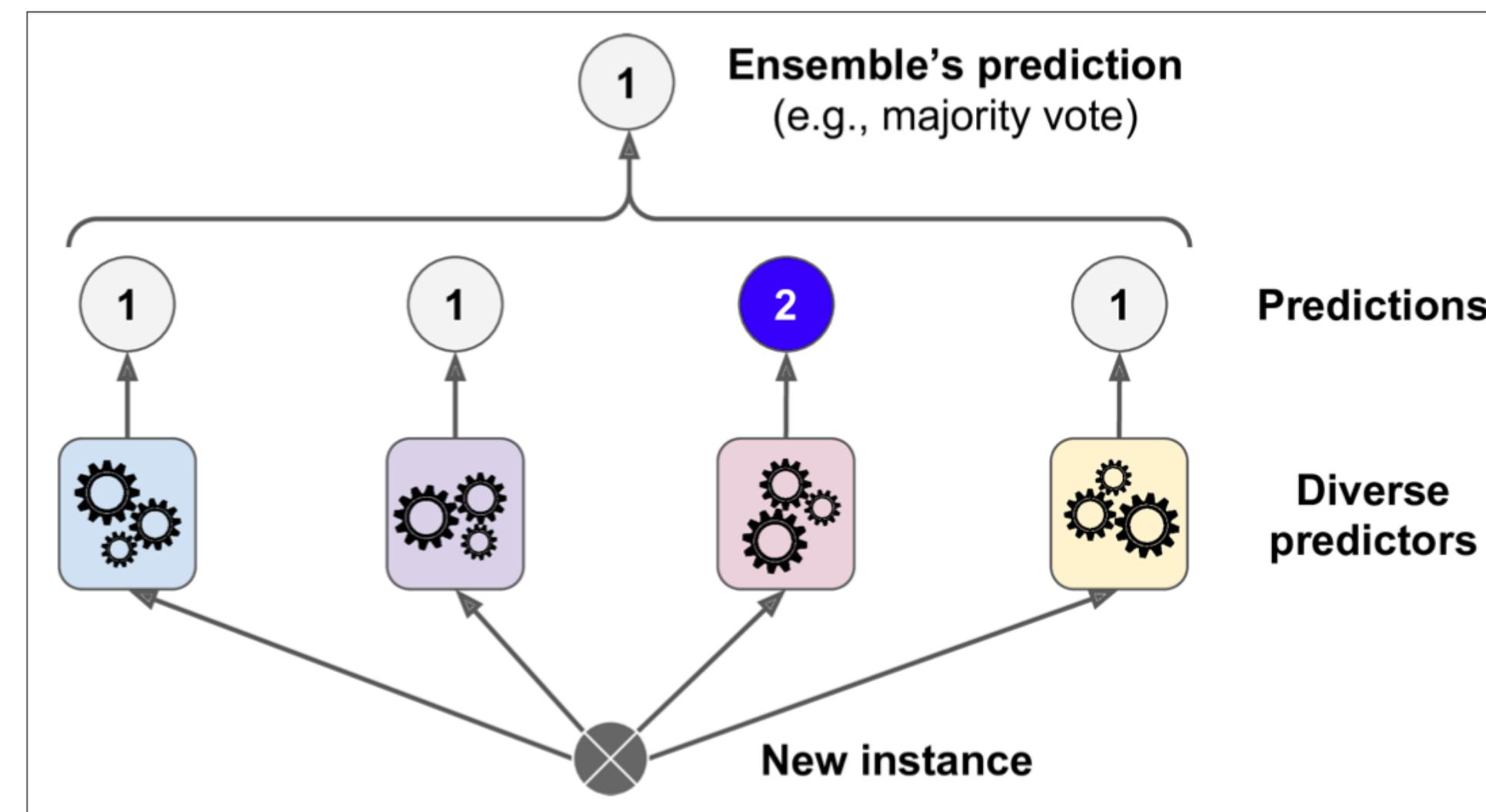


Figure 7-2. Hard voting classifier predictions

투표 기반 분류기

- 각 분류기가 약한 학습기(weak learner)일지라도 충분하게 많고 다양하다면 앙상블은 강한 학습기(strong learner)가 될 수 있음
 - ✓ 이는 큰 수의 법칙(law of large numbers)에 의해 보여짐
- 앙상블 방법은 예측기가 가능한 한 서로 독립적일 때 최고의 성능을 발휘함

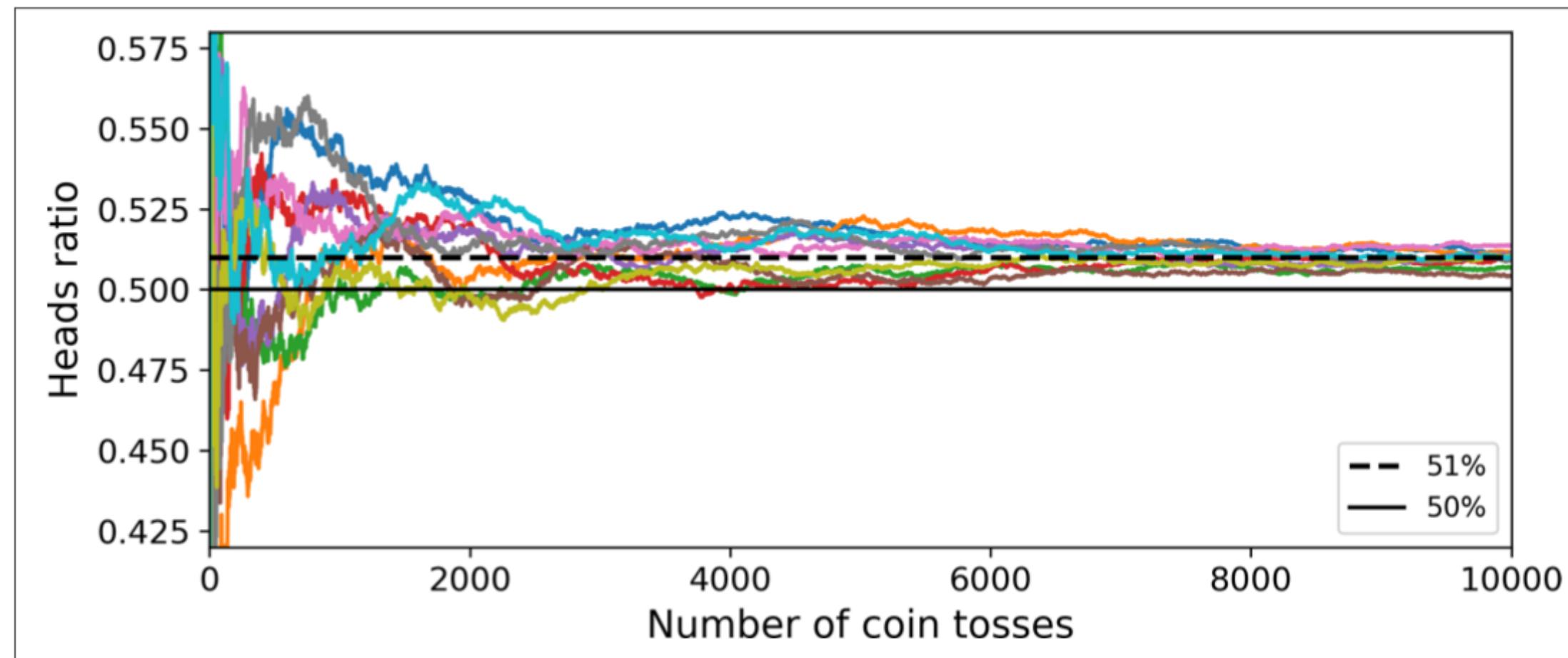


Figure 7-3. The law of large numbers

직접 투표(hard voting) 분류기

- sklearn.esnsmble 패키지 내에 VotingClassifier 클래스 사용

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression(solver="lbfgs", random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=100, random_state=42)
svm_clf = SVC(gamma="scale", random_state=42)

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')

voting_clf.fit(X_train, y_train)
```

직접 투표(hard voting) 분류기

LEVEL.2
SESSION

```
from sklearn.metrics import accuracy_score

for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```



LogisticRegression 0.864

RandomForestClassifier 0.896

SVC 0.896

VotingClassifier 0.912

간접 투표(soft voting) 분류기

LEVEL.2
SESSION

- 모든 분류기가 클래스 확률을 예측할 수 있으면(즉, predict_proba() 메서드가 있으면), 개별 분류기의 예측을 평균 내어 확률이 가장 높은 클래스를 예측할 수 있음
- 위 방식을 간접 투표(soft voting)이라 하고, 이 방식은 확률이 높은 투표에 비중을 더 두기 때문에 직접 투표 방식보다 성능이 더 높음
- VotingClassifier 클래스에서 voting 매개변수를 "soft"로 지정하면 간접 투표 방식을 사용할 수 있음

BIG.PY

배깅(bagging)과 페이스팅(pasting)

- 같은 알고리즘을 사용하는 대신, 훈련 세트의 서브셋을 무작위로 구성하여 분류기를 각기 다르게 학습시킬 수 있음
- 훈련 세트에서 중복을 허용하여 샘플링하는 방식을 배깅(bagging)이라 함
 - ✓ bootstrap aggregating의 줄임말이며, 중복을 허용한 리샘플링(resampling)을 통계학에서는 부트스트래핑(bootstrapping)이라고 함
- 중복을 허용하지 않고 샘플링 하는 방식을 페이스팅(pasting)이라고 함

BIG.PY

배깅(bagging)과 페이스팅(pasting)

LEVEL.2
SESSION

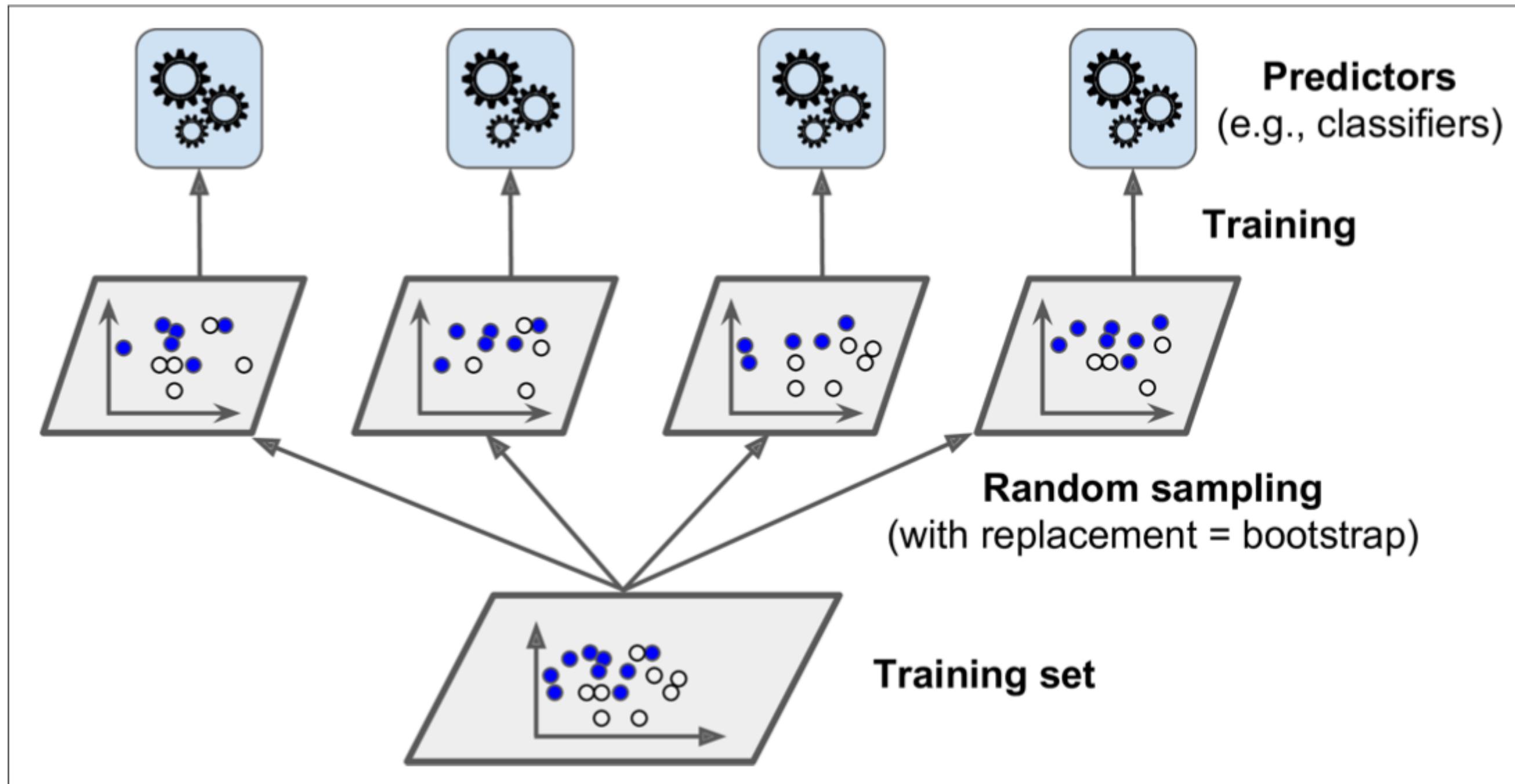


Figure 7-4. Pasting/bagging training set sampling and training

배깅(bagging)과 페이스팅(pasting)

- sklearn.ensemble 패키지 내에 BaggingClassifier 클래스를 활용하여 배깅 알고리즘 활용 가능
- 배깅이 아닌 페이스팅을 사용하고 싶으면, bootstrap=False로 지정

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

배깅(bagging)과 페이스팅(pasting)

- 단일 결정 트리의 결정 경계와 500개의 트리를 사용한 배깅 양상블의 결정 경계 비교
- 양상블의 예측이 결정 트리 하나의 예측보다 일반화가 훨씬 잘된 것을 볼 수 있음

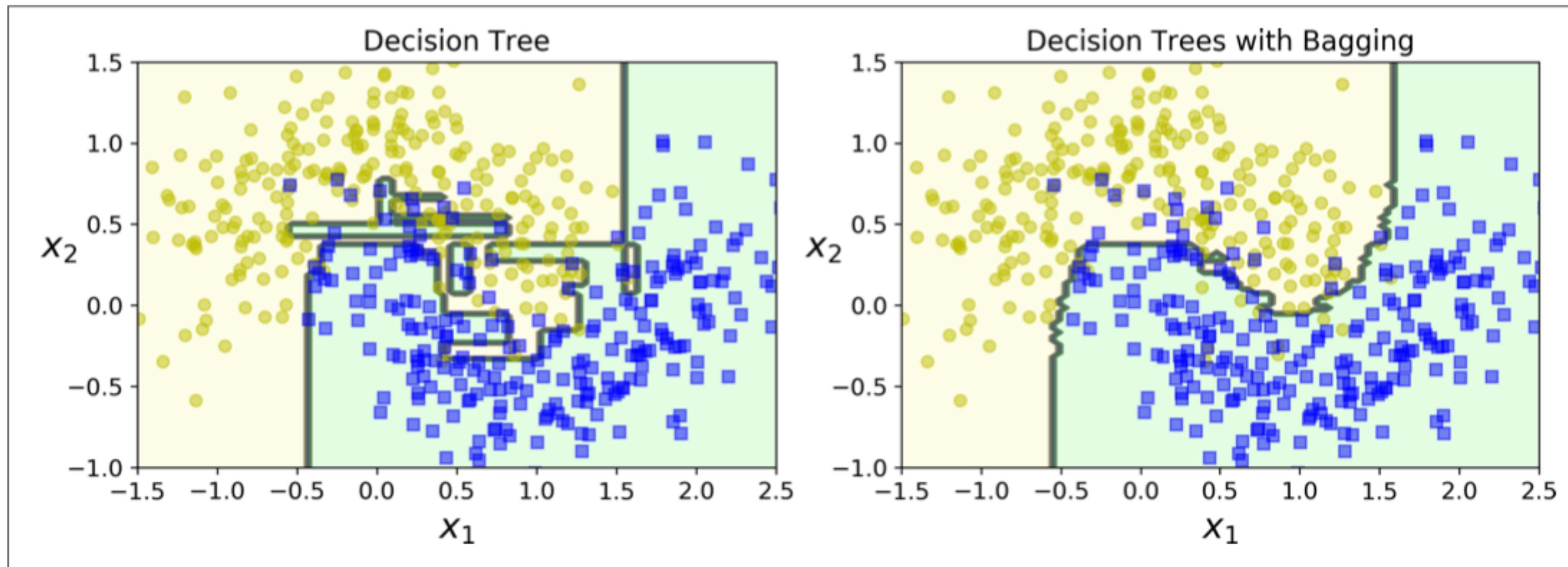


Figure 7-5. A single Decision Tree versus a bagging ensemble of 500 trees

배깅(bagging)과 페이스팅(pasting)

LEVEL.2
SESSION

- 부트스트래핑은 각 예측기가 학습하는 서브셋에 다양성을 증가시키므로 배깅이 페이스팅보다 편향이 조금 더 높음
- 하지만 다양성을 추가한다는 것은 예측기들의 상관관계를 줄이므로 양상블의 분산을 감소시킴
- 전반적으로 배깅이 더 나은 모델을 만들기 때문에 일반적으로 더 선호함
- 시간과 CPU 파워에 여유가 있다면 교차 검증으로 배깅과 페이스팅을 모두 평가해서 더 나은 쪽을 선택하는 것이 좋음



oob(out-of-bag) 평가

- 배깅을 사용하면 어떤 샘플은 한 예측기를 위해 여러 번 샘플링되고 어떤 것은 전혀 선택되지 않을 수 있음
- BaggingClassifier 클래스에서 bootstrap=True로 지정하면 훈련 세트의 크기만큼인 m개 샘플을 선택함
- 이는 평균적으로 전체 샘플에서 63% 정도만 샘플링 된다.
- 이때 나머지 37%를 oob(out-of-bag) 샘플이라고 부름
- 예측기가 훈련되는 동안에는 oob 샘플을 사용하지 않으므로 별도의 검증 세트를 사용하지 않고 oob 샘플을 사용해 평가할 수 있음

BIG.PY

oob(out-of-bag) 평가

- BaggingClassifier 클래스의 인스턴스를 만들 때 oob_score=True로 지정하면 훈련이 끝난 후 자동으로 oob 평가를 수행함

```
bag_clf = BaggingClassifier(  
    DecisionTreeClassifier(), n_estimators=500,  
    bootstrap=True, oob_score=True, random_state=40)  
bag_clf.fit(X_train, y_train)  
print(bag_clf.oob_score_) 0.8986666666666666  
  
from sklearn.metrics import accuracy_score  
y_pred = bag_clf.predict(X_test)  
accuracy_score(y_test, y_pred)  
array([[0.32275132, 0.67724868],  
       [0.34117647, 0.65882353],  
       [1.        , 0.        ],  
       [0.        , 1.        ],  
       [0.        , 1.        ],  
       [0.09497207, 0.90502793],  
       [0.31147541, 0.68852459].
```

랜덤 패치와 랜덤 서브스페이스

- BaggingClassifier는 특성 샘플링도 지원함
 - ✓ max_features, bootstrap_features 두 매개변수로 조절됨
- 이미지와 같은 매우 고차원의 데이터셋을 다룰 때 특히 유용함
- 훈련 특성(feature)과 샘플(sample)을 모두 샘플링하는 것을 랜덤 패치 방식(random patches method)라고 함
 - ✓ bootstrap=False이고 max_samples=1.0으로 설정
- 훈련 샘플을 모두 사용하고 특성만 샘플링하는 것을 랜덤 서브스페이스 방식(random subspaces method)라고 함
 - ✓ bootstrap_features=True 그리고/또는 max_features를 1.0보다 작게 설정
- 특성 샘플링은 더 다양한 예측기를 만들며 편향을 늘리는 대신 분산을 낮춤

랜덤 포레스트(Random Forest)

- 랜덤 포레스트는 일반적으로 배깅 방법(또는 페이스팅)을 적용한 결정 트리의 앙상블임
- 전형적으로 max_samples를 훈련 세트의 크기로 지정함
- sklearn.ensemble 패키지 내에 RandomForestClassifier 클래스 활용

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500,
                                  max_leaf_nodes=16,
                                  random_state=42)

rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

랜덤 포레스트(Random Forest)

LEVEL.2
SESSION

- 랜덤 포레스트 알고리즘은 트리의 노드를 분할할 때 전체 특성 중에서 최선의 특성을 찾는 대신 무작위로 선택한 특성 후보 중에서 최적의 특성을 찾는 식으로 무작위성을 더 주입함
- 이는 트리를 더욱 다양하게 만들고 편향을 손해보는 대신 분산을 낮추어 전체적으로 더 훌륭한 모델을 만들어냄

```
bag_clf = BaggingClassifier(  
    DecisionTreeClassifier(max_features="sqrt",  
                          max_leaf_nodes=16),  
    n_estimators=500,  
    random_state=42)
```

엑스트라 트리(Extra Trees)

LEVEL.2
SESSION

- 트리를 더욱 무작위하게 만들기 위해 후보 특성을 사용해 무작위로 분할한 다음 그중에서 최상의 분할을 선택
- 이와 같이 극단적으로 무작위한 트리의 랜덤 포레스트를 익스트림 랜덤 트리(extremely randomized trees) 양상블(또는 줄여서 엑스트라 트리extra-trees)라고 부름
- 모든 노드에서 특성마다 가장 최적의 임계값을 찾는 것이 가장 시간이 많이 소요되는 작업 중 하나이므로 일반적인 랜덤 포레스트보다 엑스트라 트리가 훨씬 빠름
- sklearn.ensemble 패키지 내에 ExtraTreesClassifier 클래스 활용하며, RandomForestClassifier와 사용법이 같음
- RandomForestClassifier와 ExtraTreesClassifier중에 어떤게 더 나을지는 직접 시도하여 비교해야 함



- 랜덤 포레스트는 특성의 상대적 중요도를 측정하기 쉬움
 - ✓ 결정 트리를 기반으로 하는 모델은 모두 특성 중요도를 제공함
- 사이킷런은 어떤 특성을 사용한 노드가 평균적으로 불순도를 얼마나 감소시키는지 확인하여 특성의 중요도를 측정함
- 사이킷런은 훈련이 끝난 뒤 특성마다 자동으로 이 점수를 계산하고 중요도의 전체 합이 1이 되도록 결과값을 정규화함
 - ✓ 이 값은 feature_importances_ 변수에 저장되어 있음
- 랜덤 포레스트는 특히 특성을 선택해야 할 때 어떤 특성이 중요한지 빠르게 확인할 수 있어 매우 편리함

특성 중요도

- iris 데이터셋에 RandomForestClassifier를 훈련시키고 각 특성의 중요도를 출력하는 코드
- 가장 중요한 특성은 꽃잎의 길이와 너비고, 나머지는 덜 중요해보인다고 해석할 수 있음

```
from sklearn.datasets import load_iris
iris = load_iris()
rnd_clf = RandomForestClassifier(n_estimators=500, random_state=42)
rnd_clf.fit(iris["data"], iris["target"])
for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
    print(name, score)
```

```
sepal length (cm) 0.11249225099876375
sepal width (cm) 0.02311928828251033
petal length (cm) 0.4410304643639577
petal width (cm) 0.4233579963547682
```

- MNIST 데이터셋에 랜덤 포레스트 분류기를 훈련시키고 각 픽셀의 중요도를 그래프로 나타냄

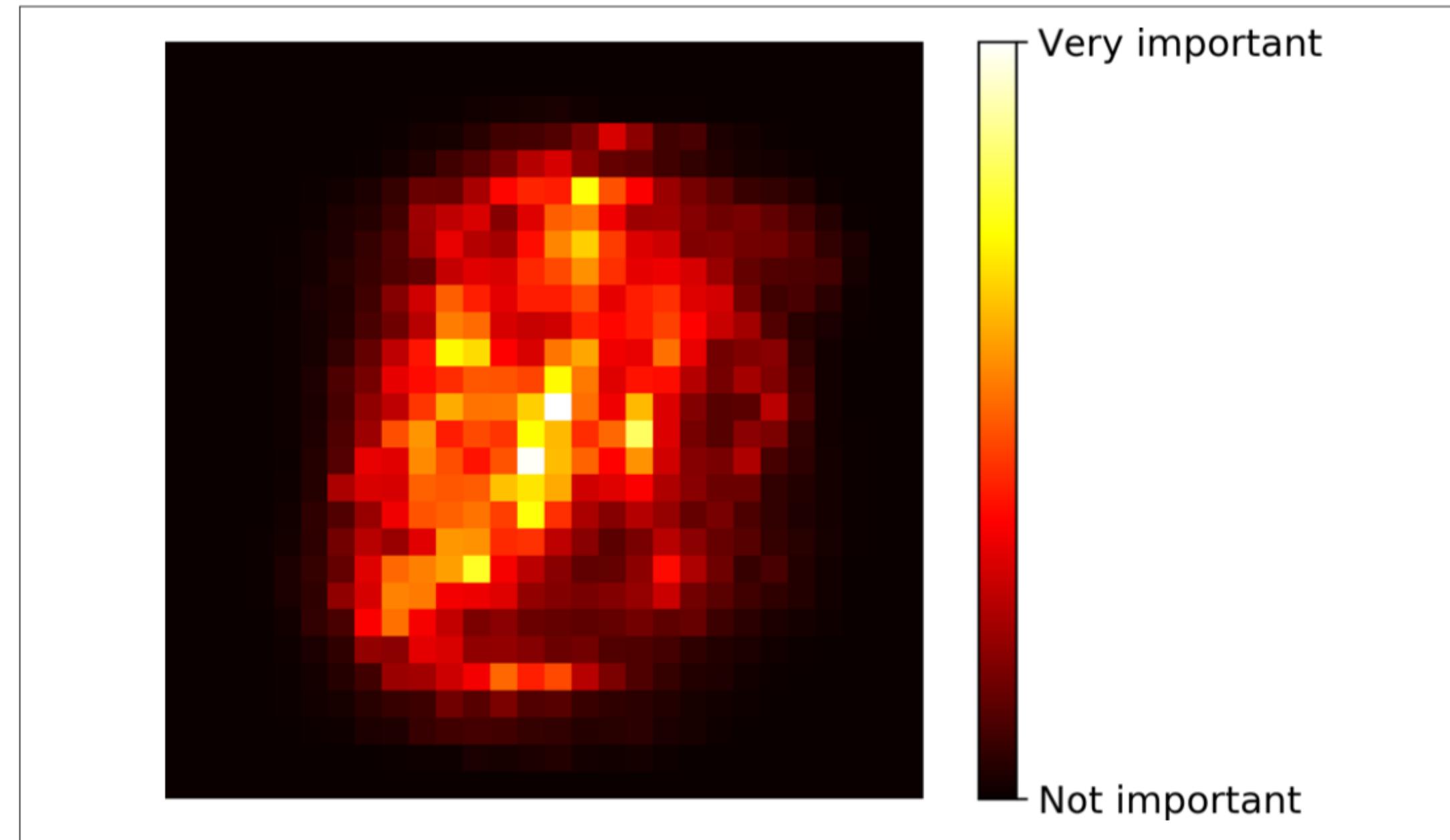


Figure 7-6. MNIST pixel importance (according to a Random Forest classifier)

부스팅(Boosting)

LEVEL.2
SESSION

- 부스팅은 약한 학습기를 여러 개 연결하여 강한 학습기를 만드는 양상별 방법
- 부스팅 방법의 아이디어는 앞의 모델을 보완해나가면서 일련의 예측기를 학습시키는 것
- 가장 인기 있는 것은 에이다부스트(AdaBoost)와 그레디언트 부스팅(gradient boosting)

BIG.PY

에이다부스트(AdaBoost)

- 이전 모델이 과소적합했던 훈련 샘플의 가중치를 더 높혀 이전 예측기를 보완하는 방법
- 이렇게 하면 새로운 예측기는 학습하기 어려운 샘플에 점점 더 맞춰지게 됨
- 각 예측기는 이전 예측기가 훈련되고 평가된 후에 학습될 수 있기 때문에 병렬화를 할 수 없다는 단점이 있음

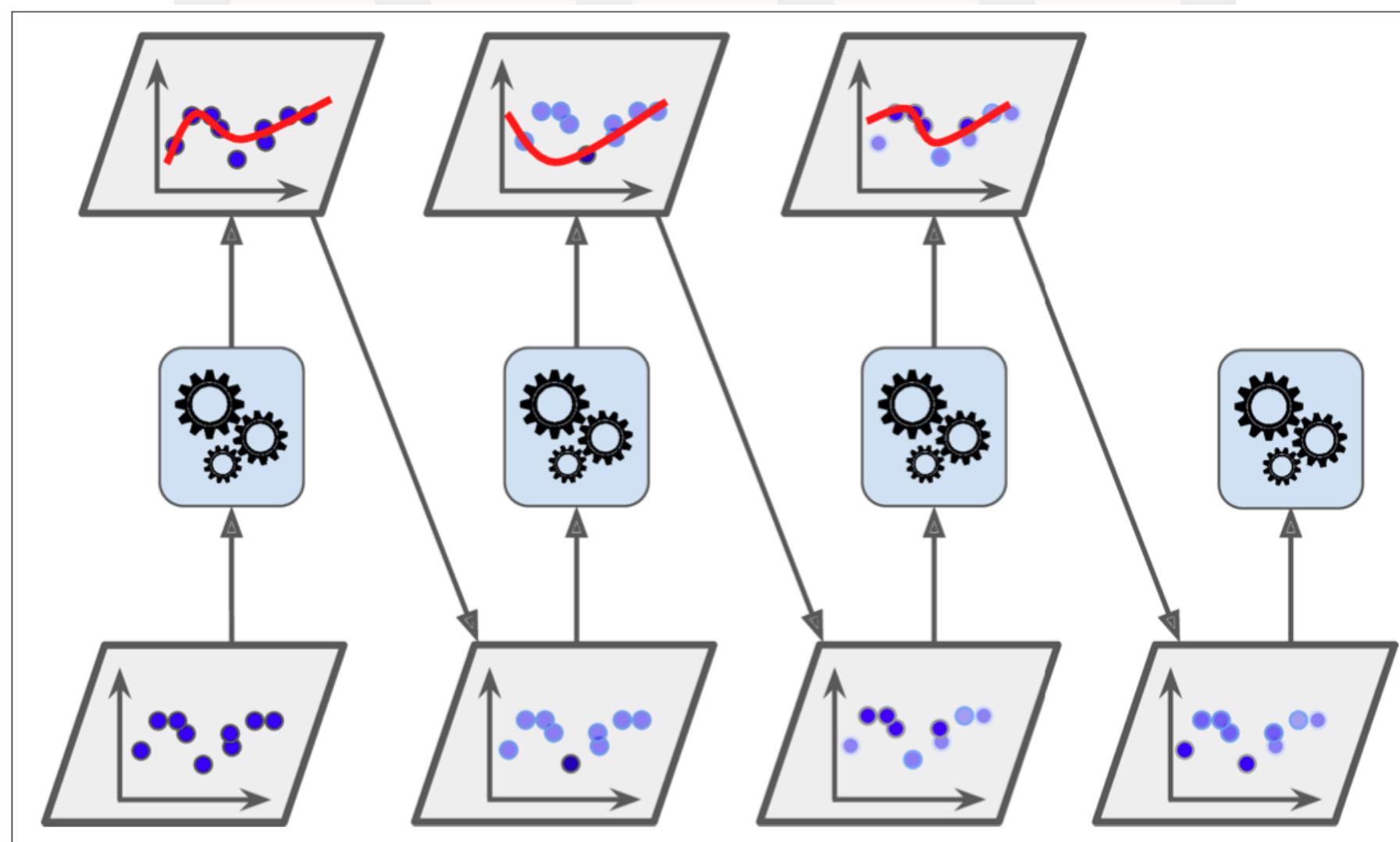


Figure 7-7. AdaBoost sequential training with instance weight updates

에이다부스트(AdaBoost)

LEVEL.2
SESSION

- moons 데이터셋에 훈련시킨 다섯 개의 연속된 예측기의 결정 경계 모습
 - ✓ 모델은 구제를 강하게 한 RBF 커널 SVM 분류기

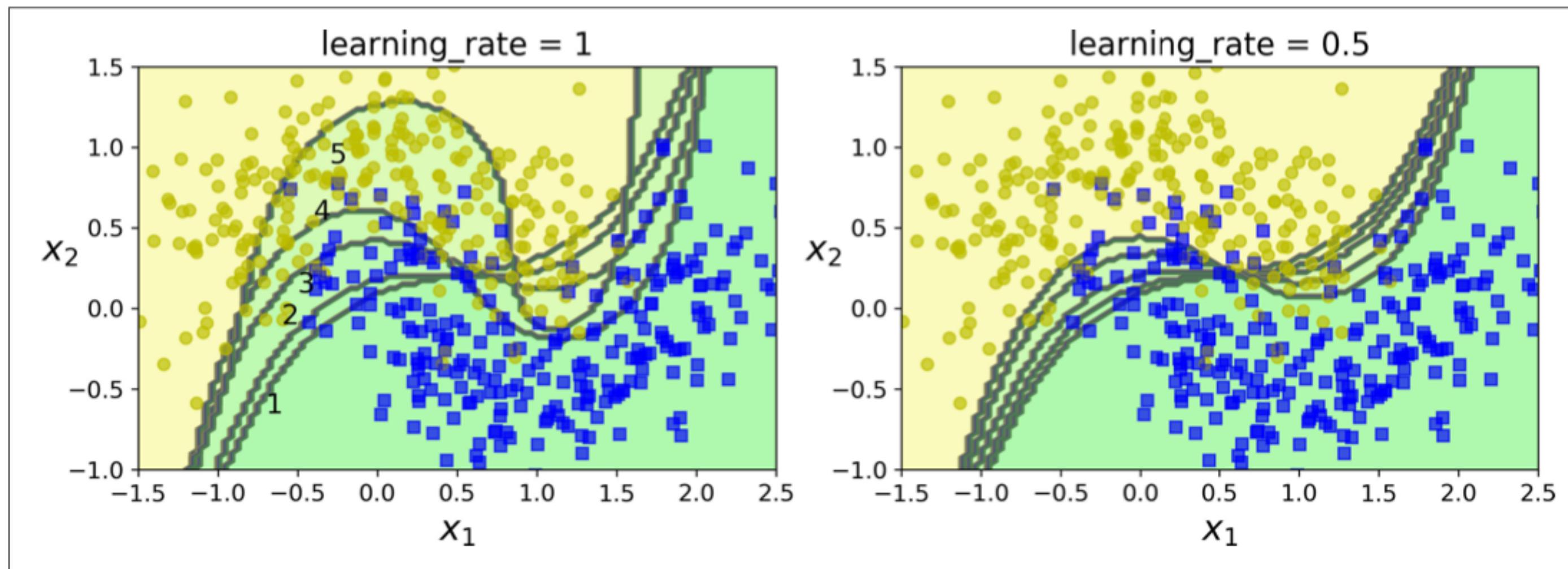
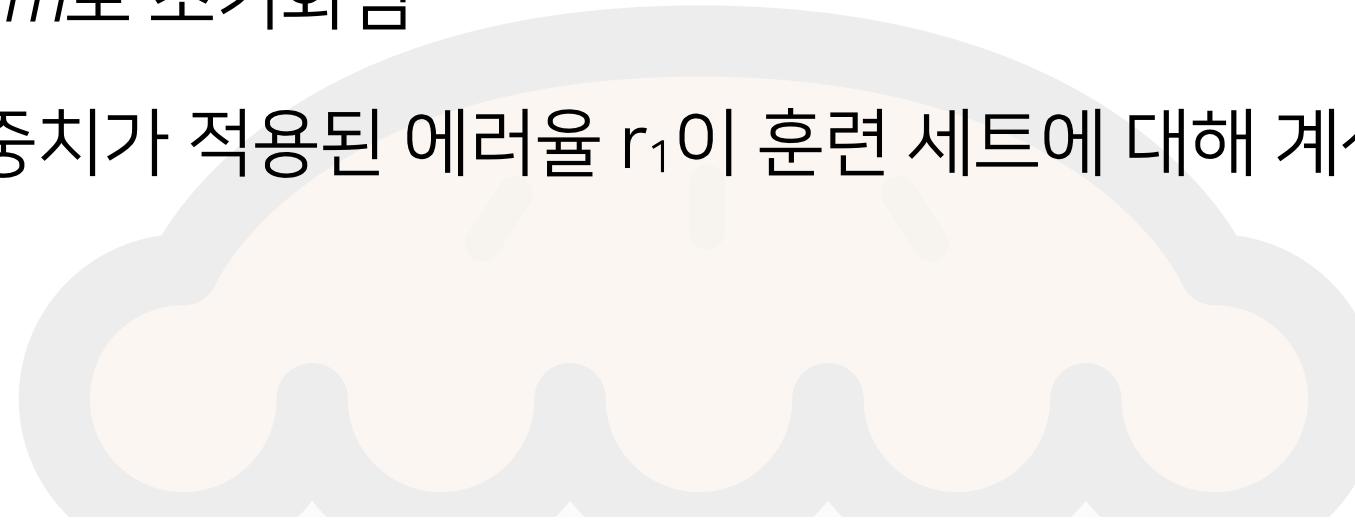


Figure 7-8. Decision boundaries of consecutive predictors

에이다부스트(AdaBoost)

- 각 샘플 가중치 $w^{(i)}$ 는 초기에 $1/m$ 로 초기화됨
- 첫 번째 예측기가 학습되고, 가중치가 적용된 에러율 r_1 이 훈련 세트에 대해 계산됨



Equation 7-1. Weighted error rate of the j^{th} predictor

$$r_j = \frac{\sum_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^m w^{(i)}}{\sum_{i=1}^m w^{(i)}} \quad \text{where } \hat{y}_j^{(i)} \text{ is the } j^{\text{th}} \text{ predictor's prediction for the } i^{\text{th}} \text{ instance.}$$

에이다부스트(AdaBoost)

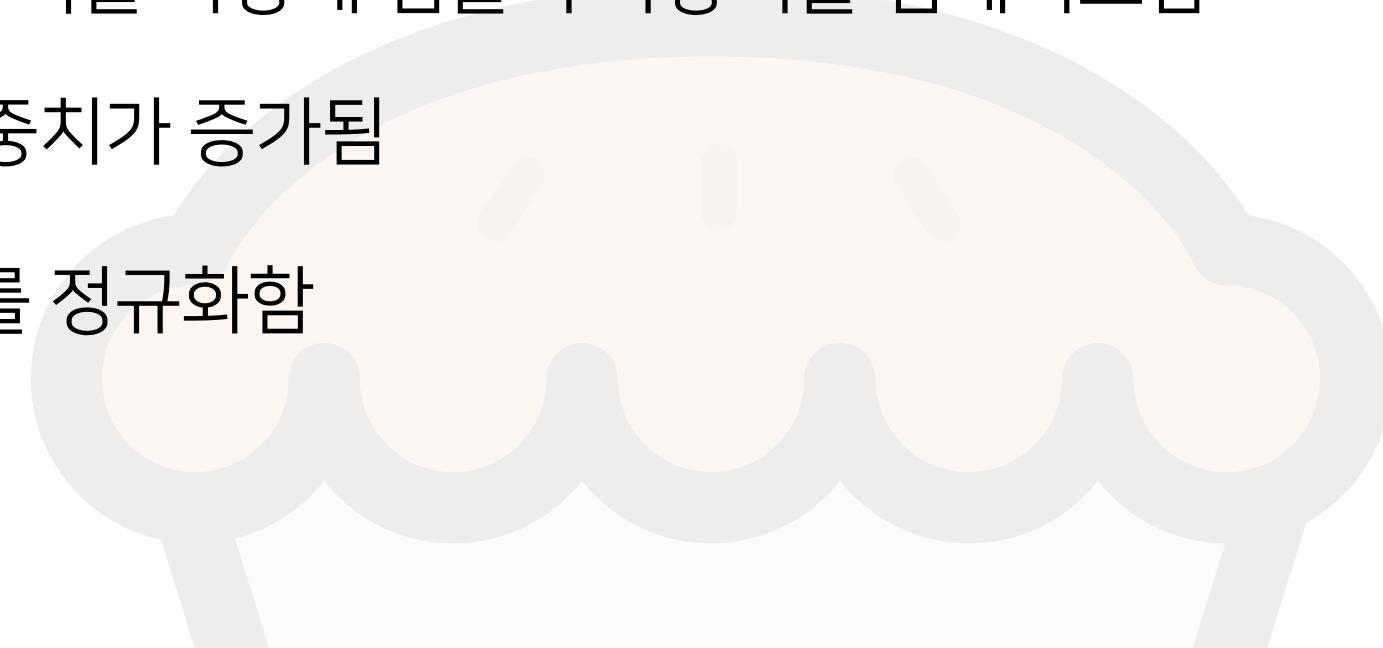
- 예측기의 가중치 a_j 는 아래 식을 사용해 계산됨
 - ✓ 여기서 η 는 학습률 하이퍼파라미터(기본값은 1)
 - ✓ 예측기가 정확할수록 가중치가 더 높아지게 됨
- 만약 무작위로 예측하는 정도라면 가중치가 0에 가깝고, 그보다 더 나쁘면 음수가 될 수 있음

Equation 7-2. Predictor weight

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

에이다부스트(AdaBoost)

- 에이다부스트 알고리즘이 아래 식을 사용해 샘플의 가중치를 업데이트함
 - ✓ 즉, 잘못 분류된 샘플의 가중치가 증가됨
- 그런 다음 모든 샘플의 가중치를 정규화함



Equation 7-3. Weight update rule

for $i = 1, 2, \dots, m$

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} \exp(\alpha_j) & \text{if } \hat{y}_j^{(i)} \neq y^{(i)} \end{cases}$$

에이다부스트(AdaBoost)

- 마지막으로 새 예측기가 업데이트된 가중치를 사용해 훈련되고 전체 과정이 반복됨
 - ✓ 새 예측기의 가중치가 계산되고 샘플의 가중치를 업데이트해서 또 다른 예측기를 훈련시키는 식
- 이 알고리즘은 지정된 예측기 수에 도달하거나 완벽한 예측기가 만들어지면 중지됨
- 예측을 할 때 에이다부스트는 단순히 모든 예측기의 예측을 계산하고 예측기 가중치 α_j 를 더해 예측 결과를 만듬
- 가중치 합이 가장 큰 클래스가 예측 결과가 됨

Equation 7-4. AdaBoost predictions

$$\hat{y}(\mathbf{x}) = \operatorname{argmax}_k \sum_{j=1}^N \alpha_j \quad \text{where } N \text{ is the number of predictors.}$$
$$\hat{y}_j(\mathbf{x}) = k$$

에이다부스트(AdaBoost)

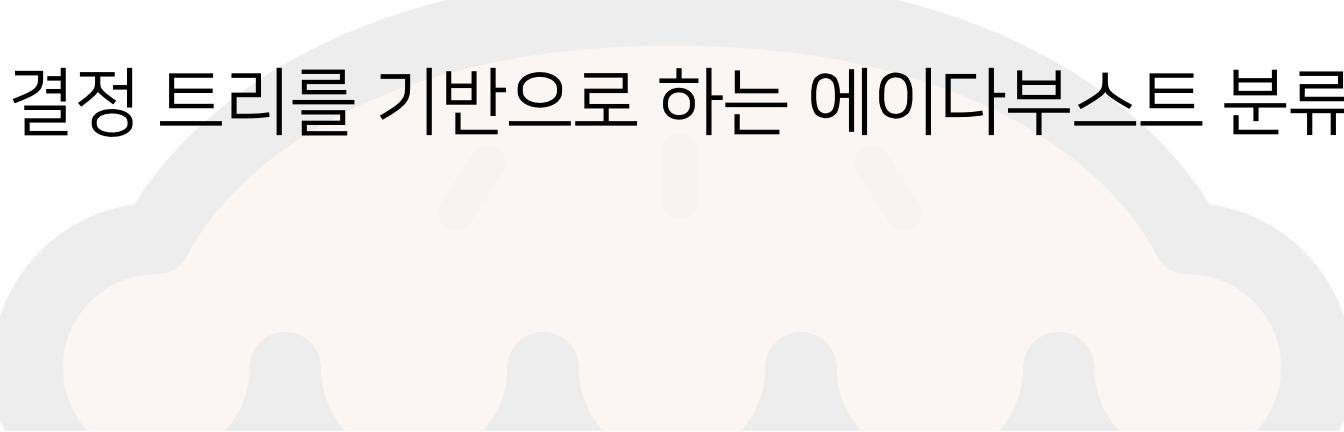
LEVEL.2
SESSION

- 사이킷런은 SAMME라는 에이다부스트의 다중 클래스 버전을 사용함
 - ✓ 클래스가 두 개뿐일 때는 SAMME가 에이다부스트와 동일함
- 만약 예측기가 클래스의 확률을 추정할 수 있다면(predict_proba() 메서드가 있다면) 사이킷런은 SAMME.R이라는 변종 방법을 사용함
 - ✓ 이 알고리즘은 예측값 대신 클래스 확률에 기반하여 일반적으로 성능이 더 좋음

BIG.PY

에이다부스트(AdaBoost)

- sklearn.ensemble 패키지 내에 AdaBoostClassifier 클래스 활용
- 아래 코드는 200개의 아주 얇은 결정 트리를 기반으로 하는 에이다부스트 분류기를 훈련



```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.5, random_state=42)
ada_clf.fit(X_train, y_train)
```

그레디언트 부스팅(gradient boosting)

LEVEL.2
SESSION

- 그레디언트 부스팅은 양상블에 이전까지의 오차를 보정하도록 예측기를 순차적으로 추가함
- 샘플의 가중치를 수정하는 대신, 이전 예측기가 만든 잔여 오차(residual error)에 사로운 예측기를 학습시킴
- 결정 트리를 기반 예측기로 사용하는 것을 그레디언트 트리 부스팅(gradient tree boosting) 또는 그레디언트 부스티드 회귀 트리(gradient boosted regression tree, GBRT) 라고 함

```
from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3,
                                  learning_rate=1.0, random_state=42)
gbrt.fit(X, y)
```

그레디언트 부스팅(gradient boosting)

LEVEL.2
SESSION

- 그레디언트 부스팅을 적용한 예측 그래프
- 트리가 양상블에 추가될수록 양상블의 예측이 점자 좋아지는 것을 알 수 있음

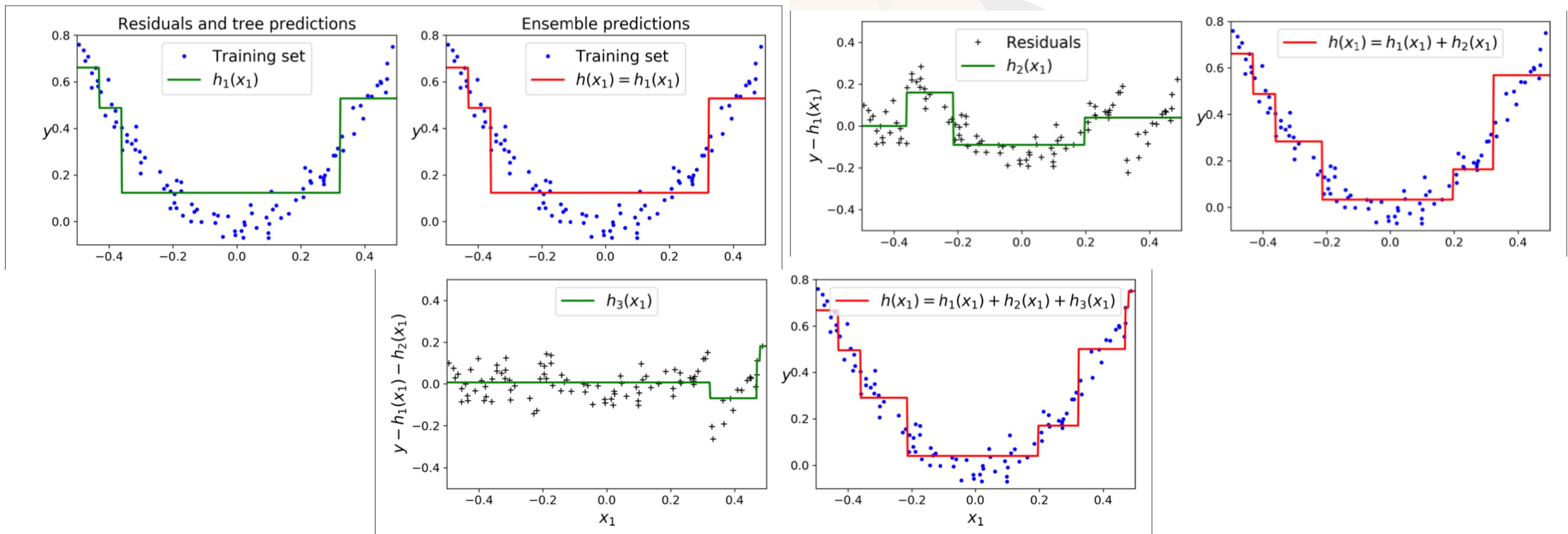


Figure 7-9. Gradient Boosting

그레디언트 부스팅(gradient boosting)

- sklearn.ensemble 패키지 내에 GradientBoostingRegressor 클래스 활용
- learning_rate 매개변수가 각 트리의 기여 정도를 조절함
 - ✓ 이를 0.1처럼 낮게 설정하면 많은 트리가 필요하지만 일반적으로 예측의 성능은 좋아짐
 - ✓ 이를 축소(shrinkage)라고 부르는 규제 방법중 하나임

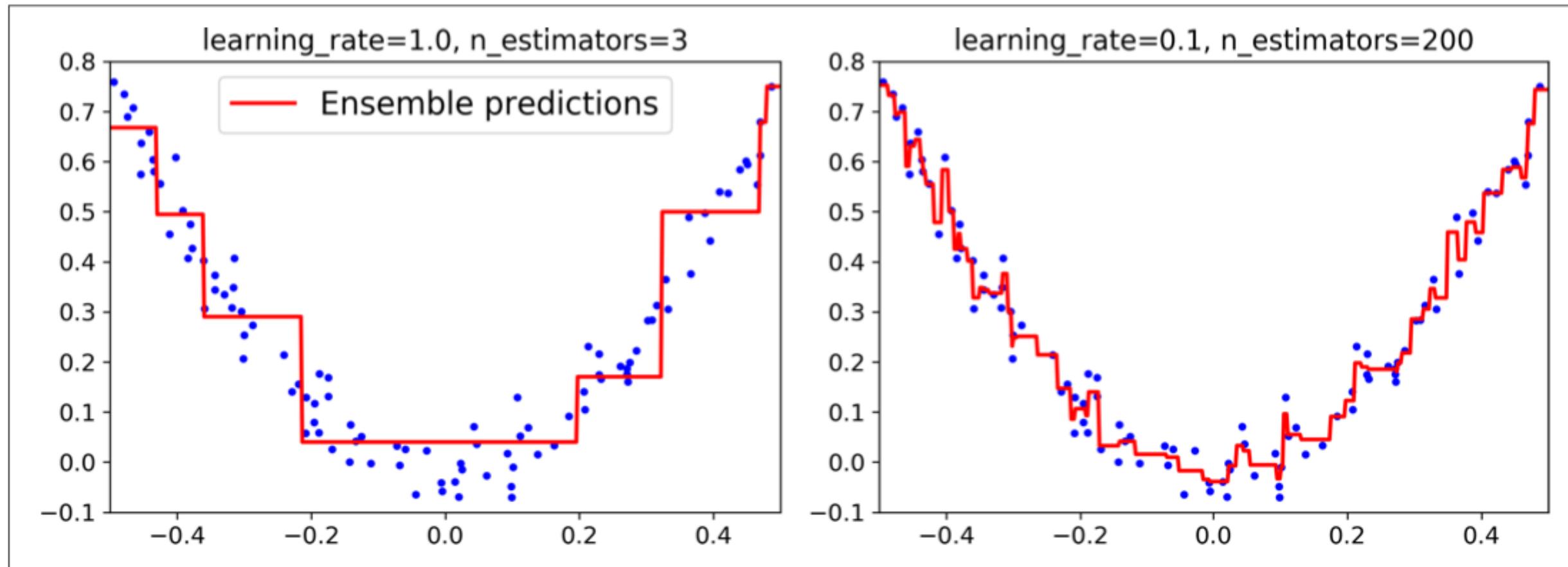


Figure 7-10. GBRT ensembles with not enough predictors (left) and too many (right)

그레디언트 부스팅(gradient boosting)

LEVEL.2
SESSION

- 최적의 트리 수를 찾기 위해서 staged_predict() 메서드를 활용하여 조기 종료 기법을 사용할 수 있음
- 이 메서드는 훈련의 각 단계(트리 하나, 트리 두 개 등)에서 양상브렝 의해 만들어진 예측기를 순회하는 반복자(iterator)를 반환함

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=49)

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=120, random_state=42)
gbrt.fit(X_train, y_train)

errors = [mean_squared_error(y_val, y_pred)
          for y_pred in gbrt.staged_predict(X_val)]
bst_n_estimators = np.argmin(errors) + 1

gbrt_best = GradientBoostingRegressor(max_depth=2,
                                       n_estimators=bst_n_estimators,
                                       random_state=42)
gbrt_best.fit(X_train, y_train)
```

그레디언트 부스팅(gradient boosting)

LEVEL.2
SESSION

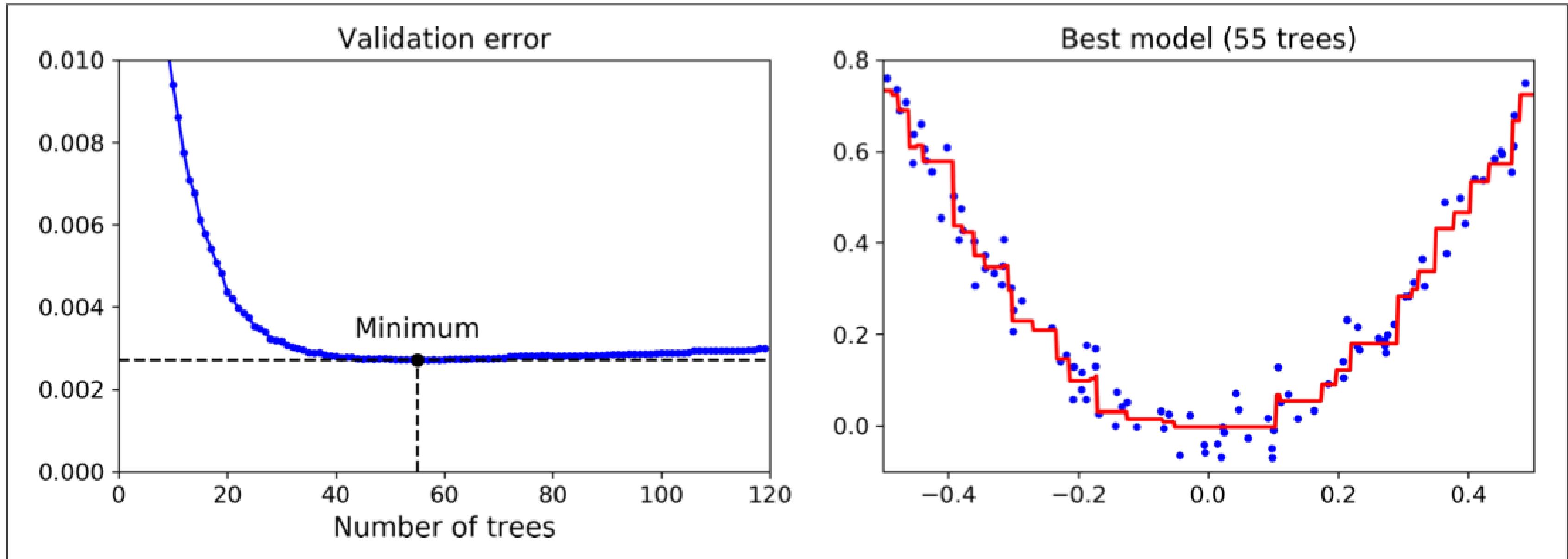


Figure 7-11. Tuning the number of trees using early stopping

그레디언트 부스팅(gradient boosting)

- 실제로 훈련을 중지하는 방법으로 조기 종료를 구현할 수도 있음
- warm_start=True로 설정하면 사이킷런이 fit() 메서드가 호출될 때 기존 트리를 유지하고 훈련을 추가할 수 있도록 해줌

```
gbrt = GradientBoostingRegressor(max_depth=2, warm_start=True, random_state=42)

min_val_error = float("inf")
error_going_up = 0
for n_estimators in range(1, 120):
    gbrt.n_estimators = n_estimators
    gbrt.fit(X_train, y_train)
    y_pred = gbrt.predict(X_val)
    val_error = mean_squared_error(y_val, y_pred)
    if val_error < min_val_error:
        min_val_error = val_error
        error_going_up = 0
    else:
        error_going_up += 1
        if error_going_up == 5:
            break # early stopping
```

그레디언트 부스팅(gradient boosting)

LEVEL.2
SESSION

- GradientBoostingRegressor는 각 트리가 훈련할 때 사용할 훈련 샘플의 비율을 지정할 수 있는 subsample 매개변수도 지원함
- 편향이 높아지는 대신 분산이 낮아지는 효과가 있으며, 훈련 속도를 상당히 높일 수 있음
- 이러한 기법을 확률적 그레디언트 부스팅(stochastic gradient boosting)이라고 함

BIG.PY

그레디언트 부스팅(gradient boosting)

LEVEL.2
SESSION

- 최적화된 그레디언트 부스팅 구현으로 XGBoost 파이썬 라이브러리가 유명함
- 마이크로소프트에서 만든 LightBGM 패키지도 요즘 많이 사용하는 추세

```
import xgboost

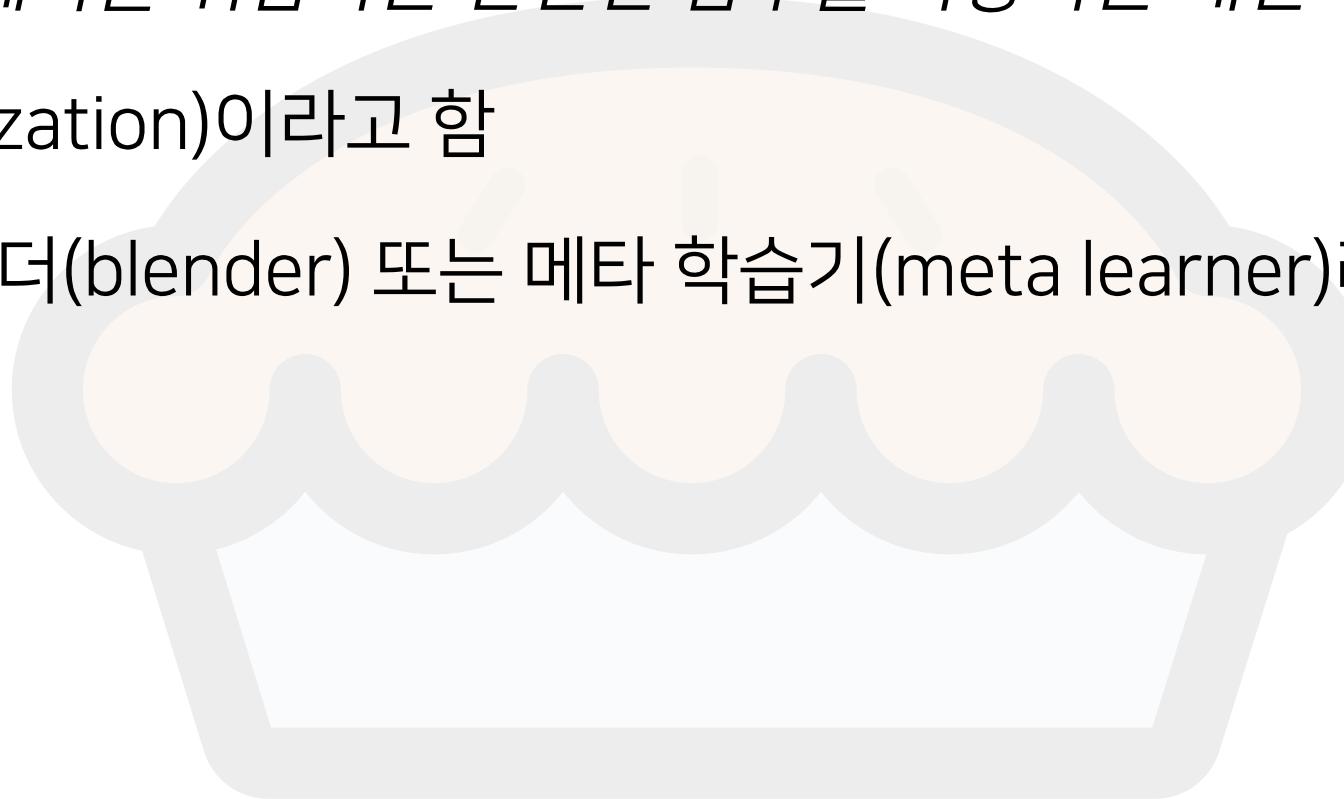
xgb_reg = xgboost.XGBRegressor(random_state=42)
xgb_reg.fit(X_train, y_train)
y_pred = xgb_reg.predict(X_val)

xgb_reg.fit(X_train, y_train, eval_set=[(X_val, y_val)], early_stopping_rounds=2)
y_pred = xgb_reg.predict(X_val)
```

스태킹(stacking)

LEVEL.2
SESSION

- '앙상블에 속한 모든 예측기의 예측을 취합하는 간단한 함수를 사용하는 대신 취합하는 모델을 훈련시킬 수 없을까?'
- 이를 스태킹(stacked generalization)이라고 함
- 예측을 수합하는 예측기를 블렌더(blender) 또는 메타 학습기(meta learner)라고 함



스태킹(stacking)

LEVEL.2
SESSION

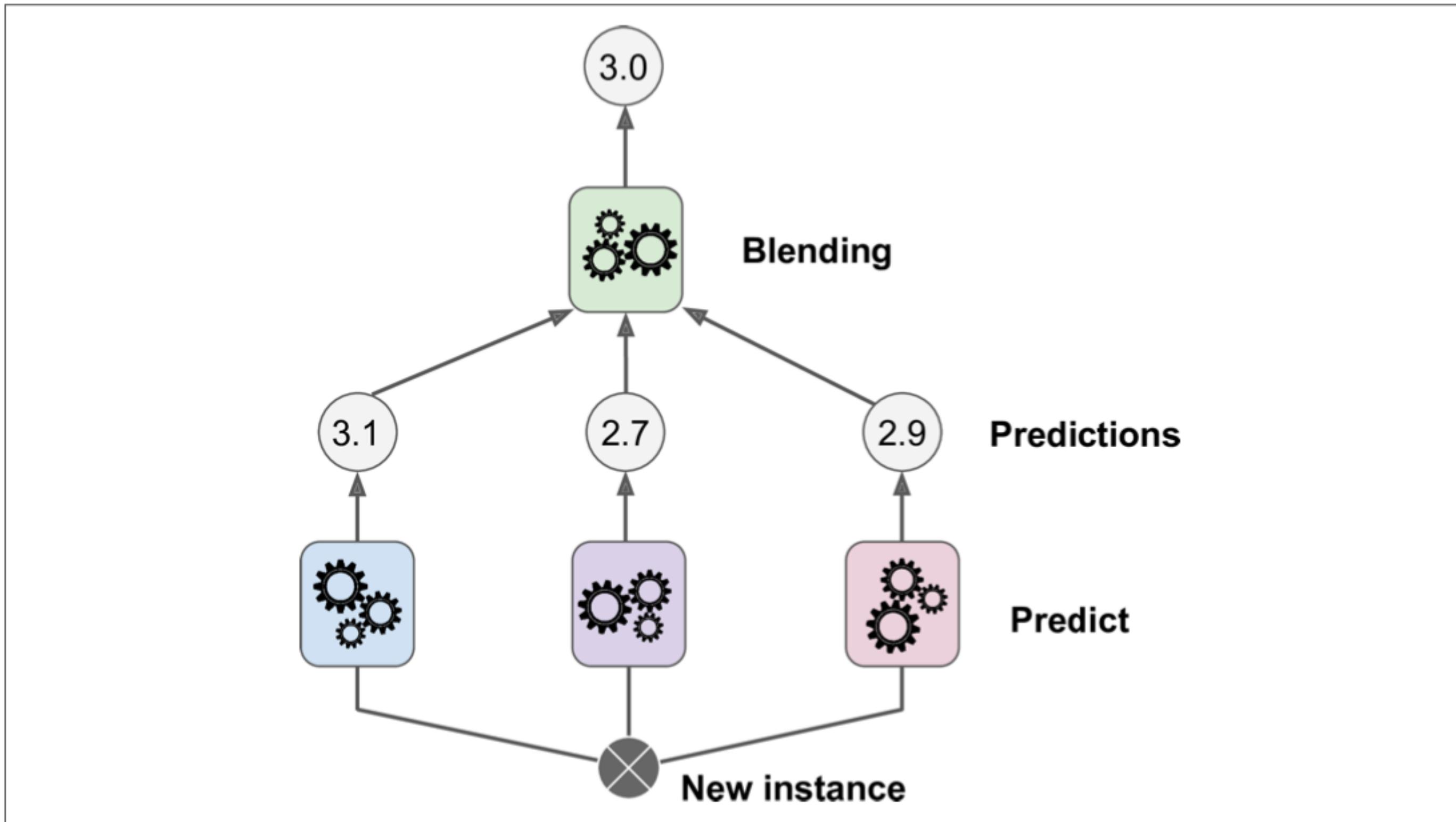


Figure 7-12. Aggregating predictions using a blending predictor

스태킹(stacking)

- 블렌더를 학습시키는 일반적인 방법은 홀드 아웃(hold-out) 세트를 사용하는 것

- ✓ 먼저 훈련 세트를 두 개의 서브셋으로 나눔
- ✓ 첫 번째 서브셋은 첫 번째 레이어의 예측을 훈련시키기 위해 사용됨
- ✓ 첫 번째 레이어의 예측기를 사용해 두 번째 세트에 대한 예측을 만듬
- ✓ 즉, 첫번째 레이어의 예측을 가지고 타깃값을 예측하도록 학습됨

BIG.PY

스태킹(stacking)

LEVEL.2
SESSION

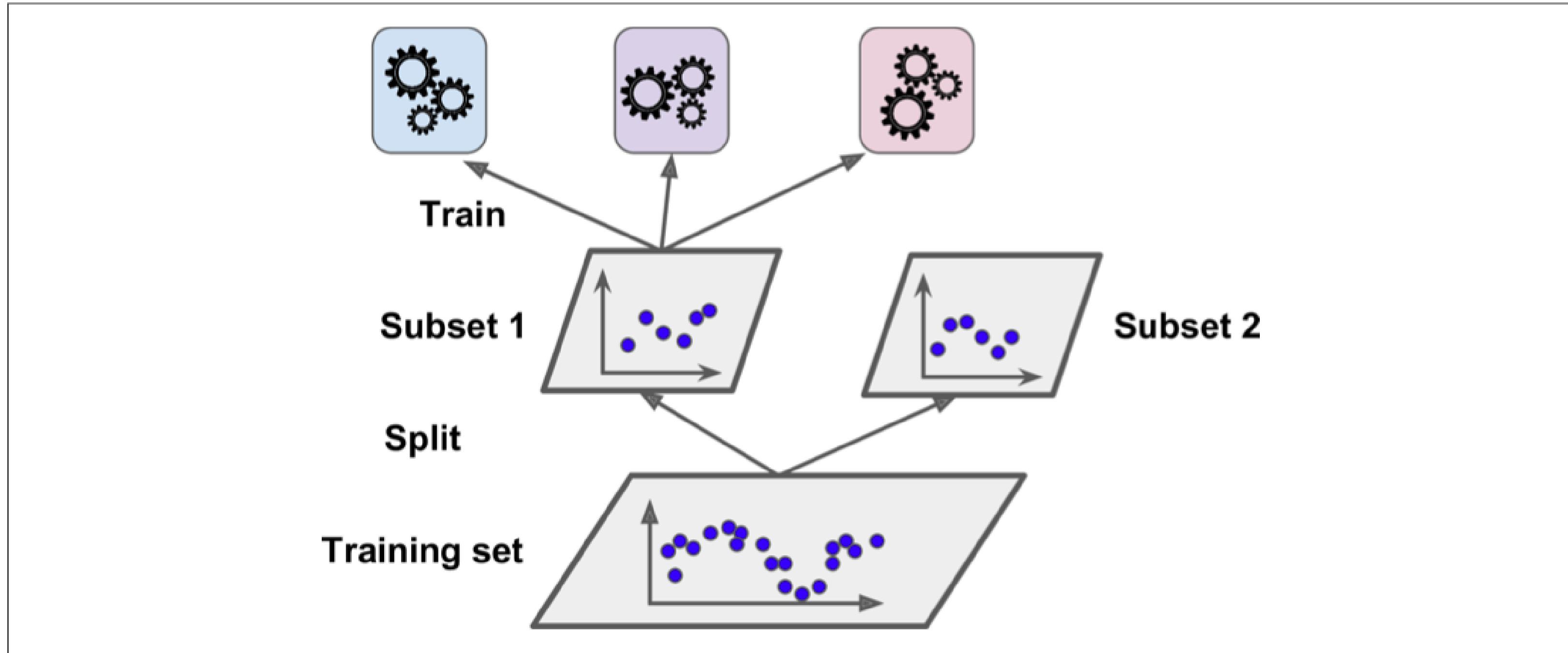


Figure 7-13. Training the first layer

스태킹(stacking)

LEVEL.2
SESSION

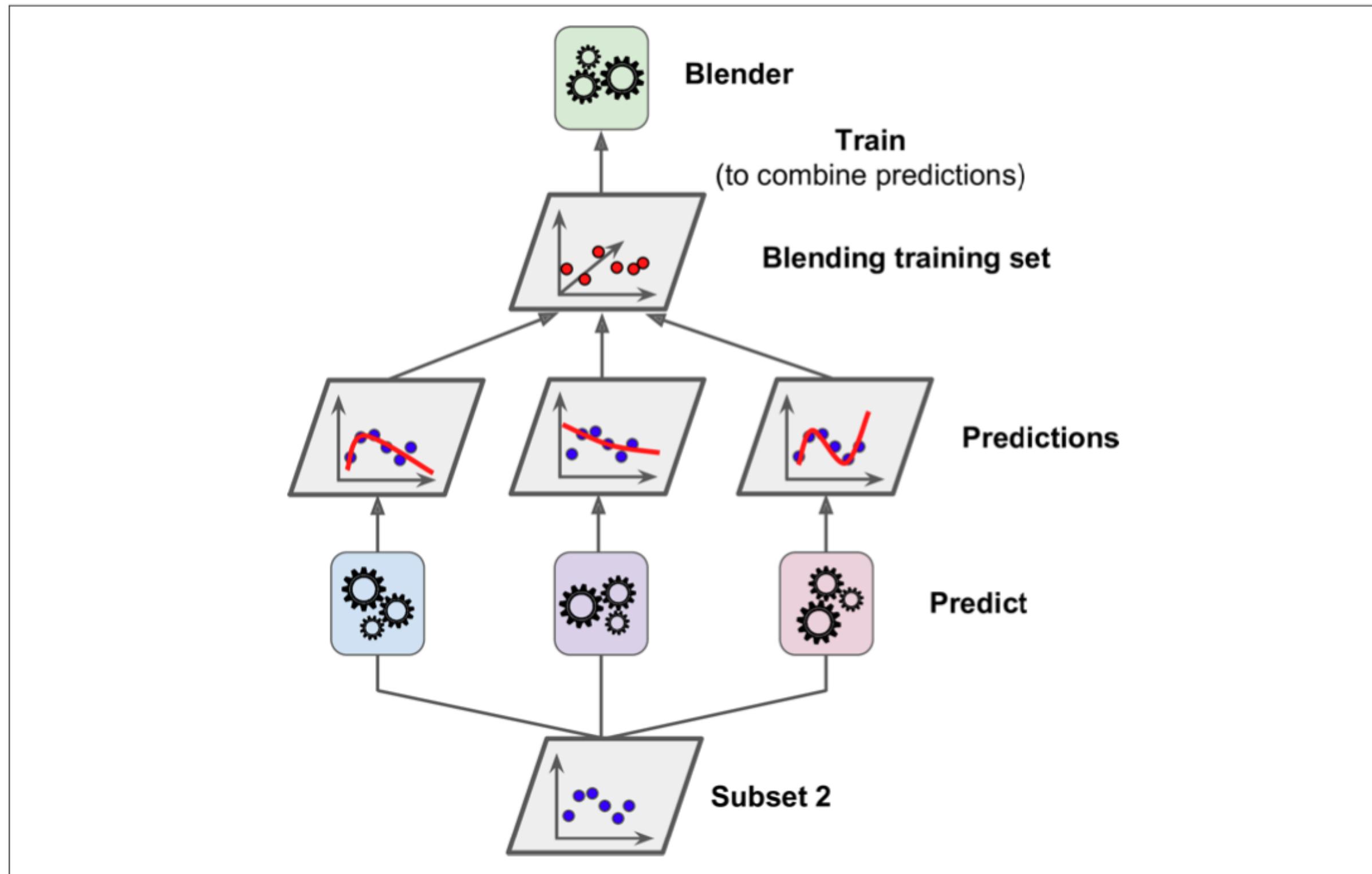


Figure 7-14. Training the blender

스태킹(stacking)

LEVEL.2
SESSION

- 이런 방식의 블렌더를 여러 개 훈련시키는 것도 가능

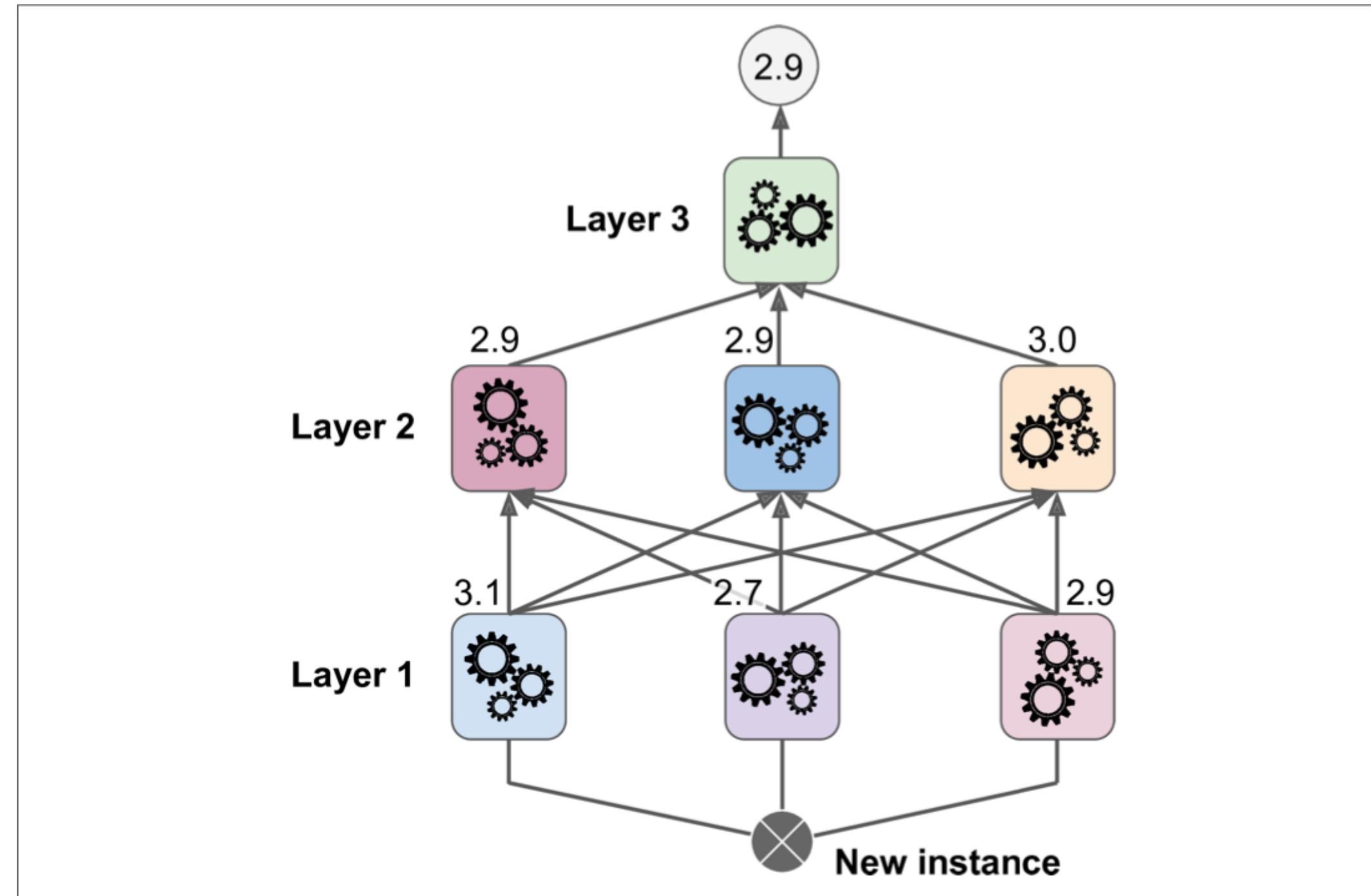


Figure 7-15. Predictions in a multilayer stacking ensemble

