

7장

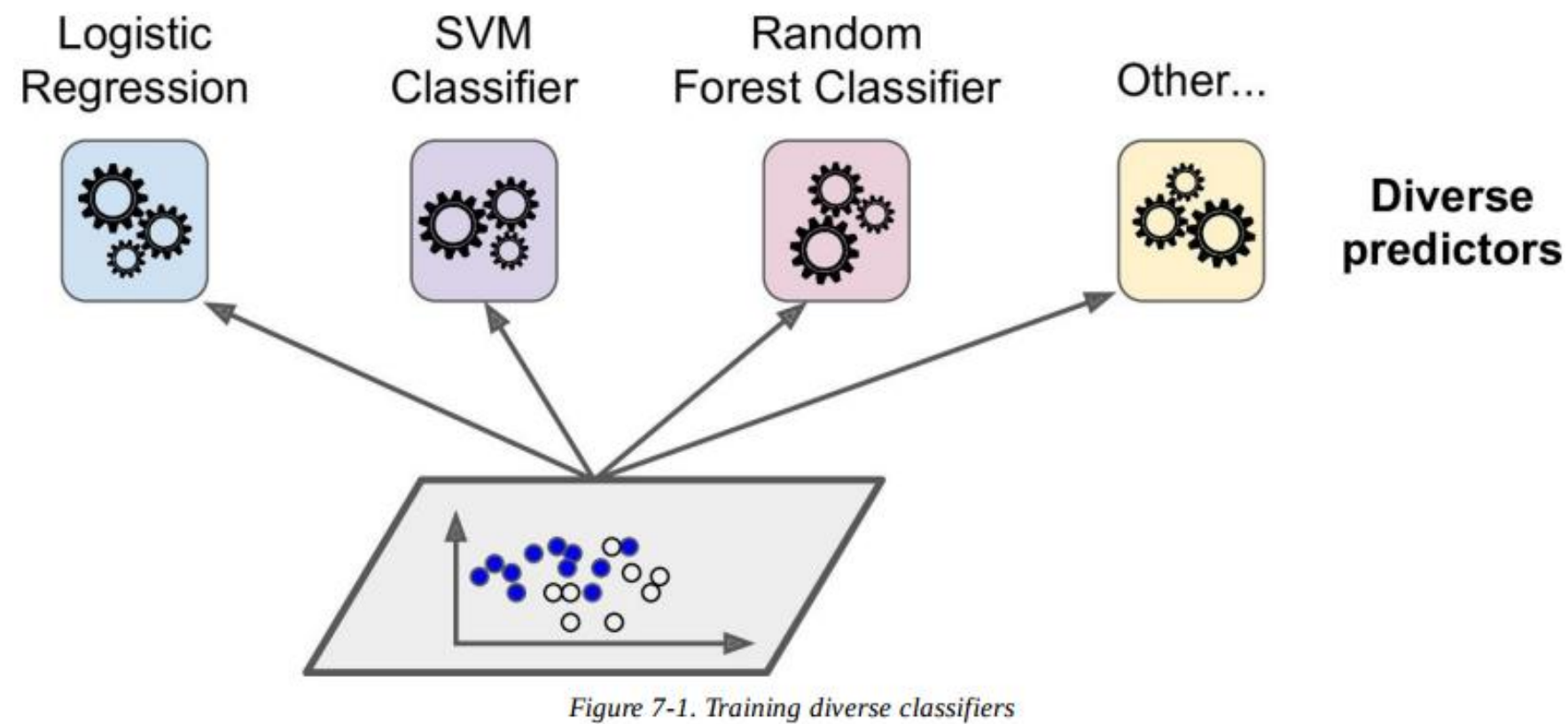
앙상블 학습 (Ensemble Learning) 과

랜덤 포리스트 (Random Forests)

앙상블(Ensemble Method)

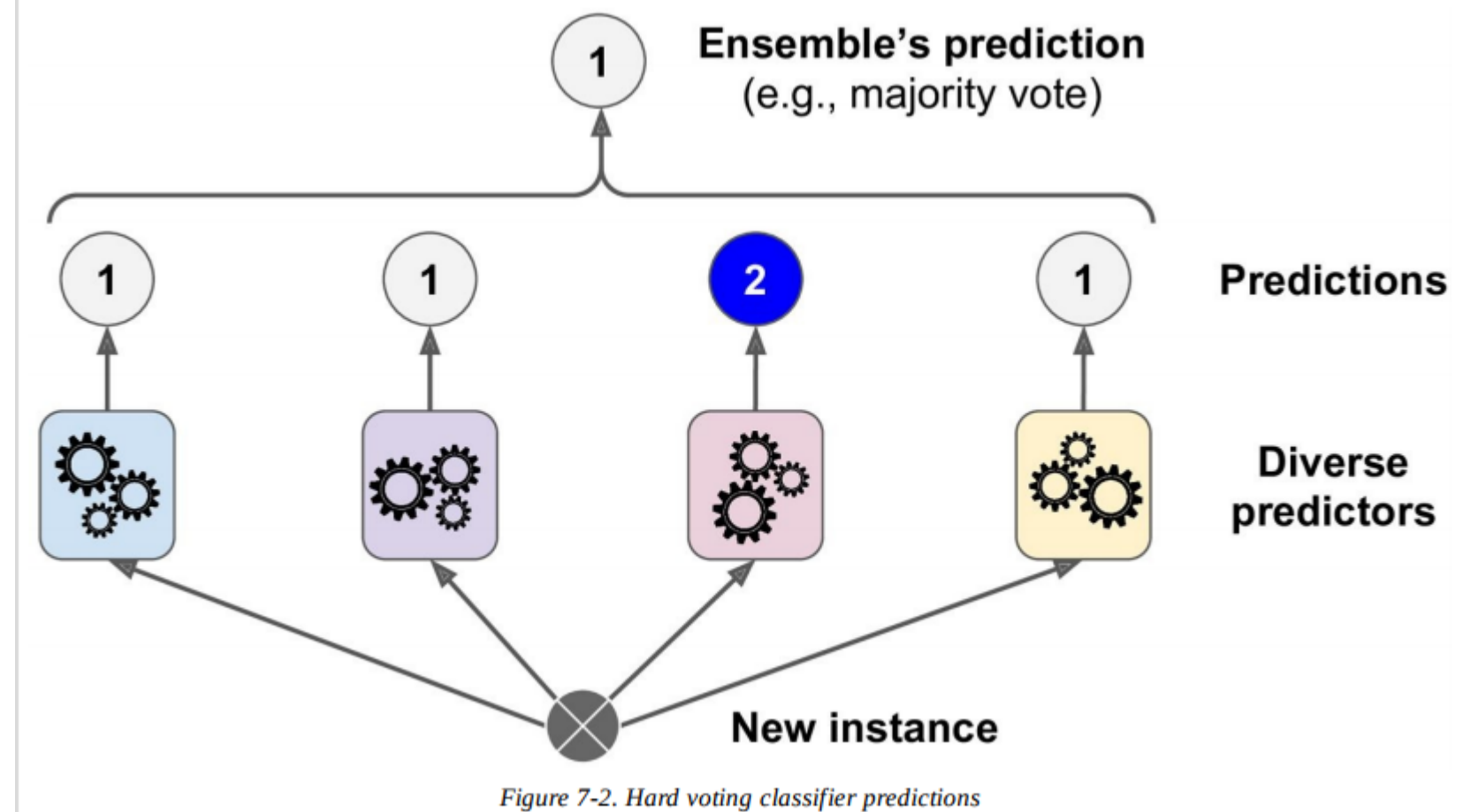
- 앙상블 : 여러 개의 모델을 합친 것
- 여러 개의 모델을 합쳐 일반화 성능 향상 : 대중의 지혜(Wisdom of Crowd)
- 랜덤 포리스트 : 결정트리의 앙상블 → 매우 강력한 머신러닝 방법
- 앙상블 방법
 - 배깅 (bagging) : 중복을 허용하는 샘플링
 - 페이스팅(pasting) : 중복을 허용하지 않는 샘플링
 - 부스팅(boosting) : 이전 예측기의 오차를 보완해서 샘플링
 - 스택킹(stack) : 앙상블 결과 위에 예측을 위한 모델 추가

투표기반 분류기(Voting classifiers)



기존 방법 :

- 다양한 분류기를 학습.
- 정확도 가장 높은 분류기 선택하여 결과 예측



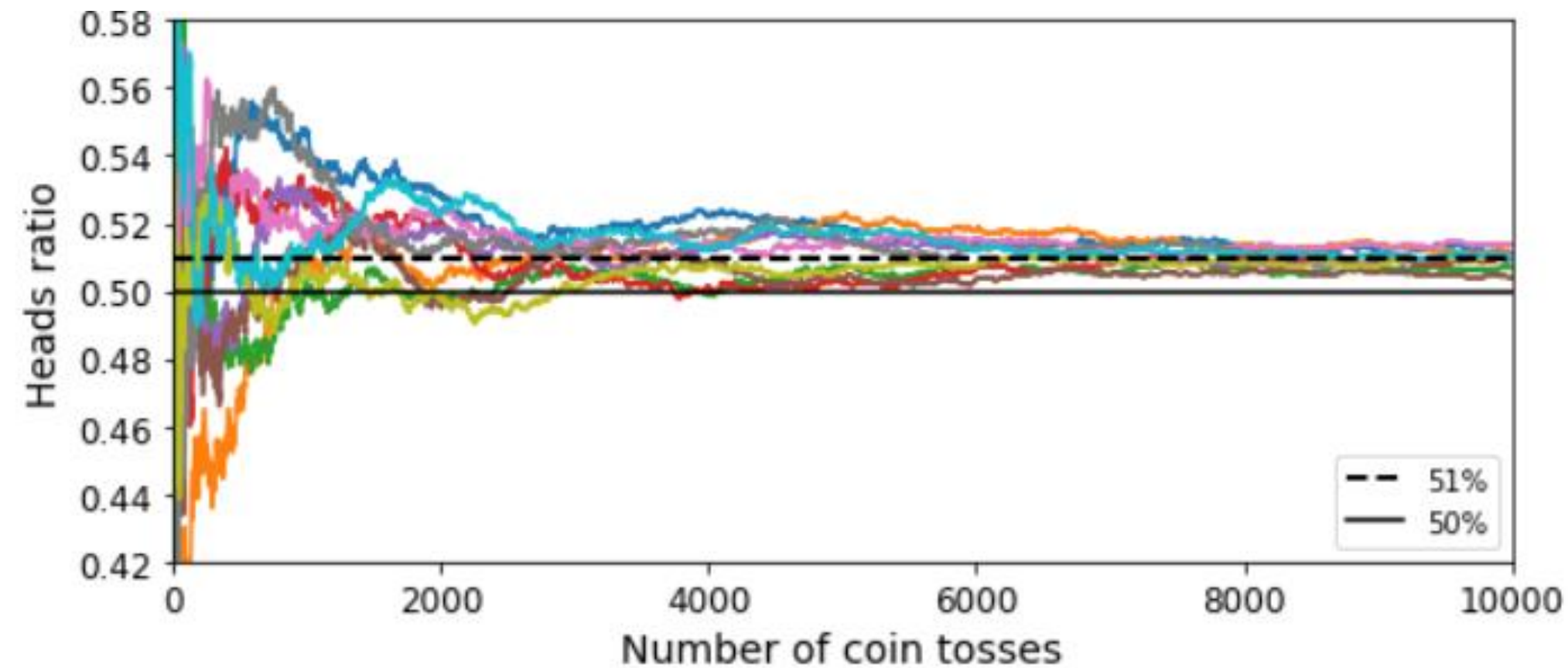
앙상블 방법 :

- 다양한 분류기를 학습.
- 각 분류기 예측 결과를 혼합해서 최종 예측 결과를 결정 (위 예에서는 다수결 투표 했음)

Law of Large Numbers

샘플수가 많아지면 샘플 평균은 모집단 평균과 비슷해진다.

예) 앞면이 나올 확률이 51%인 동전 3개를 동시에 던져
앞면이 나온 비율? 할 때마다 변화 큼
10,000개를 동시에 던져 앞면이 나온 비율? 51%에 근접



예) 앞면이 나올 확률이 51%인 동전 n 개를 동시에 던져 앞면이 더 많이 나오는 확률?

n	3	10	100	1,000	10,000	1,000(앞면 나올 확률 0.55경우)
확률	51.50%	64.74%	61.81%	74.68%	97.78%	99.93%

➔ 정확도가 각각 55%인 분류기 1,000개로 분류한 결과를 다수결로 앙상블한 결과 정확도는 얼마가 되겠는가?

투표기반 분류기 : 프로그램

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

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard'
)
voting_clf.fit(X_train, y_train)
```

```
# 학습, 정확도 계산
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

학습 :

- 3개 분류기 앙상블
- voting='hard' : 다수결 방법(default)

voting='soft' 로 변경하면 :

- 각 분류기의 클래스 확률 이용

분류 결과 :

- 앙상블한 것이 분류기 1개 사용한 것보다 정확도 높음
- voting='soft'로 했으면 정확도 92%.
➔ 'hard'보다 높음

Soft voting : 각 분류기의 클래스 확률을 평균내서 가장 높은 클래스로 분류. 분류기들이 predict_proba()를 사용할 수 있어야 함

배깅(Bagging) 과 페이스팅(Pasting)

- 모델이 같아도 다른 학습 데이터 셋을 사용하여 학습하면 다른 분류기가 만들어 짐
→ 학습 데이터 셋을 다양하게 해서 여러 개 분류기 만들어 앙상블하자
- 원 학습 데이터 셋에서 새로운 학습 데이터 셋을 샘플링
- 배깅(bagging) : 중복을 허용하는 샘플링. Bootstrap Aggregation
 - 통계학에서는 중복을 허용하는 샘플링을 부트스트래핑(bootstrapping)이라고 함*
- 페이스팅(pasting) : 중복을 허용하지 않는 샘플링
- BaggingClassifier : 통계적 최빈값(statistical mode) 계산
- BaggingRegressor : 예측값의 평균 계산

배깅과 페이스팅: 프로그램

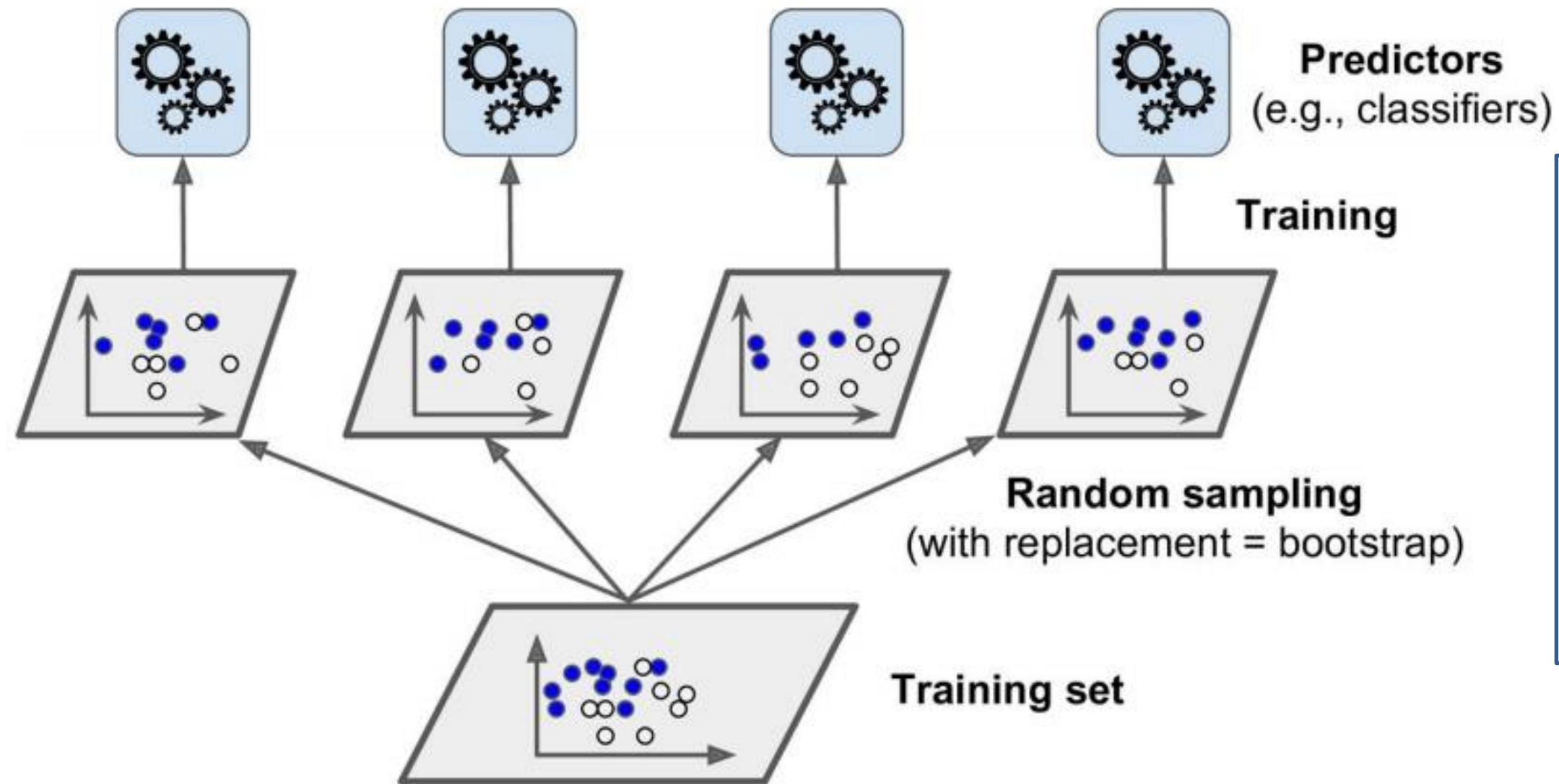
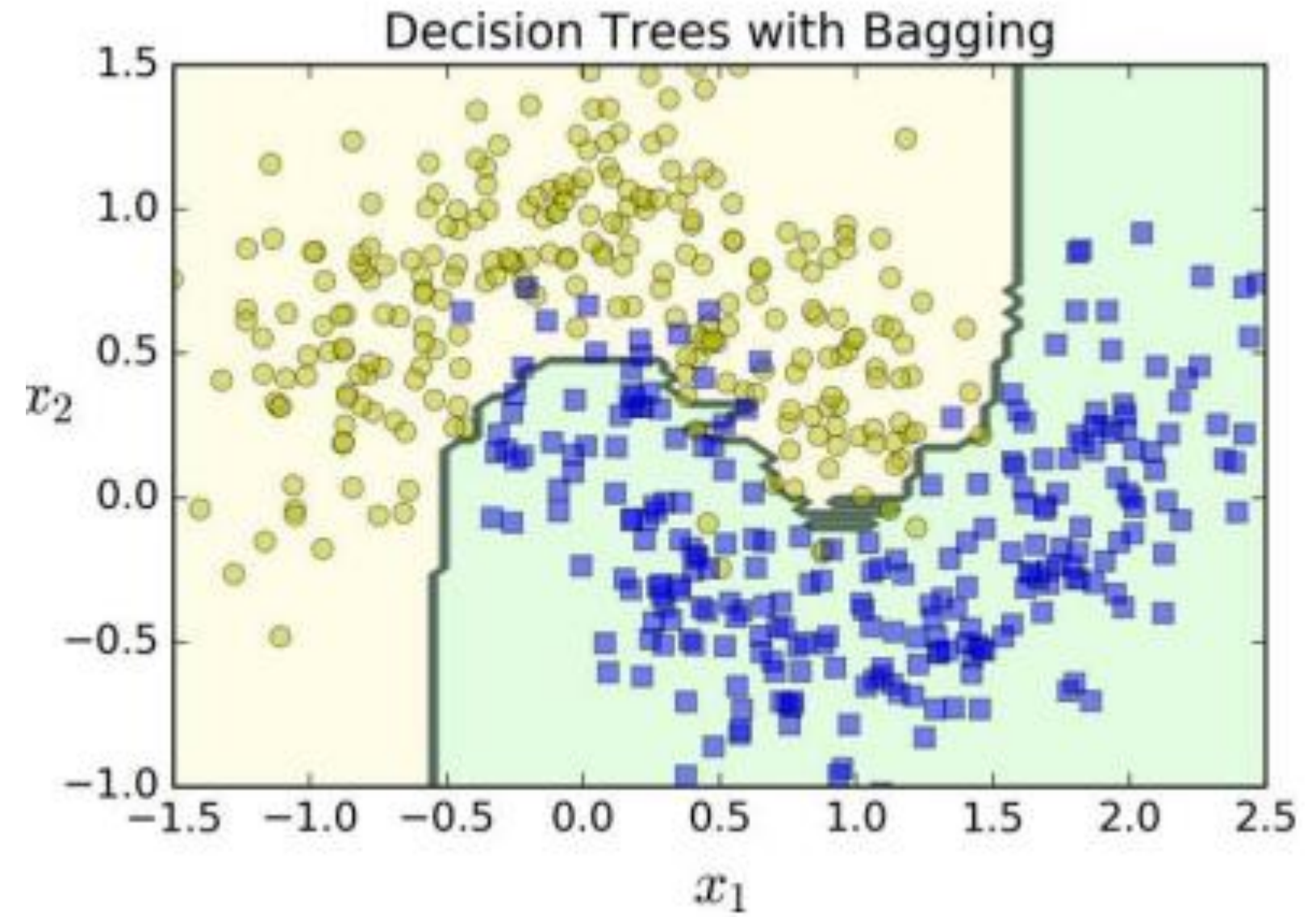
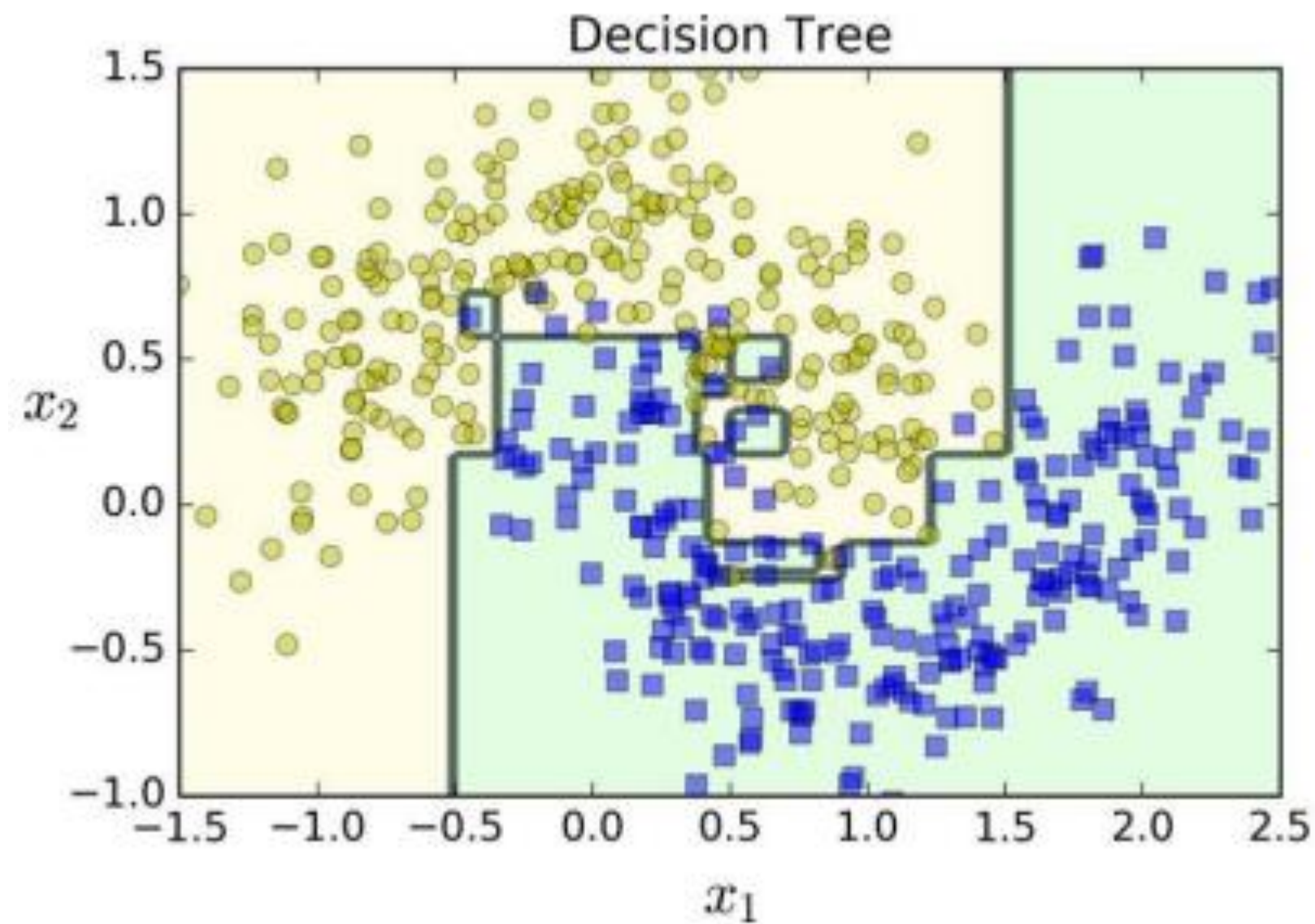


Figure 7-4. Pasting/bagging training set sampling and training

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

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1
)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

n_estimators : 학습할 분류기 개수
max_samples : 무작위로 선택할 샘플 수
bootstrap=True : Bagging (default),
False : Pasting
n_jobs : 사용할 코어수(-1:모두)



한 개 결정트리 \leftrightarrow 500개 결정트리의 앙상블

- 결정경계가 더 부드러운 것은?
- 복잡도가 낮은 것은? 분산이 작은 것은? 편향이 큰 것은?
- 일반화가 잘 되는 것은?
- 예측하는데 시간이 더 걸리는 것은?
- 계산시간이 문제가 안되면 어떤 것을 선택하겠는가?

Out-of-bag evaluation (oob 평가)

1,000개 샘플에서 한번에 하나씩 무작위로 1,000번 꺼냈을 때 (bagging)
한번도 선택이 되지 않은 샘플의 개수는?


한번 샘플링할 때 어떤 샘플이 선택되지 않을 확률: $1 - \frac{1}{1000}$

1,000 번 샘플링할 때 어떤 샘플이 선택되지 않을 확률: $\left(1 - \frac{1}{1000}\right)^{1000} \approx 0.368$

63.2%만 선택됨

- 학습에 한번도 선택이 안된 샘플들 (out-of-bag instances)들을 평가에 사용하자

```
>>> bag_clf = BaggingClassifier(  
>>>     DecisionTreeClassifier(), n_estimators=500,  
>>>     bootstrap=True, n_jobs=-1, oob_score=True)  
>>> bag_clf.fit(X_train, y_train)  
>>> bag_clf.oob_score_  
0.9306666666666666
```



랜덤 서브스페이스(random subspace method) 와 랜덤패치(random patch method)

- 특징이 많은 경우 특징에 대해서도 배깅 가능 : 랜덤 서브스페이스
(샘플에 대해서는 배깅 안 함 : 모든 샘플 사용)

bootstrap=False, max_samples=1.0, **bootstrap_features=True**, max_features=0.3

모든 샘플 사용

특징 30% (변경가능) 배깅

- 샘플과 특징 모두를 배깅 : 랜덤 패치

bootstrap=True, max_samples=0.4, **bootstrap_features=True**, max_features=0.3

샘플 40% (변경가능) 배깅

특징 30% (변경가능) 배깅

랜덤 포리스트 (Random Forest)

- RandomForestClassifier, RandomForestRegressor
- BaggingClassifier + DecisionTreeClassifier와 비슷
 - 노드를 분할할 때 DecisionTreeClassifier는 최적 특징 선택
 - RandomForestClassifier는 임의의 특징 선택

```
from sklearn.ensemble import RandomForestClassifier
```

```
rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1)  
rnd_clf.fit(X_train, y_train)
```

```
y_pred_rf = rnd_clf.predict(X_test)
```

이렇게 하면 RandomForestClassifier와 비슷해 짐

```
bag_clf = BaggingClassifier(  
    DecisionTreeClassifier(splitter="random", max_leaf_nodes=16),  
    n_estimators=500, max_samples=1.0, bootstrap=True, n_jobs=-1)
```

Bootstrap_features = False (default)

엑스트라 트리 (Extra- Trees)

- 극단적으로 무작위한 트리 (Extremely Randomized Tree) 앙상블
- 랜덤 포리스트는 특징 선택은 랜덤, 선택된 특징에 대한 임계값은 최적값 계산
- 엑스트라 트리는 임계값 조차 랜덤 → 학습속도가 빠름
- 사용법은 랜덤포리스트와 비슷
 - RandomForestClassifier → ExtraTreesClassifier
 - RandomForestRegressor → ExtraTreesRegressor

랜덤포리스트 : 특징중요도

- 랜덤 포리스트는 모든 특징에 대해 중요도 알려줌 : `feature_importances_`
특징으로 분할할 때 지니 불순도를 계산하므로, 그 특징이 불순도를 얼마나 낮추는가를 (가중치 평균으로) 계산할 수 있음. (한 노드에서 특징의 중요도 = 노드의 샘플 개수 * 불순도 - 왼쪽 자식노드의 샘플 개수 * 불순도 - 오른쪽 자식노드의 샘플 개수 * 불순도)

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1)
>>> 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) 0.112492250999
sepal width (cm) 0.0231192882825
petal length (cm) 0.441030464364
petal width (cm) 0.423357996355
```

Iris 데이터 특징
중요도

MNIST 각 화소
(특징) 중요도

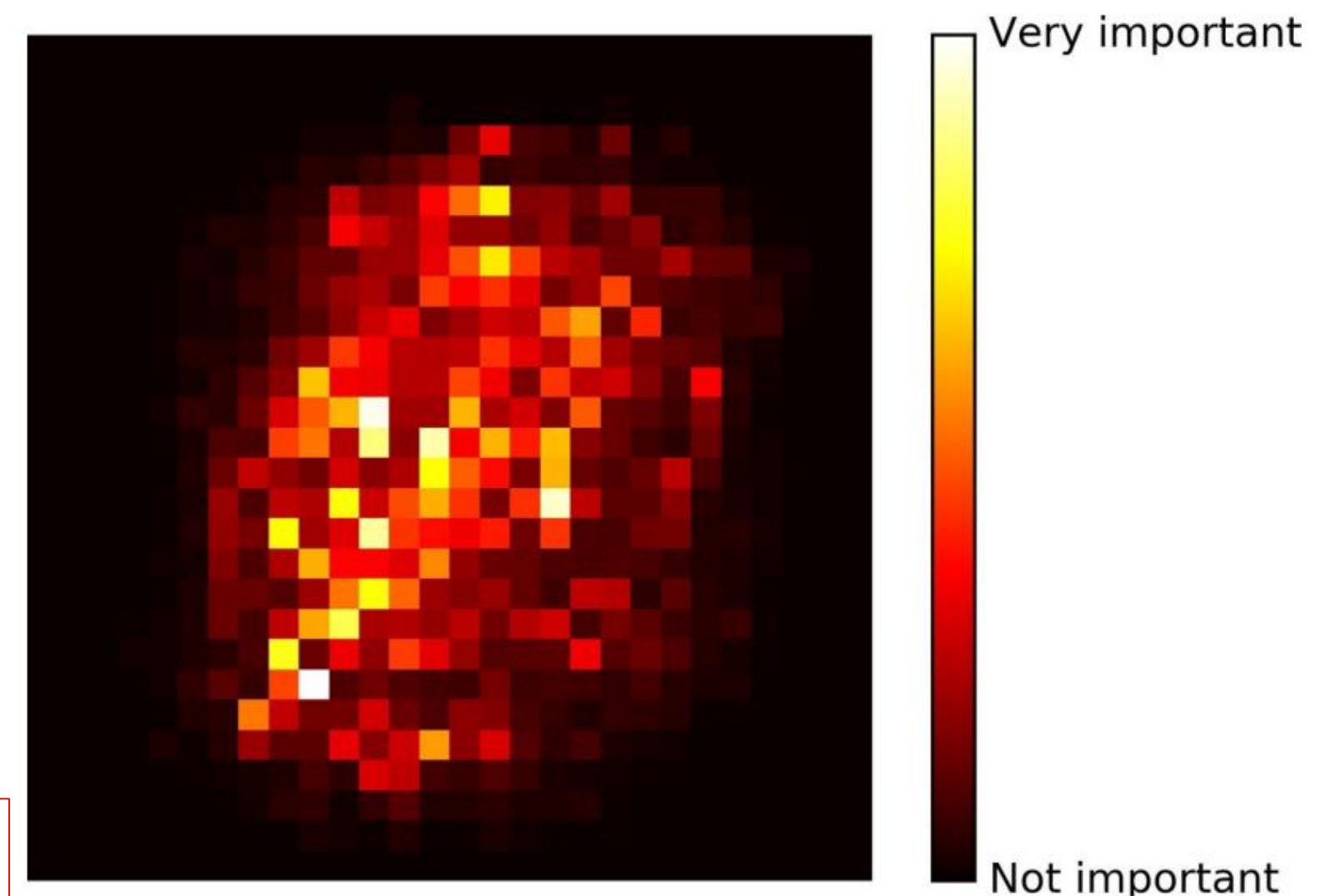


Figure 7-6. MNIST pixel importance (according to a Random Forest classifier)

부스팅 (Boosting)

- 가설 부스팅 (Hypothesis Boosting)
- 약한 학습기를 여러 개 연결해서 강한 학습기를 만드는 앙상블 방법
 - ➔ 앞 모델을 보완해 나가도록 모델을 학습하여 연결해 나감
- 연속해서 모델을 학습해야 하므로 병렬화 하지 못함
- AdaBoost (Adaptive Boosting), Gradient Boosting 이 있음

에이다부스트(아다부스트. AdaBoost)

- 가설 부스팅 (Hypothesis Boosting)
- 이전 모델이 놓친 샘플에 가중치를 크게 해서 다시 학습

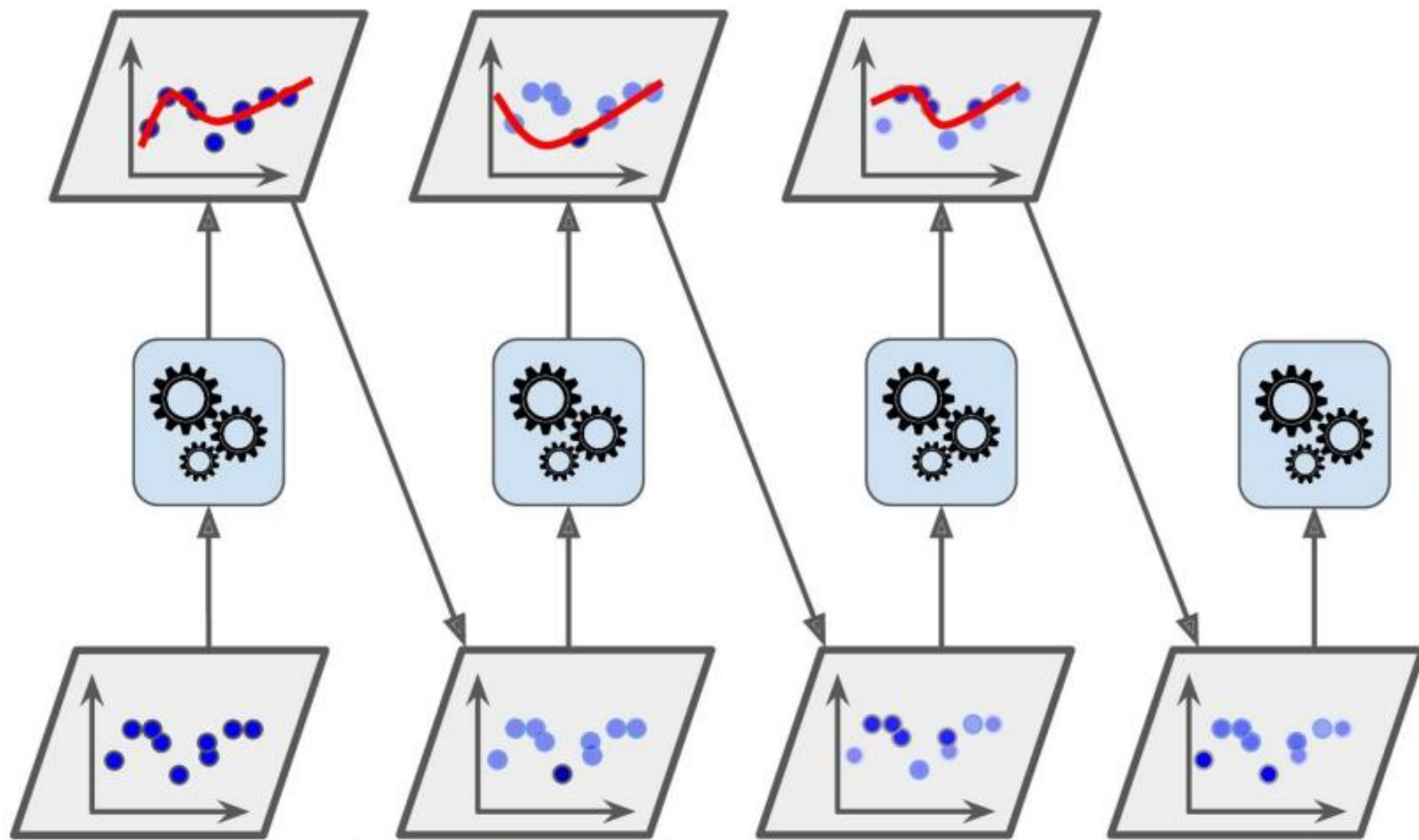
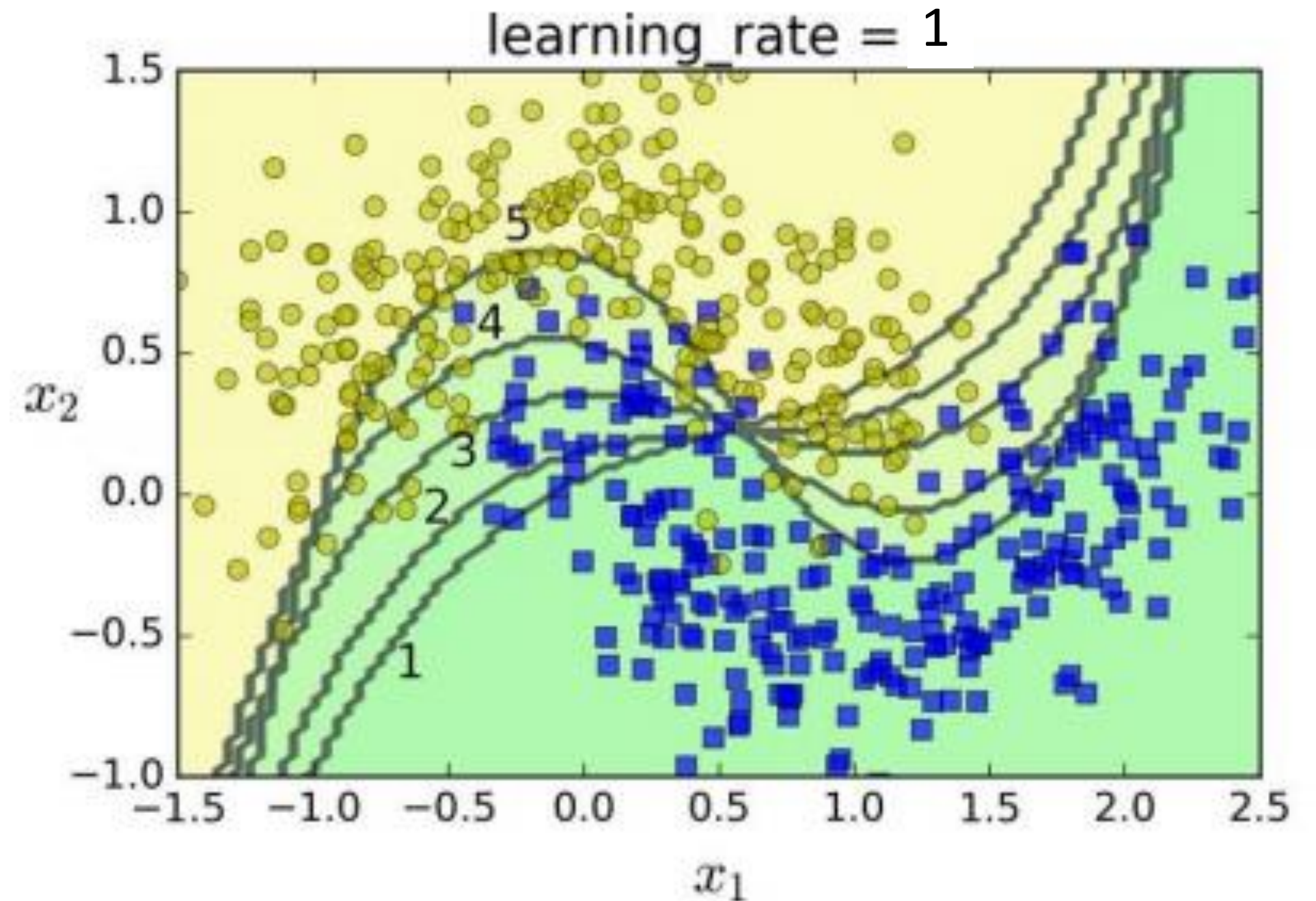


Figure 7-7. AdaBoost sequential training with instance weight updates



- 1번째 모델에서 틀린 샘플에 대해 가중치 주어 2번째 모델 학습...
- 계속하면 오차가 작은 방향으로 찾아감
➔ 경사하강법과 비슷한 모양

아다부스트 알고리즘

가중치 적용된 에러율(w 초깃값은 1/m)

$$r_j = \frac{\sum_{i=1}^m w^{(i)} \mathbb{1}_{\hat{y}_j^{(i)} \neq y^{(i)}}}{\sum_{i=1}^m w^{(i)}}$$

j번째 예측기의
에러, 가중치

예측기 가중치 계산

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

사이킷런의 아다부스트 = SAMME의 이진 분류(K=2) 버전

$$\alpha_j = \eta \left(\log \frac{1-r_j}{r_j} + \log(K-1) \right)$$

predict_proba() 메서드가 있을 때: SAMME.R 알고리즘 사용

$$\alpha_j = -\eta \frac{K-1}{\nu} y \log \hat{y}_j$$

$$\hat{y}(x) = \operatorname{argmax}_k \sum_{j=1}^N (K-1) \left(\log \hat{y}_j - \frac{1}{K} \sum_{k=0}^K \hat{y}_j \right)$$

가중치 업데이트

i번째 샘플의
가중치

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

여기서 $i = 1, 2, \dots, m$

예측

$$\hat{y}(\mathbf{x}) = \operatorname{argmax}_k \sum_{\substack{j=1 \\ \hat{y}_j(\mathbf{x})=k}}^N \alpha_j$$

여기서 N은 예측기 수

변경된 가중치로 새 예측기 학습

AdaBoostClassifier의 algorithm 매개변수 기본값이 'SAMME.R'이고 'SAMME'로도 지정할 수 있음

아다부스트 : 프로그램

- 사이킷런 : SAMME(Stagewise Additive Modeling using Multiclass loss function) 사용.
다중 클래스 지원
- 예측기가 클래스 확률을 추정할 수 있으면 SAMME.R 을 사용하는 것이 성능 향상됨

```
from sklearn.ensemble import AdaBoostClassifier

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

아주 얇은 결정트리.
AdaBoostClassifier의 기본 예측기

그래디언트 부스팅 (Gradient Boosting)

- 이전까지의 오차를 보정하도록 예측기를 추가 : 아다부스트와 동일
- 잔여오차(residual error)에 새로운 예측기 학습 :
아다부스트는 샘플에 가중치 부여 (오차가 큰 샘플에 큰 가중치)

```
from sklearn.tree import DecisionTreeRegressor
```

```
tree_reg1 = DecisionTreeRegressor(max_depth=2)  
tree_reg1.fit(X, y)
```

```
y2 = y - tree_reg1.predict(X) 잔여오차  
tree_reg2 = DecisionTreeRegressor(max_depth=2)  
tree_reg2.fit(X, y2)
```

```
y3 = y2 - tree_reg2.predict(X) 잔여오차  
tree_reg3 = DecisionTreeRegressor(max_depth=2)  
tree_reg3.fit(X, y3)
```

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

오른쪽 코드와 같은 결과

```
from sklearn.ensemble import GradientBoostingRegressor
```

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

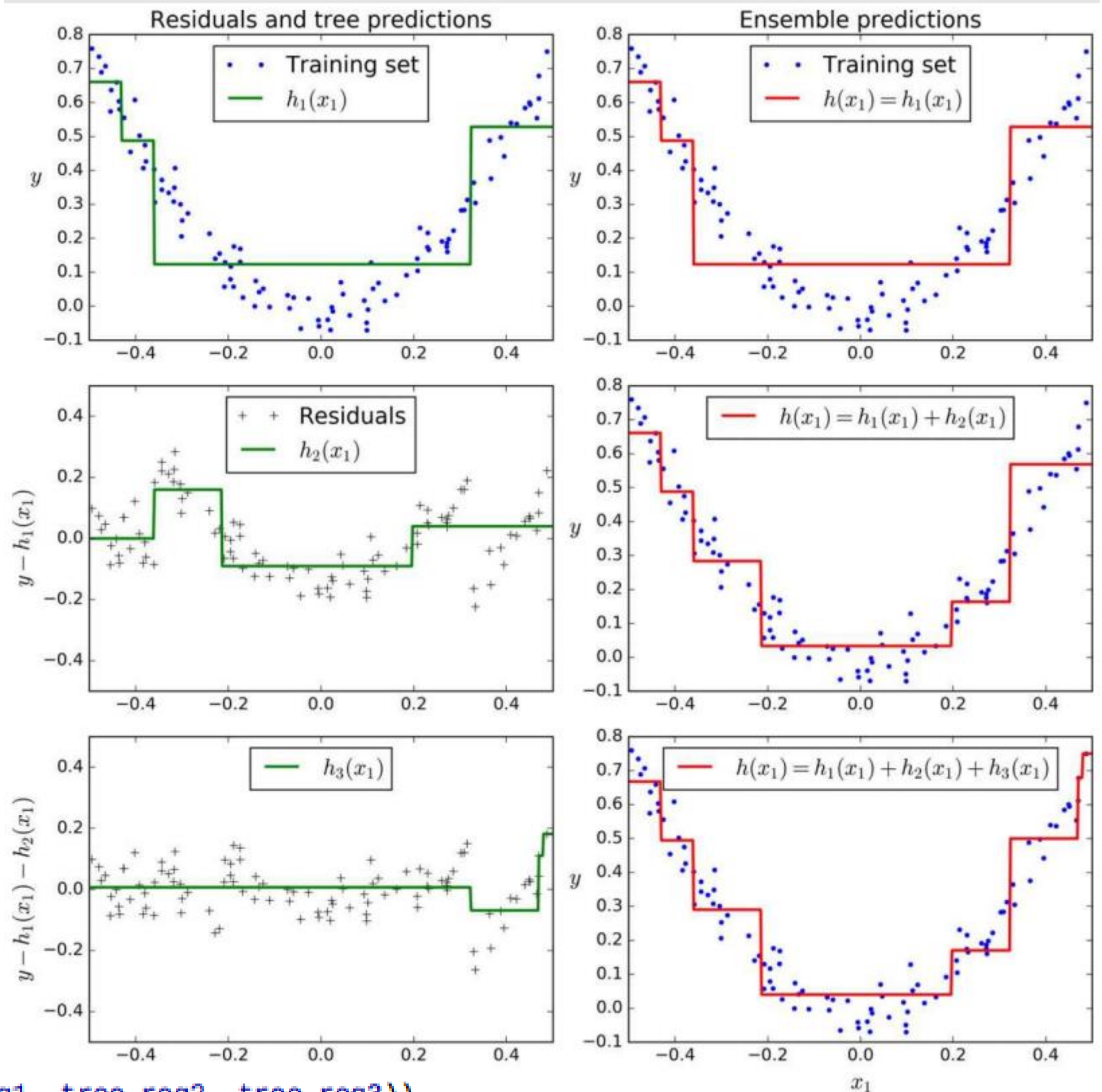
```
from sklearn.tree import DecisionTreeRegressor
```

```
tree_reg1 = DecisionTreeRegressor(max_depth=2)  
tree_reg1.fit(X, y)  $h_1$ 
```

```
y2 = y - tree_reg1.predict(X)  
tree_reg2 = DecisionTreeRegressor(max_depth=2)  
tree_reg2.fit(X, y2)  $h_2$ 
```

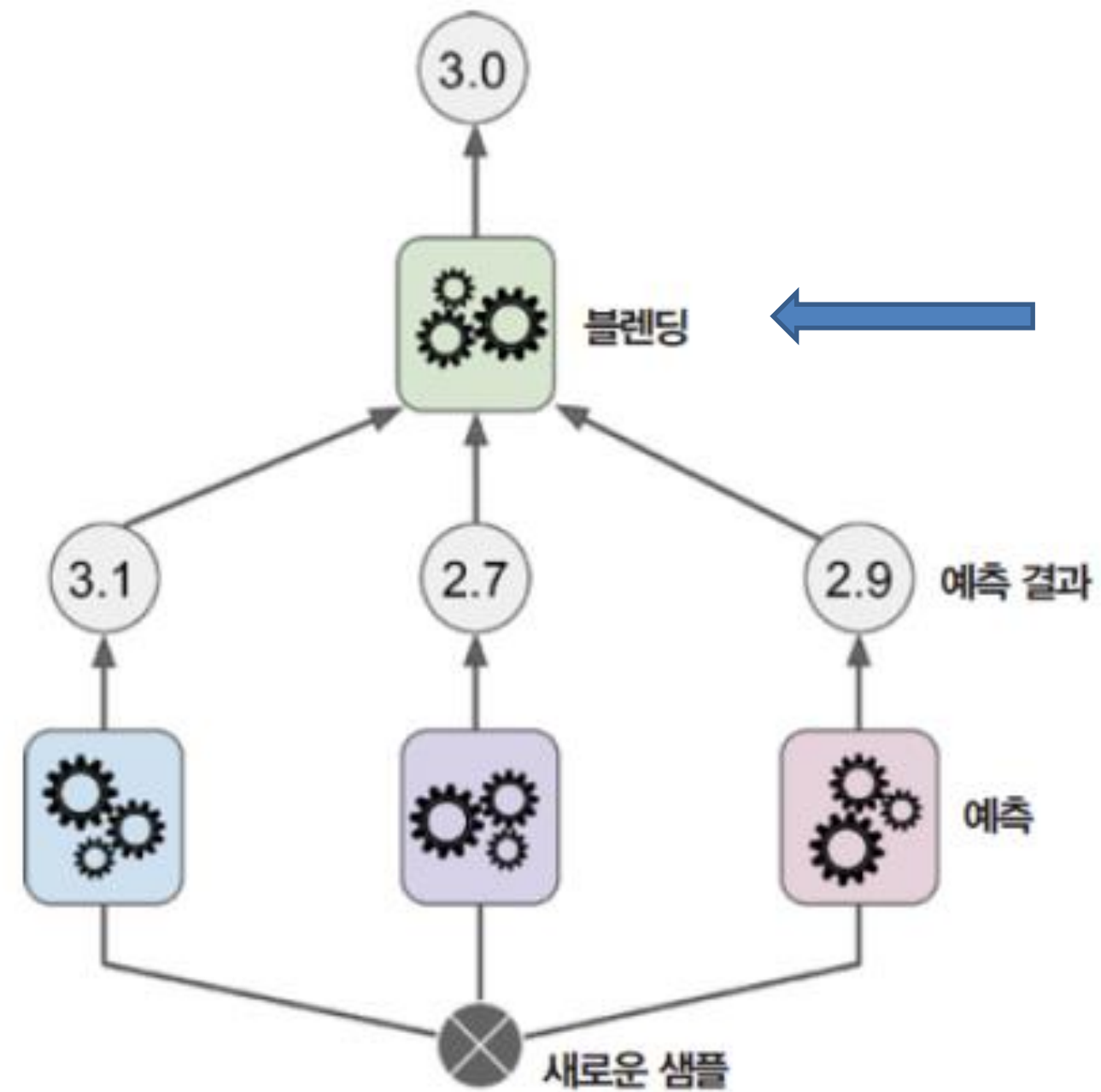
```
y3 = y2 - tree_reg2.predict(X)  
tree_reg3 = DecisionTreeRegressor(max_depth=2)  
tree_reg3.fit(X, y3)  $h_3$ 
```

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```



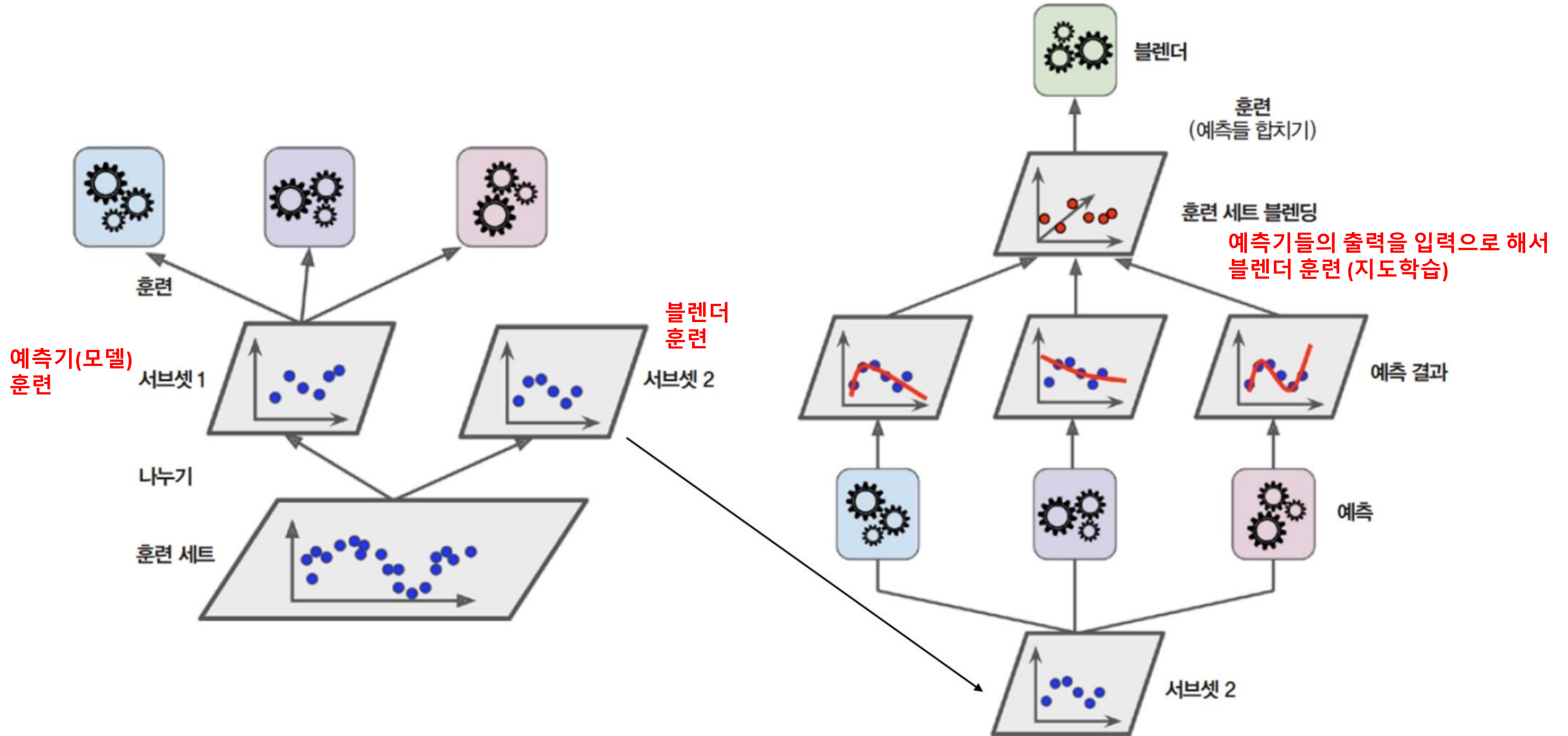
스태킹(Stacking)

- 예측기를 혼합하는 방법을 학습 : 블렌더(Blender) 혹은 메타학습기(Meta Learner)



고정된 방법대신 학습으로 결정

블렌더 훈련



감사합니다