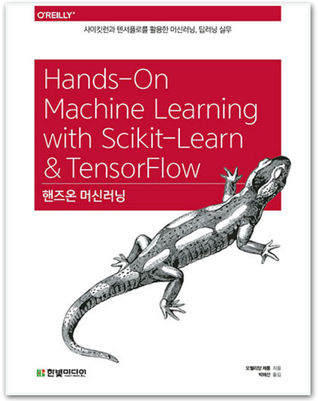


핸즈온 머신러닝

7장. 앙상블 학습과 랜덤 포레스트

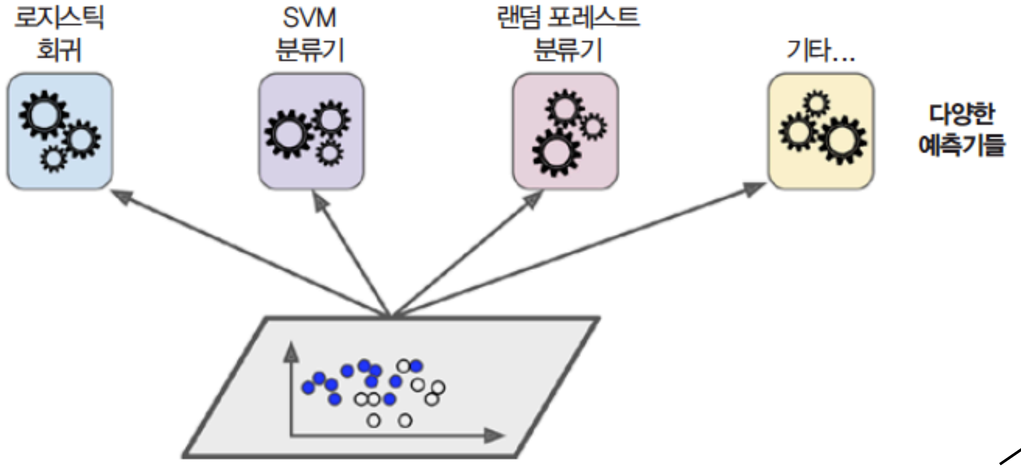
박해선(옮긴이)
haesun.park@tensorflow.blog
https://tensorflow.blog

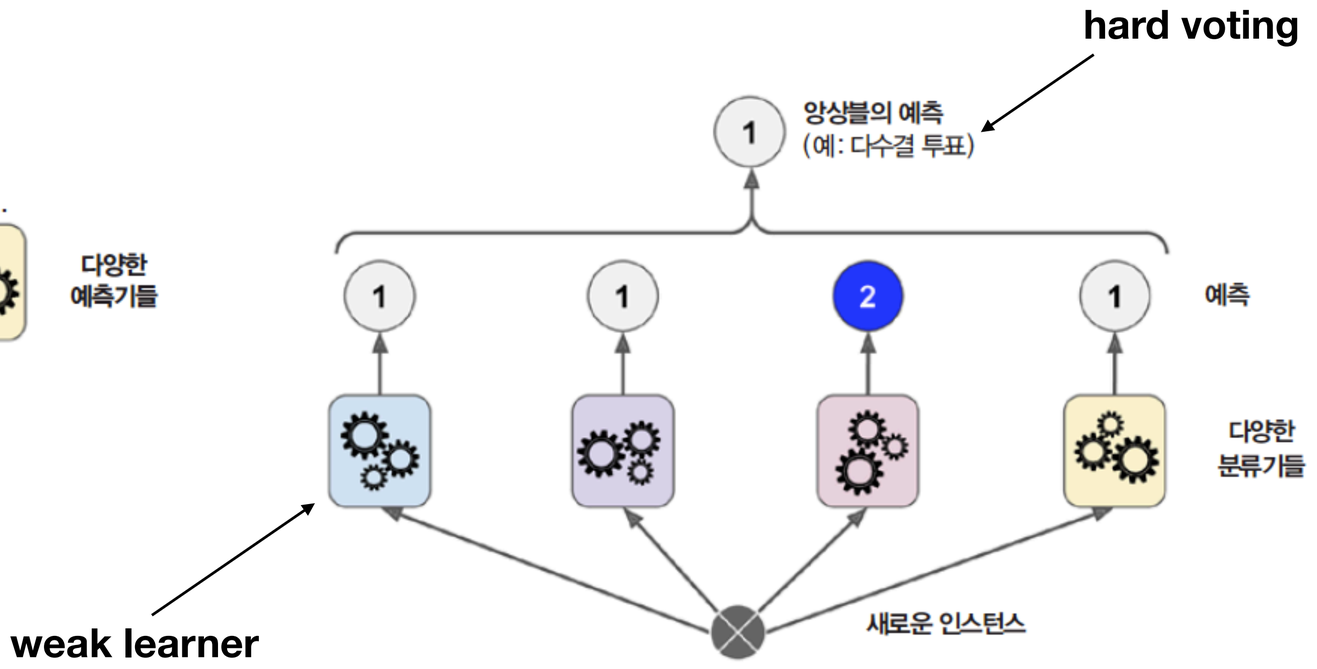

케라스 창시자에게 배우는 딥러닝
창시자의 철학까지 담았다!
DEEP LEARNING WITH PYTHON
프랑소와 숄레 지음
박해선 옮김
딥러닝, 신경망, 머신 러닝 기초부터
컴퓨터 비전과 텍스트, 시퀀스, 생성 모델을 위한 딥러닝까지
입문하면서 알아야 할 모든 것을 설명한다!
길벗

앙상블(Ensemble method)

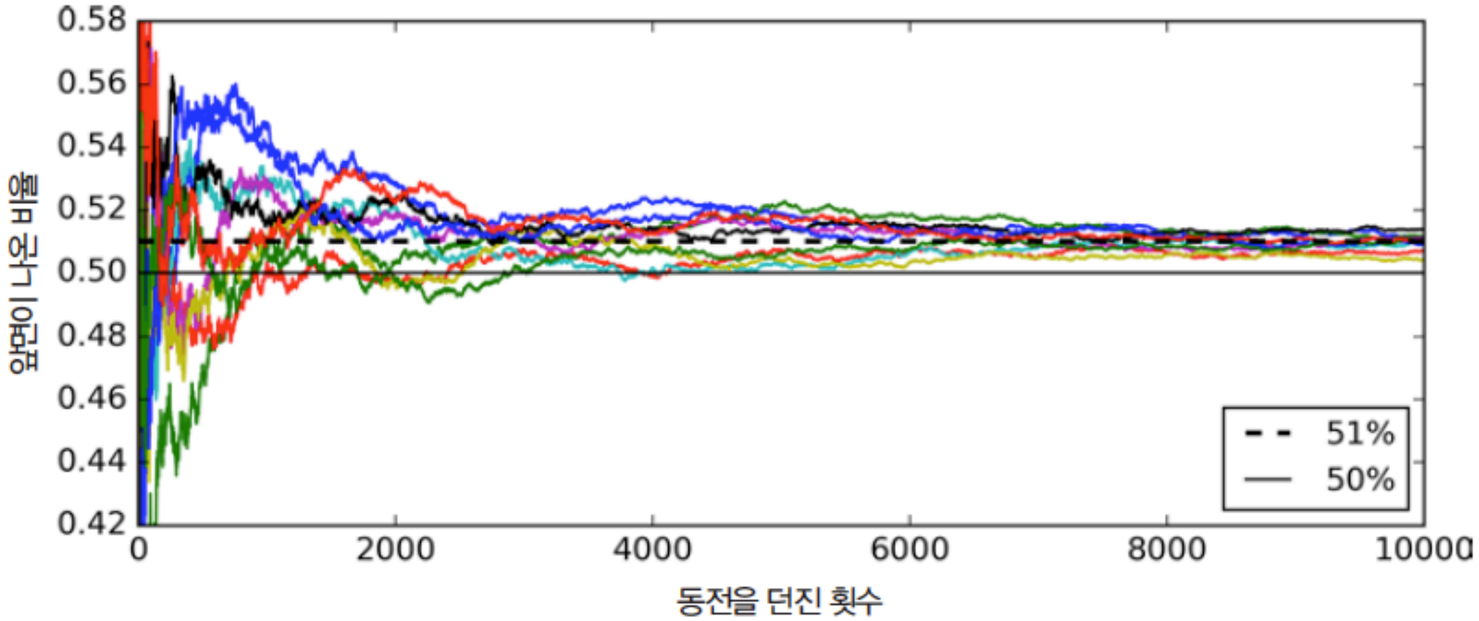
- 여러 개의 모델을 합쳐 보다 나은 일반화 성능을 달성합니다.
- 지각에 관한 문제(딥러닝)를 제외하고 적용할 수 있는 가장 강력한 방법입니다.
- 배깅(bagging): 중복을 허용한 샘플링
- 페이스팅(pasting): 중복을 허용하지 않은 샘플링
- 부스팅(boosting): 이전 예측기의 오차를 보완
- 스태킹(stacking): 앙상블 결과 위에 예측을 위한 모델을 추가

투표 기반 분류기


로지스틱
회귀
SVM
분류기
랜덤 포레스트
분류기
기타...
다양한
예측기들


hard voting
1
앙상블의 예측
(예: 다수결 투표)
1
1
2
1
예측
다양한
분류기들
weak learner
새로운 인스턴스

큰 수의 법칙



성공할 확률 질량 함수 $= \sum_{i=1}^{k} \binom{n}{k} p^k (1-p)^{(n-k)}$

$$1 - \sum_{i=1}^{499} \binom{1000}{k} 0.51^k (1-0.51)^{(1000-k)} = 0.747$$

1-scipy.stats.binom.cdf(499, 1000, 0.51) = 0.747

실패할 확률 질량 함수 $= \sum_{i=k}^{n} \binom{n}{k} \epsilon^k (1-\epsilon)^{(n-k)}$

VotingClassifier

```
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)
```

probability=True
voting='soft'

moons 데이터셋

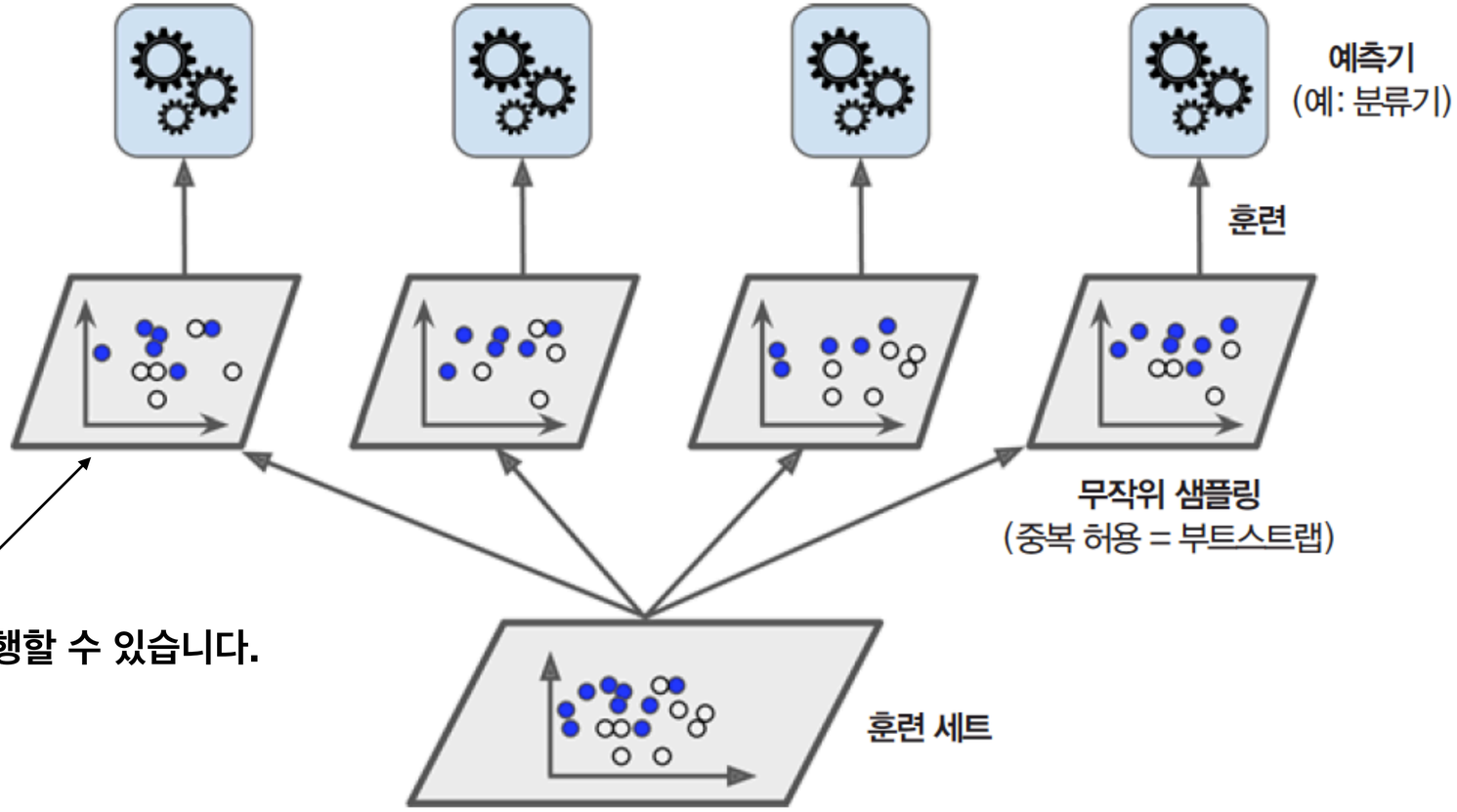
```
>>> from sklearn.metrics import accuracy_score
>>> for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
...     clf.fit(X_train, y_train)
...     y_pred = clf.predict(X_test)
...     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
...
LogisticRegression 0.864
RandomForestClassifier 0.872
SVC 0.888
VotingClassifier 0.896
```

- predict_proba() 메서드가 있는 모델은 분류기의 확률을 평균내는 간접 투표(soft voting)을 사용할 수 있습니다.

배깅 & 페이스팅

- 훈련 세트에서 무작위로 서브 세트를 만들어 여러 개의 분류기를 학습합니다.
- 배깅(Bagging): 훈련 세트에서 중복을 허용하여 샘플링합니다. Boostrap aggregating의 줄임말입니다.
- 페이스팅(Pasting): 훈련 세트에서 중복을 허용하지 않고 샘플링합니다.
- BaggingClassifier: 기본적으로 간접 투표(soft voting)를 사용하지만 predict_proba 메서드가 없는 모델의 경우 직접 투표처럼 작동합니다(soft+hard mix).
- BaggingRegressor: 예측 값의 평균

배깅, 페이스팅 훈련 방식



훈련과 예측을 병렬로 수행할 수 있습니다.

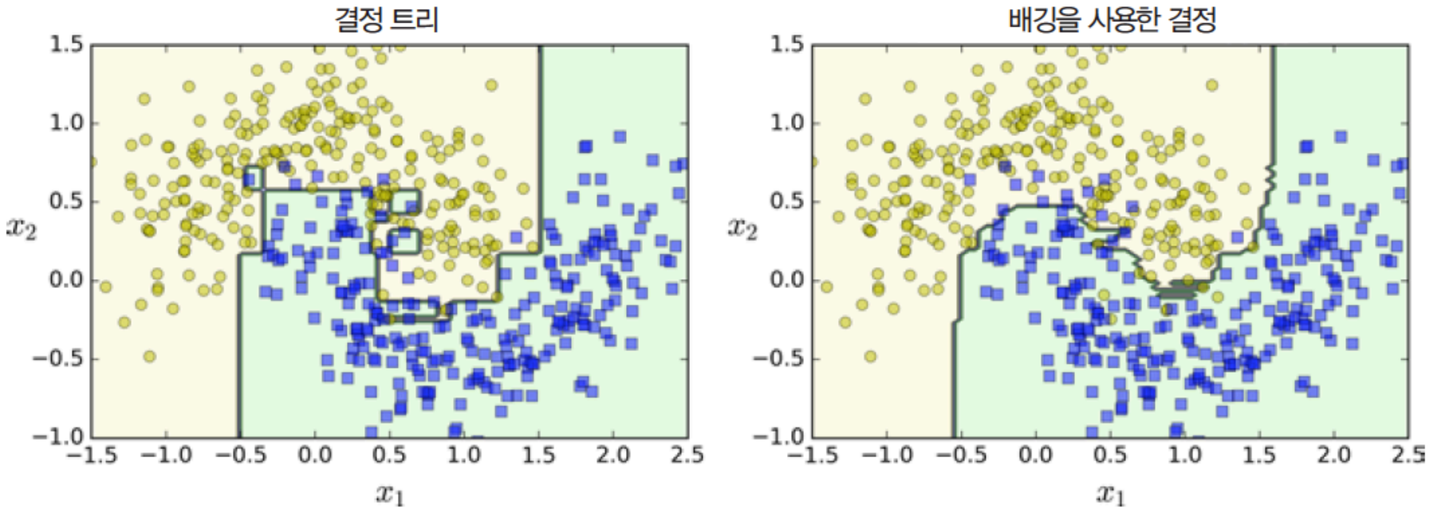
BaggingClassifier

```
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)
```

default: 1

훈련 세트 비율:0~1.0

False: 페이스팅



배깅 vs 페이스팅

- 배깅(부스트래핑)이 더 다양한 서브셋을 만듭니다.
- 분산: 배깅 < 페이스팅
- 편향: 배깅 > 페이스팅
- 보통 배깅을 선호하나 교차 검증으로 두 모델을 모두 확인해 보는 것이 좋습니다.

oob 평가

- BaggingClassifier(bootstrap=True, m_samples=1.0)
- 아주 큰 m개의 샘플을 m번 선택했을 때 한 번도 포함되지 않을 확률

$$y = \left(1 - \frac{1}{m}\right)^m = e^{-1} = 0.367$$

- 교차 검증 대신 남겨진 샘플(out of bag, oob)을 사용하여 평가할 수 있습니다.

https://tensorflow.blog/랜덤-포레스트에서-데이터가-누락될-확률/

BaggingClassifier + oob

```
>>> 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_
0.9013333333333333

>>> from sklearn.metrics import accuracy_score
>>> y_pred = bag_clf.predict(X_test)
>>> accuracy_score(y_test, y_pred)
0.912

>>> bag_clf.oob_decision_function_
array([[ 0.31746032,  0.68253968],
       [ 0.34117647,  0.65882353],
       [ 1.        ,  0.        ],
       ...
       [ 1.        ,  0.        ],
       [ 0.03108808,  0.96891192],
       [ 0.57291667,  0.42708333]])
```

랜덤 패치와 랜덤 서브스페이스

- bootstrap_features: 중복을 허용한 특성 샘플링 여부
- max_features: 랜덤하게 사용할 최대 특성 수
- 랜덤 패치(random patch): bootstrap=True, max_samples < 1.0, bootstrap_features=True, max_featrues < 1.0
- 랜덤 서브스페이스(random subspace): bootstrap=False, max_samples = 1.0, bootstrap_features=True, max_featrues < 1.0
- 마찬가지로 특성 샘플링은 분산을 낮추고 편향을 늘립니다.

랜덤 포레스트

- RandomForestClassifier, RandomForestRegressor
- BaggingClassifier + DecisionTreeClassifier와 비슷

```
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)


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

엑스트라 트리

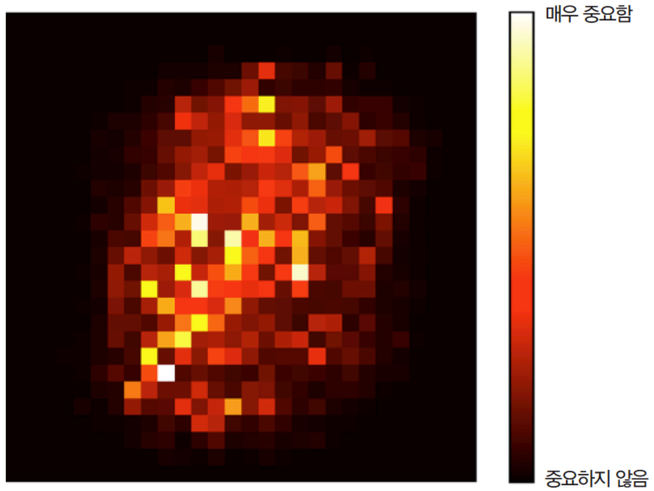
- ExtraTreesClassifer, ExtraTreesRegressor
- ->ExtraTreeClassifier->DecisionTreeClassifier(splitter='random')
- 노드를 무작위로 분할합니다. 분산을 낮추고 편향을 손해 봅니다.

특성 중요도

```
>>> 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
```

모든 특성에 대해 골고루 평가

노드에 사용된 특성의 중요도 = 현재 노드 샘플 비율 * 불순도
- 왼쪽 노드 샘플 비율 * 불순도 - 오른쪽 노드 샘플 비율 * 불순도

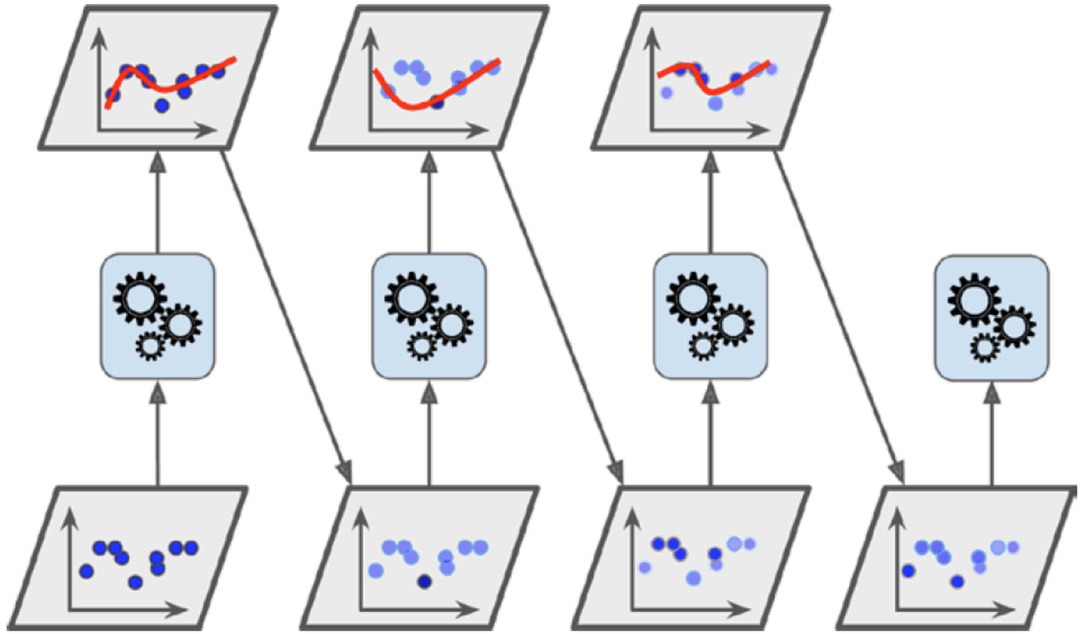


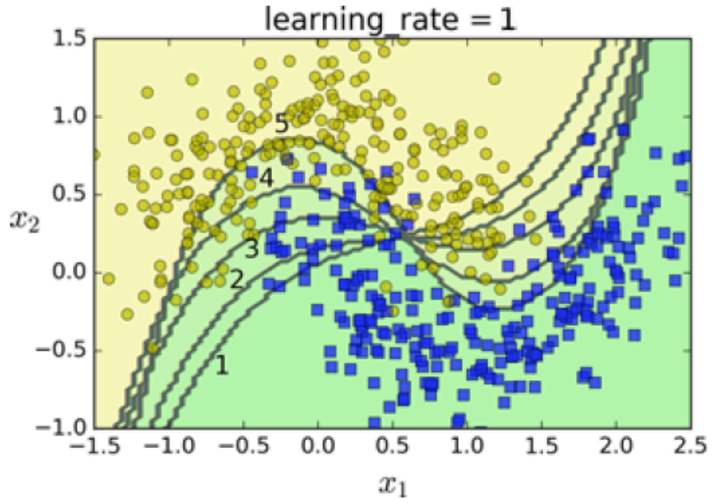
부스팅(boosting)

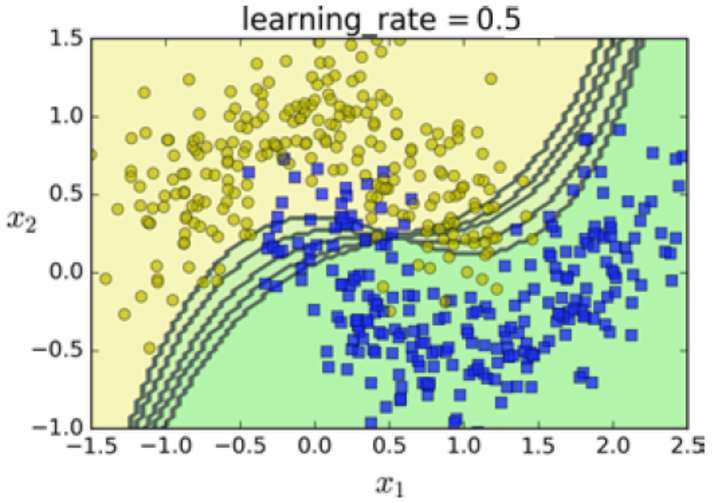
- 약한 앞의 모델을 보완해나가면서 일련의 예측기를 학습합니다.
- 아다부스트(AdaBoost)와 그래디언트 부스팅(Gradient Boosting)
- 연속하여 모델을 학습해야 하기 때문에 병렬화되지 못합니다.

아다부스트

- 이전 모델이 놓친 샘플에 가중치를 크게하여 다시 학습합니다.







아다부스트 알고리즘

가중치 적용된 에러율(w 초깃값은 1/m)

$$r_j = \frac{\sum_{i=1}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}$$

가중치 업데이트

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

여기서 $i = 1, 2, \cdots, m$

예측기 가중치 계산

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

예측

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

여기서 N은 예측기 수

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

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

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

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

그래디언트 부스팅

- 이전의 예측기가 만든 잔여 오차를 학습하는 회귀 트리(DecisionTreeRegressor)를 추가합니다.

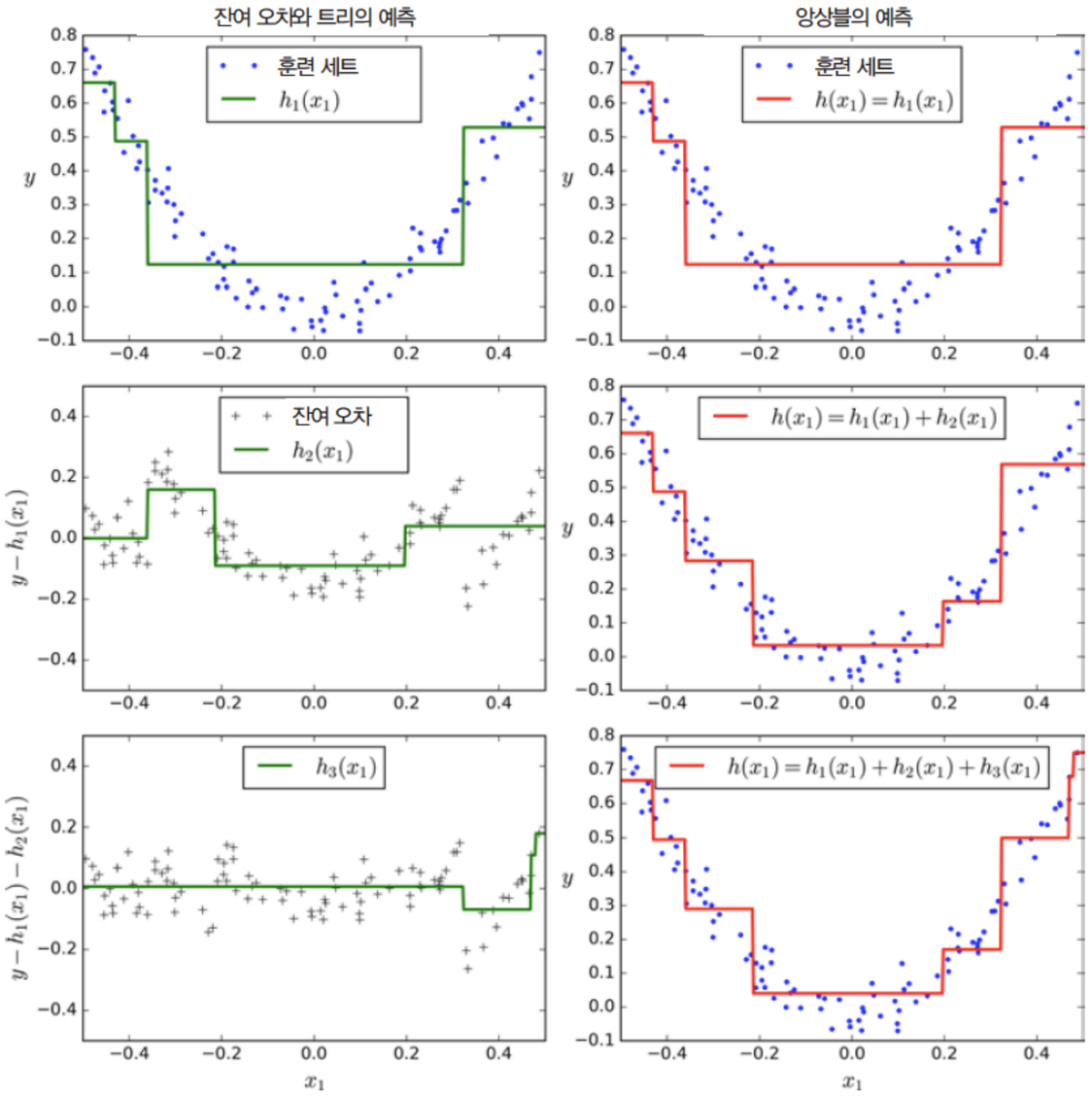
- GradientBoostingClassifier(loss='deviance'), GradientBoostingRegressor(loss='ls')

그래디언트 부스팅 예제

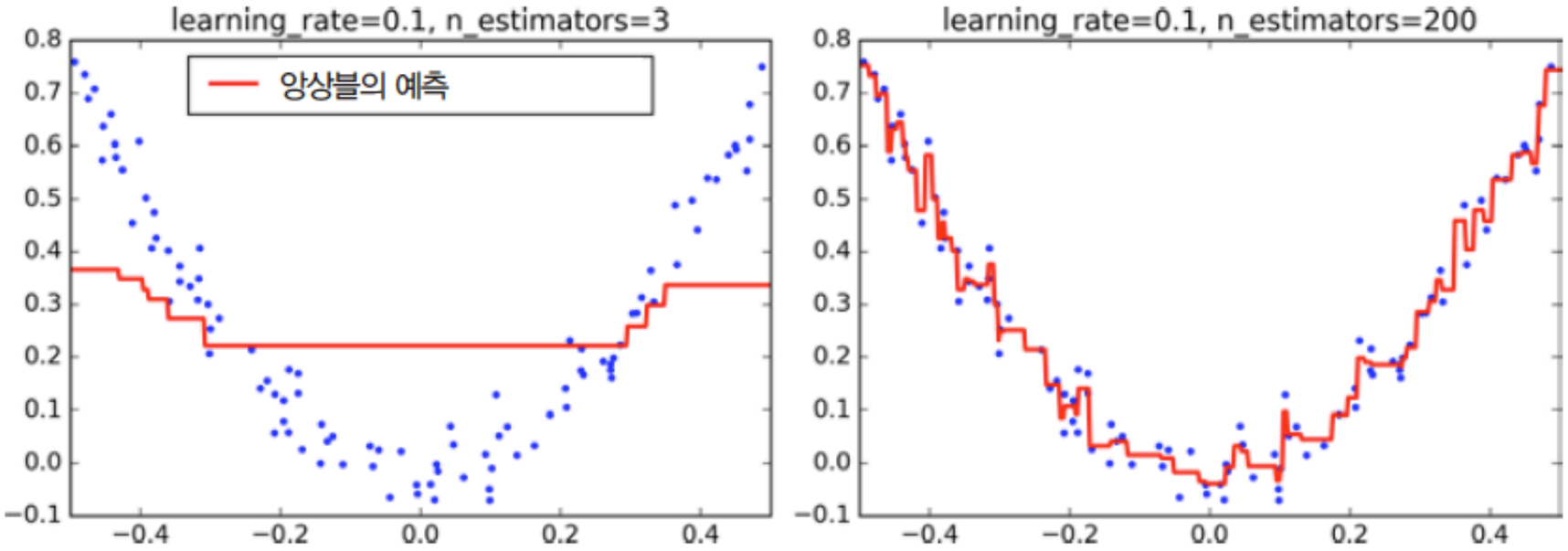
```
from sklearn.tree import DecisionTreeRegressor

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

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

y3 = y2 - tree_reg2.predict(X)
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))
```

```
from sklearn.ensemble import GradientBoostingRegressor

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

잔여 오차와 예측

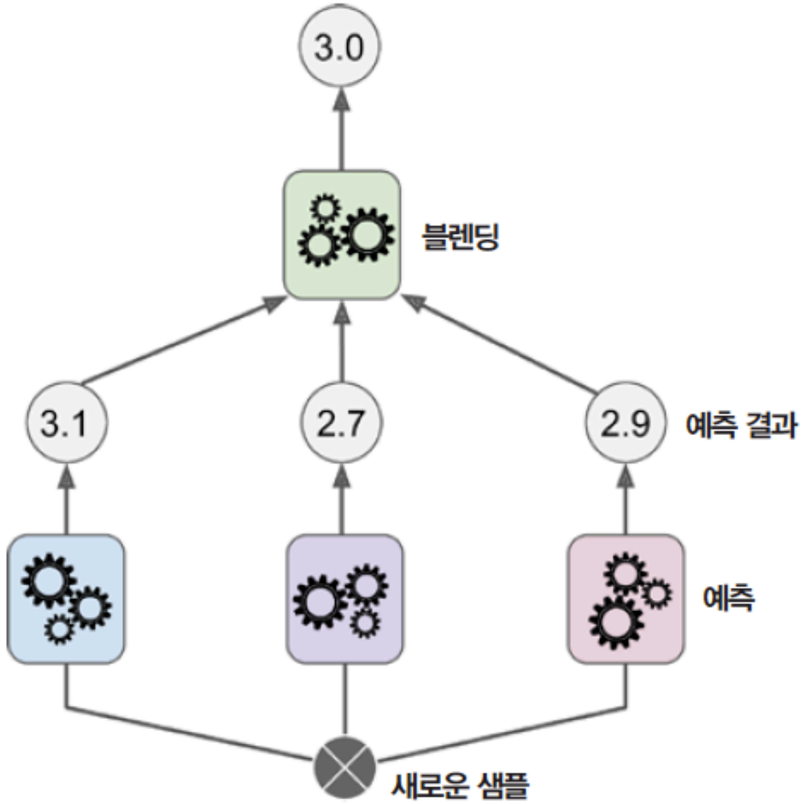

잔여 오차와 트리의 예측
훈련 세트
$h_1(x_1)$
잔여 오차
$h_2(x_1)$
$h_3(x_1)$
y
$y - h_1(x_1)$
$y - h_1(x_1) - h_2(x_1)$
x_1
앙상블의 예측
훈련 세트
$h(x_1) = h_1(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1) + h_3(x_1)$
y
x_1

예측기 개수의 영향

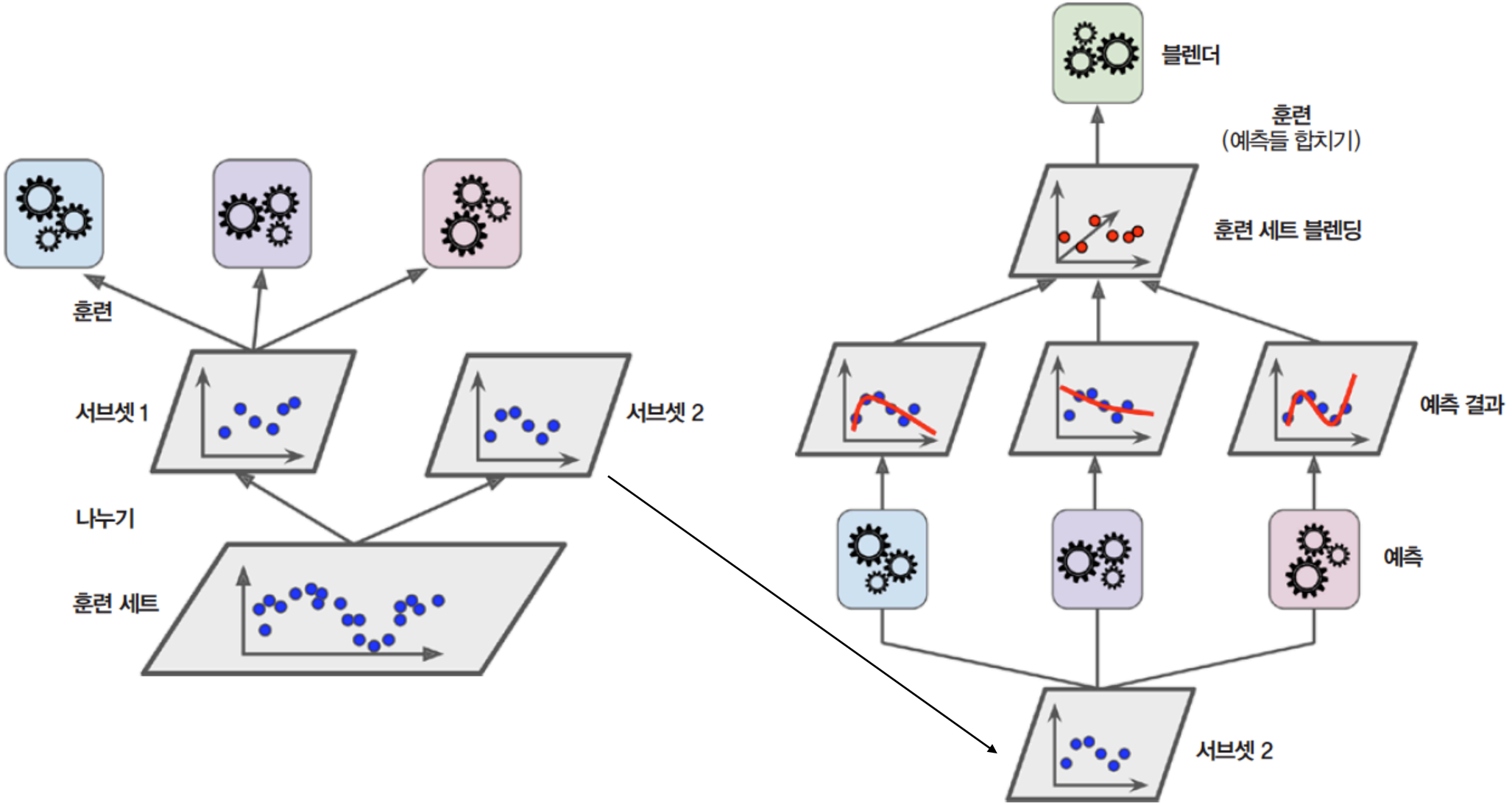

learning_rate=0.1, n_estimators=3
앙상블의 예측
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.4
−0.2
0.0
0.2
0.4
learning_rate=0.1, n_estimators=200
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.4
−0.2
0.0
0.2
0.4

스태킹(stacking)

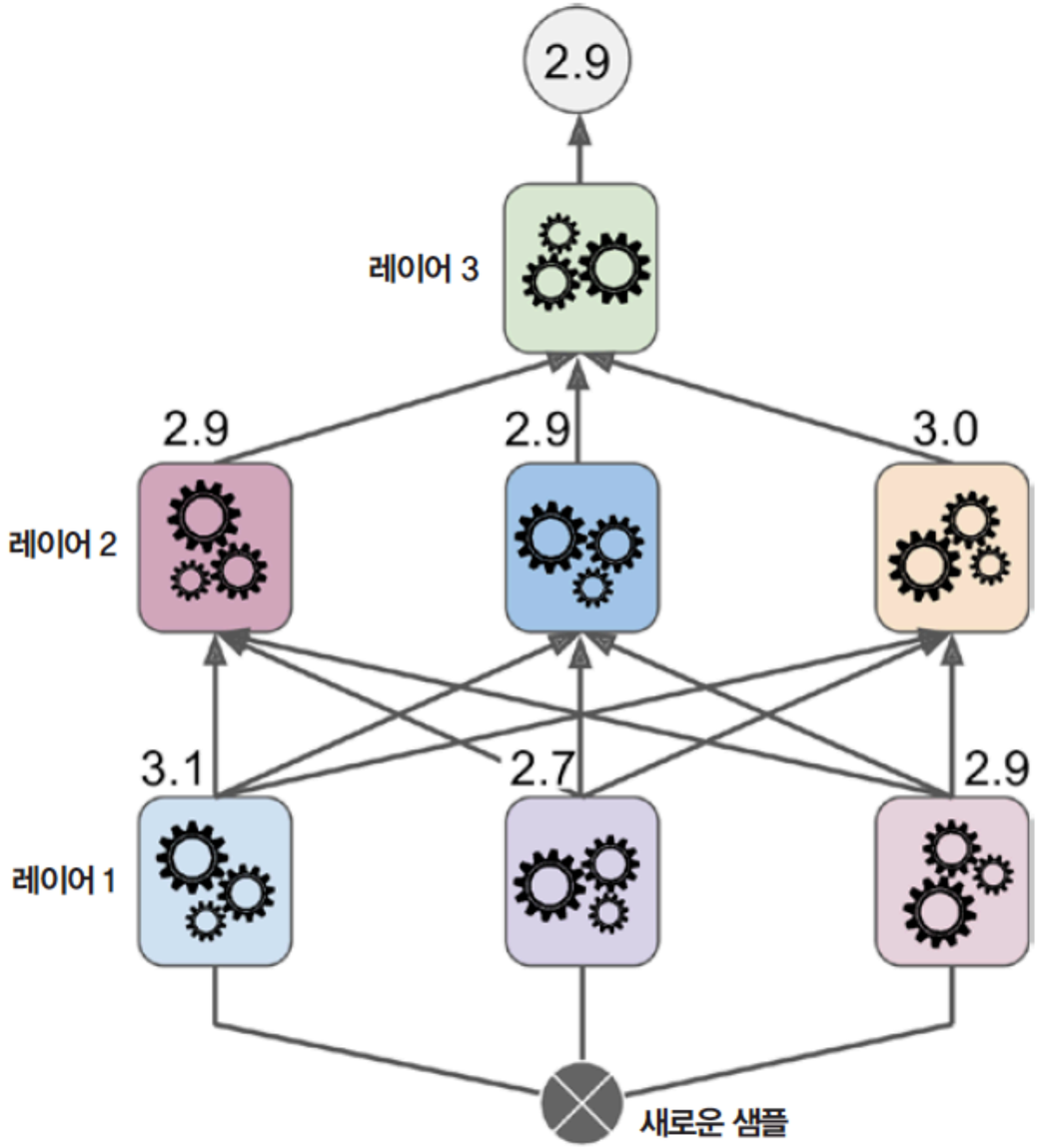
- 앙상블의 예측 결과를 사용하여 새로운 예측기(블렌더 혹은 메타학습기)를 훈련합니다.



블렌더 훈련



멀티 레이어 스태킹



감사합니다