Chap7. 앙상블 학습과 랜덤 포레스트

RTLab Seongsu Keum

CONTENTS

7.0 Introduction

대중의 지혜 (wisdom of the crowd)

무작위로 선택한 수천 명의 사람에게 질문을 하고 대답을 모은 경우, 옳은 답을 이끌어낼 가능성이 높다.

앙상블 학습 (Ensemble Learning)

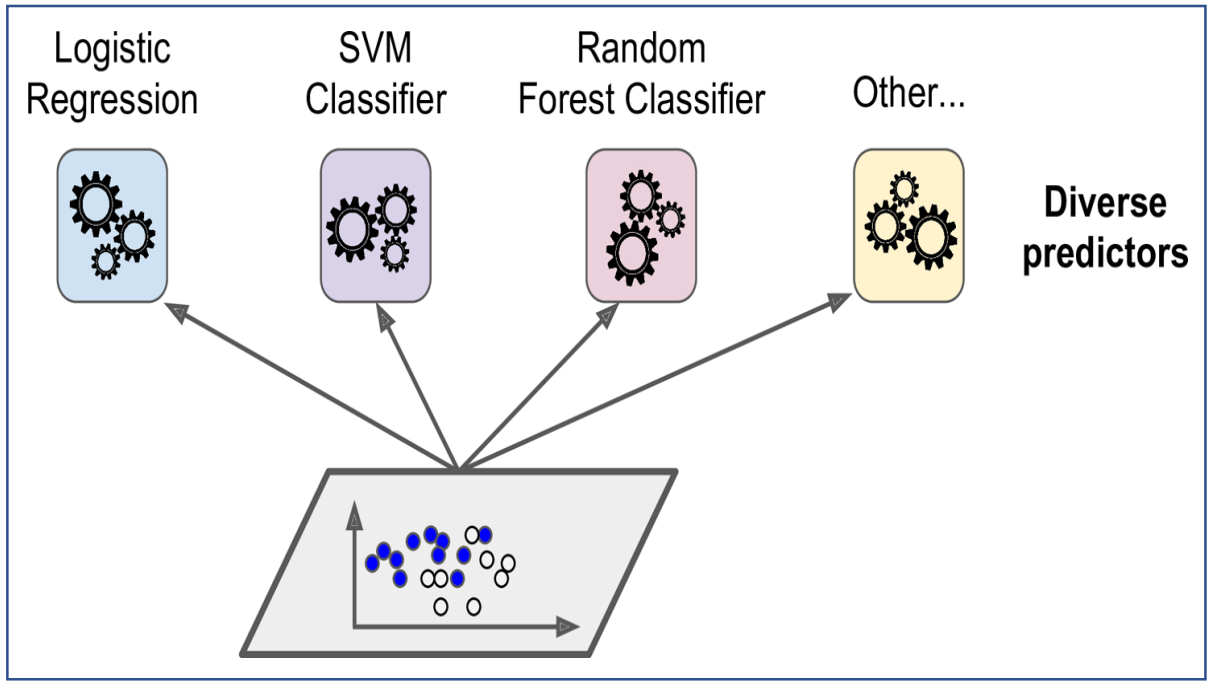
일련의 예측기(분류나 회귀 모델)로부터 예측을 수집하면 가장 좋은 모델 하나보다 좋은 예측을 얻을 수 있다.

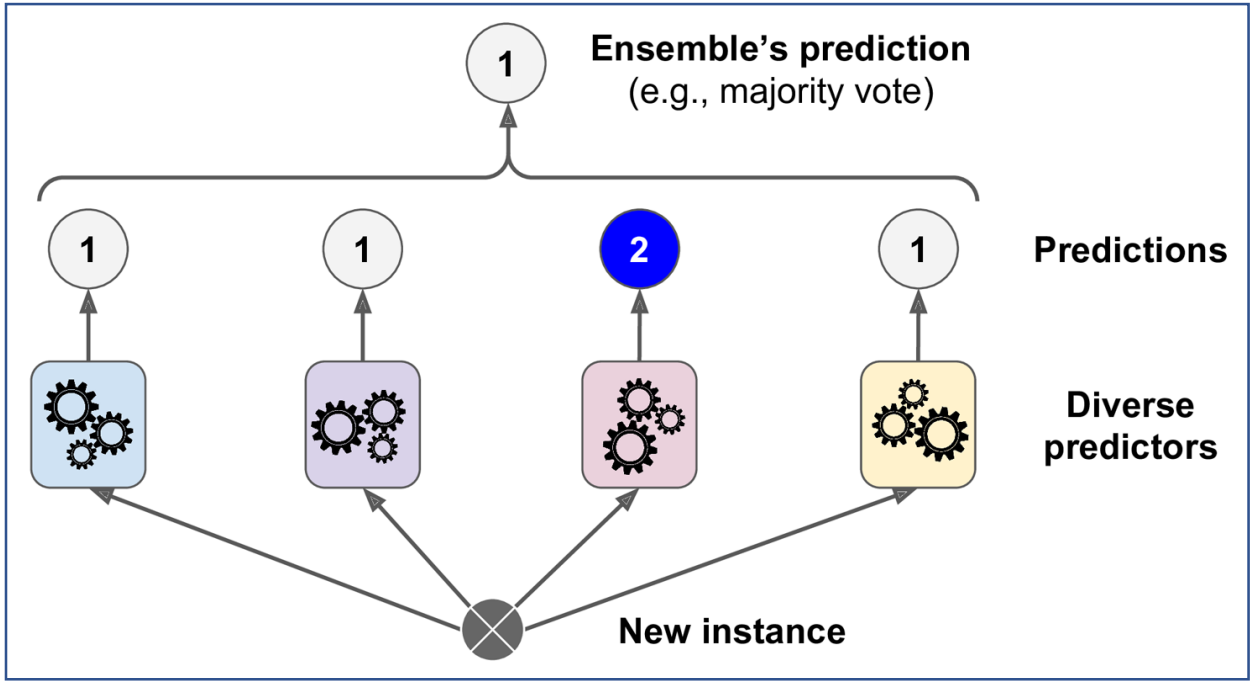
일련의 예측기를 앙상블이라고 부르며, 앙상블을 이용한 학습을 앙상블 학습이라 한다.

랜덤 포레스트 (Random Forest)

결정 트리의 앙상블을 랜덤 포레스트라고 한다. 간단한 방법임에도 랜덤 포레스트는 오늘날 가장 강력한 머신러닝 알고리즘 중 하나이다.

7.1 투표 기반 분류기





더 좋은 분류기를 만드는 아주 간단한 방법

: 각 분류기의 예측을 모아서 가장 많이 선택된 클래스로 예측하는 것.

이렇게 다수결 투표로 정해지는 분류기를 직접 투표(hard voting) 분류기라고 함.

7.1 투표 기반 분류기

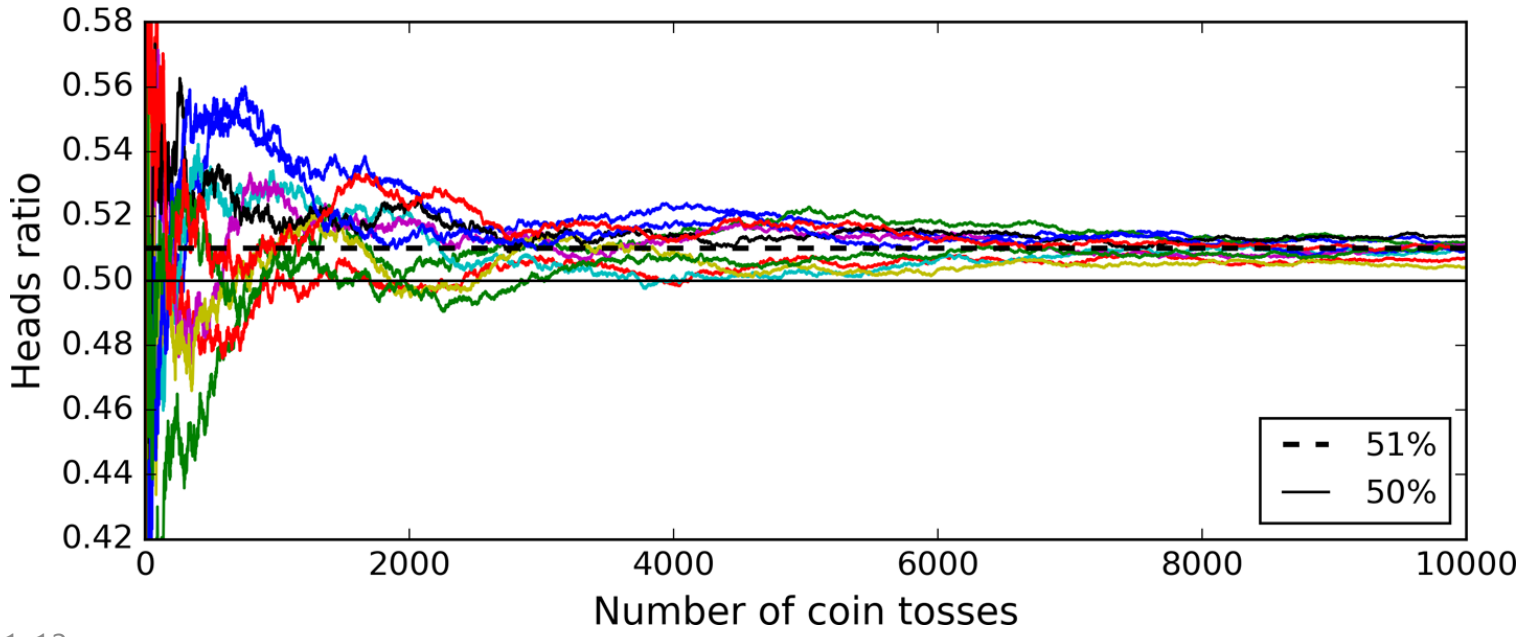
이 다수결 투표 분류기는 보통 앙상블에 포함된 개별 분류기 중 가장 뛰어난 것보다 정확도가 높음.

이것이 어떻게 가능할까?

큰 수의 법칙 (law of large numbers)

던졌을 때 앞면이 51%, 뒷면이 49%가 나오는 동전

-> 동전을 더 많이 던질수록 앞면이 다수가 될 확률이 높아진다.



7.1 투표 기반 분류기

이와 비슷하게 각각 51% 정확도를 가진 여러 개의 분류기라면..?

-> 각 분류기의 예측을 모아서 가장 많이 선택된 클래스로 예측.

이 경우 분류기 각각이 가능한 한 서로 독립적일 때 최고의 성능을 발휘함.

7.1 투표 기반 분류기

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

```
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='liblinear', random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=10, random_state=42)
svm_clf = SVC(gamma='auto', 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)
```

```
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
```

7.1 투표 기반 분류기

```
log_clf = LogisticRegression(solver='liblinear', random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=10, random_state=42)
svm_clf = SVC(gamma='auto', probability=True, random_state=42)

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

```
VotingClassifier(estimators=[('lr', LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=42, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)), ('rf', RandomFor...bf',
  max_iter=-1, probability=True, random_state=42, shrinking=True,
  tol=0.001, verbose=False))],
       flatten_transform=None, n_jobs=1, voting='soft', weights=None)
```

```
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))
```

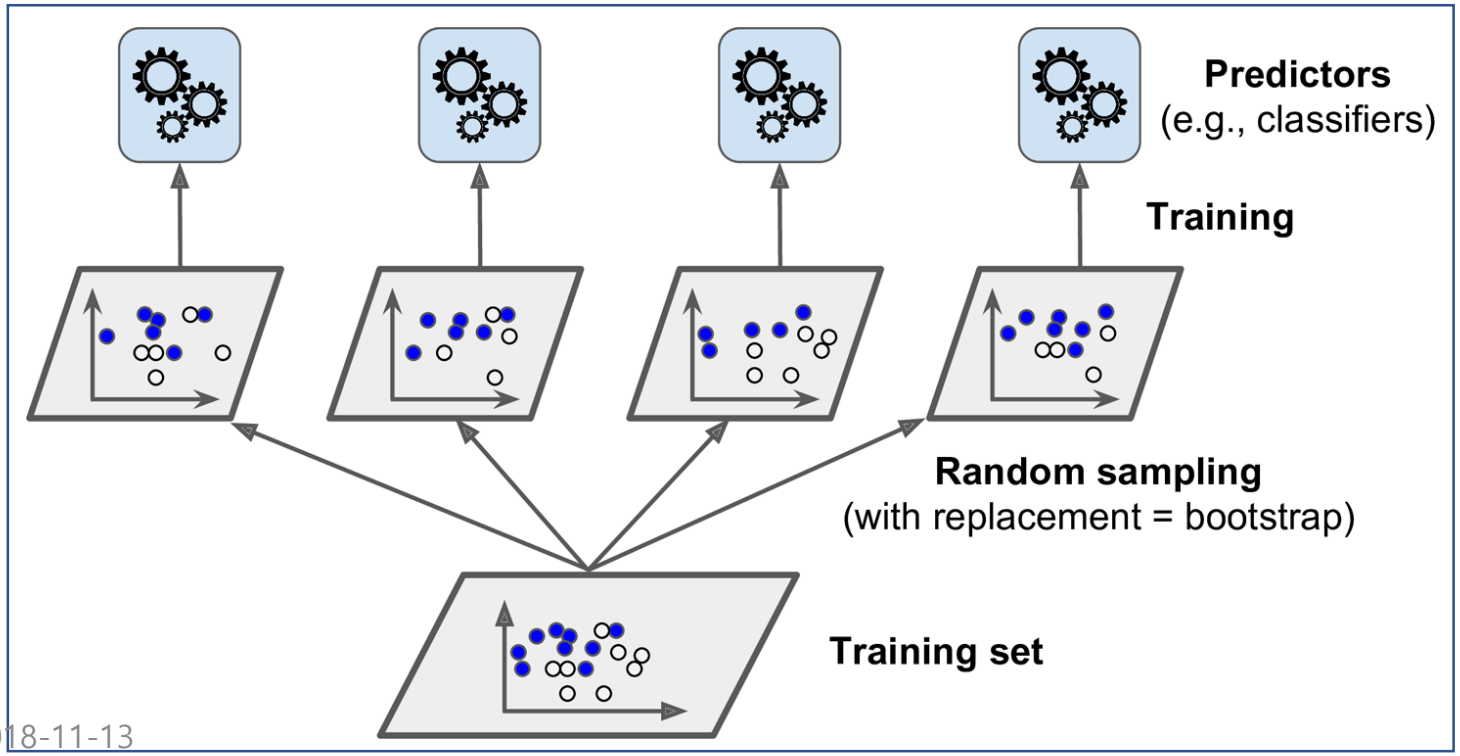
```
LogisticRegression 0.864
RandomForestClassifier 0.872
SVC 0.888
VotingClassifier 0.912
```

7.2 배깅과 페이스팅

다양한 분류기를 만드는 방법

1) 각기 다른 훈련 알고리즘을 사용하는 것
2) 같은 알고리즘을 사용하지만 훈련 세트의 서브셋을 무작위로 구성 -> 분류기를 각기 다르게 학습

훈련세트에서 중복을 허용하여 샘플링하는 방식을 배깅(bagging, bootstrap aggregatin의 줄임)이라 하며,
중복을 허용하지 않고 샘플링하는 방식을 페이스팅(pasting)이라 함.



7.2 배깅과 페이스팅

7.2.1 사이킷런의 배깅과 페이스팅

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

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.904
```

```
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)
y_pred_tree = tree_clf.predict(X_test)
print(accuracy_score(y_test, y_pred_tree))
```

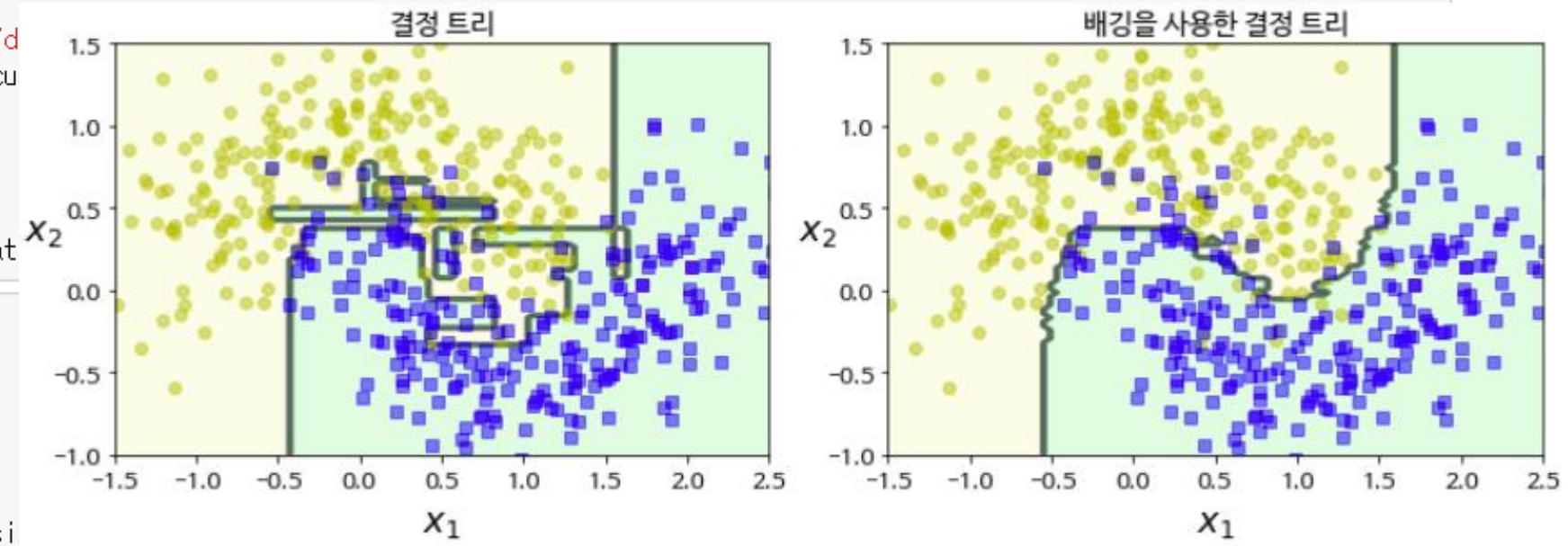
```
0.856
```

7.2 배깅과 페이스팅

```
from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, axes=[-1.5, 2.5, -1, 1.5], alpha=0.5, contour=True):
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0','#9898ff','#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)
    if contour:
        custom_cmap2 = ListedColormap(['#7d
        plt.contour(x1, x2, y_pred, cmap=cu
    plt.plot(X[:, 0][y==0], X[:, 1][y==0],
    plt.plot(X[:, 0][y==1], X[:, 1][y==1],
    plt.axis(axes)
    plt.xlabel(r"$x_1$", fontsize=18)
    plt.ylabel(r"$x_2$", fontsize=18, rotat
```

```
plt.figure(figsize=(11,4))
plt.subplot(121)
plot_decision_boundary(tree_clf, X, y)
plt.title("결정 트리", fontsize=14)
plt.subplot(122)
plot_decision_boundary(bag_clf, X, y)
plt.title("배깅을 사용한 결정 트리", fontsi
save_fig("decision_tree_without_and_with_bagging_plot")
plt.show()
```



7.2 배깅과 페이스팅

7.2.2 oob 평가

배깅을 사용하면 어떤 예측기는 하나의 샘플을 여러 번 샘플링하고,
다른 어떤 샘플은 아예 선택하지 않을 수 있음
-> 이는 평균적으로 각 예측기가 훈련 샘플의 63% 정도만 샘플링한다는 것을 의미.

$$\lim_{m\to\infty}\left(1-\frac{1}{m}\right)^{m} = e^{-1} = 0.37$$

선택되지 않은 훈련 샘플의 나머지 37%를 oob (out-of-bag) 샘플이라고 부름.
예측기마다 남겨진 37%는 모두 다르다.

사이킷런에서 BaggingClassifier를 만들 때 oob_score=True로 지정하면 훈련이 끝난 후 자동으로 oob평가를 수행

```
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_
```

```
0.8986666666666666
```

```
y_pred = bag_clf.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.904
```

7.2 배깅과 페이스팅

```
bag_clf.oob_decision_function_
```

```
array([[0.38636364, 0.61363636],
       [0.36138614, 0.63861386],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.        , 1.        ],
       [0.08152174, 0.91847826],
       [0.34594595, 0.65405405],
       [0.00636943, 0.99363057],
       [1.        , 0.        ],
       [0.97093023, 0.02906977],
       [0.76836158, 0.23163842],
       [0.00543478, 0.99456522],
       [0.79459459, 0.20540541],
       [0.84183673, 0.15816327],
       [0.95287958, 0.04712042],
       [0.0212766 , 0.9787234 ],
       [0.        , 1.        ],
       [0.98876404, 0.01123596],
       [0.94535519, 0.05464481],
       [0.99489796, 0.00510204],
       [0.01117318, 0.98882682],
       [0.33009709, 0.66990291],
       [0.9076087 , 0.0923913 ],
       [1.        , 0.        ],
       [0.97727273, 0.02272727],
       [0.        , 1.        ],
       [0.99411765, 0.00588235],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.62962963, 0.37037037],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.        , 1.        ],
```

7.3 랜덤 패치와 랜덤 서브스페이스

BaggingClassifier는 특성 샘플링도 지원한다. 각 예측기는 무작위로 선택한 입력 특성의 일부분으로 훈련된다.

특히 (이미지와 같은) 매우 고차원의 데이터셋을 다룰 때 유용함.

훈련 특성과 샘플을 모두 샘플링하는 것을 랜덤 패치 방식,
훈련샘플을 모두 사용하고 특성은 샘플링하는 것을 랜덤 서브스페이스 방식이라고 함.

특성 샘플링은 더 다양한 예측기를 만들며 편향을 늘리는 대신 분산을 낮춘다.

7.4 랜덤 포레스트

랜덤 포레스트는 배깅 방법(또는 페이스팅)을 적용한 결정 트리의 앙상블이다.

BaggingClassifier에 DecisionTreeClassifier를 넣어 만들거나, RandomForestClassifier를 사용할 수 있다.

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(splitter="random", max_leaf_nodes=16, random_state=42),
    n_estimators=500, max_samples=1.0, bootstrap=True, n_jobs=-1, random_state=42)

bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)

accuracy_score(y_test, y_pred)
```

```
0.92
```

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1, random_state=42)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)

accuracy_score(y_test, y_pred_rf)
```

```
0.912
```

7.4 랜덤 포레스트

7.4.1 엑스트라 트리

랜덤 포레스트에서 트리를 만들 때 각 노드는 무작위로 특성의 서브셋을 만들어 분할에 사용한다.
트리를 더욱 무작위하게 만들기 위해 최적의 임곗값을 찾는 대신, 후보 특성을 사용해 무작위로 분할한 다음 그 중에서 최상의 분할을 선택한다.

이와 같이 극단적으로 무작위한 트리의 랜덤 포레스트를 익스트림 랜덤 트리 앙상블 (또는 줄여서 엑스트라 트리) 라고 한다.

엑스트라 트리는 편향이 늘어나지만 분산을 낮추는 효과.
일반적으로 랜덤 포레스트보다 엑스트라 트리가 훨씬 빠르다.

7.4 랜덤 포레스트

7.4.2 특성 중요도

랜덤 포레스트의 또 다른 장점은 특성의 상대적 중요도를 측정하기 쉽다는 것이다.

사이킷런은 훈련이 끝난 뒤 특성마다 자동으로 이 점수를 계산하고 중요도의 합이 1이 되도록 정규화한다.

```
from sklearn.datasets import load_iris
iris = load_iris()
rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1, 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) 0.11249225099876374
sepal width (cm) 0.023119288282510326
petal length (cm) 0.44103046436395765
petal width (cm) 0.4233579963547681
```

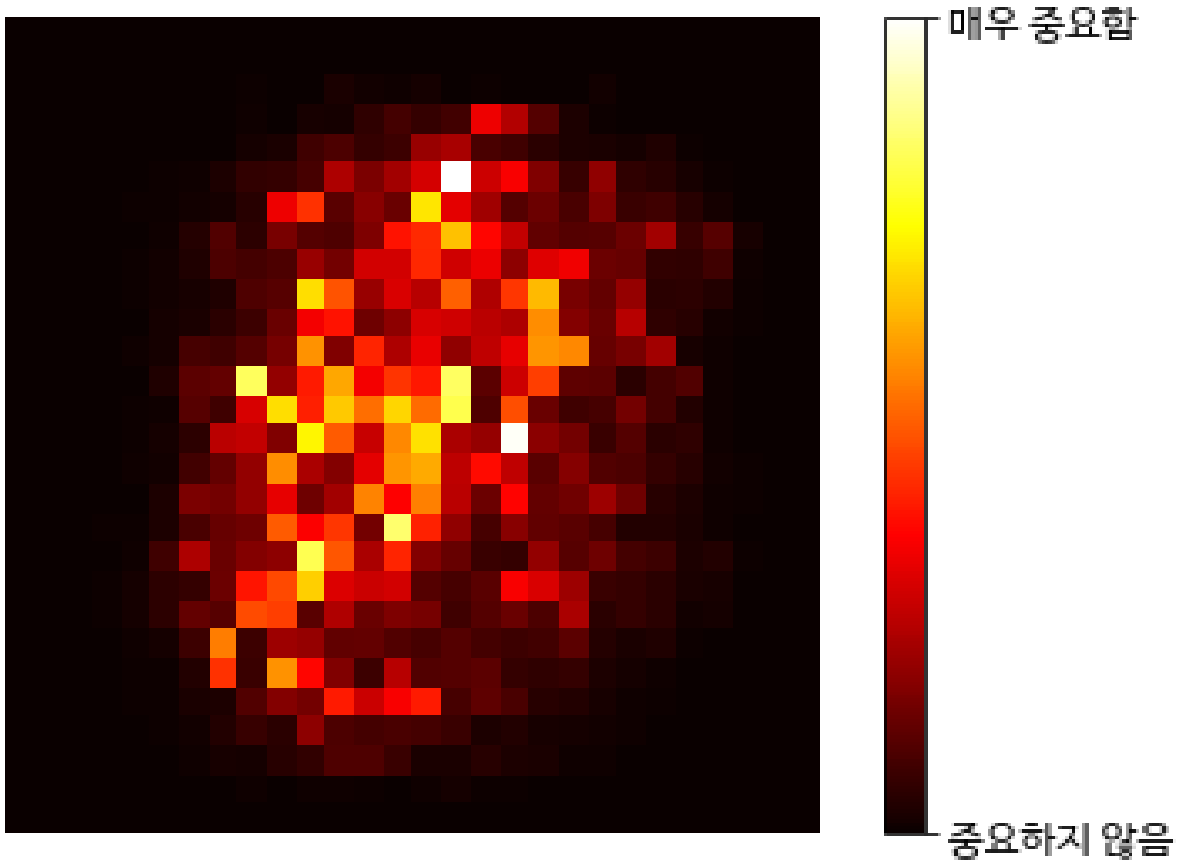
```
rnd_clf.feature_importances_
```

```
array([0.11249225, 0.02311929, 0.44103046, 0.423358  ])
```

7.4 랜덤 포레스트

MNIST 데이터셋을 랜덤 포레스트 분류기로 훈련
-> 각 픽셀의 중요도를 그래프화

랜덤 포레스트는 특히 특성을 선택해야 할 때
어떤 특성이 중요한지 빠르게 확인할 수 있다.



7.5 부스팅

부스팅 (boosting)

약한 학습기를 여러 개 연결하여 강한 학습기를 만드는 앙상블 방법.

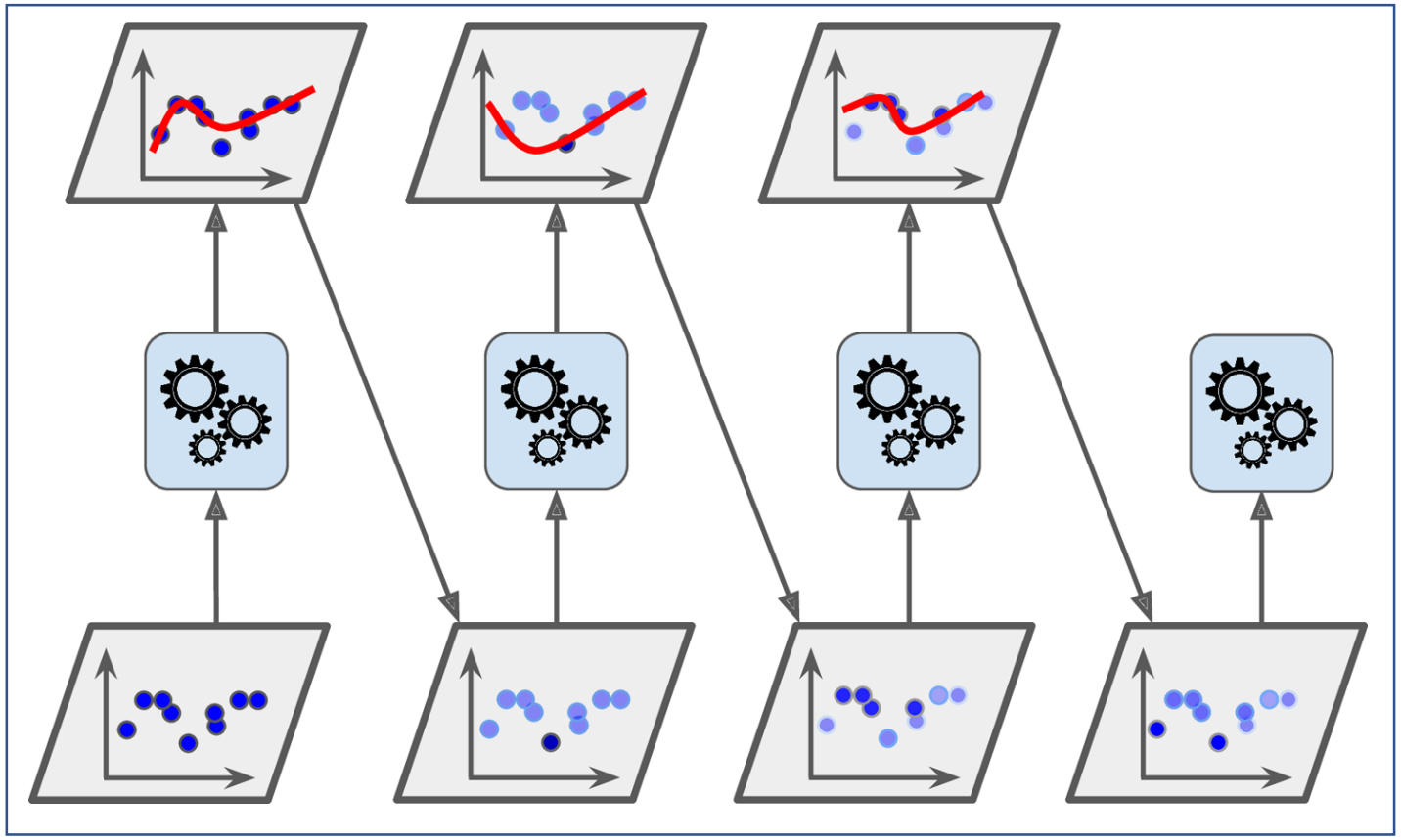
앞의 모델을 보완해 나가면서 예측기를 학습시킨다.

아다부스트 (Adaptive Boosting)와 그래디언트 부스팅 (Gradient Boosting)

7.5 부스팅

7.5.1 아다부스트

이전 모델이 과소적합했던 훈련 샘플의 가중치를 더 높인다. 새로운 예측기는 학습하기 어려운 샘플에 점점 더 맞춰지게 된다.



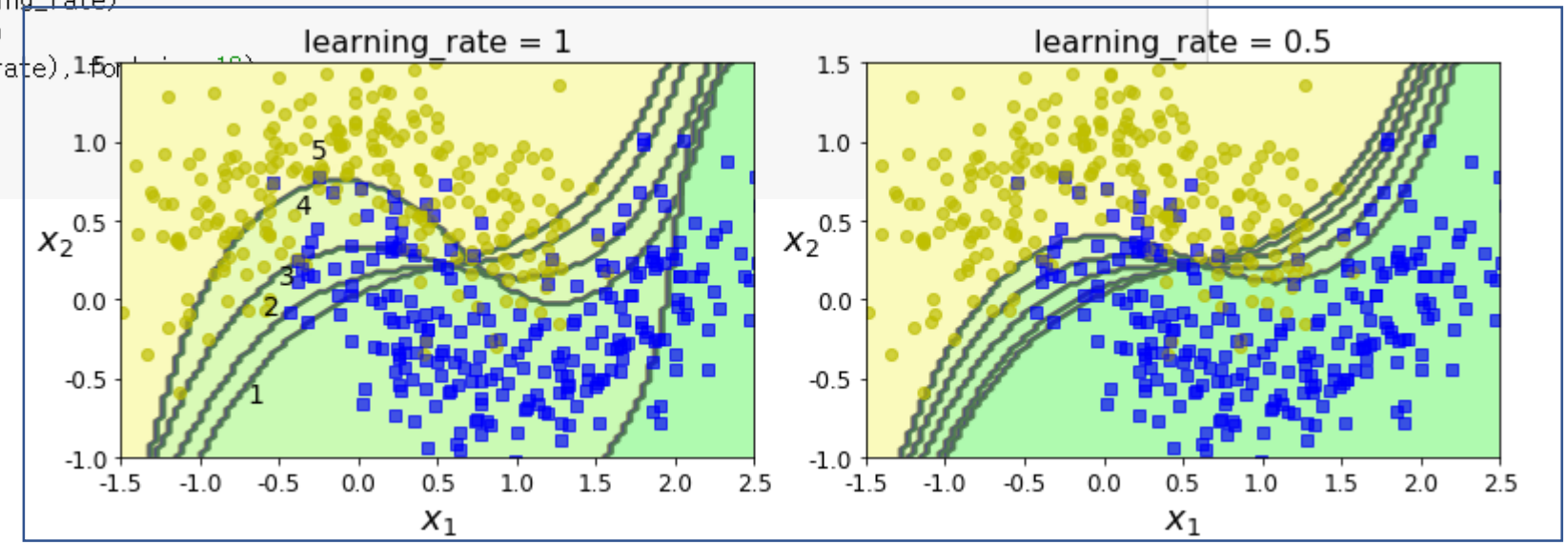
1) 첫 번째 분류기를 훈련 세트에서 훈련시키고 예측을 만든다.
2) 잘못 분류된 훈련 샘플의 가중치를 상대적으로 높인다.
3) 두 번째 분류기는 업데이트된 가중치를 사용해 훈련 세트에서 훈련하고 다시 예측을 만든다
4) ...

7.5 부스팅

```
m = len(X_train)

plt.figure(figsize=(11, 4))
for subplot, learning_rate in ((121, 1), (122, 0.5)):
    sample_weights = np.ones(m)
    plt.subplot(subplot)
    if subplot == 121:
        plt.text(-0.7, -0.65, "1", fontsize=14)
        plt.text(-0.6, -0.10, "2", fontsize=14)
        plt.text(-0.5,  0.10, "3", fontsize=14)
        plt.text(-0.4,  0.55, "4", fontsize=14)
        plt.text(-0.3,  0.90, "5", fontsize=14)
    for i in range(5):
        svm_clf = SVC(kernel="rbf", C=0.05, gamma='auto', random_state=42)
        svm_clf.fit(X_train, y_train, sample_weight=sample_weights)
        y_pred = svm_clf.predict(X_train)
        sample_weights[y_pred != y_train] *= (1 + learning_rate)
        plot_decision_boundary(svm_clf, X, y, alpha=0.2)
        plt.title("learning_rate = {}".format(learning_rate), fo

save_fig("boosting_plot")
plt.show()
```



7.5 부스팅

j번째 예측기의 가중치가 적용된 에러율

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

예측기의 가중치 (예측기가 정확할수록 가중치가 더 높음)

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

가중치 업데이트 규칙

$$\text{for } i = 1, 2, \cdots, m$$

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

지정된 예측기 수에 도달하거나 완벽한 예측기가 만들어지면 중지

가중치의 합이 가장 큰 클래스가 예측결과가 됨.

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

7.5 부스팅

7.5.2 그래디언트 부스팅

앙상블에 이전까지의 오차를 보정하도록 예측기를 순차적으로 추가한다. 하지만 아다부스트처럼 반복마다 샘플의 가중치를 수정하는 게 아니라 이전 예측기가 만든 잔여 오차를 새로운 예측기를 학습시킨다.

```
np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)
```

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg1.fit(X, y)
```

```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg2.fit(X, y2)
```

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg3.fit(X, y3)
```

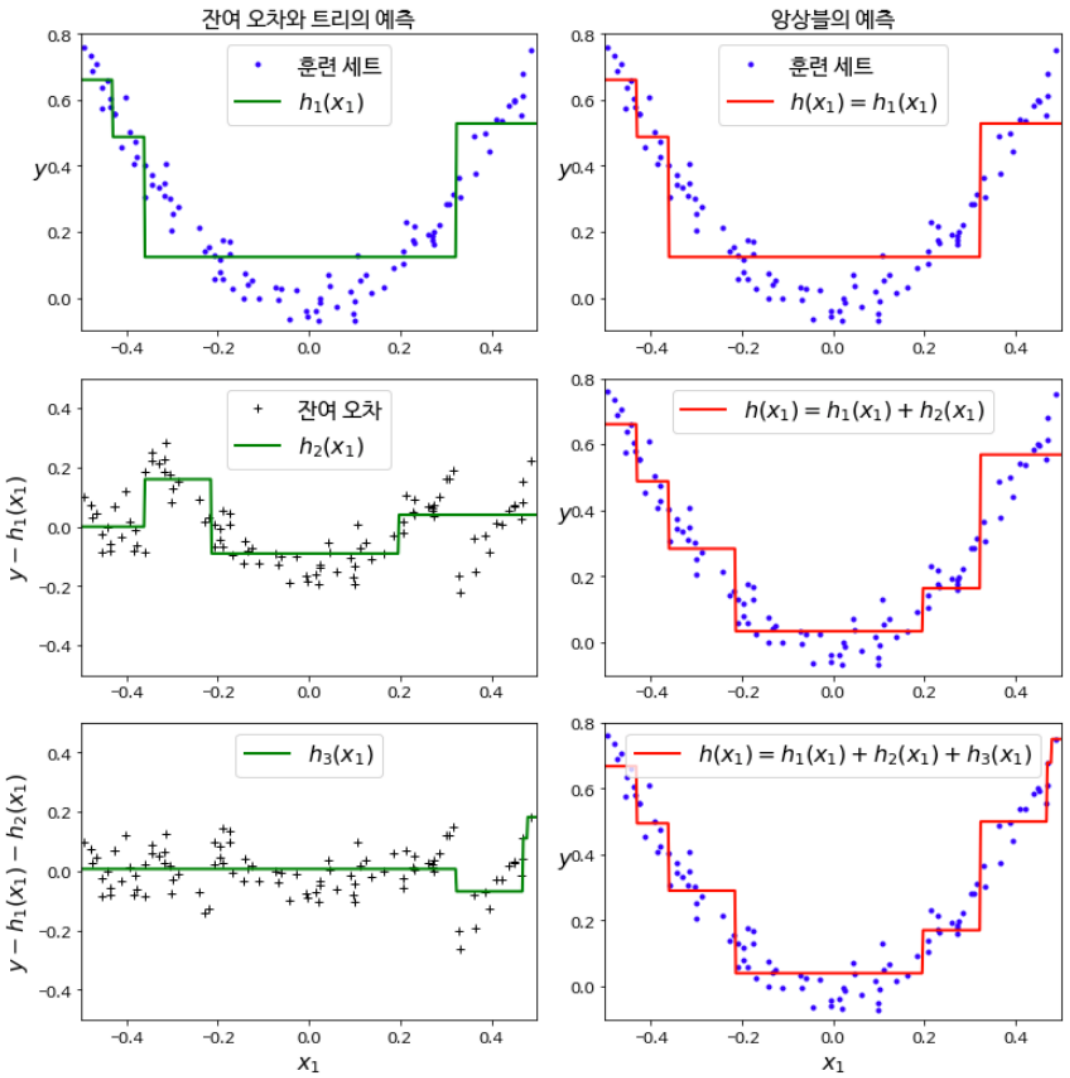
```
X_new = np.array([[0.8]])
```

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

```
y_pred
```

```
array([0.75026781])
```

7.5 부스팅


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

7.5 부스팅

```
from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=0.1, random_state=42)
gbrt.fit(X, y)
```

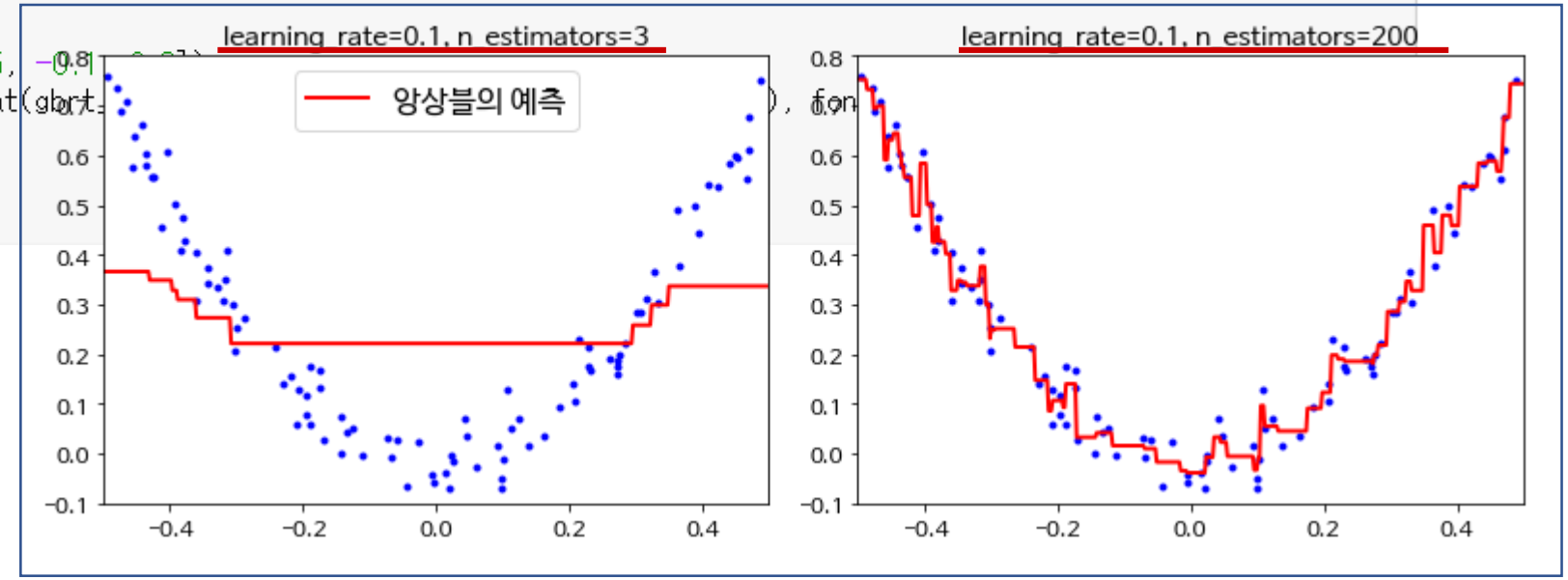
```
gbrt_slow = GradientBoostingRegressor(max_depth=2, n_estimators=200, learning_rate=0.1, random_state=42)
gbrt_slow.fit(X, y)
```

```
plt.figure(figsize=(11,4))

plt.subplot(121)
plot_predictions([gbrt], X, y, axes=[-0.5, 0.5, -0.1, 0.8], label="앙상블의 예측")
plt.title("learning_rate={}, n_estimators={}".format(gbrt.learning_rate, gbrt.n_estimators), fontsize=14)

plt.subplot(122)
plot_predictions([gbrt_slow], X, y, axes=[-0.5, 0.5, -0.1
plt.title("learning_rate={}, n_estimators={}".format(gbrt_

save_fig("gbrt_learning_rate_plot")
plt.show()
```



7.5 부스팅

```
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)
```

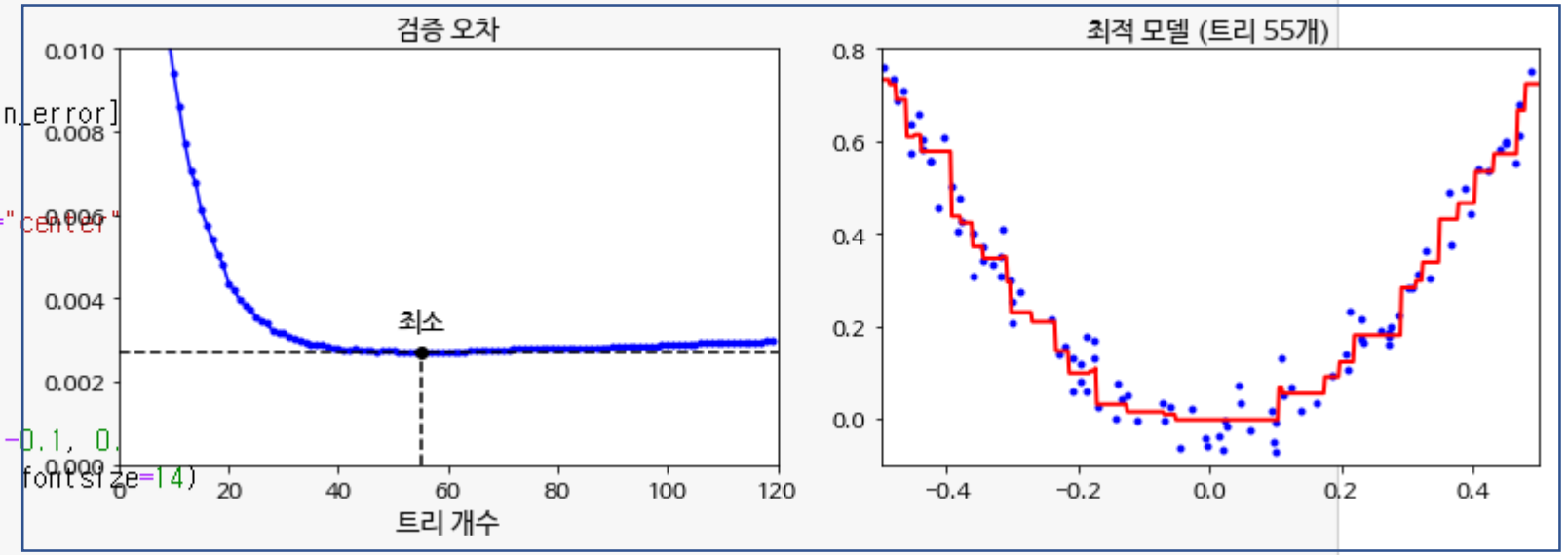
```
min_error = np.min(errors)
```

```
plt.figure(figsize=(11, 4))

plt.subplot(121)
plt.plot(errors, "b.-")
plt.plot([bst_n_estimators, bst_n_estimators], [0, min_error],
plt.plot([0, 120], [min_error, min_error], "k--")
plt.plot(bst_n_estimators, min_error, "ko")
plt.text(bst_n_estimators, min_error*1.2, "최소", ha="center"
plt.axis([0, 120, 0, 0.01])
plt.xlabel("트리 개수")
plt.title("검증 오차", fontsize=14)

plt.subplot(122)
plot_predictions([gbrt_best], X, y, axes=[-0.5, 0.5, -0.1, 0.
plt.title("최적 모델 (트리 %d개)" % bst_n_estimators, fontsize=14)

save_fig("early_stopping_gbrt_plot")
plt.show()
```



7.5 부스팅

```
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  # 조기 종료
```

```
print(gbrt.n_estimators)
```

```
61
```

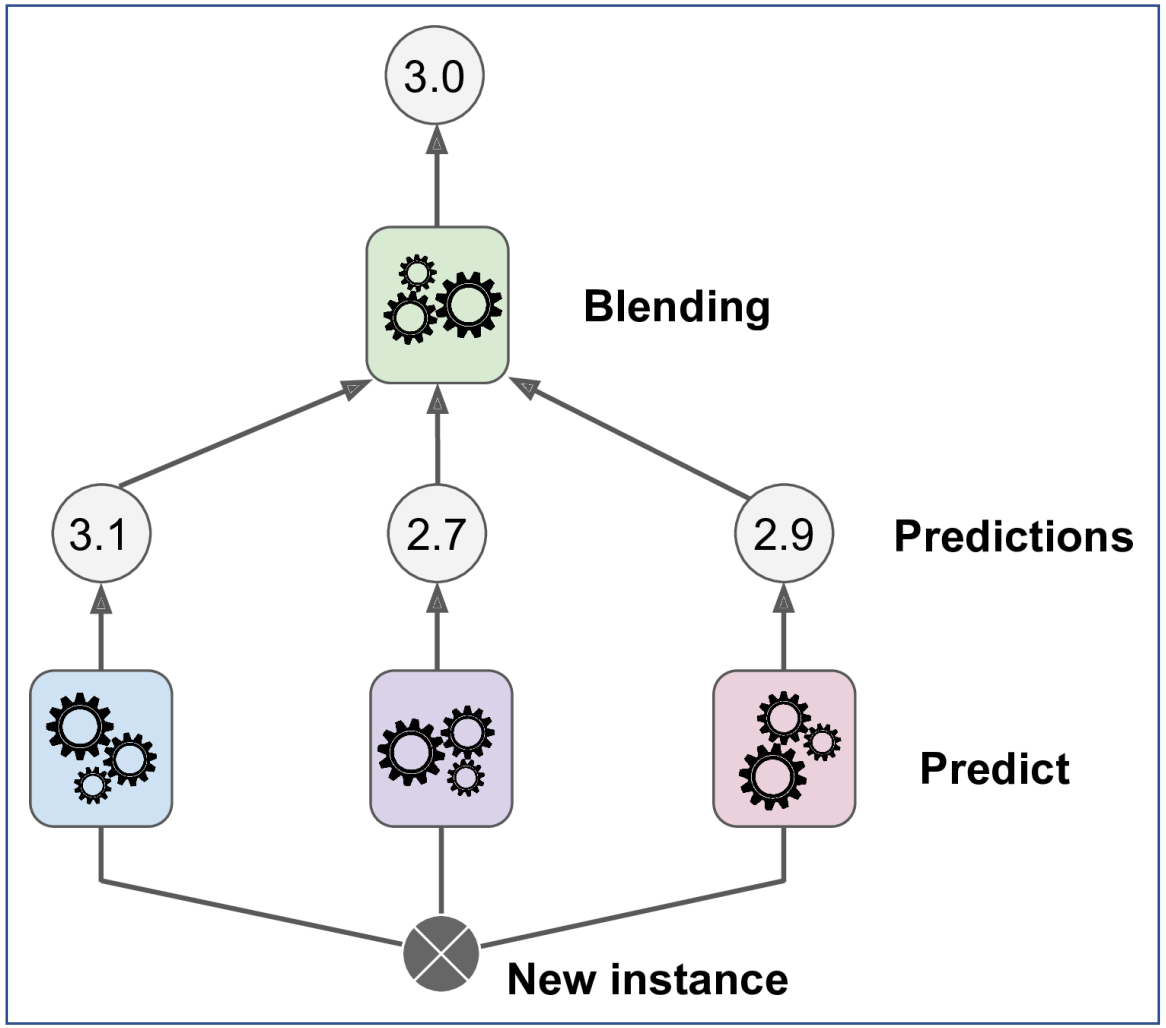
7.6 스태킹

스태킹 (stacking)

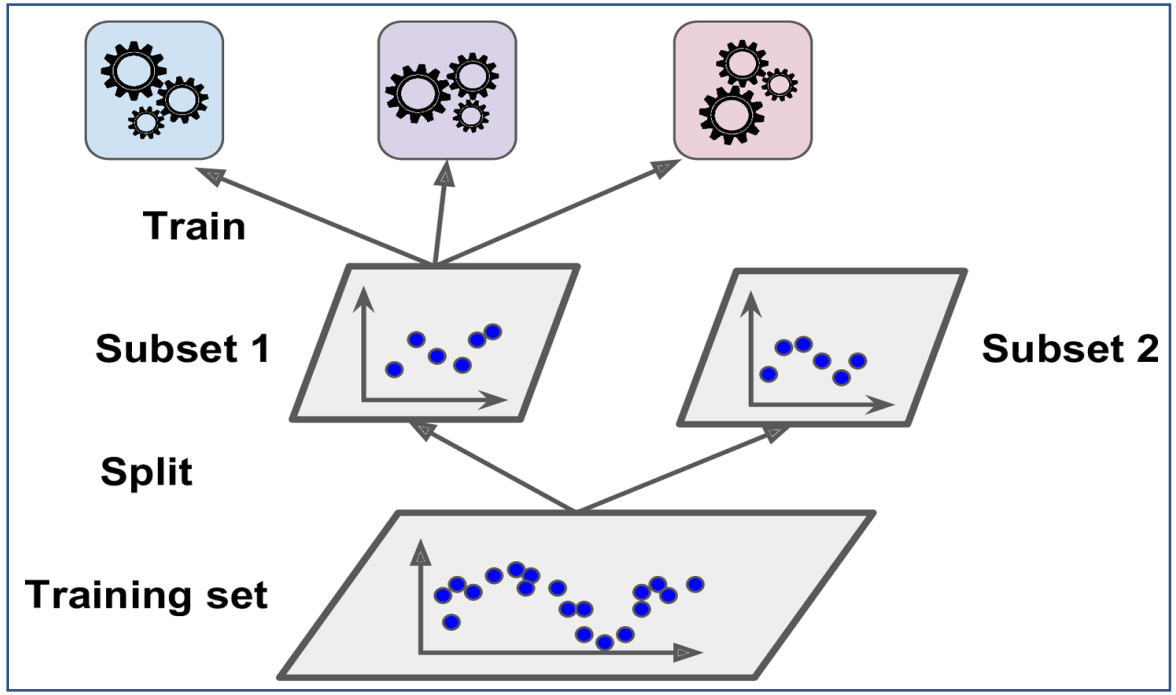
앙상블에 속한 모든 예측기의 예측을 취합하는 간단한 함수를 사용하는 대신 취합하는 모델을 훈련시킬 수 없을까? 라는 기본 아이디어

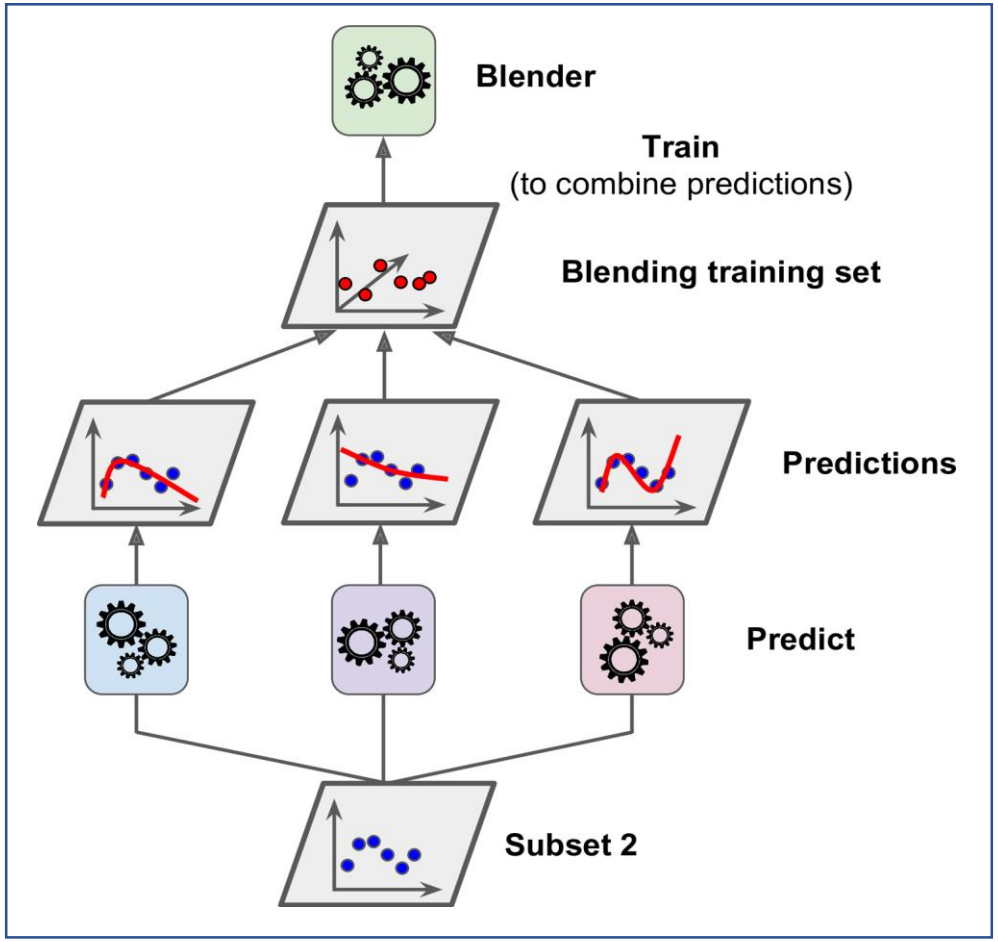
마지막 예측기(블렌더)가 예측들을 입력으로 받아 최종 예측(3.0)을 만든다.

안타깝게도 사이킷런은 스태킹을 직접 지원하지는 않는다.



7.6 스태킹

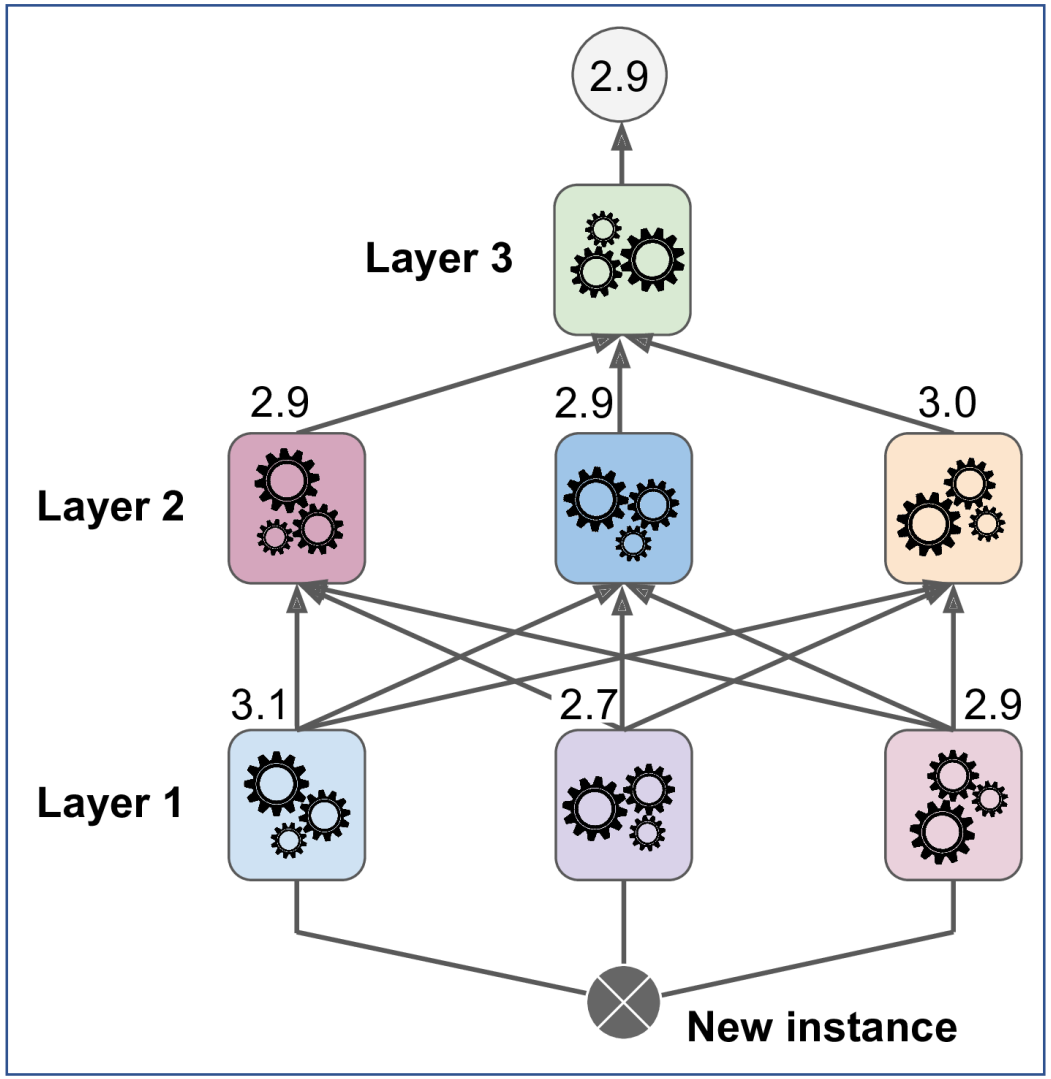




1) 먼저 훈련 세트를 두 개의 서브셋으로 나눈다.
2) 첫 번째 서브셋은 첫 번째 레이어를 훈련시키기 위해 사용된다.
3) 첫 번째 레이어의 예측기를 통해 두 번째 (홀드 아웃) 세트에 대한 예측을 만든다.
4) 세 개의 예측값은 새로운 훈련 세트로 만들 수 있다. (3차원)
5) 블렌더가 이 새로운 훈련세트로 훈련한다.

7.6 스태킹

멀티 레이어 스태킹 앙상블의 예측



Thank You!