Ch.7

Ensemble Learning and Random Forest

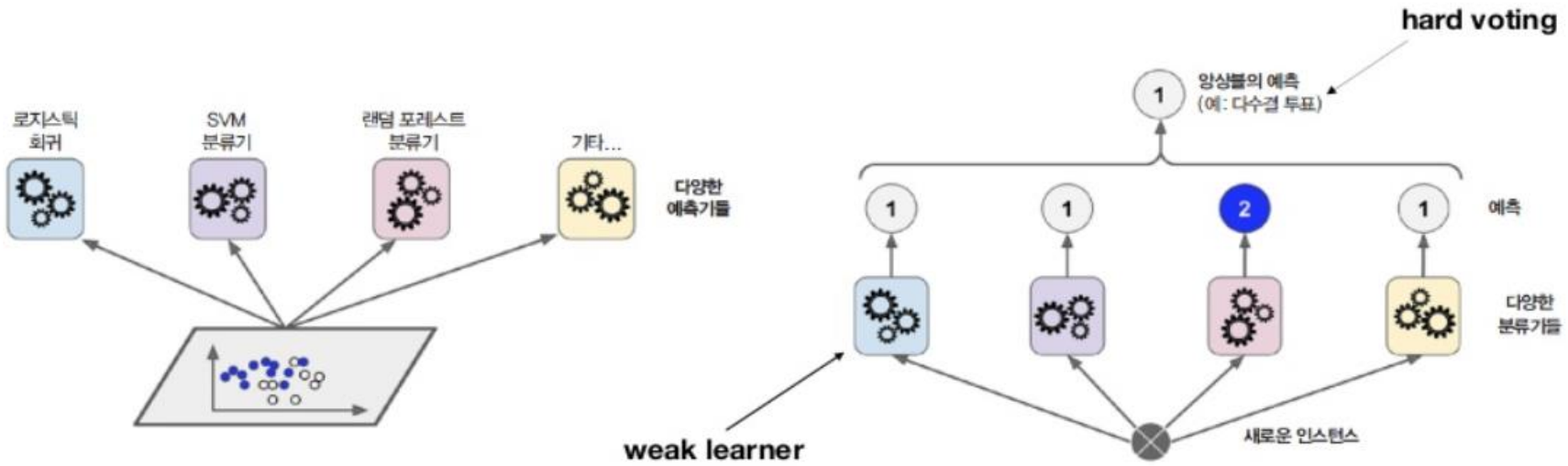
최우진

2019/01/23

앙상블 방법(Ensemble method)

- 여러 개의 모델을 합쳐 보다 나은 일반화 성능을 달성
- 앙상블 방법의 종류
- 배깅(bagging): 훈련 세트에서 중복을 허용하여 샘플링하는 방식
- 페이스팅(pasting): 훈련 세트에서 중복을 허용하지 않고 샘플링하는 방식
- 부스팅(boosting): 약한 학습기를 여러 개 연결하여 강한 학습기를 만드는 방식
  (즉, 이전 예측기의 오차를 보안하는 방식)
- 스태킹(stacking): 앙상블의 예측 결과를 사용하여 새로운 예측기를 훈련시키는 방식
  (즉, 앙상블 결과 위에 예측을 위한 모델을 추가하는 방식)

7.1 투표 기반 분류기

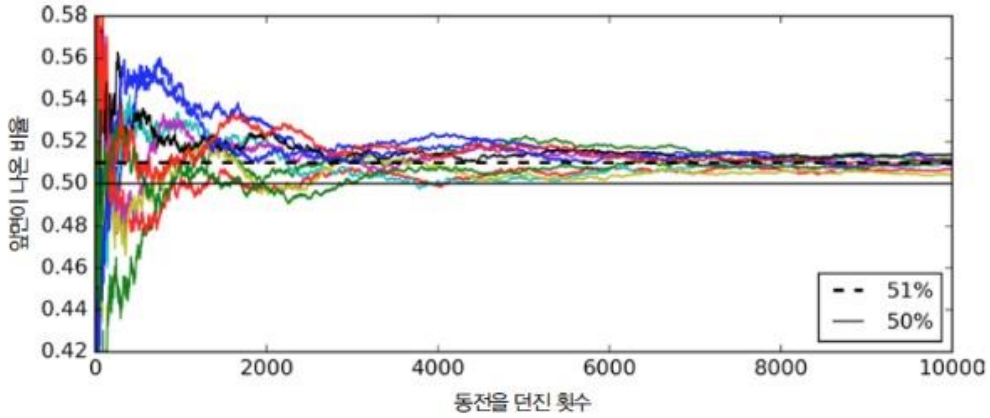


→ 다수결 투표 분류기가 앙상블에 포함된 개별 분류기 중 가장 뛰어난 것보다도 정확도가 높은 경우가 많다.

- 직접 투표(hard voting) 분류기: 다수결 투표로 정해지는 분류기
- 약한 학습기(weak learner): 랜덤 추측보다 조금 더 높은 성능을 내는 분류기
- 강한 학습기(strong learner): 높은 정확도를 내는 분류기

7.1 투표 기반 분류기

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

7.1 투표 기반 분류기

- VotingClassifier - hard voting

```
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)
```

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

7.1 투표 기반 분류기

- VotingClassifier - soft voting

```
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='warn',
          n_jobs=None, penalty='l2', random_state=42, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)), ('rf', Rando...bf',
  max_iter=-1, probability=True, random_state=42, shrinking=True,
  tol=0.001, verbose=False))],
         flatten_transform=None, n_jobs=None, voting='soft', weights=None)
```

간접 투표(soft voting)

- 모든 분류기가 클래스의 확률을 예측할 수 있으면(즉, predict_proba() 메소드가 있으면), 개별 분류기의 예측을 평균 내어 확률이 가장 높은 클래스를 예측하는 분류기
- 확률이 높은 투표에 비중을 더 두기 때문에 직접 투표 방식보다 성능이 높다.

7.1 투표 기반 분류기

- VotingClassifier의 결과

Hard voting의 결과

```
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
```

Soft voting의 결과

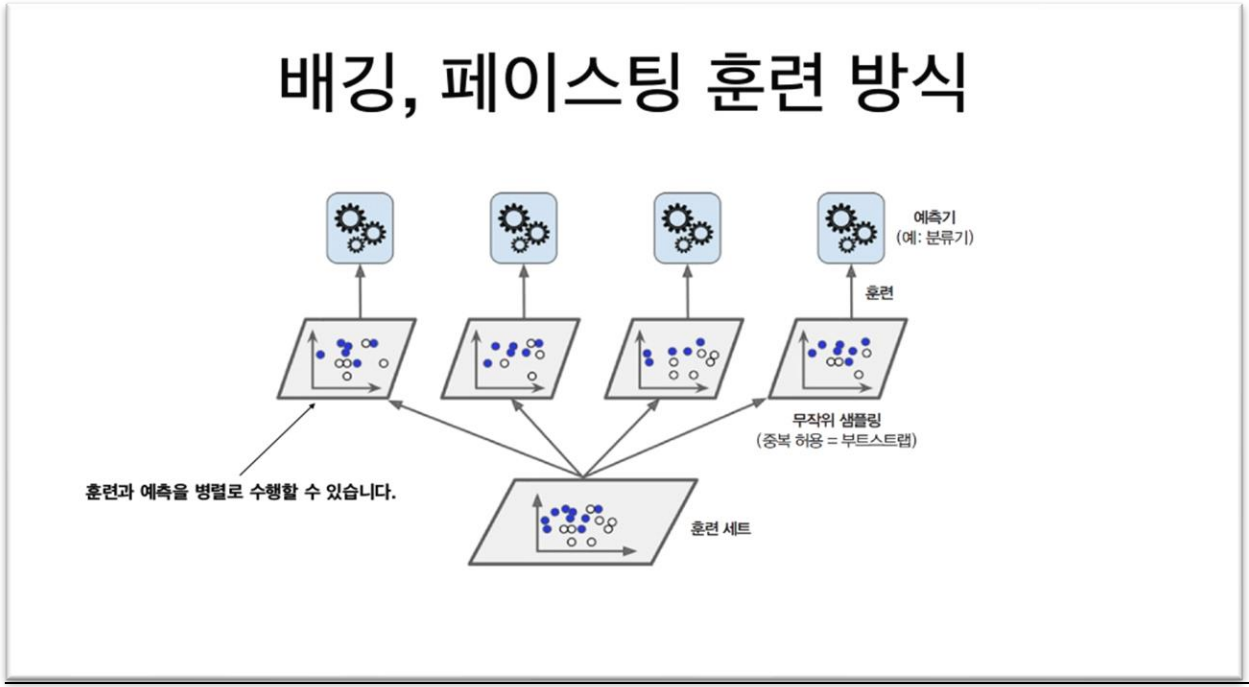
```
from sklearn.metrics import accuracy_score

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

- 같은 알고리즘을 사용하지만 훈련 세트의 서브셋을 무작위로 구성하여 여러 개의 분류기를 각기 다르게 학습
- 배깅(Boostrap aggregating): 훈련 세트에서 중복을 허용하여 샘플링하는 방식
- 페이스팅(Pasting): 훈련 세트에서 중복을 허용하지 않고 샘플링하는 방식



7.2.1 사이킷런의 배깅과 페이스팅

- BaggingClassifier: 기본적으로 간접 투표 방식을 사용하지만 분류기가 확률을 추정할 수 없으면(즉, predict_proba() 함수가 없으면) 직접 투표 방식으로 작동

- 배깅

```
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)
```

- 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=False, n_jobs=-1, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

- n_jobs: 사이킷런이 훈련과 예측에 사용할 CPU 코어 수 지정

  (-1: 가용한 모든 코어를 사용, 기본값: 1)

7.2.1 사이킷런의 배깅과 페이스팅

- 배깅(부스트래핑)은 각 예측기가 학습하는 서브셋에 다양성을 증가시킨다.
- 분산: 배깅 < 페이스팅
- 편향: 배깅 > 페이스팅 (배깅이 예측기들의 상관관계를 줄이므로)
- 전반적으로 배깅이 더 나은 모델을 만들어 일반적으로 더 선호하지만,

  교차 검증으로 두 모델을 모두 확인하여 더 나은 쪽을 선택하는 것이 좋다.

7.2.2 oob 평가 (out-of-bag)

- 배깅을 사용하면 어떤 샘플은 한 예측기를 위해 여러 번 샘플링되고 어떤 것은 전혀 선택되지 않을 수 있다.
- n개의 샘플을 n번 선택했을 때 한 번도 포함되지 않을 확률
- $\boldsymbol{y = (1 - \frac{1}{n})^n}$ ➡ $\ln(y) = \ln\left(1 - \frac{1}{n}\right)^n = n\ln\left(1 - \frac{1}{n}\right)$

$$= \frac{1}{x}\ln(1-x) = \frac{\ln(1-x)}{x}$$

$x = \frac{1}{n}$

- $\boldsymbol{n \to \infty}$, 즉 x → 0일 때 로피탈의 정리를 적용하면

$$\lim_{x\to 0}\frac{\ln(1-x)}{x} = \lim_{x\to 0}\frac{\frac{d}{dx}\boldsymbol{ln(1-x)}}{\frac{d}{dx}\boldsymbol{x}} = \lim_{x\to 0}\frac{\boldsymbol{-1}}{\boldsymbol{1-x}} = \boldsymbol{-1}$$

7.2.2 oob 평가 (out-of-bag)

- $y = e^{-1} \approx 0.368$

샘플 개수 n이 아주 클 때, 어떤 샘플이 무작위한 n번의 선택에 한번도 포함되지 않을 확률 = 약 37% 이다. (oob 샘플)

- 교차 검증 대신 남겨진 샘플을 사용하여 분류기를 평가할 수 있다.

7.3 랜덤 패치와 랜덤 서브스페이스

- BaggingClassifier - 특성(feature) 샘플링 가능
- bootstrap_features: 중복을 허용한 특성 샘플링 여부
- max_features: 랜덤하게 사용할 최대 특성수
- 랜덤 패치 방식 - 훈련 특성과 샘플을 모두 샘플링하는 방식
- 랜덤 서브스페이스 방식 - 훈련 샘플을 모두 사용하고 특성은 샘플링하는 방식
- 특성 샘플링은 더 다양한 예측기를 만들며 편향을 늘리는 대신 분산을 낮춘다.

	랜덤 패치	랜덤 서브스페이스
bootstrap	True	False
max_samples	< 1.0	= 1.0
bootstrap_features	True	True
max_features	< 1.0	< 1.0

7.4 랜덤 포레스트

- 일반적으로 배깅(또는 페이스팅)을 적용한 결정 트리의 앙상블
- 랜덤 포레스트 알고리즘

  트리의 노드를 분할할 때 전체 특성 중에서 최신의 특성을 찾는 대신

  무작위로 선택한 특성 후보 중에서 최적의특성을 찾는 방식

- 결국 트리를 더욱 다양하게 만들고 편향은 높아지고 분산을 낮춘다.
- RandomForestClassifier(또는 RandomForestRegressor)
- BaggingClassifier + DecisionTreeClassifier 와 비슷하게 구현 가능

7.4.1 엑스트라 트리

- 트리를 더욱 무작위하게 만들기 위해 최적의 임곗값을 찾는 대신 후보 특성을 사용해 무작위로 분할한 다음 그중에서 최상의 분할을 선택하는 방식
- 극단적으로 무작위한 트리의 랜덤 포레스트
- 편향이 늘어나지만 대신 분산을 낮추게 된다.
- 속도: 랜덤 포레스트 < 엑스트라 트리
- 이유: 모든 노드에서 특성마다 가장 최적의 임곗값을 찾는 것이 트리-알고리즘에서 가장 시간이 많이 소요되는 작업 중 하나이기 때문이다.

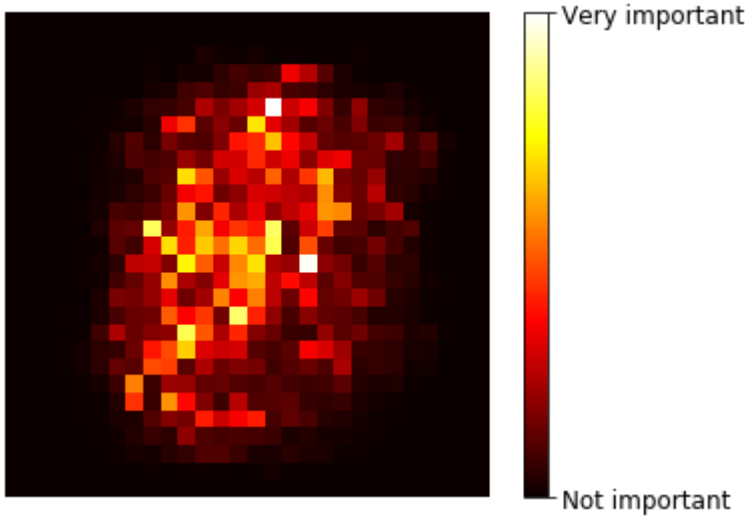
7.4.2 특성 중요도

- 랜덤 포레스트 장점: 특성의 상대적 중요도를 측정하기 쉽다.
- 평균적으로 불순도를 얼마나 감소시키는지 확인하여 특성 중요도를 측정한다.
- 가중치 평균이며 각 노드의 가중치는 연관된 훈련 샘플 수와 동일하다.

- 사이킷런 - 훈련이 끝난 뒤 특성마다 자동으로 중요도를 계산하고 중요도의 전체 합이 1이 되도록 결괏값을 정규화한다.
- (결과 값은 feature_importances_ 변수에 저장)

7.4.2 특성 중요도

- 노드에 사용된 결정 트리의 특성 중요도 = (현재 노드의 샘플 비율 * 불순도) - (왼쪽자식 노드의 샘플 비율 *불순도) - (오른쪽자식 노드의 샘플 비율 *불순도)
- 샘플 비율은 트리 전체 샘플 수에 대한 비율
- 랜덤 포레스트의 특성 중요도 = 각 결정 트리의 특성 중요도의 합 / 트리 수



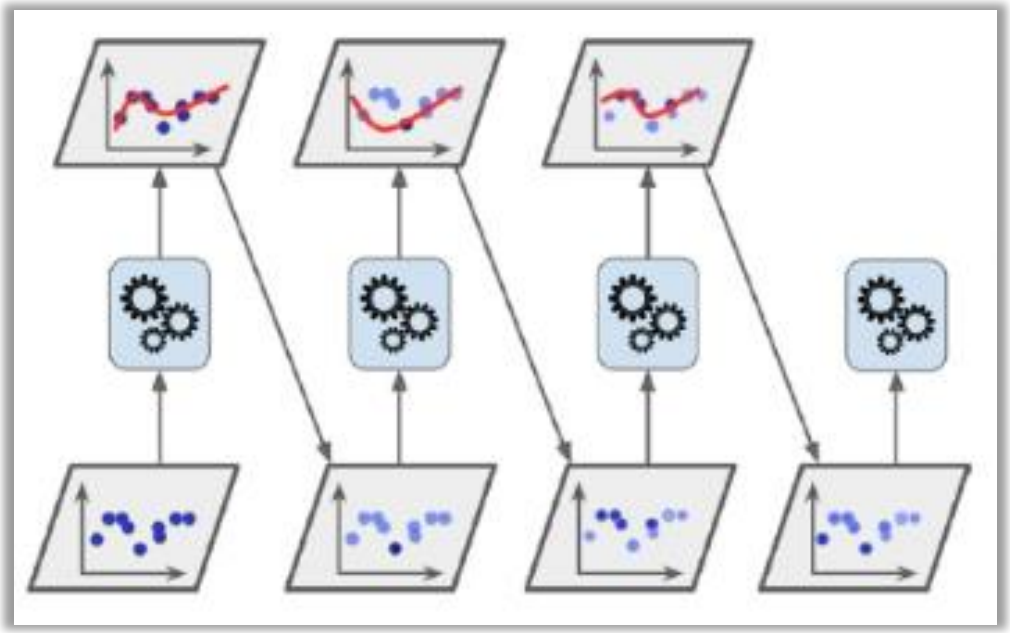
특성의 중요도를
빠르게 확인할 수 있다

7.5 부스팅

- 약한 학습기를 여러 개 연결하여 강한 학습기를 만드는 앙상블 방법
- 부스팅 방법: 앞의 모델을 보완해나가면서 일련의 예측기를 학습시킨다.
- 종류: 아다부스트(AdaBoost), 그래디언트 부스팅(Gradient Boosting)
- 연속된 학습 기법이다보니 병렬화를 하지 못하여, 배깅이나 페이스팅만큼 확장성이 높지는 않습니다.

7.5.1 아다부스트

- 이전 모델이 과소적합했던 훈련 샘플의 가중치를 더 높여 학습시키는 모델
- 새로운 예측기는 학습하기 어려운 샘플에 점점 더 맞춰지게 된다.



7.5.1 아다부스트

- **아다부스트 알고리즘**

가중치 적용된 에러율(w 초깃값은 1/m)

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

예측기 가중치 계산

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

- $\hat{y}_j^{(i)}$는 i번째 샘플에 대한 j번째 예측기의 예측
- 예측기가 정확할수록 가중치가 더 높아진다.
- 무작위로 예측하여 에러율(r)이 0.5에 가까워지면 $\frac{1-r}{r} \approx 1$ 이 되므로 예측기 가중치가 0에 가까워진다.
- 에러율(r) > 0.5 이면 $\frac{1-r}{r} < 1$ 이 되어 가중치가 음수가 된다.

7.5.1 아다부스트

- **아다부스트 알고리즘**

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 $i = 1, 2, \cdots, m$

- 잘못 분류된 샘플의 가중치가 증가된다.
- 모든 샘플의 가중치를 정규화한다.

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

7.5.1 아다부스트

- 아다부스트 알고리즘

사이킷런의 아다부스트 = SAMME의 이진 분류(K=2) 버전

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

- K : 클래스 수
- N : 예측기 수

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

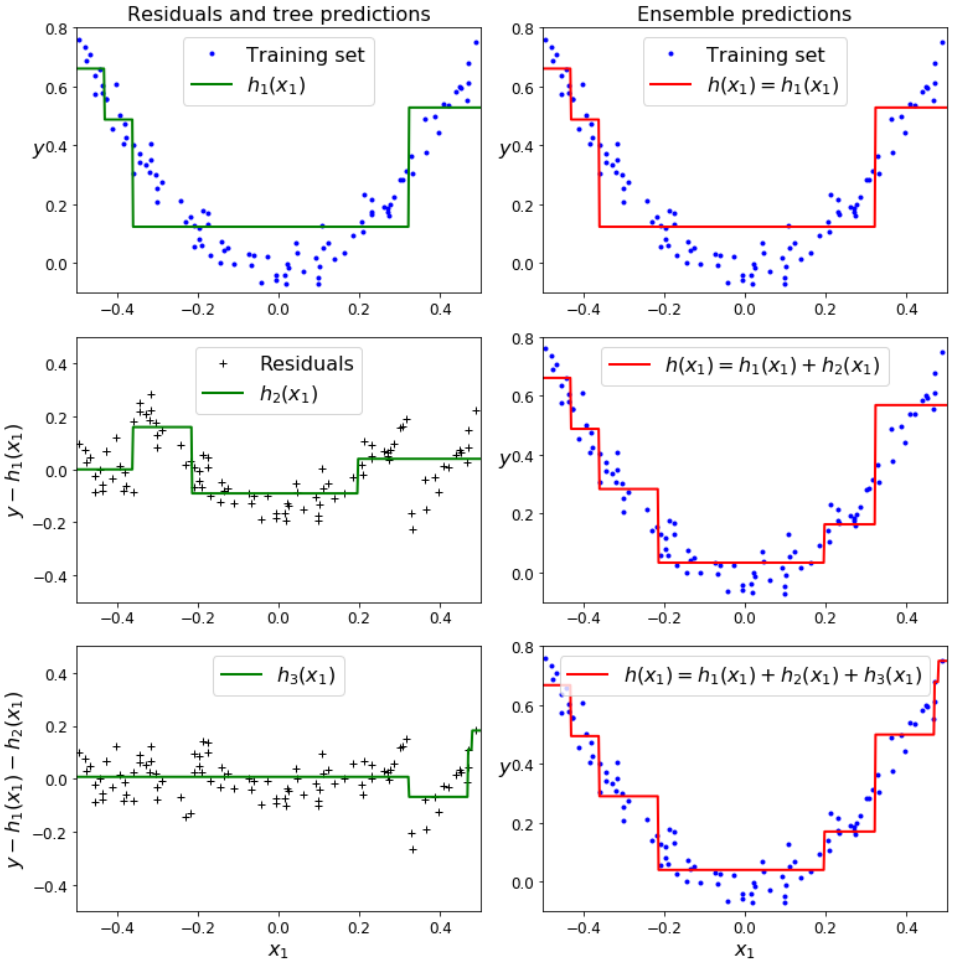
7.5.2 그래디언트 부스팅

- 앙상블에 이전까지의 오차를 보정하도록 예측기를 순차적으로 추가하지만 이전 예측기가 만든 잔여 오차에 새로운 예측기를 학습시킨다.
- 그래디언트 트리 부스팅 또는 그래디언트 부스티드 회귀 트리(GBRT)
- GradientBoostingClassifier(loss='deviance'),
- GradientBoostingRegressor(loss='ls')

  - deviance: 로지스틱 손실함수
  - ls: 최소 제곱

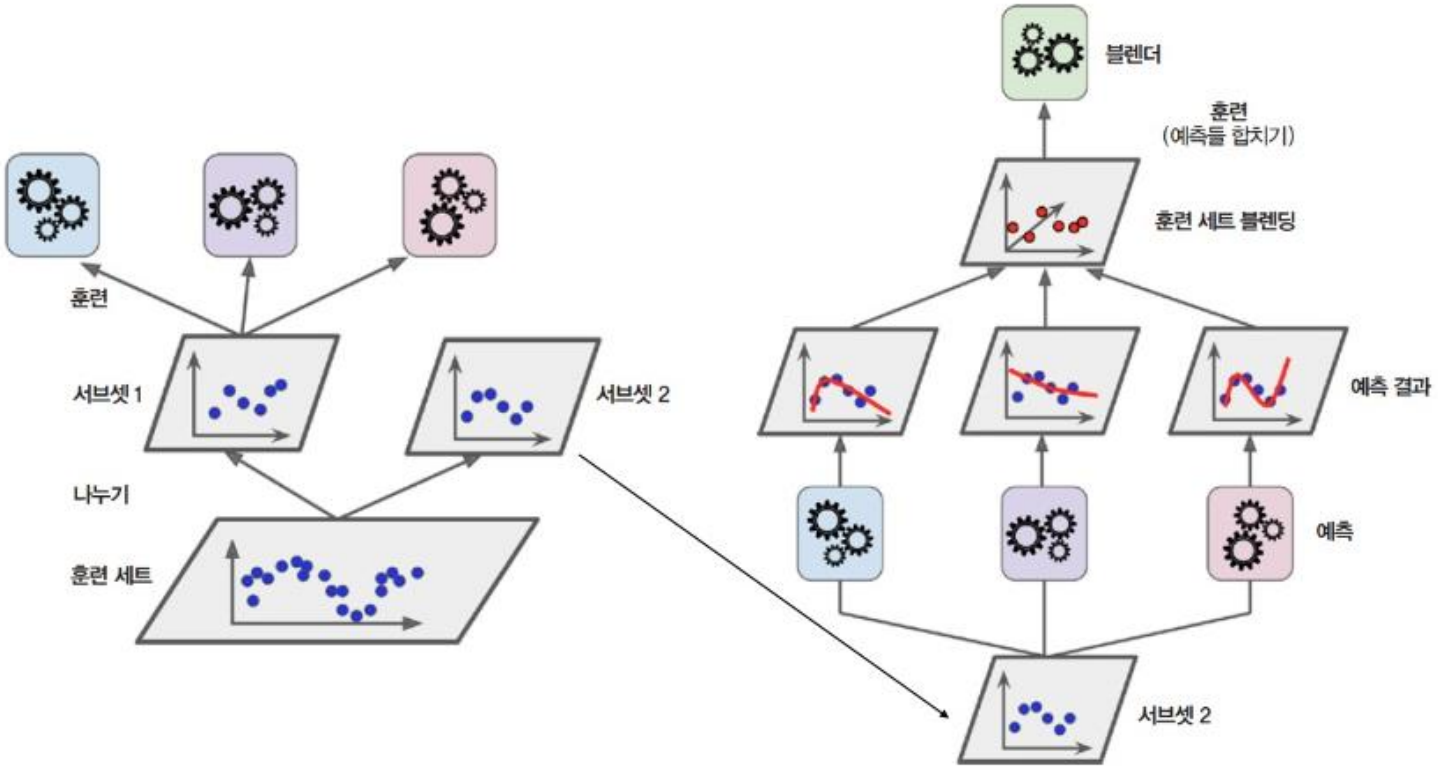
7.5.2 그래디언트 부스팅

잔여 오차와 예측



7.6 스태킹

- 앙상블의 예측 결과를 사용하여 새로운 예측기(블렌더, 메타학습기)를 훈련



감사합니다