

7장 앙상블 학습과 랜덤포레스트

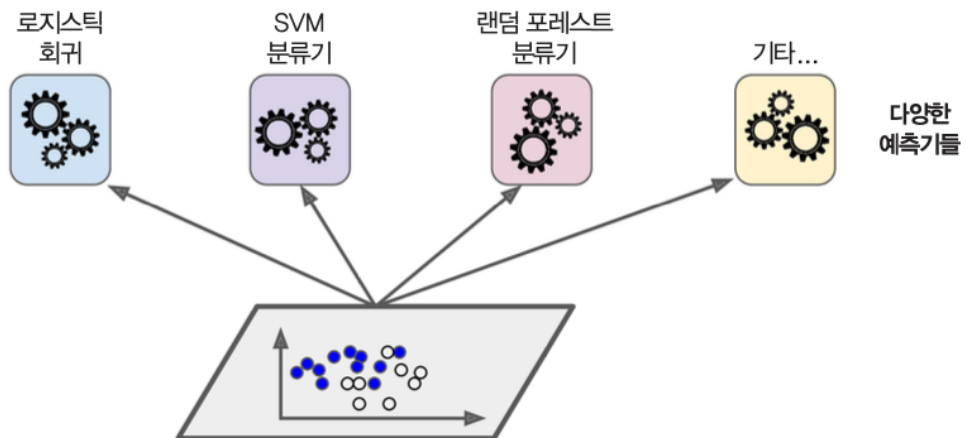
- 앙상블: 여러 개의 예측기로 이루어진 그룹
- 앙상블 학습: 예측기 여러 개의 결과를 종합하여 예측값을 지정하는 학습
- 앙상블 기법: 앙상블 학습을 지원하는 앙상블 학습 알고리즘

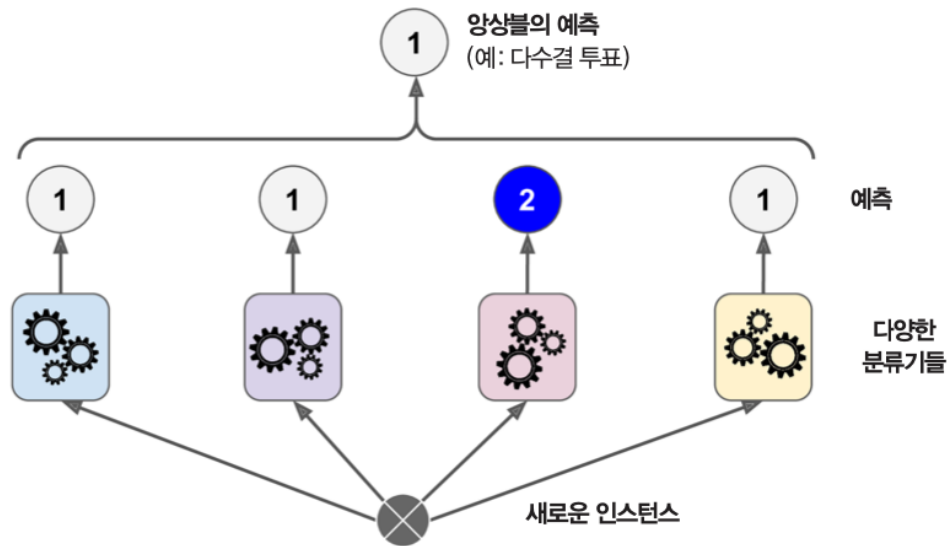
주요 내용

- 다수결 분류기
- 배깅/페이스팅
 - 랜덤 패치와 랜덤 서브스페이스
- 랜덤포레스트
 - 엑스트라 트리
- 부스팅
 - 에이다부스트
 - 그레이디언트 부스팅

7.1 다수결(투표기반) 분류기

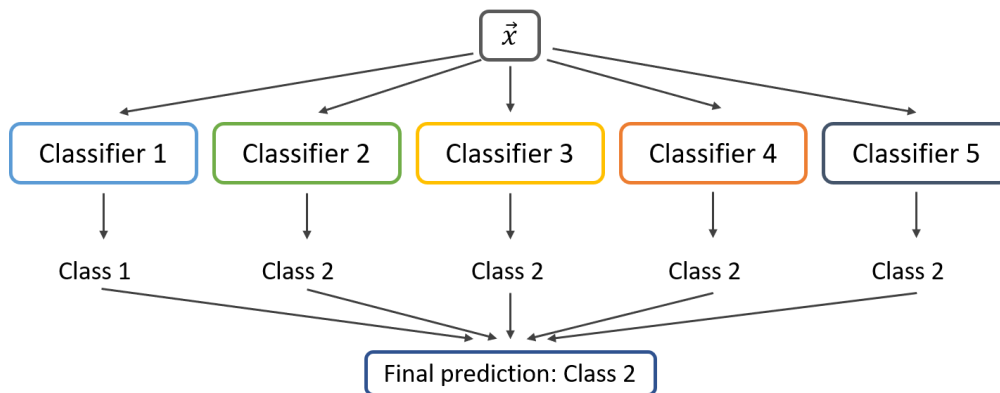
- 동일한 훈련 세트에 대해 여러 종류의 분류기 이용한 앙상블 학습 적용 후 다수결 방식으로 예측값 결정





직접투표

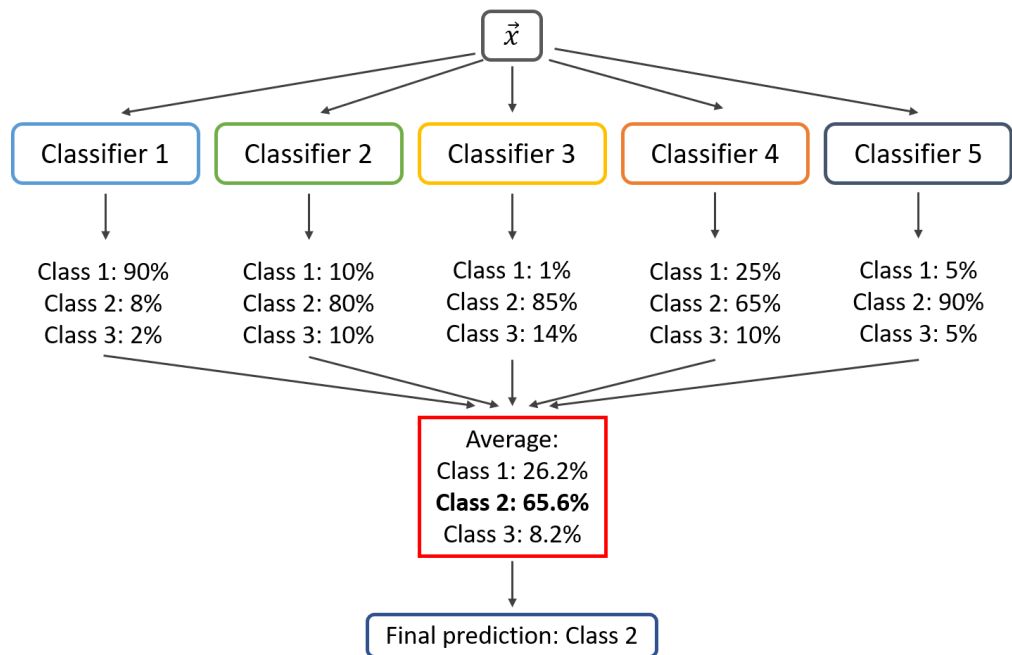
- 앙상블에 포함된 예측기들의 예측값들의 다수로 결정



<그림출처: [kaggle \(https://www.kaggle.com/fengdanye/machine-learning-6-basic-ensemble-learning\)](https://www.kaggle.com/fengdanye/machine-learning-6-basic-ensemble-learning).>

간접투표

- 앙상블에 포함된 예측기들의 예측한 확률값들의 평균값으로 예측값 결정
- 전제: 모든 예측기가 `predict_proba()` 메서드와 같은 확률 예측 기능을 지원해야 함.
- 높은 확률에 보다 비중을 두기 때문에 직접투표 방식보다 성능 좀 더 좋음.



<그림출처: [kaggle \(https://www.kaggle.com/fengdanye/machine-learning-6-basic-ensemble-learning\)](https://www.kaggle.com/fengdanye/machine-learning-6-basic-ensemble-learning)>

다수결 분류기의 특징

- 앙상블에 포함된 분류기들 사이의 독립성이 전제되는 경우 개별 분류기 보다 정확한 예측 가능
- 단, 독립성이 보장되지 못한다면 다수결 분류기의 성능이 더 낮아질 수 있음

다수결 분류기 예제

- 사이킷런의 다수결 분류기
- `VotingClassifier`: 다수결 분류기 모델 제공

```
# 선형 회귀
log_clf = LogisticRegression(solver="lbfgs", random_state=42)
# 랜덤포레스트
rnd_clf = RandomForestClassifier(n_estimators=100, random_state=42)
# 서포트벡터머신
svm_clf = SVC(gamma="scale", random_state=42)

# 다수결 분류기: 직접 투표
voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)], voting='hard')

# 다수결 분류기 학습
voting_clf.fit(X_train, y_train)
```

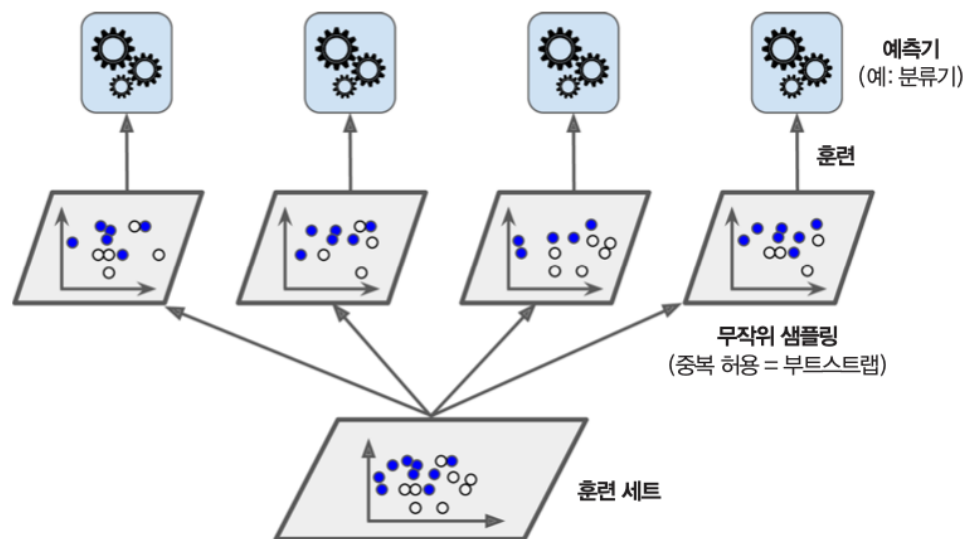
- `voting='hard'`: 직접 투표 방식 지정 하이퍼 파라미터
- `voting='soft'`: 간접 투표 방식 지정 하이퍼 파라미터
 - 주의: SVC 모델 지정할 때 `probability=True` 사용해야 `predict_proba()` 메서드 지원됨.

7.2 배깅/페이스팅

- 동일한 예측기를 훈련 세트의 다양한 부분집합을 대상으로 학습시키는 방식
- 학습에 사용되는 부분집합에 따라 훈련세트가 다른 예측기를 학습시키는 앙상블 학습 기법임.
- 부분집합을 임의로 선택할 때 중복 허용 여부에 따라 앙상블 학습 방식이 달라짐
 - 배깅: 중복 허용 샘플링
 - 페이스팅: 중복 미허용 샘플링

배깅 관련 주의사항

- 배깅(bagging): bootstrap aggregation의 줄임말
- 통계 분야에서 부트스트래핑, 즉, 중복허용 리샘플링으로 불림
- 배깅 방식만은 동일 훈련 샘플을 여러번 샘플링할 수 있음.



배깅/페이스팅 예측 방식

- 개별 예측기의 결과를 종합해서 최종 예측값 지정
- 분류 모델: 직접 투표 방식 사용. 즉, 수집된 예측값들 중에서 최빈값(mode) 선택
- 회귀 모델: 수집된 예측값들의 평균값 선택

앙상블 학습의 편향과 분산

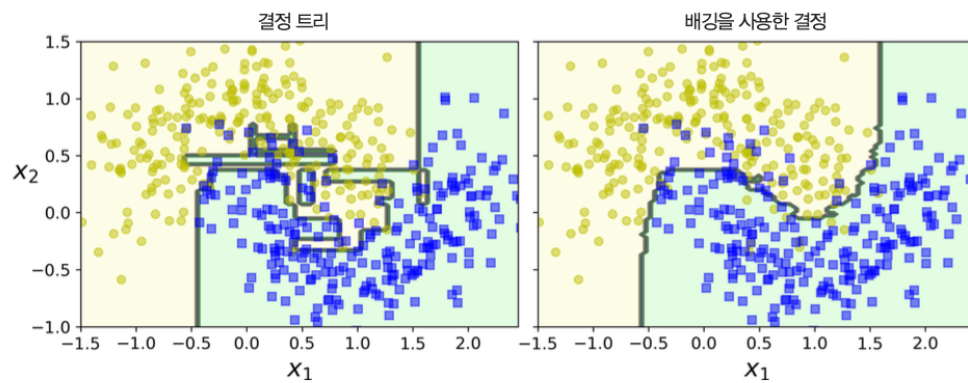
- 개별 예측기의 경우에 비해 편향은 비슷하지만 분산은 줄어듦. 즉, 과대적합의 위험성이 줄어듦.
- 개별 예측기: 배깅/페이스팅 방식으로 학습하면 전체 훈련 세트를 대상으로 학습한 경우에 비해 편향이 커짐. 따라서 과소적합 위험성 커짐.

사이킷런의 배깅/페이스팅

- 분류 모델: `BaggingClassifier`
- 회귀 모델: `BaggingRegressor`

예제

```
bag_clf = BaggingClassifier(  
    DecisionTreeClassifier(random_state=42),  
    n_estimators=500,          # 의사결정나무 500개 사용  
    max_samples=100,          # 부분집합 크기: 100개 샘플 사용. 기본값: 1.0. 즉, 훈련 샘플 전  
    체 선택.  
    bootstrap=True,           # 기본값:배깅방식. 페이스팅은 False 지정  
    n_jobs=-1,  
    random_state=42)  
  
bag_clf.fit(X_train, y_train)
```



배깅 대 페이스팅

- 일반적으로 배깅 방식이 편향은 키우고, 분산은 줄임
 - 표본 샘플링의 무작위성으로 인해 예측기들 사이의 상관관계 정도가 약화되기 때문
- 전반적으로 배깅 방식이 좀 더 나은 모델 생성
- 하지만 교차검증을 통해 확인 필요

oob 평가

- oob(out-of-bag) 샘플: 선택되지 않은 훈련 샘플
- oob 샘플을 활용하여 앙상블 학습에 사용된 개별 예측기의 성능 평가 가능
- `BaggingClassifier` 의 `oob_score=True` 로 설정하면 oob 평가를 자동으로 실행

7.3 랜덤 패치와 랜덤 서브스페이스

- BaggingClassifier 는 특성 대상 샘플링 기능 지원: max_features 와 bootstrap_features
- 이미지 등과 같이 매우 높은 차원의 데이터셋을 다룰 때 유용하게 사용
- 더 다양한 예측기를 만들며, 편향이 커지지만 분산은 낮아짐

max_features

- 학습에 사용할 특성 수 지정
- 특성 선택은 무작위
 - 정수인 경우: 지정된 수만큼 특성 선택
 - 부동소수점($\in [0, 1]$)인 경우: 지정된 비율만큼 특성 선택
- max_samples와 유사 기능 수행

bootstrap_features

- 학습에 사용할 특성을 선택할 때 중복 허용 여부 지정
- 기본값은 False. 즉, 중복 허용하지 않음.
- botostrap과 유사 기능 수행

랜덤 패치 기법

- 훈련 샘플과 훈련 특성 모두를 대상으로 중복을 허용하며 임의의 샘플 수와 임의의 특성 수만큼 샘플링해서 학습하는 기법
- 아래 두 조건이 참이어야 함
 - `bootstrap=True` 또는 `max_samples < 1.0`
 - `bootstrap_features=True` 또는 `max_features < 1.0`

랜덤 서브스페이스 기법

- 전체 훈련 세트를 학습 대상으로 삼지만 훈련 특성은 임의의 특성 수만큼 샘플링해서 학습하는 기법
- 아래 두 조건이 참이어야 함
 - `bootstrap=False` 그리고 `max_samples=1.0`
 - `bootstrap_features=True` 또는 `max_features < 1.0`

7.4 랜덤포레스트

- 배깅/페이스팅 방법을 적용한 의사결정나무의 앙상블을 최적화한 모델
- 분류 용도: `RandomForestClassifier`
- 회귀 용도: `RandomForestRegressor`

예제

```
rnd_clf = RandomForestClassifier(  
    n_estimators=500,  
    max_leaf_nodes=16,          # 최대 16개의 나뭇잎 허용  
    n_jobs=-1,  
    random_state=42)  
  
rnd_clf.fit(X_train, y_train)
```

엑스트라 트리

랜덤포레스트의 마디 분할 방식

- 특성: 무작위 선택
- 특성 임계값: 무작위로 분할한 다음 최적값 선택

엑스트라 트리(extra-trees)의 마디 분할 방식

- 특성과 특성 임계값 모두 무작위 선택
- 일반적인 랜덤포레스트보다 속도가 훨씬 빠름
- 이 방식을 사용하면 편향은 늘고, 분산은 줄어듦

예제

```
extra_clf = ExtraTreesClassifier(  
    n_estimators=500,  
    max_leaf_nodes=16,  
    n_jobs=-1,  
    random_state=42)  
  
extra_clf.fit(X_train, y_train)
```

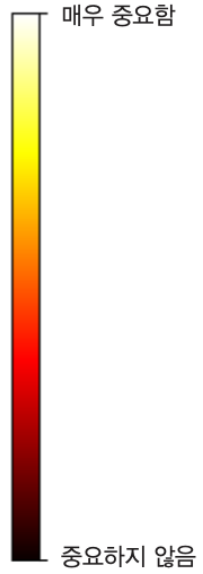
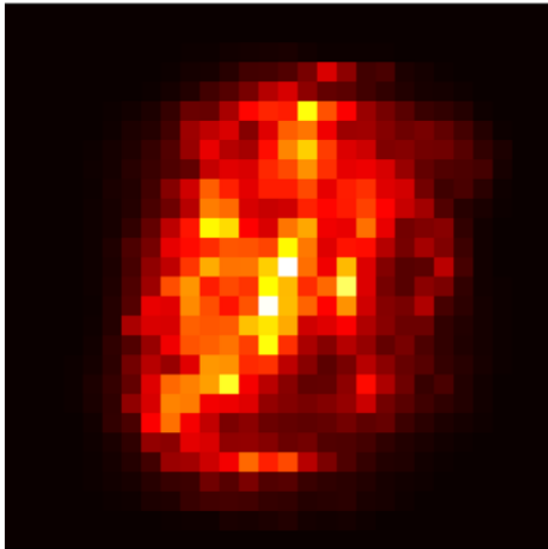
특성 중요도

- 랜덤포레스트는 학습에 사용된 특성의 상대적 중요도도 함께 측정
- 특성 중요도: 해당 특성을 사용한 마디가 평균적으로 불순도를 얼마나 감소시키는지 측정
 - 즉, 불순도를 많이 줄이면 그만큼 중요도가 커짐
- 사이킷런의 `RandomForestClassifier`: 훈련이 끝난 뒤 특성별 중요도의 전체 합이 1이 되도록 하는 방식
 - 특성별 상대적 중요도를 측정한 후 `feature_importances_` 속성에 저장

예제: 붓꽃 데이터셋

- 꽃잎 길이(petal length): 44.1%
- 꽃잎 너비(petal width): 42.3%
- 꽃받침 길이(sepal length): 11.3%
- 꽃받침 너비(sepal width): 2.3%

예제: MNIST

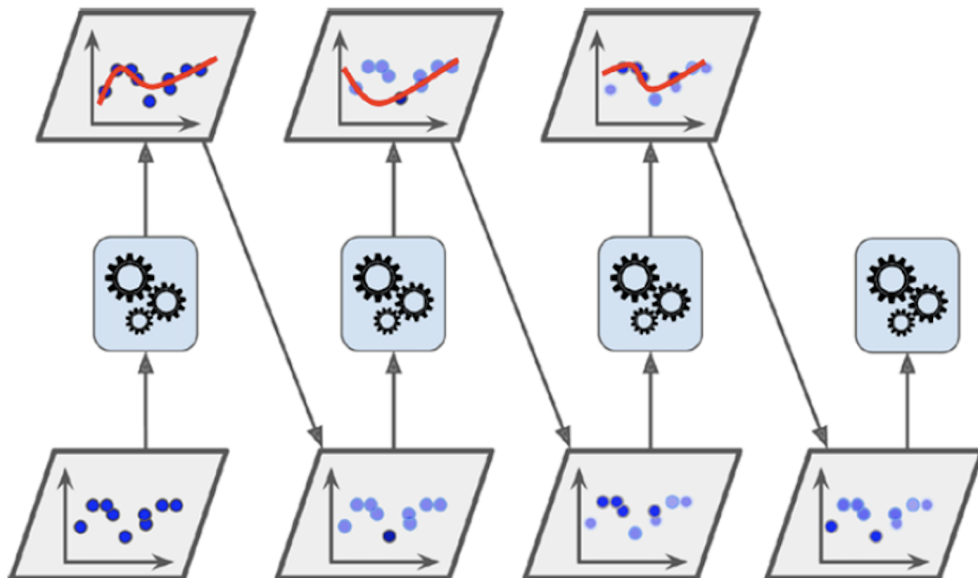


7.5 부스팅

- 부스팅(boosting): 성능이 약한 학습기의 여러 개를 선형으로 연결하여 강한 성능의 학습기를 만드는 앙상블 기법
 - 순차적으로 이전 학습기의 결과를 바탕으로 성능을 조금씩 높여가는 방식
 - 순차적으로 학습하기에 배깅/페이스팅에 비해 확장성이 떨어짐
- 성능이 약한 예측기의 단점을 보완하여 좋은 성능의 예측기를 훈련해 나가는 것이 부스팅의 기본 아이디어

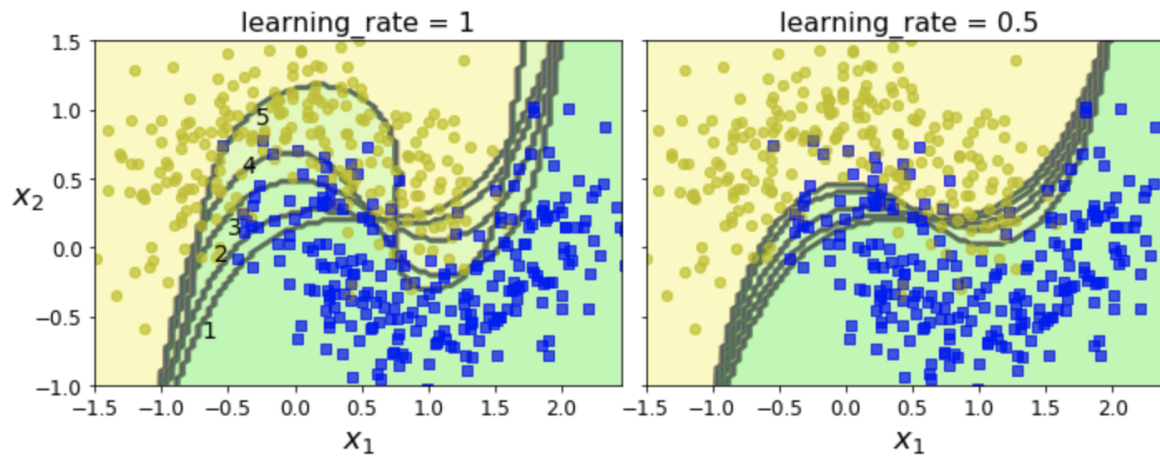
에이다부스트(AdaBoost)

- 좀 더 나은 예측기를 생성하기 위해 잘못 적용된 가중치를 조정하여 새로운 예측기를 추가하는 앙상블 기법.
- 이전 모델이 제대로 학습하지 못한, 즉 과소적합했던 샘플들에 대한 가중치를 더 높이는 방식으로 새로운 모델 생성.
- 따라서 새로운 예측기는 학습하기 어려운 샘플에 조금씩 더 잘 적응하는 모델이 연속적으로 만들어져 감.



예제: 에이다 부스트 + SVC

- moons 데이터셋에 rbf 커널을 사용하는 SVC 모델 적용
- 5번 연속으로 학습한 결과



사이키런의 에이다부스트

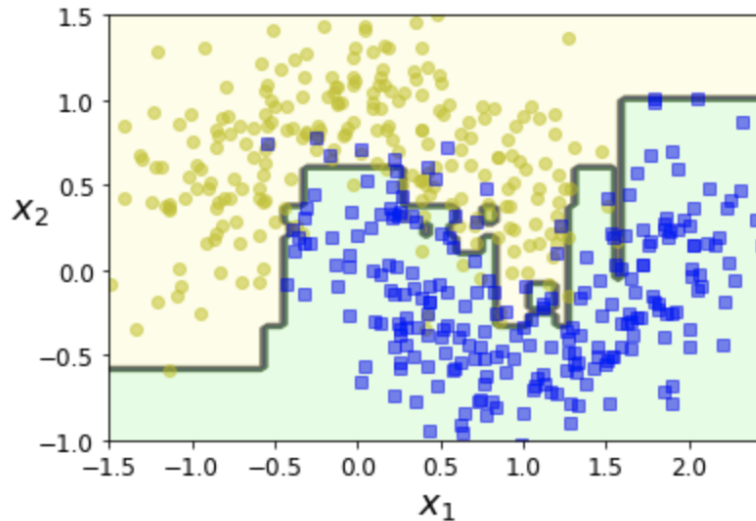
- 분류 모델: AdaBoostClassifier
- 회귀 모델: AdaBoostRegressor

예제: 에이다 부스트 + 의사결정나무

- moons 데이터셋에 의사결정나무 사용

```
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)
```



그레이디언트 부스팅

- 이전 학습기에 의한 오차를 보정하도록 새로운 예측기를 순차적으로 추가하는 아이디어는 에이다 부스트와 동일
- 샘플의 가중치를 수정하는 대신 이전 예측기가 만든 잔차(residual error)에 대해 새로운 예측기를 학습시킴
- 잔차(residual error): 예측값과 실제값 사이의 오차

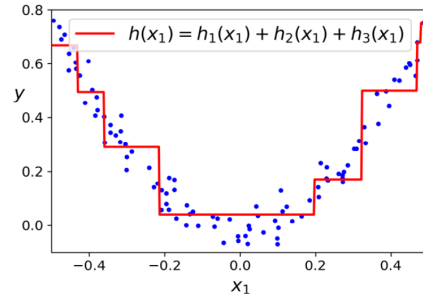
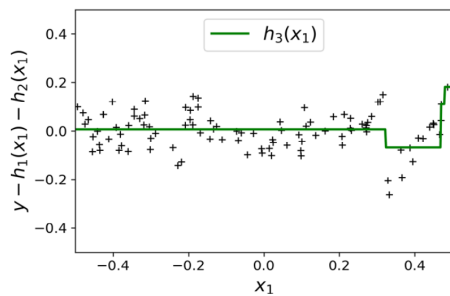
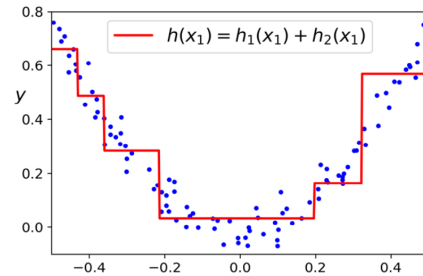
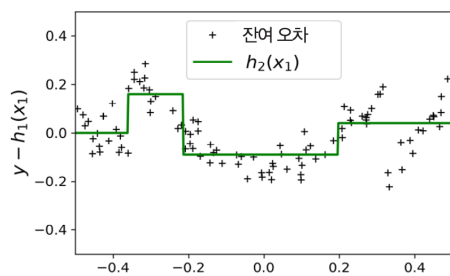
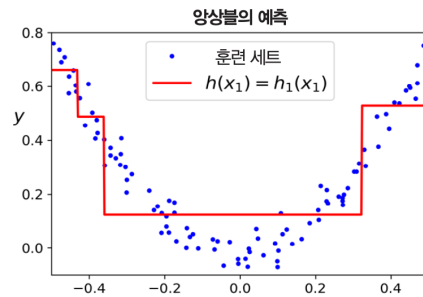
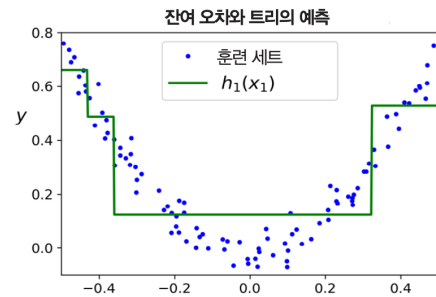
사이킷런 그레이디언트 부스팅 모델

- 분류 모델: GradientBoostingClassifier
 - RandomForestClassifier와 비슷한 하이퍼파라미터를 제공
- 회귀 모델: GradientBoostingRegressor
 - RandomForestRegressor와 비슷한 하이퍼파라미터를 제공

그레이디언트 부스티드 회귀 나무(GBRT) 예제: 그레이디언트 부스팅 (회귀)+ 의사결정나무

- 2차 다항식 데이터셋에 의사결정나무 3개를 적용한 효과와 동일하게 작동

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

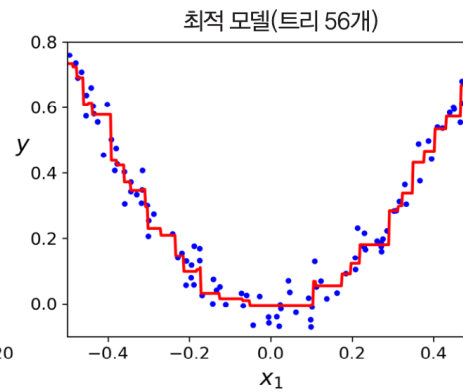
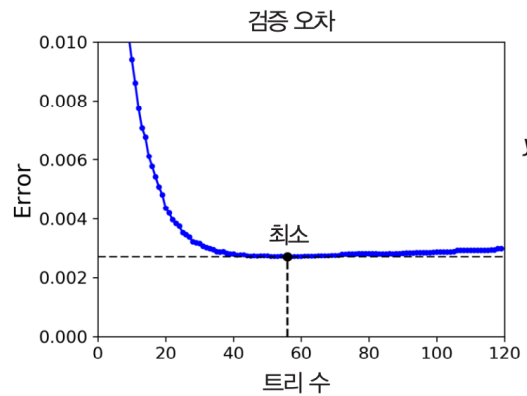


learning_rate(학습률)

- `learnign_rate`는 기존에 설명한 학습률과 다른 의미의 학습률.
 - 하지만 기본 아이디어는 동일.
- 이전 의사결정나무에서 학습된 값을 전달할 때 사용되는 비율
 - 1.0이면 그대로 전달
 - 1.0보다 작으면 해당 비율 만큼 조금만 전달

최적의 의사결정나무 수 확인법

- 조기종료 기법 활용



확률적 그레이디언트 부스팅

- 각 의사결정나무가 훈련에 사용할 훈련 샘플의 비율을 지정하여 학습
- 속도 빠름
- 편향 높아지지만, 분산 낮아짐.

XGBoost

- Extreme Gradient Boosting의 줄임말.
- 빠른 속도, 확장성, 이식성 뛰어남.
- 사이킷런과 비슷한 API 제공
- 조기종료 등 다양한 기능 제공.

7.6 스택킹

- 스택킹 아이디어는 간단함.
- 사이킷런이 지원하지 않지만 구현은 어렵지 않음.
- 하지만 스택킹 설명은 생략.