7장 앙상블 학습과 랜덤포레스트

감사의 글

자료를 공개한 저자 오렐리앙 제롱과 강의자료를 지원한 한빛아카데미에게 진심어린 감사를 전합니다.

주요 내용

- 앙상블 학습이란?
- 배깅/페이스팅
- 램덤포레스트
- 부스팅
- 스태킹

앙상블 학습이란?

- 여러 개 모델의 예측값을 조합하여 일반화 성능이 좋은 보다 강력한 모델을 구현하며, 대표적으로 평균화 기법과 부스팅 기법이 사용된다.
- 평균화 기법: 독립적으로 학습된 예측기 여러 개의 예측값들의 평균값을 예측값으로 사용하여 분산이 줄어든 모델을 구현한다.
    - 예제: 배깅(Bagging) 기법, 랜덤 포레스트(Random Forest)등
- 부스팅 기법: 예측기 여러 개를 순차적으로 쌓아 예측값의 편차를 줄인 모델을 구현한다.
    - 예제: 에이다 부스트(AdaBoost), 그레이디언트 부스팅(Gradient Tree Boosting) 등

앙상블 학습에 대한 확률적 근거

이항분포의 누적분포함수를 이용하여 앙상블 학습의 성능이 향상되는 이유를 설명할 수 있음.

- p: 예측기 하나의 성능
- n: 예측기 개수
- 반환값: 다수결을 따를 때 성공할 확률, 즉 다수결 의견이 보다 정확할 확률. 이항 분포의 누적분포 함수 활용.

In [1]:

```
from scipy.stats import binom

def ensemble_win_proba(n, p):
    """
    p: 예측기 하나의 성능
    n: 앙상블 크기, 즉 예측기 개수
    반환값: 다수결을 따를 때 성공할 확률. 이항 분포의 누적분포함수 활용.
    """

    return 1 - binom.cdf(int(n*0.4999), n, p)
```

적중률 51% 모델 1,000개의 다수결을 따르면 74.7% 정도의 적중률 나옴.

In [2]:
```
ensemble_win_proba(1000, 0.51)
```

Out[2]:
```
0.7467502275561786
```

적중률 51% 모델 10,000개의 다수결을 따르면 97.8% 정도의 적중률 나옴.

In [3]:
```
ensemble_win_proba(10000, 0.51)
```

Out[3]:
```
0.9777976478701533
```

적중률 90% 모델 10개의 다수결을 따르면 100%에 가까운 성능이 가능함.

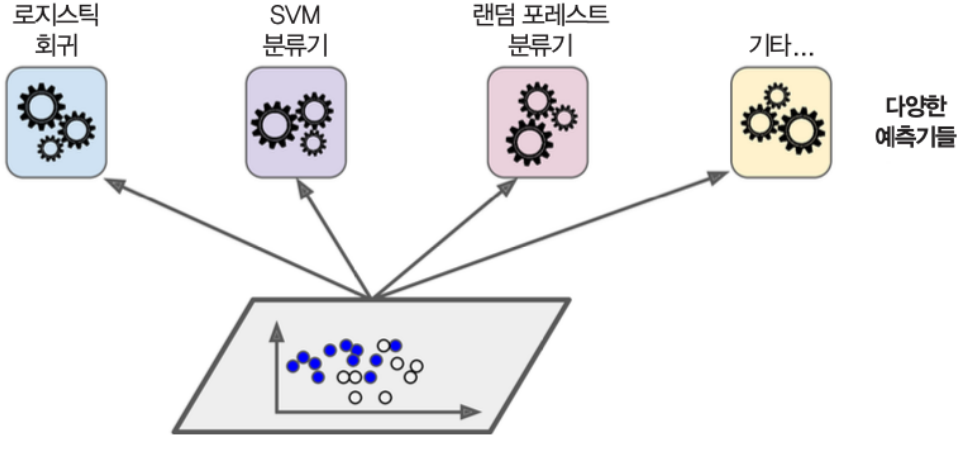
In [4]:
```
ensemble_win_proba(10, 0.9)
```

Out[4]:
```
0.9998530974
```

- **주의사항:** 앙상블 학습에 포함된 각각의 모델이 서로 독립인 것을 전제로한 결과임.
- 동일한 데이터를 사용할 경우 독립성이 보장되지 않으며, 경우에 따라 성능이 하락할 수 있음.
- 독립성을 높이기 위해 매우 다른 알고리즘을 사용하는 여러 모델을 사용해야 함.

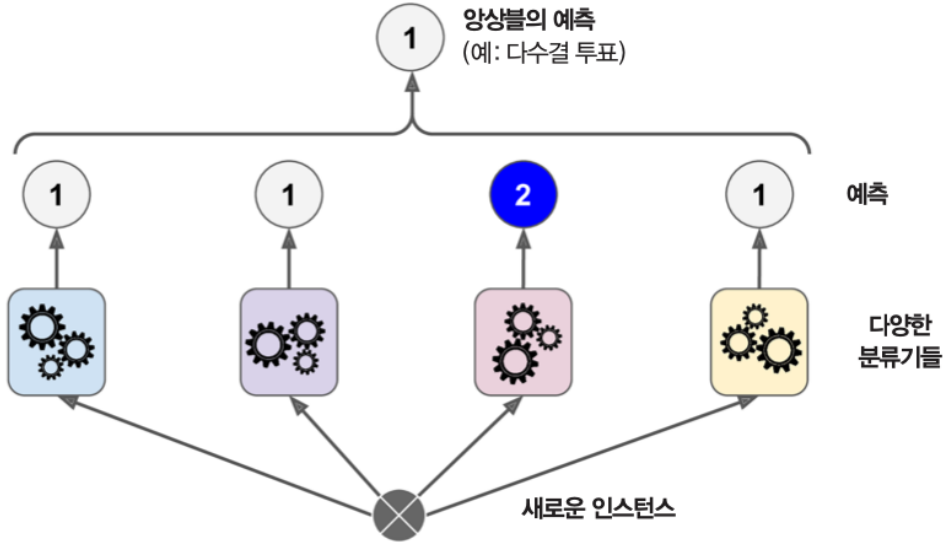
7.1 투표식 분류기

- 동일한 훈련 세트에 대해 여러 종류의 분류기 이용한 앙상블 학습 적용 후 직접 또는 간접 투표를 통해 예측값 결정.



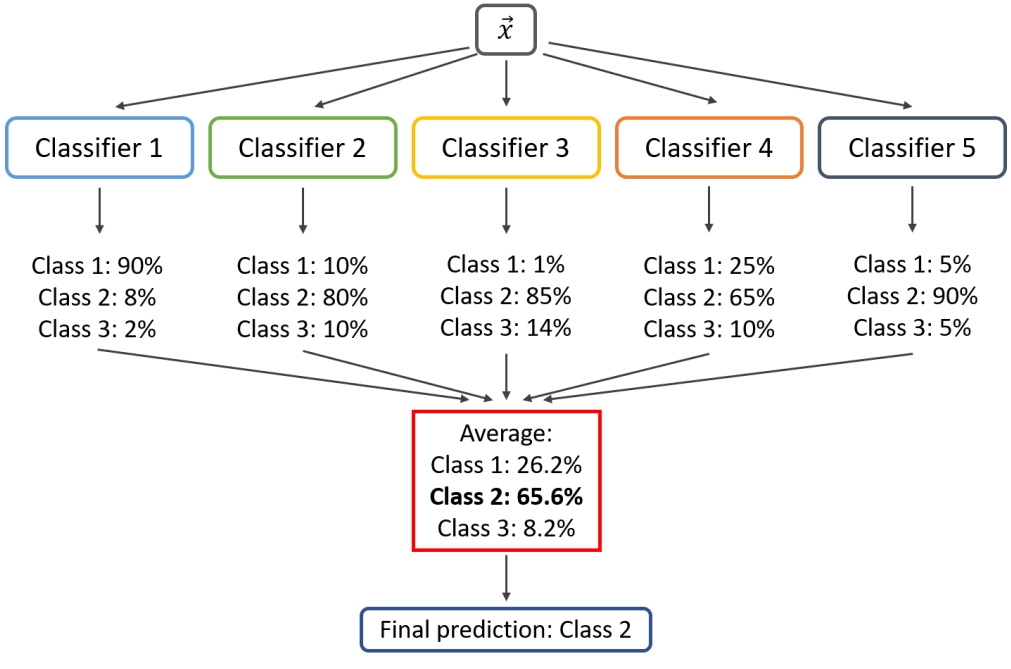
직접투표

- 앙상블에 포함된 예측기들의 예측값들의 다수로 결정



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>

투표식 분류기 예제

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

```python
# 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)

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

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

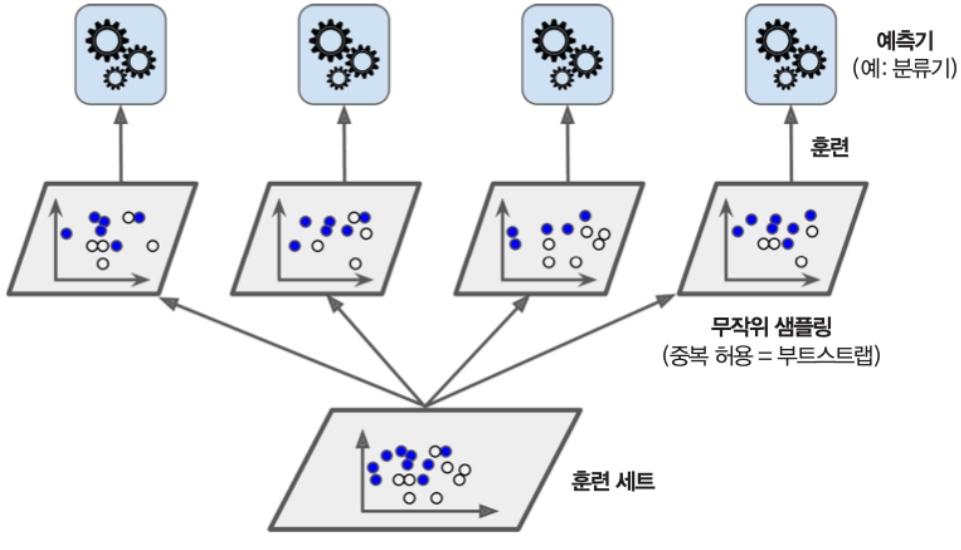
- `voting='hard'` : 직접 투표 방식 지정 하이퍼 파라미터
- `voting='soft'` : 간접 투표 방식 지정 하이퍼 파라미터
    - 주의: `SVC` 모델 지정할 때 `probability=True` 사용해야 `predict_proba()` 메서드 지원됨.

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

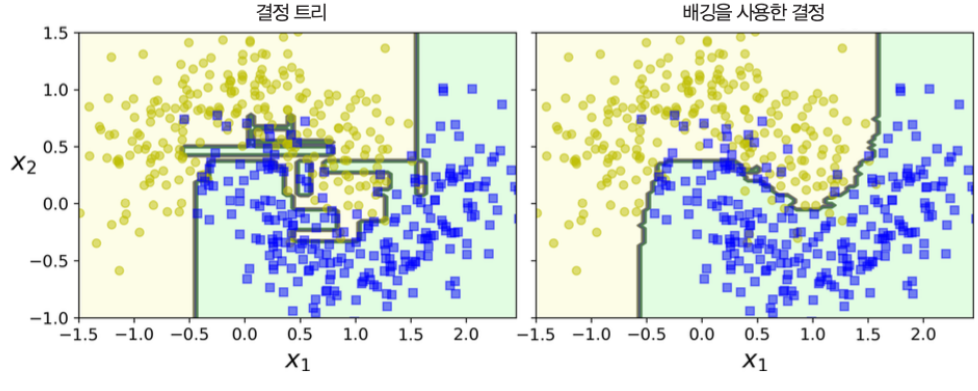
- 개별 예측기의 경우에 비해 편향은 비슷하지만 분산은 줄어듦. 즉, 과대적합의 위험성이 줄어듦.
- 개별 예측기: 배깅/페이스팅 방식으로 학습하면 전체 훈련 세트를 대상으로 학습한 경우에 비해 편향이 커짐. 따라서 과소적합 위험성 커짐.

참고: Single estimator versus bagging: bias-variance decomposition의 번역본

예제: 사이킷런의 배깅/페이스팅

- 훈련세트(`X_train`) 크기: 500
- `n_estimators=500` : 결정트리 500개 사용
- `max_samples=100` : 각 예측기가 100개 샘플 사용. 기본값은 1.0, 즉 전체 훈련 샘플 선택
- `bootstrap=True` : 배깅 방식 사용(기본값). `False` 면 페이스팅 방식 사용.
- `n_jobs=-1` : 모든 사용가능한 cpu 사용하여 훈련을 **병렬처리**함. 양의 정수일 경우 정해진 수의 cpu 사용.

```
bag_clf = BaggingClassifier(DecisionTreeClassifier(random_state=42),
                            n_estimators=500, max_samples=100, bootstrap=True,
n_jobs=-1, random_state=42)
```



배깅(Bagging) vs. 페이스팅(Pasting)

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

oob 평가

- oob(out-of-bag) 샘플: 선택되지 않은 훈련 샘플. 예측기별 약 63% 정도.
- oob 샘플을 활용하여 앙상블 학습에 사용된 개별 예측기의 성능 평가 가능 앙상블 모델의 oob 평가는 개별 예측기의 oob 점수의 평균값 사용.
- `BaggingClassifier` 의 `oob_score=True` 로 설정하면 oob 평가를 자동으로 실행 평가점수는 `oob_score_` 속성에 저정됨. 테스트세트에 대한 정확도와 비슷한 결과가 나옴.

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1, oob_score=True, random_state=42)

bag_clf.fit(X_train, y_train)
bag_clf.oob_score_
```

`oob_decision_function_` 속성

각 훈련 샘플의 클래스 확률 또한 계산한다.

```
>>> 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]])
```

7.3 랜덤 패치와 랜덤 서브스페이스

- `BaggingClassifier`는 특성 샘플링 기능 지원: `max_features`와 `bootstrap_features`
- 이미지 등 매우 높은 차원의 데이터셋을 다룰 때 유용
- 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 수만큼을 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 결정트리의 앙상블을 최적화한 모델
- 분류 용도: `RandomForestClassifier`
- 회귀 용도: `RandomForestRegressor`

```
rnd_clf = RandomForestClassifier(n_estimators=500,
                                 max_leaf_nodes=16,
                                 n_jobs=-1,
                                 random_state=42)

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

옵션

- `BaggingClassifier`와 `DecisionTreeClassifier`의 옵션을 거의 모두 가짐.
- 예외
    - `DecisitionClassifier`의 옵션 중: `splitter='random'`, `presort=False`, `max_samples=1.0`
    - `BaggingClassifier`의 옵션 중: `base_estimator=DecisionClassifier(지정된옵션,...)`
- `max_features='auto'`가 `RandomForestClassifier`의 기본값임. 따라서 특성 선택에 무작위성 사용됨. 결정트리에 비해 편향은 크게, 분산은 낮게.
    - 선택되는 특성 수:

$$\sqrt{\text{전체 특성 수}}$$

    - `splitter='random'` 옵션과 함께 사용되어 특성은 무작위적으로 선택되나 임곗값은 최적의 값을 선택함.

엑스트라 트리

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

랜덤포레스트의 마디 분할 방식

- 특성: 무작위 선택
- 특성 임곗값: 선택된 특성별로 무작위로 선택한 1개의 임곗값을 이용하여 분할한 다음 최적값 선택
- 참고: ExtraTreeClassifier 클래스 정의 라이브러리에 따라 1개가 아닌 서너개의 임곗값을 무작위로 선택하는 경우도 있음.

예제

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

특성 중요도

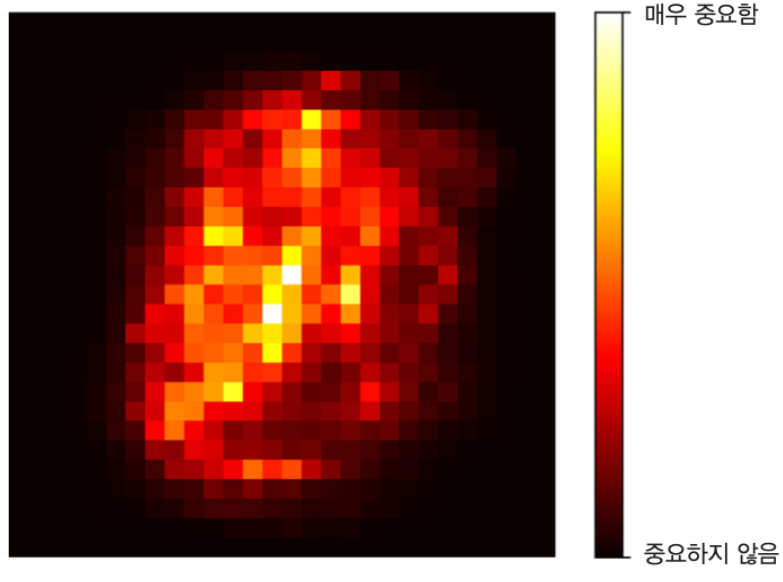
- 특성 중요도: 해당 특성을 사용한 마디가 평균적으로 불순도를 얼마나 감소시키는지를 측정
    - 즉, 불순도를 많이 줄이면 그만큼 중요도가 커짐
- 사이킷런의 `RandomForestClassifier` : 훈련이 끝난 뒤 특성별 중요도의 전체 합이 1이 되도록 하는 방식
    - 특성별 상대적 중요도를 측정한 후 `feature_importances_` 속성에 저장

예제: 붓꽃 데이터셋

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

예제: MNIST

아래 이미지는 각 픽셀의 중요도를 보여준다.

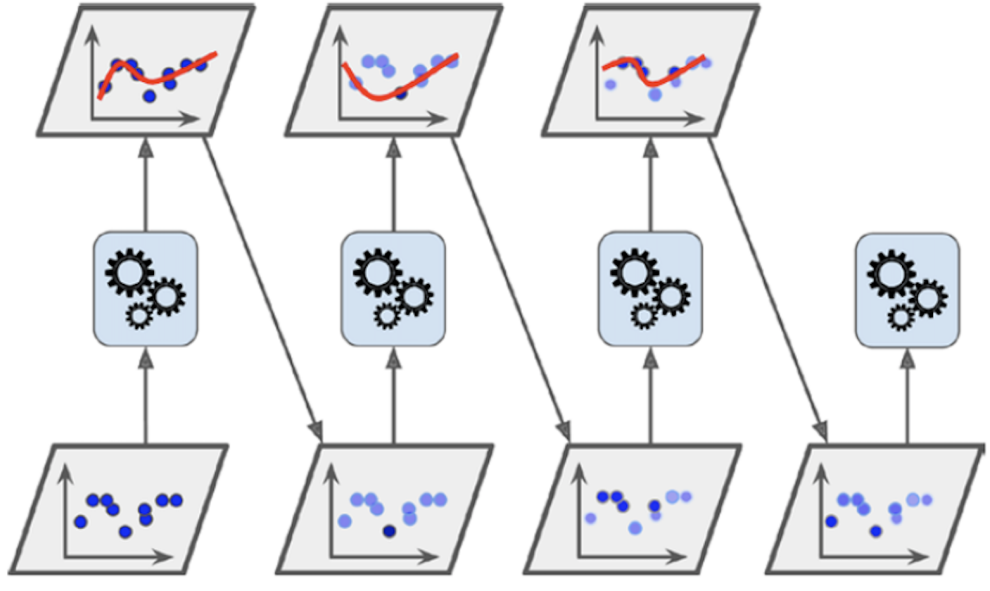


7.5 부스팅

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

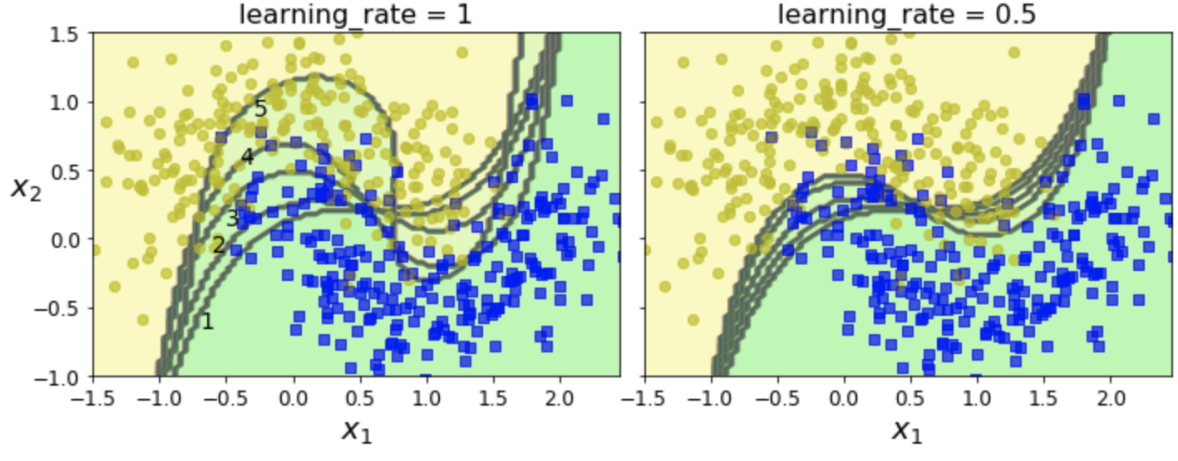
에이다부스트(AdaBoost)

- 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

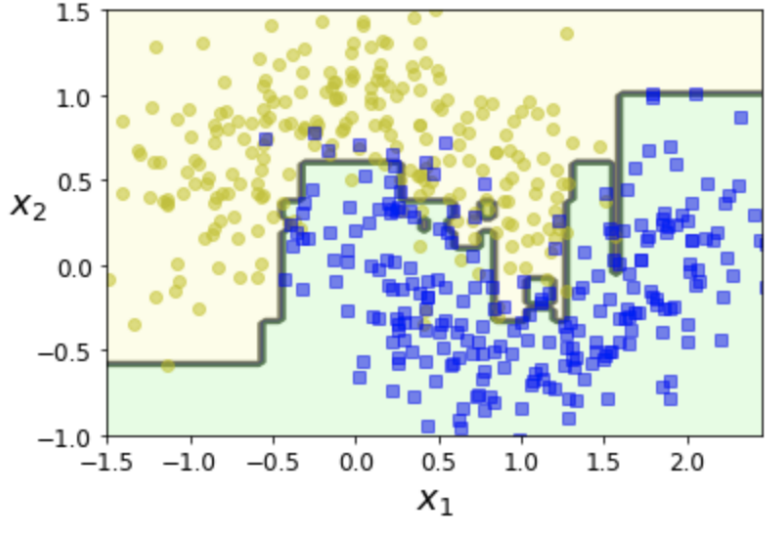
- 분류 모델: `AdaBoostClassifier`
- 회귀 모델: `AdaBoostRegressor`

예제: 에이다 부스트 + 결정트리

- moons 데이터셋에 결정트리 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 에이다부스트와 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

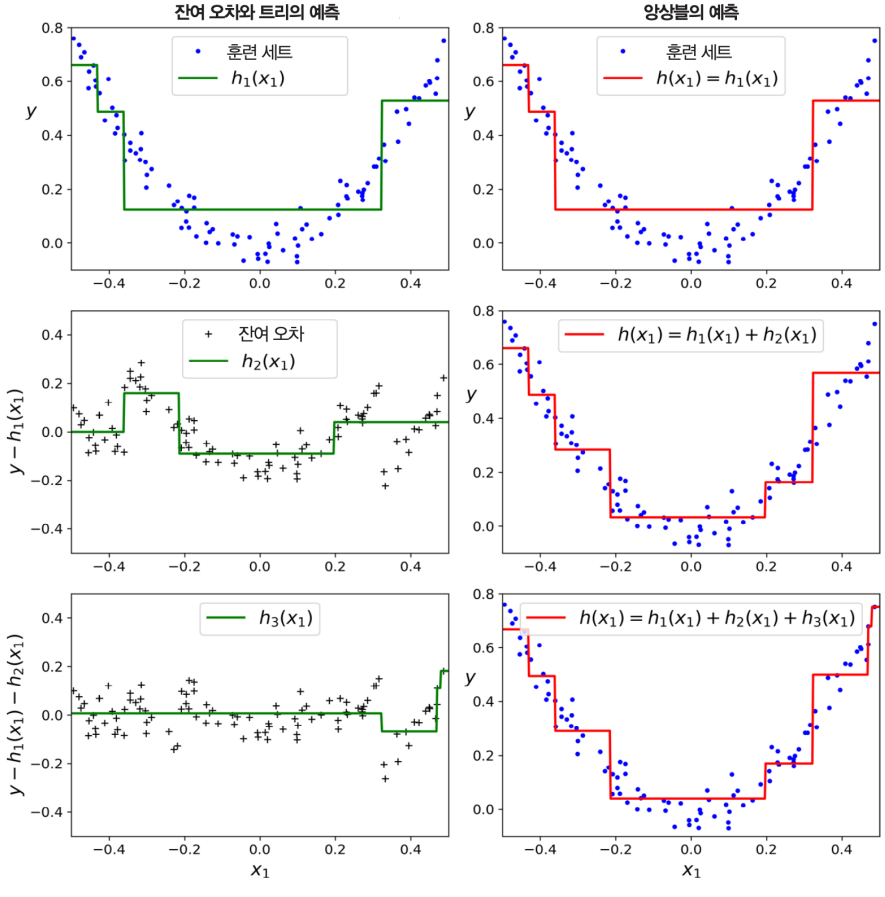
- 분류 모델: `GradientBoostingClassifier`
    - `RandomForestClassifier`와 비슷한 하이퍼파라미터를 제공
- 회귀 모델: `GradientBoostingRegressor`
    - `RandomForestRegressor`와 비슷한 하이퍼파라미터를 제공

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

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

```
gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0)

gbrt.fit(X, y)
```

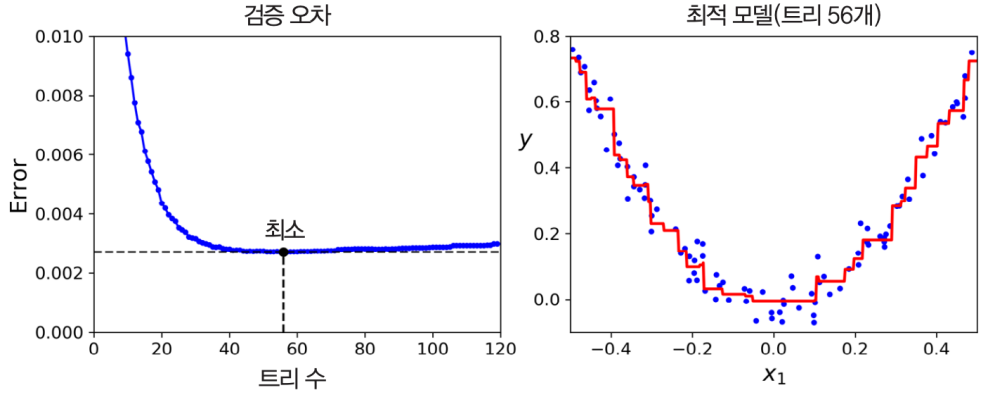

잔여 오차와 트리의 예측
앙상블의 예측
훈련 세트
$h_1(x_1)$
$h(x_1) = h_1(x_1)$
잔여 오차
$h_2(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1)$
$h_3(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1) + h_3(x_1)$
y
$y - h_1(x_1)$
$y - h_1(x_1) - h_2(x_1)$
x_1

`learning_rate`(학습률)

- `learnign_rate`는 기존에 설명한 학습률과 다른 의미의 학습률.
    - 각 결정트리의 기여도 조절에 사용
- 축소 규제: 학습률을 낮게 정하면 많은 수의 결정트리 필요하지만 성능 좋아짐.
- 이전 결정트리에서 학습된 값을 전달할 때 사용되는 비율
    - 1.0이면 그대로 전달
    - 1.0보다 작으면 해당 비율 만큼 조금만 전달

최적의 결정트리 수 확인법

- 조기종료 기법 활용



확률적 그레이디언트 부스팅

- 각 결정트리가 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 스태킹

- 스태킹 아이디어는 간단하지만, 여기서 설명은 생략함.
- 사이킷런 버전 0.22부터 지원됨.
- 참고:
    - Stacked generalization
    - sklearn.ensemble.StackingClassifier
    - 연습문제 9번