앙상블 학습과 랜덤 포레스트

둘러보기

1. 투표 기반 분류기
2. 배깅과 페이스팅
3. 랜덤 패치와 랜덤 서브스페이스
4. 랜덤 포레스트
5. 부스팅
6. 스태킹

0. 기초 용어

앙상블 학습(Ensemble Lerning)

'대중의 지혜' 로 불리는 원리를 기반으로 분류나 회귀 모델과 같은 일련의 예측기를 '앙상블'이라고 부르며, '앙상블 학습' 이라고 한다.

랜덤 포레스트(Random Forest)

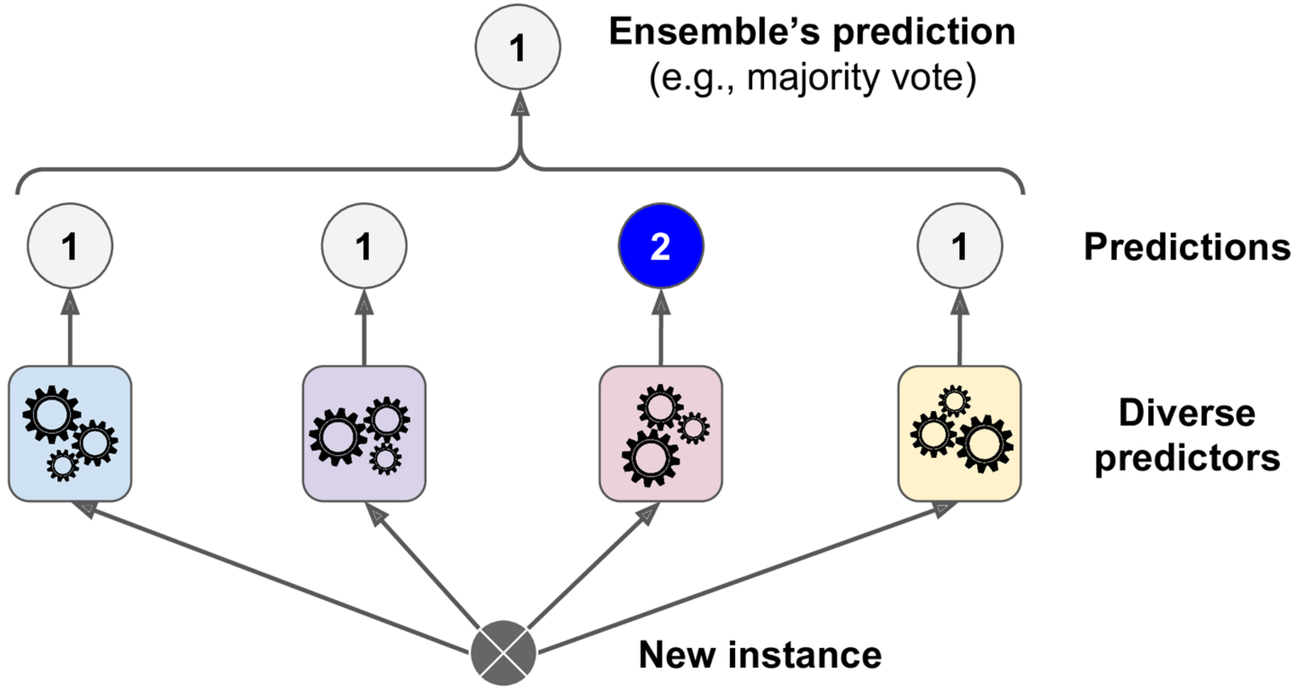
결정 트리의 앙상블을 '랜덤 포레스트'라고 한다. 오늘날 강력한 머신러닝 알고리즘 중 하나이다.

흔히 앙상블 방법을 사용하여 이미 만든 여러 괜찮은 예측기를 연결하여 더 좋은 예측기를 만든다.

1. 투표 기반 분류기

직접 투표(Hard Voting)

각 분류기의 예측을 모아서 다수결에 따라 가장 많이 선택된 클래스를 예측하는 방식으로 정해진 분류기를 말한다. 이 분류기는 앙상블에 포함된 개별 분류기 중 가장 뛰어난 것보다도 정확도가 높을 수 있다. 이것이 가능한 이유는 '**큰 수의 법칙**' 원리 때문이다.



1. 투표 기반 분류기

직접 투표(Hard Voting)

큰 수의 법칙(Law of Large Numbers)

"큰 모집단에서 무작위로 뽑은 표본의 평균이 전체 모집단의 평균과 가까울 가능성이 높다"

$$Pr(K = k) = f(k; n, p) = \binom{n}{k} p^k (1-p)^{n-k}$$

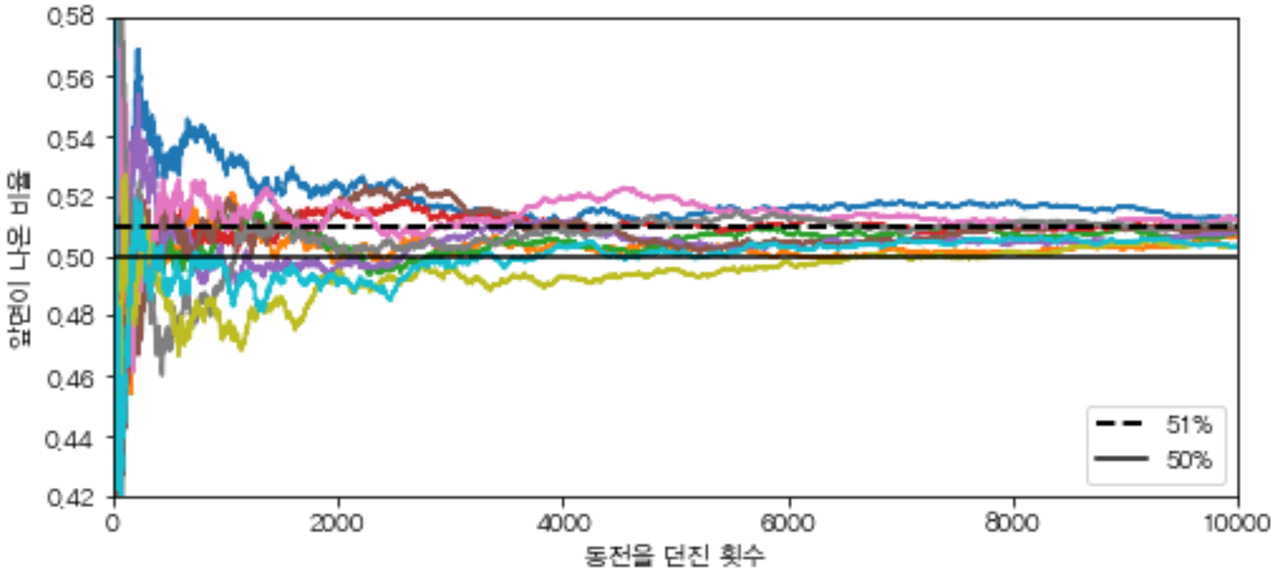
▲ 이항분포의 확률질량함수

1. 투표 기반 분류기

직접 투표(Hard Voting)

큰 수의 법칙(Law of Large Numbers)

"큰 모집단에서 무작위로 뽑은 표본의 평균이 전체 모집단의 평균과 가까울 가능성이 높다"




```
from scipy.stats import binom

print(1 - binom.cdf(49,100,0.51))
print(1 - binom.cdf(499,1000,0.51))
print(1 - binom.cdf(4999,10000,0.51))
```

```
0.6180787124933089
0.7467502275561786
0.9777976478701533
```

1. 투표 기반 분류기

간접 투표(Soft Voting)

모든 분류기가 클래스의 확률을 예측할 수 있으면 개별 분류기의 예측을 평균 내어 확률이 가장 높은 클래스를 예측할 수 있는데, 이를 '간접 투표'라고 한다. 앞서 직접 투표 방식으로 앙상블 모델을 구축하게 되면 '모든 분류기가 완벽하게 독립적이고 오차에 상관관계가 없다'는 가정에 맞지 않으므로 앙상블의 정확도가 낮아진다. 그러므로 '간접 투표'가 더 적합하다.

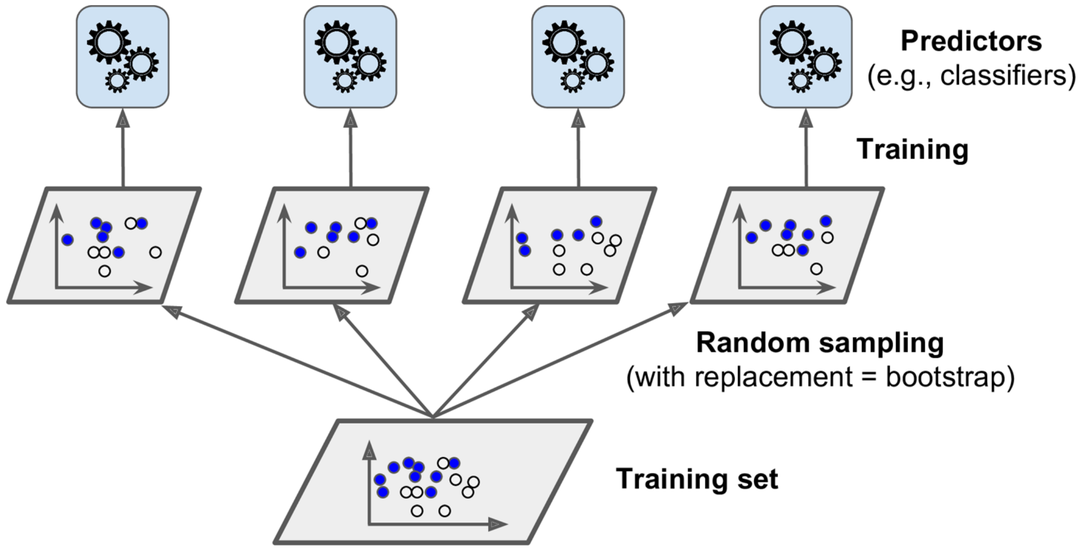
2. 배깅과 페이스팅

배깅(Bagging [Bootstrap Aggregating])

훈련 세트에서 중복을 허용하여 샘플링하는 방식.

페이스팅(Pasting)

훈련 세트에서 중복을 허용하지 않고 샘플링하는 방식.



배깅과 페이스팅에서는 같은 훈련 샘플을 여러 개의 예측기에 걸쳐 사용할 수 있다.

배깅은 한 예측기를 위해 같은 훈련 샘플을 여러 번 샘플링할 수 있다.

2. 배깅과 페이스팅

수집 함수

분류 -〉 통계적 최빈값

회귀 -〉 평균

개별 예측기는 원본 훈련 세트로 훈련시킨 것보다 훨씬 크게 편향되어 있지만, 수집 함수를 통과하면 편향과 분산이 모두 감소한다.

편향

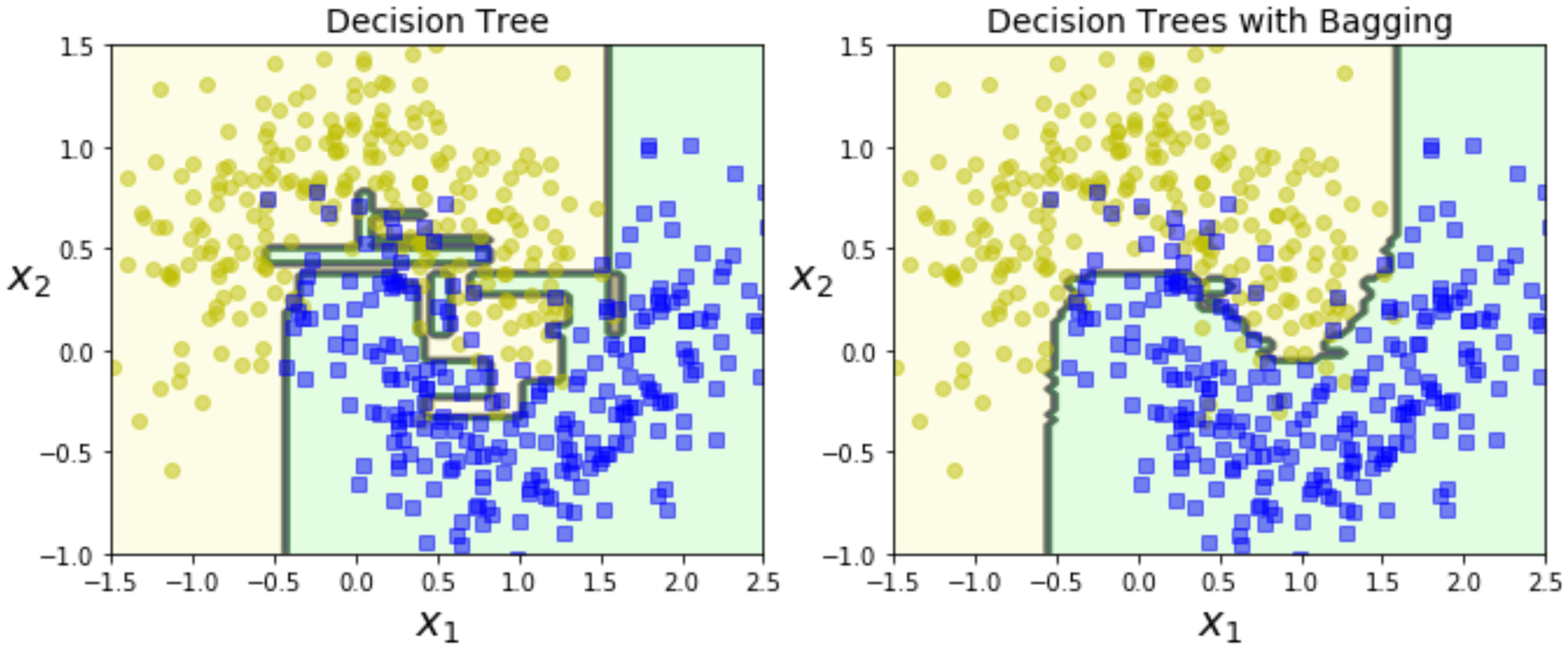
잘못된 가정으로 인한 것으로, 편향이 큰 모델은 훈련데이터에 과소적합되기 쉽다.

분산

훈련데이터에 있는 작은 변동에 모델이 과도하게 민감하기 때문에 발생.

2. 배깅과 페이스팅

사이킷런의 배깅과 페이스팅



▲ 단일 결정 트리(왼쪽)와 배깅 앙상블(오른쪽) 비교. 앙상블의 예측이 Decision Tree 단일 모델의 예측보다 오른쪽 모델이 일반화가 잘 되어 있다고 볼 수 있다.

2. 배깅과 페이스팅

사이킷런의 배깅과 페이스팅

부트스트래핑(중복을 허용한 샘플링)은 배깅이 페이스팅보다 편향이 조금 더 높다. 하지만 예측기 간의 상관관계를 줄이므로 앙상블의 분산을 감소시킨다.

2. 배깅과 페이스팅

oob 평가 (out-of-bag)

배깅을 사용하면 어떤 샘플은 한 예측기를 위해 여러 번 샘플링되고 어떤 것은 전혀 선택되지 않을 수 있다. 여기서 선택되지 않은 샘플의 비율을 oob(out-of-bag)샘플이라고 부른다.

```
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
```

▲ 사이킷런에서 BaggingClassifier를 만들 때 oob_score=True로 지정하면 훈련이 끝난 후 자동으로 oob 평가를 수행한다.

2. 배깅과 페이스팅

oob 샘플에 대한 결정 함수의 값

oob_decision_function_ 변수를 통해 확인할 수 있다. 이 경우 결정 함수는 각 훈련 샘플의 클래스 확률을 반환한다.

▼ 첫 번째 훈련 샘플이 양성 클래스에 속할 확률은 약 64%, 음성 클래스에 속할 확률은 약 36%로 추정하고 있다.

```
bag_clf.oob_decision_function_[:10]
```

```
array([[0.359375  , 0.640625  ],
       [0.32941176, 0.67058824],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.        , 1.        ],
       [0.08994709, 0.91005291],
       [0.35384615, 0.64615385],
       [0.02162162, 0.97837838],
       [0.98823529, 0.01176471],
       [0.97354497, 0.02645503]])
```

3. 랜덤 패치와 랜덤 서브스페이스

특성 샘플링

BaggingClassifier 를 이용하여 특성 샘플링이 가능하다. 샘플이 아닌 특성에 대한 샘플링으로, 각 예측기는 무작위로 선택한 입력 특성의 일부분으로 훈련된다. 특성 샘플링은 더 다양한 예측기를 만들며 편향을 늘리는 대신 분산을 낮춘다.

랜덤 패치 방식 (Random Patches Method)

훈련 특성과 샘플을 모두 샘플링하는 것

랜덤 서브스페이스 방식 (Random Subspaces Method)

훈련 샘플을 모두 사용하고 특성은 샘플링하는 것

4. 랜덤 포레스트

랜덤 포레스트(Random Decision Forests)

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

랜덤 포레스트의 장점

- 편향을 손해보는 대신 분산을 낮추어 전체적으로 더 훌륭한 모델을 만든다.
- 특성의 상대적 중요도를 측정하기 쉽다.
- 특성을 선택해야 할 때 어떤 특성이 중요한지 빠르게 확인할 수 있다.

4. 랜덤 포레스트

랜덤 포레스트(Random Decision Forests)

```
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)
```

▲ Bagging Classifier에 DecisionTreeClassifier를 넣어 만드는 대신 결정 트리에 최적화되어 사용하기 편리한 RandomForestClassifier를 사용할 수 있다.

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(max_features='auto', max_leaf_nodes=16),
    n_estimators=500, max_samples=1.0, bootstrap=True, n_jobs=-1)
```

▲ BaggingClassifier를 사용하여 앞의 RandomForestClassifier와 유사하게 만든 코드.

4. 랜덤 포레스트

엑스트라 트리(extra-trees, Extremely Randomized)

극단적으로 무작위한 트리의 랜덤 포레스트. 랜덤 포레스트는 트리를 더욱 무작위하게 만들기 위해, 일반적인 결정트리모델처럼 최적의 임계값을 찾는 대신 후보 변수를 사용해 무작위로 분할해 그 중 최상의 분할을 선택하는 원리인데, 이를 더 극단적으로 사용했다고 볼 수 있다.

엑스트라 트리의 장점

- 일반적인 랜덤 포레스트보다 작업 속도가 훨씬 빠르다.

4. 랜덤 포레스트

특성 중요도

사이킷런은 어떤 특성을 사용한 노드가 평균적으로 불순도를 얼마나 감소시키는지 확인하여 특성의 중요도를 측정한다.

특성 중요도 구하기

- 중요도 = (현재 노드의 샘플 비율 X 불순도) - (왼쪽 자식 노드의 샘플 비율 X 불순도) - (오른쪽 자식노드의 샘플 비율 X 불순도) 를 계산하여 더한다. (샘플 비율 : 트리 전체 샘플 수에 대한 해당 노드의 샘플 수의 비율)
- 특성 중요도의 합이 1이 되도록 전체 합으로 나누어 정규화
- 랜덤 포레스트의 특성 중요도 = (중요도 전체합) / (트리 수)

4. 랜덤 포레스트

특성 중요도

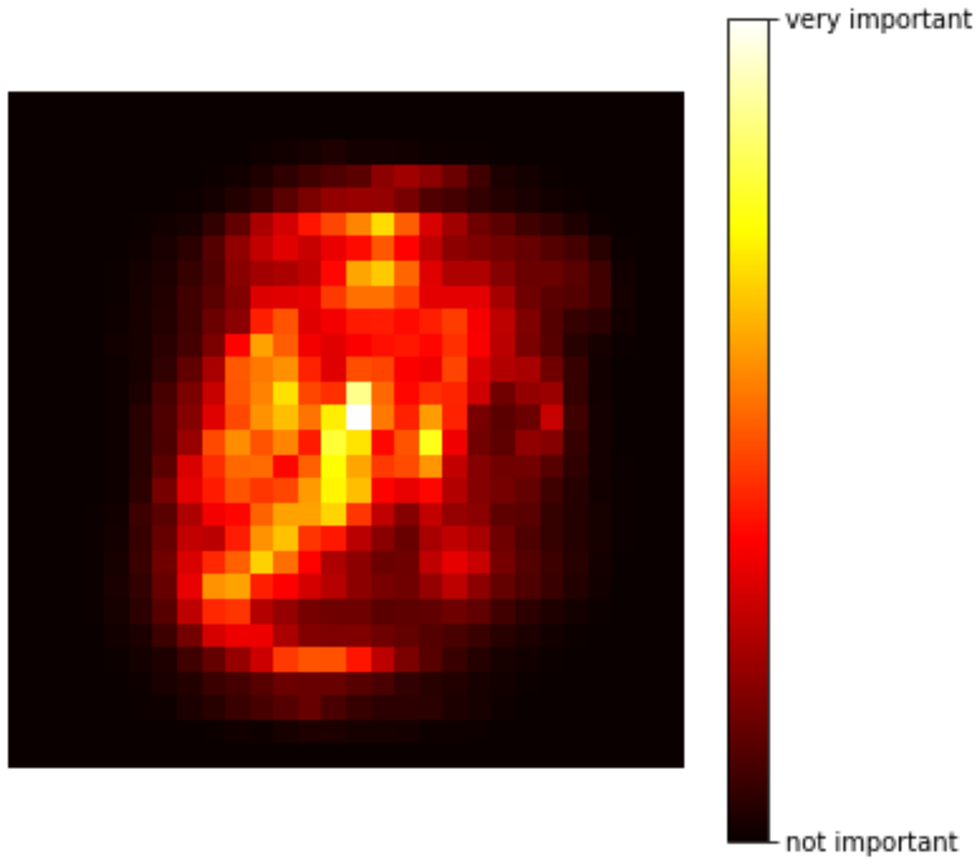
```
from sklearn.datasets import load_iris
iris = load_iris()
rnd_clf = RandomForestClassifier(n_estimators=500,n_jobs=-1)
rnd_clf.fit(iris["data"], iris["target"])
for name, score in zip(iris["feature_names"],
rnd_clf.feature_importances_):
    print(name, score)
```

```
sepal length (cm) 0.09193700831457226
sepal width (cm) 0.02303133424063586
petal length (cm) 0.4503310122507622
petal width (cm) 0.43470064519402973
```

▲ iris 데이터셋에 RandomForestClassifier를 훈련시키고 각 특성의 중요도를 출력한 모습.
꽃잎의 길이(petal length) : 45%, 꽃잎의 너비(petal width) : 43%로 가장 높다.

4. 랜덤 포레스트

특성 중요도



▲ MNIST 데이터셋에 랜덤 포레스트 분류기를 훈련시키고 앞서 활동한 것과 같이 각 픽셀의 중요도를 구해 그래프로 나타낸 모습.

5. 부스팅

부스팅 (Boosting)

약한 학습기를 여러 개 연결하여 강한 학습기를 만드는 앙상블 방법. 앞의 모델을 보완해나가면서 일련의 예측기를 학습시키는 원리이다.

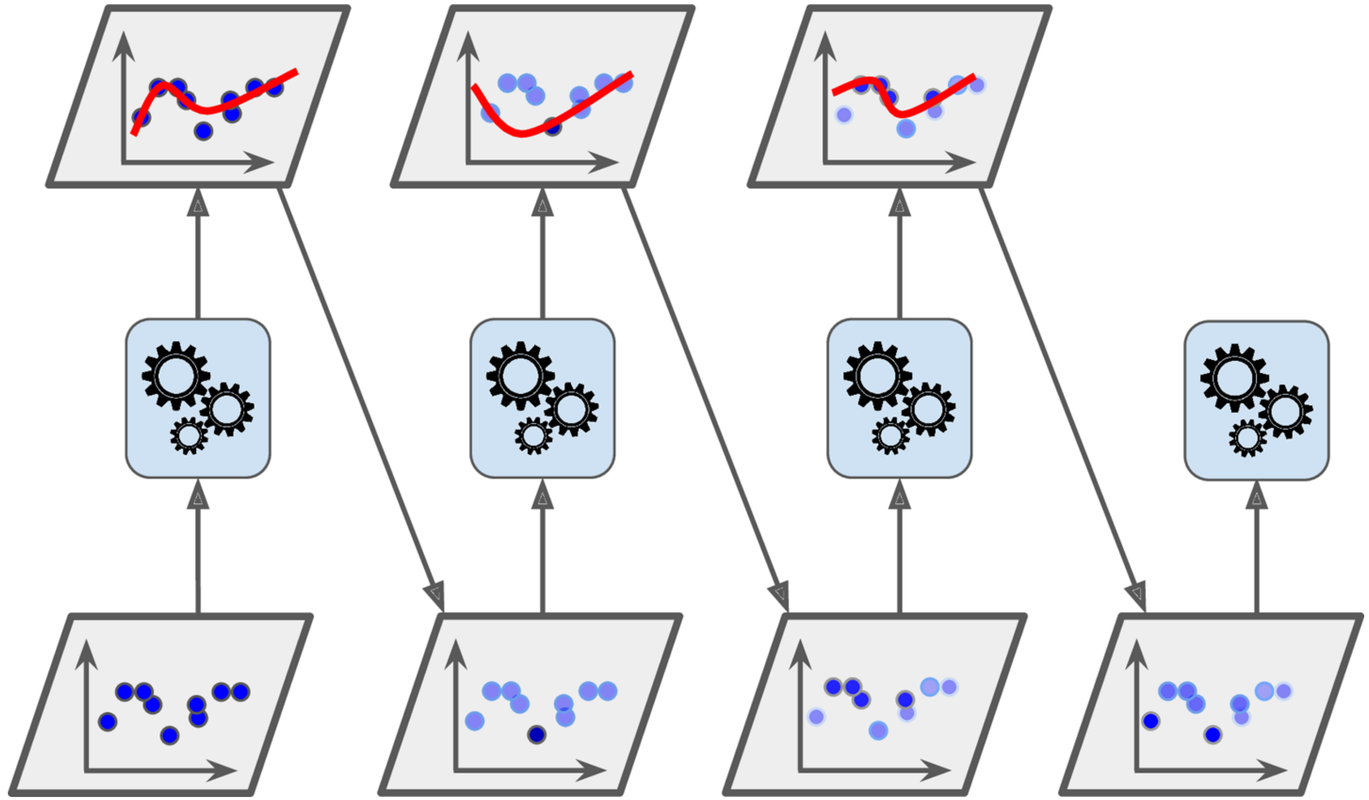
부스팅 방법 (인기있는 두 가지)

- 에이다부스트 (AdaBoost, Adaptive Boosting)
- 그레이디언트 부스팅 (Gradient Boosting)

5. 부스팅

에이다부스트(AdaBoost)

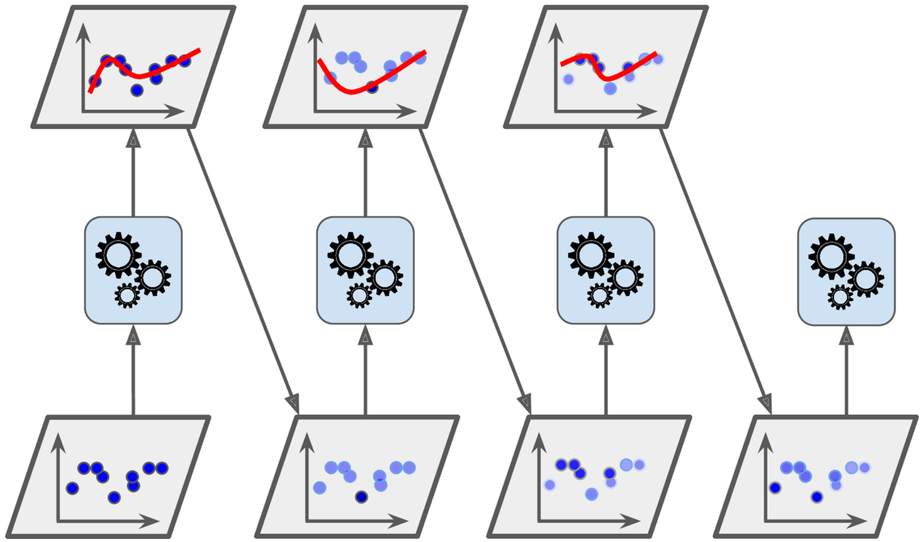
이전 모델이 과소적합했던 훈련 샘플의 가중치를 더 높이는 방법으로 예측기를 보완한다.



5. 부스팅

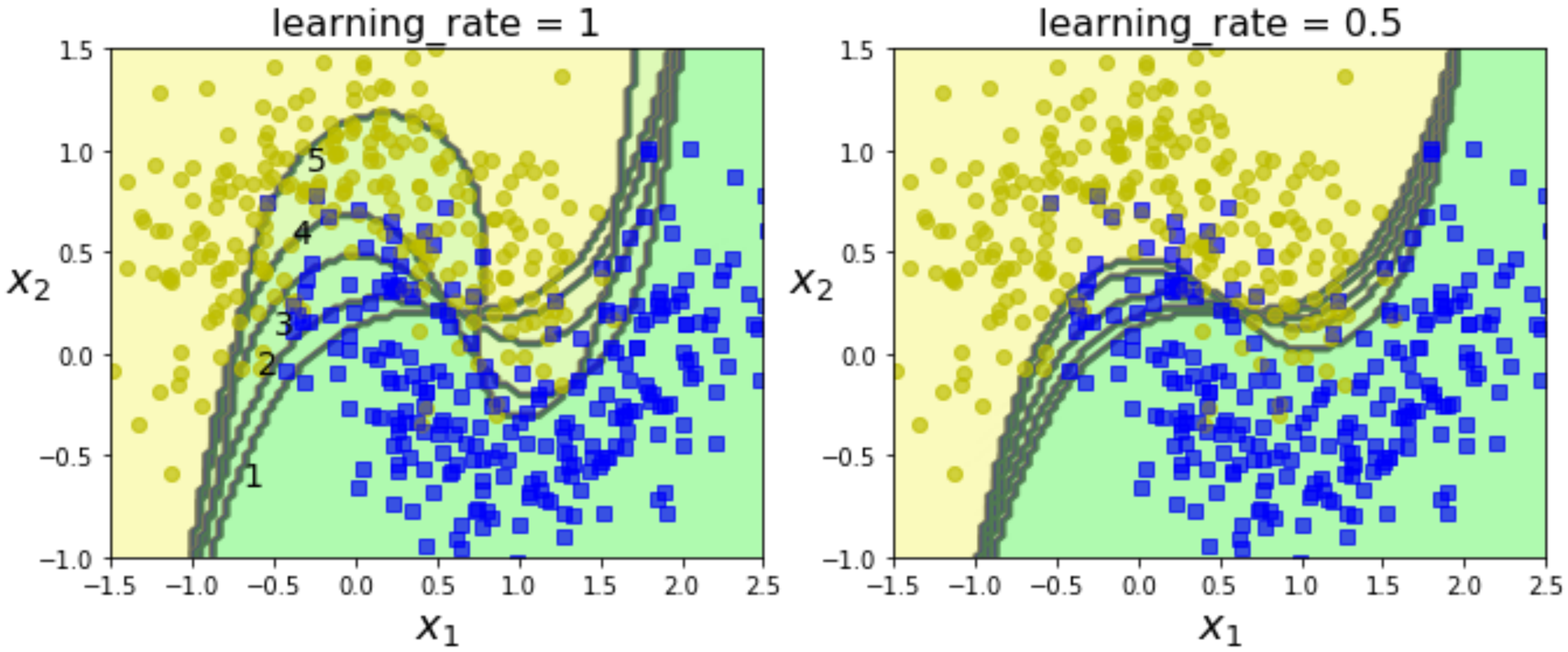
에이다부스트 과정

1. 기반이 되는 첫 번째 분류기를 훈련 데이터셋으로 훈련시키고 예측값을 만든다.
2. 잘못 분류된 훈련 샘플의 가중치를 상대적으로 높인다.
3. 업데이트된 가중치를 사용해 두번째 분류기를 훈련세트로 훈련하고 다시 예측값을 만든다.
4. 2~3과정 반복



5. 부스팅

에이다부스트



▲ moons 데이터셋에 훈련시킨 다섯 개의 연속된 예측기의 결정 경계.

5. 부스팅

에이다부스트 특징

- 모든 예측기가 훈련을 마치면 배깅이나 페이스팅과 비슷한 방식으로 예측을 만든다. 이 때 가중치가 적용된 훈련 세트의 전반적인 정확도에 따라 예측기마다 다른 가중치가 적용된다.
- 병렬화(또는 분할)할 수 없어 확장성이 높지 않다.

5. 부스팅

에이다부스트 과정에 대한 분석

1

$$r_j = \frac{\sum_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}} \quad \text{where } \hat{y}_j^{(i)} \text{ is the } j^{\text{th}} \text{ predictor's prediction for the } i^{\text{th}} \text{ instance.}$$

2

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

3

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

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

4

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

5. 부스팅

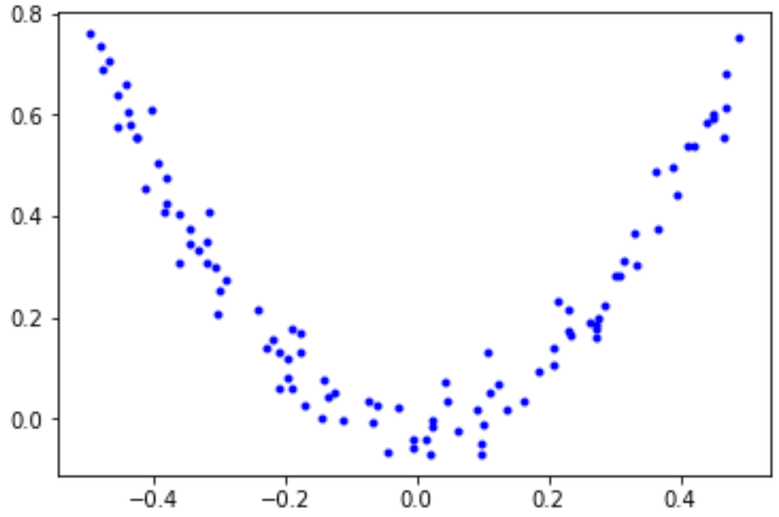
그레이디언트 부스팅(Gradient Boosting)

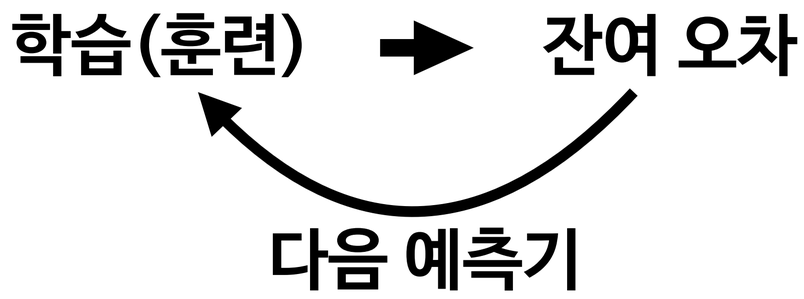
에이다부스트처럼 반복마다 샘플의 가중치를 수정하는 대신 이전 예측기가 만든 잔여 오차에 새로운 예측기를 학습시키는 방식.

5. 부스팅

그레이디언트 부스팅(Gradient Boosting)

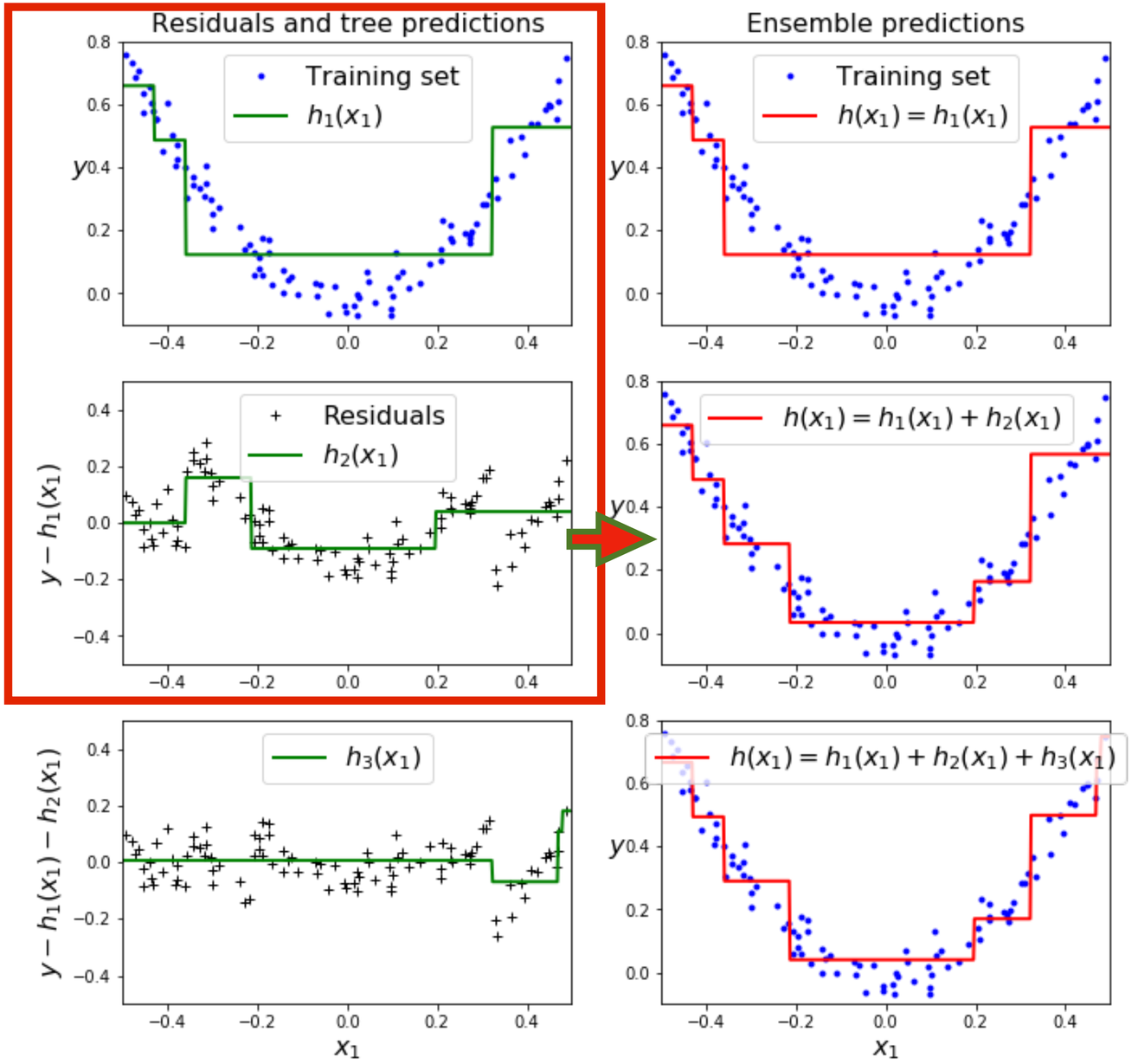
결정 트리를 기반 예측기로 사용하는 간단한 회귀 문제를 풀어 이해해보자.





▲ 그래디언트 부스티드 회귀트리(Gradient Boosted Regression Tree, GBRT)

그레이디언트 부스팅(Gradient Boosting)



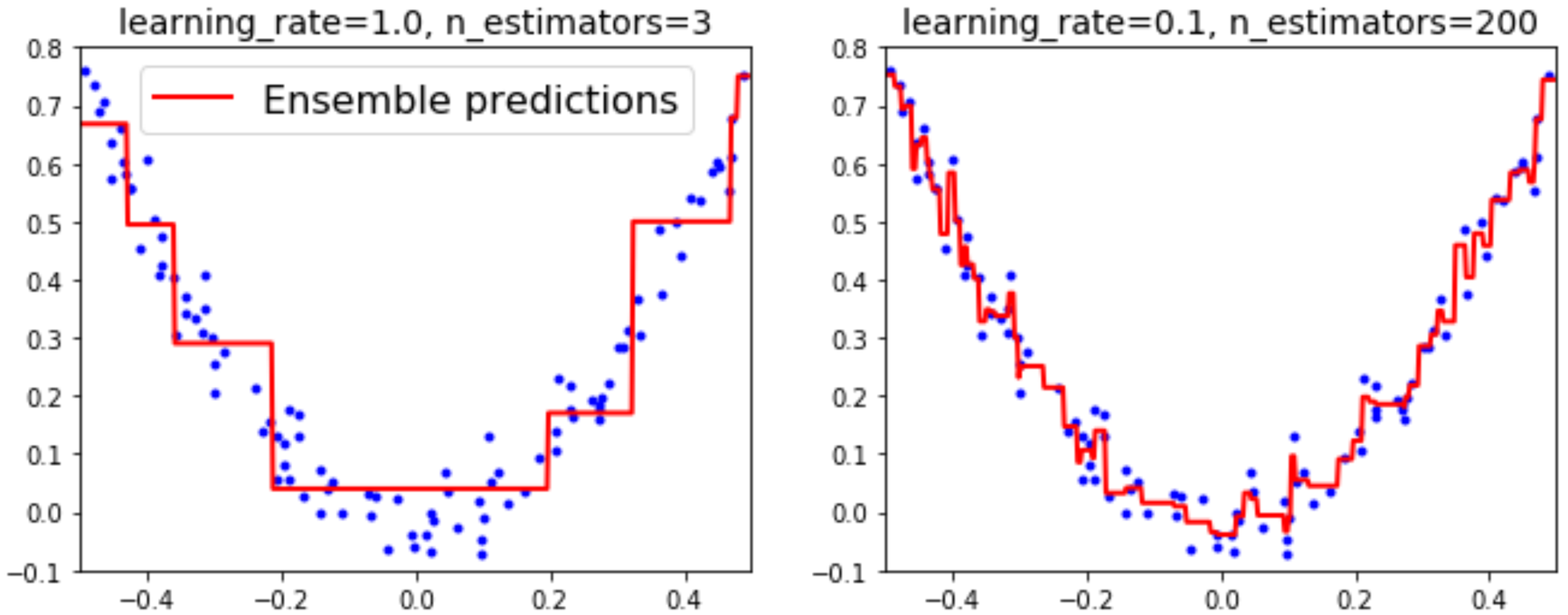
5. 부스팅

축소(Shrinkage)

learning_rate 매개변수가 각 트리의 기여 정도를 조절하는데, 이를 낮게 설정하면 앙상블을 훈련 세트에 학습시키기 위해 많은 트리가 필요하지만 일반적으로 예측의 성능은 좋아지게 된다.

5. 부스팅

그레이디언트 부스팅(Gradient Boosting)



▲ 예측기가 부족한 경우(왼쪽)와 너무 많은 경우(오른쪽) GBRT 앙상블

5. 부스팅

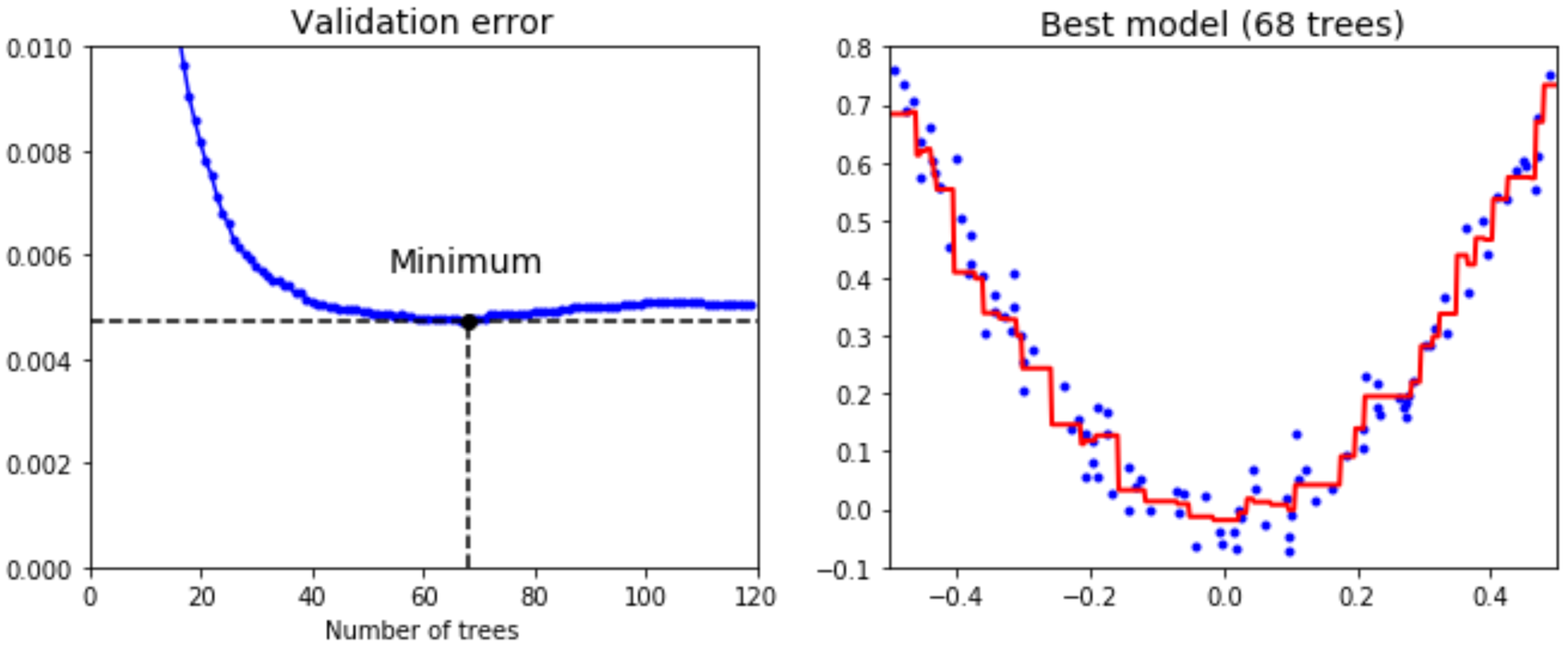
그레이디언트 부스팅(Gradient Boosting)

▼ GBRT 앙상블을 훈련시키고 최적의 트리 수를 찾기 위해 각 훈련 단계에서 검증 오차를 측정한다. (일반적인 방법)

```
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)

gbrt = GradientBoostingRegressor(max_depth=2,
                                 n_estimators= 120)
gbrt.fit(X_train, y_train)

errors = [mean_squared_error(y_val, y_pred)
          for y_pred in gbrt.staged_predict(X_val)]
bst_n_estimators = np.argmin(errors) + 1
# np.argim(array) : array에서 최솟값의 인덱스를 반환
# 즉, 최소값을 갖게하는 인덱스로 최적 트리의 수를 판단

gbrt_best = GradientBoostingRegressor(max_depth=2,
                                      n_estimators=bst_n_estimators,
                                      random_state=42)
gbrt_best.fit(X_train,y_train)
```

5. 부스팅

그레이디언트 부스팅(Gradient Boosting)



▲ 검증 오차(왼쪽), 최적 모델의 예측(오른쪽)

5. 부스팅

그레이디언트 부스팅(Gradient Boosting)

▼ GBRT 앙상블을 훈련시키고 최적의 트리 수를 찾기 위해 각 훈련 단계에서 검증 오차를 측정한다. (검증 오차가 향상되지 않으면 조기종료)

```
X_train, X_val, y_train, y_val = train_test_split(X,y)

gbrt = GradientBoostingRegressor(max_depth = 2,
                                 warm_start=True,   # 기존 트리를 유지
                                 random_state=42)

min_val_error = float("inf")
error_going_up = 0
for n_estimators in range(1,120):
    gbrt.n_estimators = n_estimators   # 트리 수 입력
    gbrt.fit(X_train, y_train)         # 훈련
    y_pred = gbrt.predict(X_val)       # 예측값 생성
    val_error = mean_squared_error(y_val, y_pred)  # MSE 계산
    if val_error < min_val_error:      # 초기 min_val_error는 무한대(inf)
        min_val_error = val_error
        error_going_up = 0
    else:
        error_going_up += 1            # 누적
        if error_going_up == 5:        # 연속해서 5번이나 MSE가 기존보다 크면
            break
```

5. 부스팅

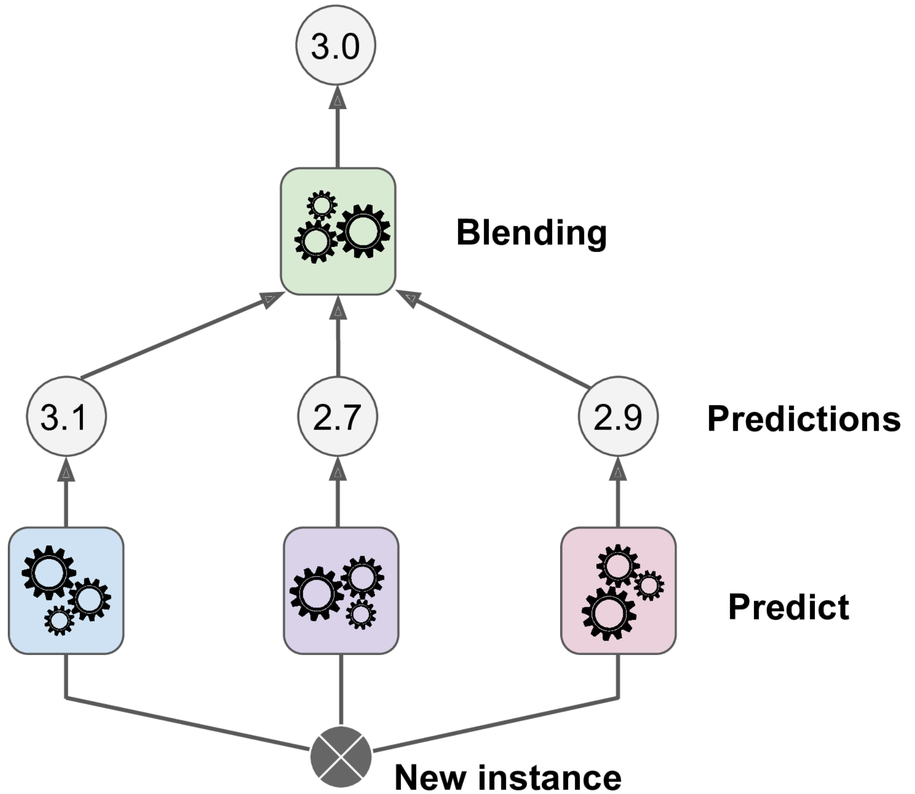
확률적 그레이디언트 부스팅(Stochastic Gradient Boosting)

훈련 샘플의 비율을 조정하여 편향이 높아지는 대신 분산이 낮아지고 훈련 속도를 높이는 부스팅 기법.

6. 스태킹

스태킹 (Stacking, stacked generalization)

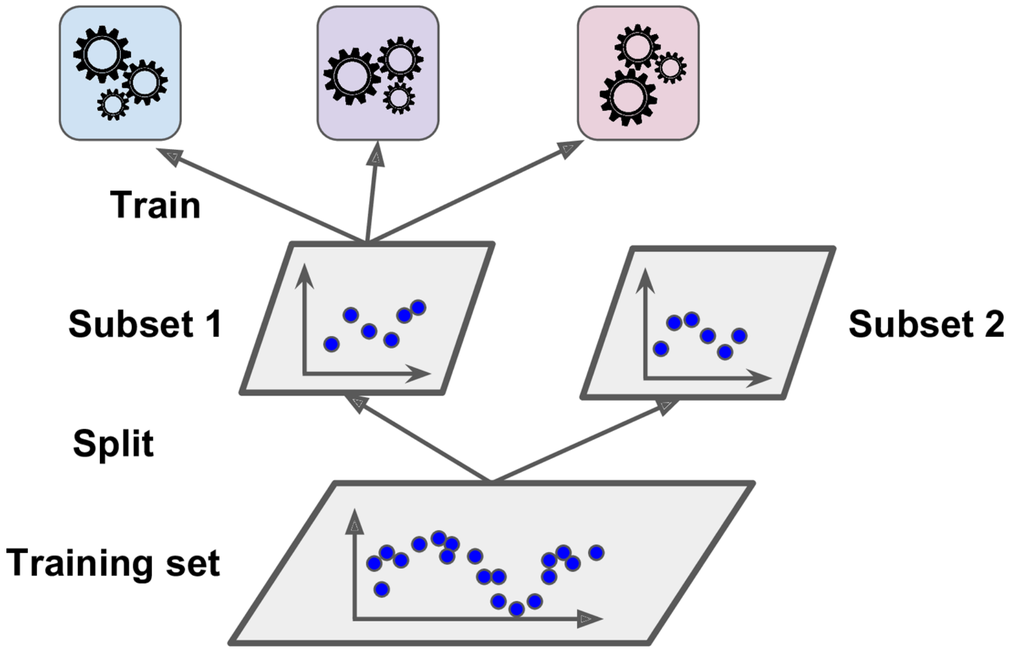
앙상블에 속하는 모든 예측기의 예측을 취합하는 방식이 아닌 취합하는 모델 자체를 훈련하는 방식. 여러 예측기가 각각 다른 값을 예측하고, 마지막 예측기(블렌더)가 최종 예측을 만든다.



6. 스태킹

블렌더를 학습시키는 일반적인 방법 : 홀드 아웃(hold-out)

1. 먼저 Training set을 2개의 Subset으로 나눈다.
2. Subset1은 첫번째 레이어의 예측기들을 훈련시키는데 사용한다.
3. 훈련된 첫번째 레이어의 예측기로 Subset2에 대한 예측을 생성한다. (Subset이지만 훈련에 사용되지 않았으므로 Test셋처럼 사용 가능)



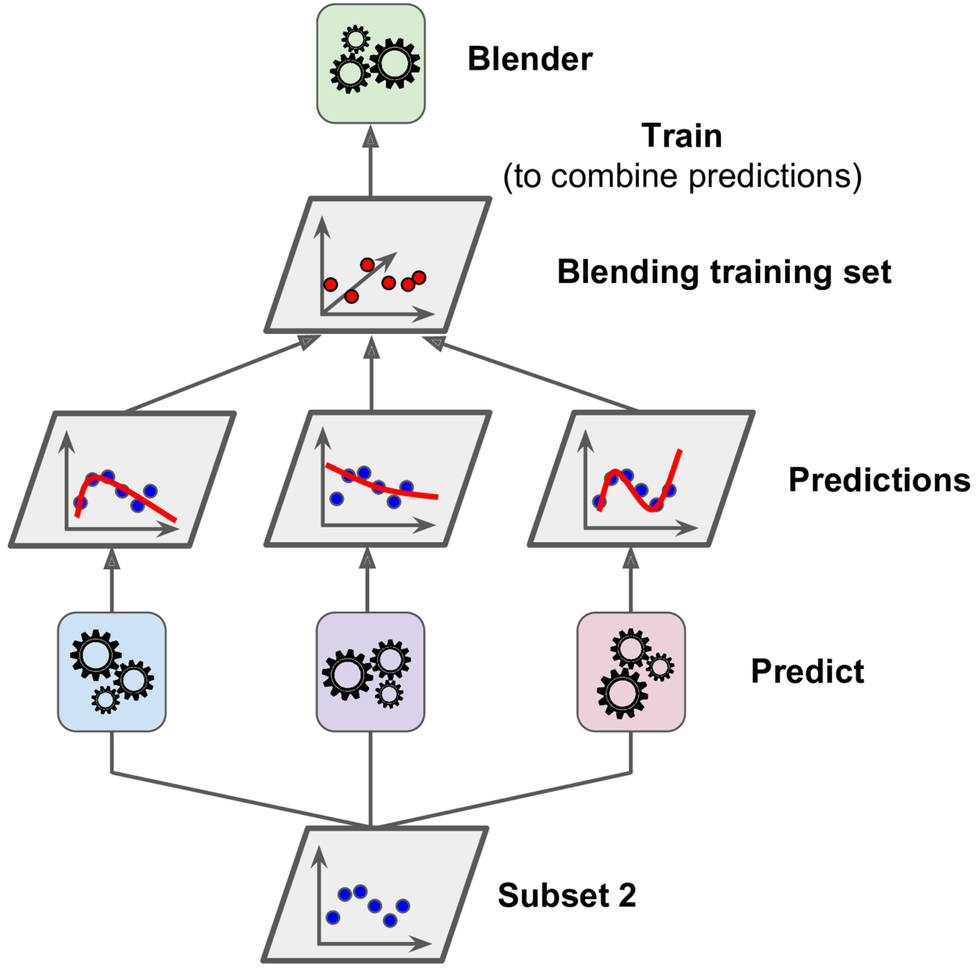
6. 스태킹

블렌더를 훈련하는 방법 : 선형 회귀 및 랜덤 포레스트 회귀

1. 세개의 예측값을 생성한다.
2. 타깃값(y)은 그대로 쓰고 앞에서 예측한 3개의 값(y_hat)을 입력 변수로 사용하는 새로운 훈련세트를 생성한다. (즉, 새로운 훈련세트는 3차원이 된다.)
3. 블렌더가 새로운 훈련 세트로 학습한다. (즉, 첫번째 레이어의 예측 3개를 이용해 y를 예측하도록 학습되는 것이다.)
4. 블렌더만의 레이어가 만들어진다. -〉 훈련 세트를 세 개의 서브셋으로 나눈다.

6. 스태킹

블렌더를 훈련하는 방법 : 선형 회귀 및 랜덤 포레스트 회귀



6. 스태킹

훈련 세트가 세 개의 서브셋으로 나누어진 이후

1. subset1은 첫번째 레이어의 예측기들을 훈련.
2. subset2로 첫번째 예측기들의 예측을 만들어 블렌더 레이어의 예측기를 위한 훈련세트 생성.
3. subset3로 두번째 예측기들의 예측을 만들어 세번째 레이어를 훈련시키기 위한 훈련세트를 만드는데 사용.

6. 스태킹

훈련 세트가 세 개의 서브셋으로 나누어진 이후