[6주차]
앙상블 학습과 랜덤포레스트

1기 오수진
1기 홍문경

EURON

목차

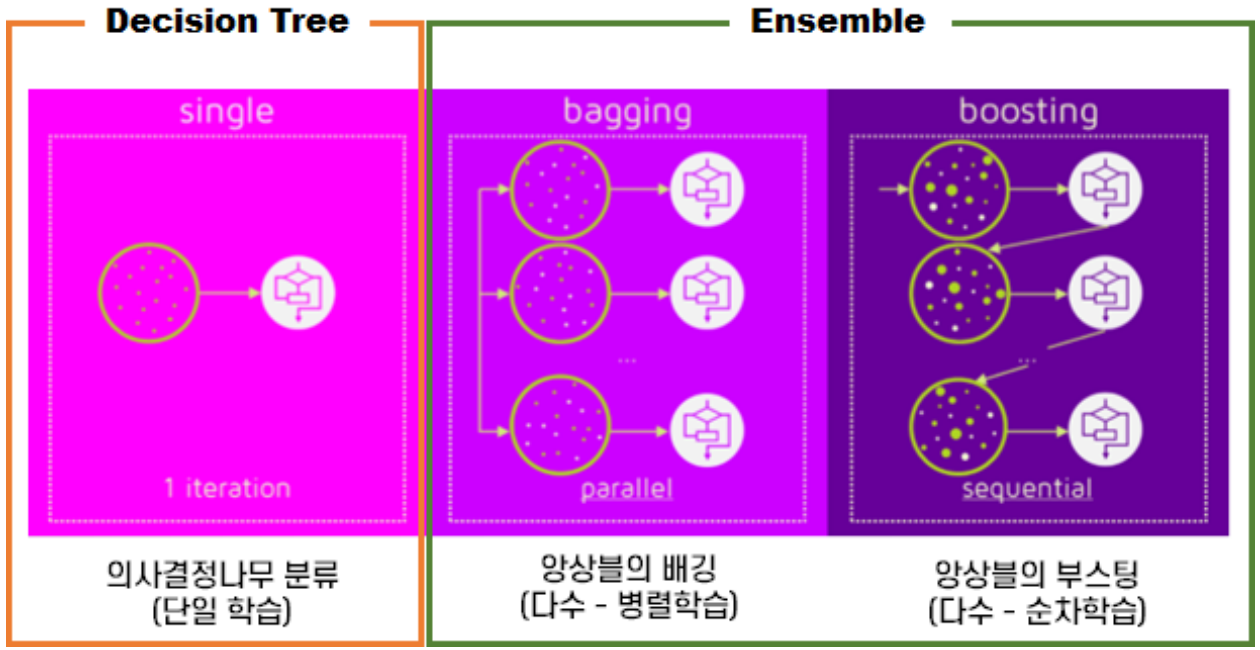
1. 앙상블 개념

여러 가지 우수한 학습 모델을 조합해 예측력을 향상시키는 모델

Assume :

〈성능 비교〉

→ *정확도가 높은 하나의 모델* << **정확도가 낮은 여러 개의 모델의 조합**



단,
1. 모델 결과의 해석이 어려움
2. 긴 예측 시간 소모

출처 : https://joyfuls.tistory.com/61

1. 앙상블 개념

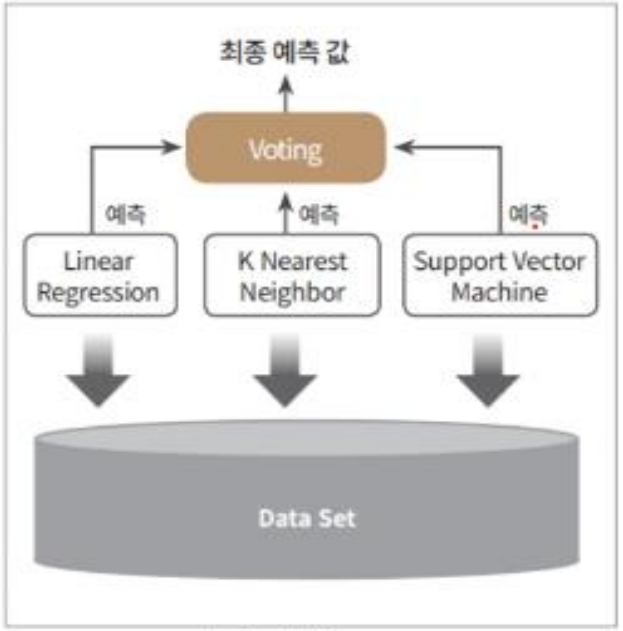
분류	Bagging	Boosting
공통점	전체 data set으로부터 복원 랜덤 샘플링으로 train set 생성	
차이점	병렬 학습	순차 학습
장점	과대 적합에 유리	높은 정확도
단점	특정 영역에서 낮은 정확도	Outlier에 취약
특징	균일한 확률 분포에 의해 train set 생성	분류하기 어려운 train set 생성

→ Random Forest

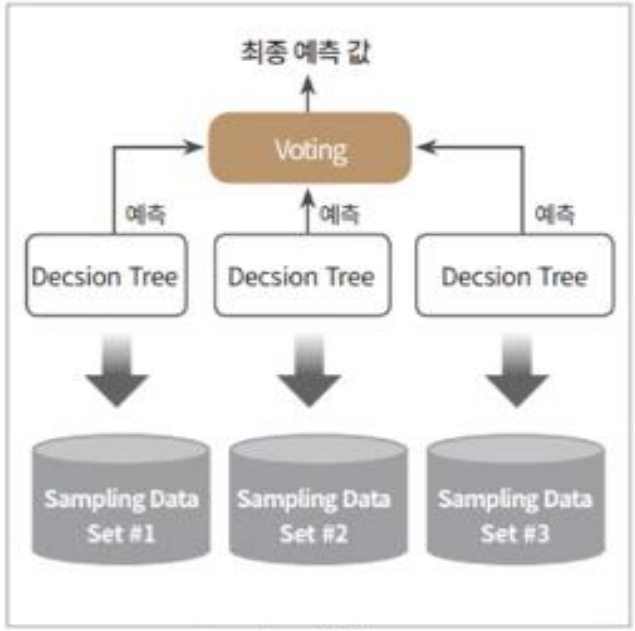
→ AdaBoost, Gradient Boost

출처 : https://joyfuls.tistory.com/61

2. 보팅이란



Voting 방식



Bagging 방식

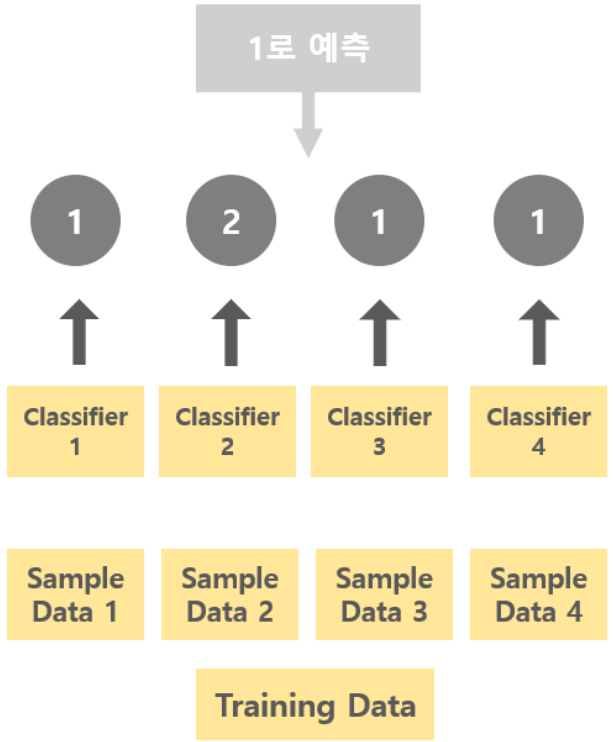
분류	Voting	Bagging
공통점	여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정	
차이점	서로 다른 알고리즘을 가진 분류기 결합	각각의 분류기가 모두 같은 유형이지만 데이터 샘플링 다르게
특징	결과가 편향되지 않게 하는 모델 선정 중요	Bias 유지하며 과적합 막고 variance 줄임, 노이즈에 강함

2. Hard voting & Soft voting

출처 : https://jaaamj.tistory.com/33

<하드 보팅>

다수결로 인해서 최종 예측은 레이블 값 1로 결정

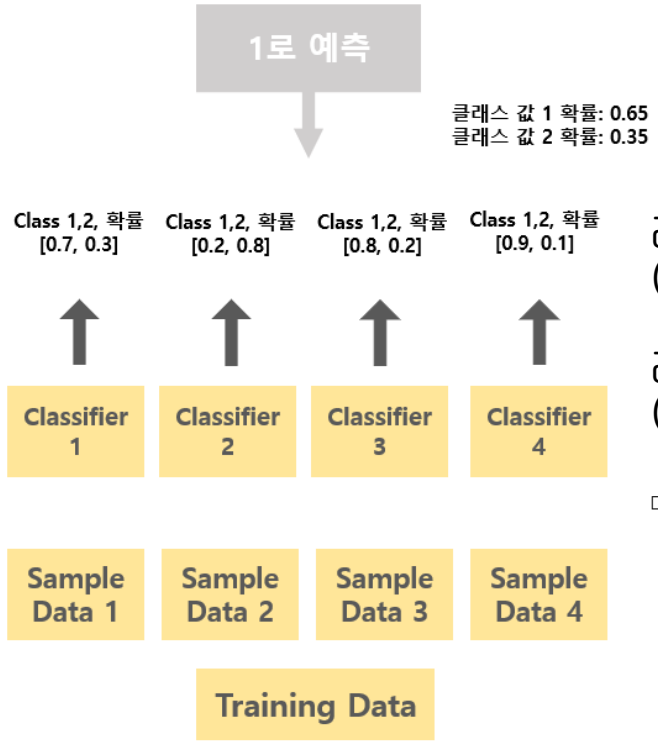


예측한 결과값들 중 다수의 분류기가 결정한 예측값을 최종 보팅 결과값으로 선정

⇒ 다수결 원칙

<소프트 보팅>

Classifier 들의 class 확률을 평균을 취해서 결정



레이블 값 1의 평균 예측 확률
(0.7+0.2+0.8+0.9)/4=0.65

레이블 값 2의 평균 예측 확률
(0.3+0.8+0.2+0.1)/4=0.35

⇨ 레이블 값 1 선정

분류기들의 레이블 값 결정 확률들을 모두 더한 후 평균을 취해 확률이 가장 높은 레이블 값을 최종 보팅 결과값으로 선정

2. VotingClassifier

```
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)
```

moons로 데이터 생성

```
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="lbfgs", random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=100, random_state=42)
svm_clf = SVC(gamma="scale", probability=True, random_state=42)

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

로지스틱회귀, 랜덤포레스트, SVM을 기반으로 보팅 분류기 생성

** voting= ‘hard’ : default

```
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 0.896
SVC 0.896
VotingClassifier 0.92
```

보팅 분류기의 정확도가 가장 높음

3. Bagging

< Bagging 알고리즘 >

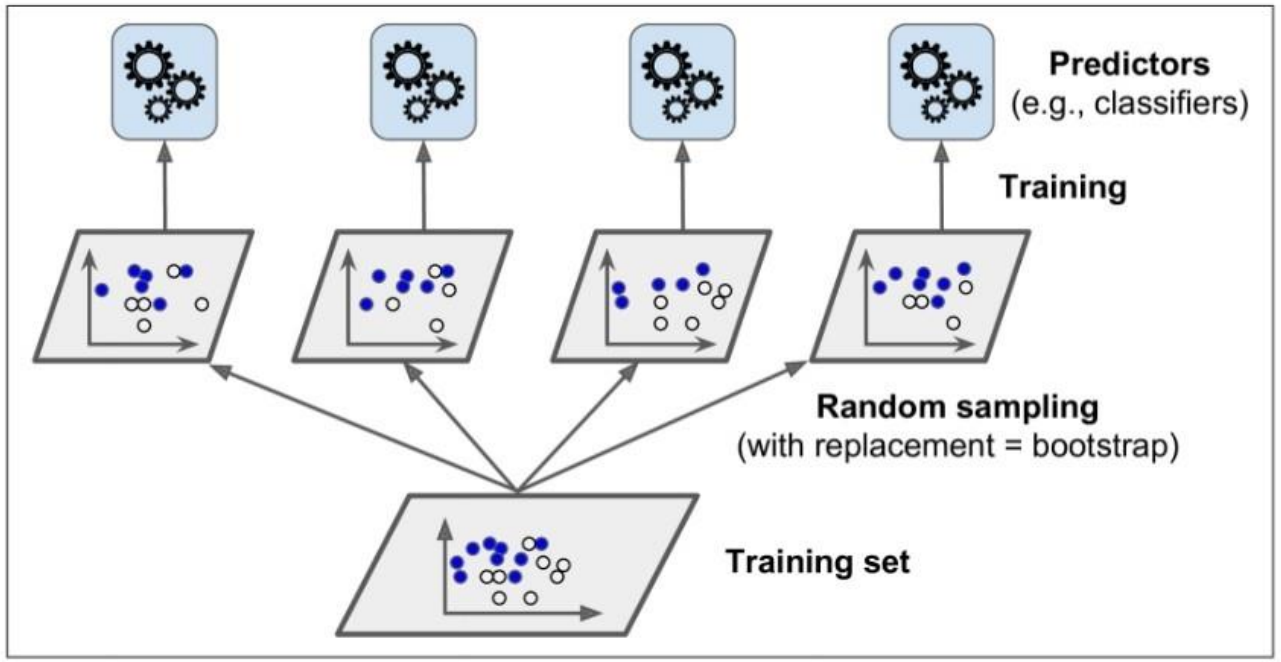
목적 : Variance ↓

Dataset에 **무작위성** 부여 → 더 **견고**하고 **안정성** 있는 모델 (성능 향상)

1. Bootstrap (복원 랜덤 샘플링) : 한 가지 **분류 모델을 여러 개** 만들어서 서로 다른 train data로 학습
2. Aggregating : 동일한 test data에 대해서 **여러 분류 모델이 예측한 값**들을 조합

→ **투표**를 통해 적절한 예측값으로 최종 결론 도출

∴ **과대 적합**이 되기 쉬운 모델을 예측할 때 적합



출처 : https://deepflowest.tistory.com/164

3. Bagging

< Bootstrap > : Random sampling with replacement에 기반한 통계 검정(혹은 추정)

>> 일반적으로 관측된 random sample에서 resampling을 수행하는 case resampling 기법
→ 통계량(추정량)의 분포를 구함

original sample X = {x1, ..., xn} → bootstrap sample Y = {y1, ..., ym}

X의 각 관측치가 선택 될 확률이 같도록
X에서 대체하여 선택한 샘플

→ **장점 : x의 분포에 대한 가정 X**

∴ **주로 모집단의 분포를 구할 수 없는 경우에 분포를 추정하기 위해 사용**

** **Bagging에서의 적용** : 추정 혹은 예측 자체의 성능을 개선시키기 위해 bootstrap 사용
즉, 주어진 train data로부터 bootstrap sample을 resampling한 후, 각 bootstrap sample 내에서 fit된 모형들의 예측 값들의 평균 도출 (classification : 다수결 원칙)

출처 : https://lee-jaejoon.github.io/stat-bootstrap/

3. Bagging

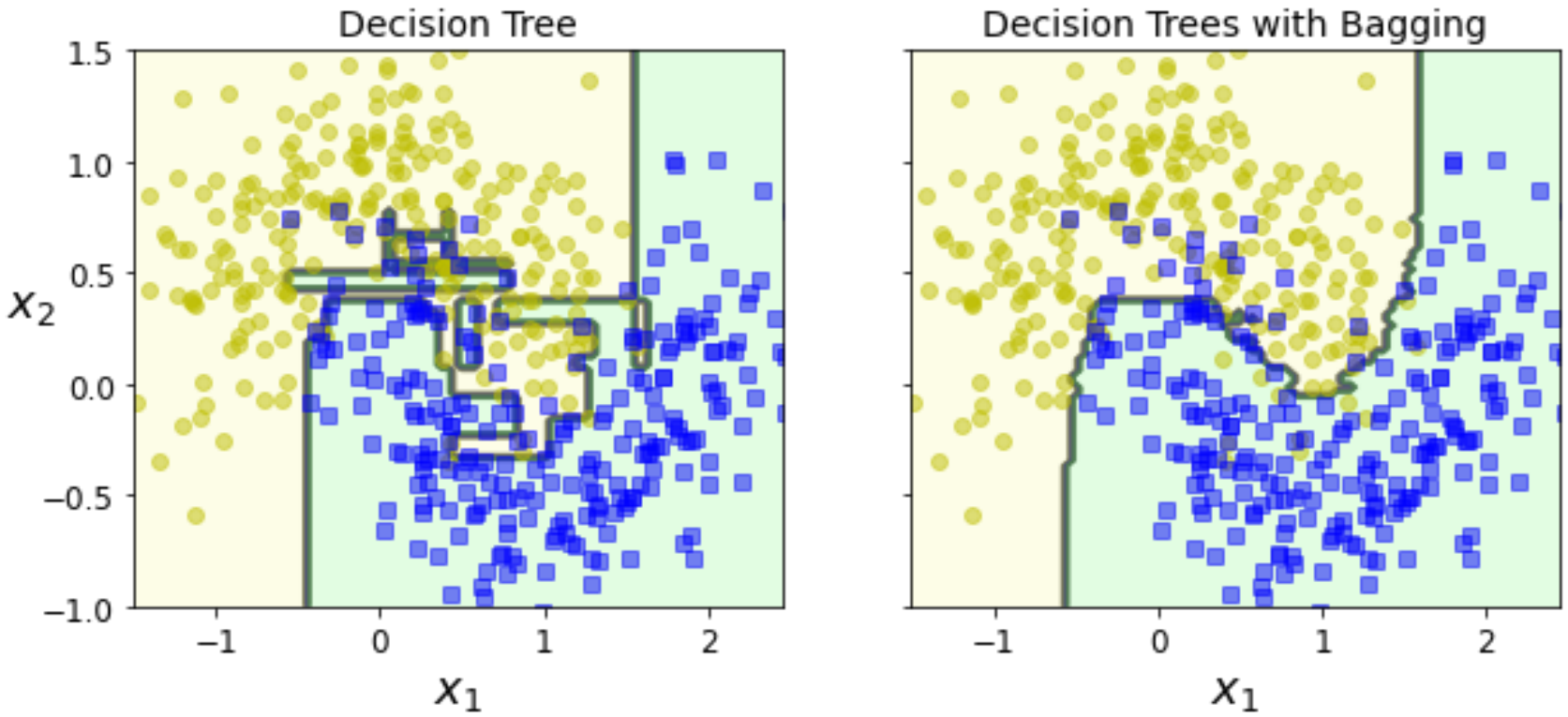
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

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

결정 트리를 기반 모델로 지정

```
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
```

3. Bagging

〈 2차원 평면에서의 결정 경계 〉



3. Bagging

< OOB 평가 : Out Of Bag >

- **OOB sample :** Original sample에서 **훈련에 전혀 사용되지 않는 sample**

original sample X = {x1, ..., xn} → bootstrap sample Y = {y1, ..., ym}에서 (일반적으로 m=n)

n개의 sample 중 bootstrap sample로 선택되지 않을 확률 $= \lim_{n\to\infty}\left(1-\frac{1}{n}\right)^{\boldsymbol{m}} = \lim_{n\to\infty}\left(1-\frac{1}{n}\right)^{\boldsymbol{n}} = e^{-1} \approx 37\%$

- **OOB 평가 :** OOB sample로 앙상블 모델을 평가하는 방식

→ **장점 : 훈련 세트를 효율적으로 사용**

3. Bagging

< OOB 평가 : Out Of Bag >

```
bag_clf=BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    bootstrap=True, oob_score=True, random_state=40
)

bag_clf.fit(X_train, y_train)
print(bag_clf.oob_score_)
# 0.8986666666666666

bag_clf.oob_decision_function_[0]
# [0.32275132, 0.67724868]
```

OOB 평가 실시

```
from sklearn.metrics import accuracy_score

y_pred=bag_clf.predict(X_test)
print(accuracy_score(y_test, y_pred))
# 0.912
```

4. 랜덤포레스트

랜덤 포레스트

배깅 방법 (또는 페이스팅) 을 적용한 결정 트리의 앙상블
max_samples: 훈련 세트 크기 지정

트리의 노드를 분할 할 때
선택한 특성 후보 중에서 **최적의 특성을 찾는** 식으로 무작위
성을 더 주입

4. 랜덤포레스트

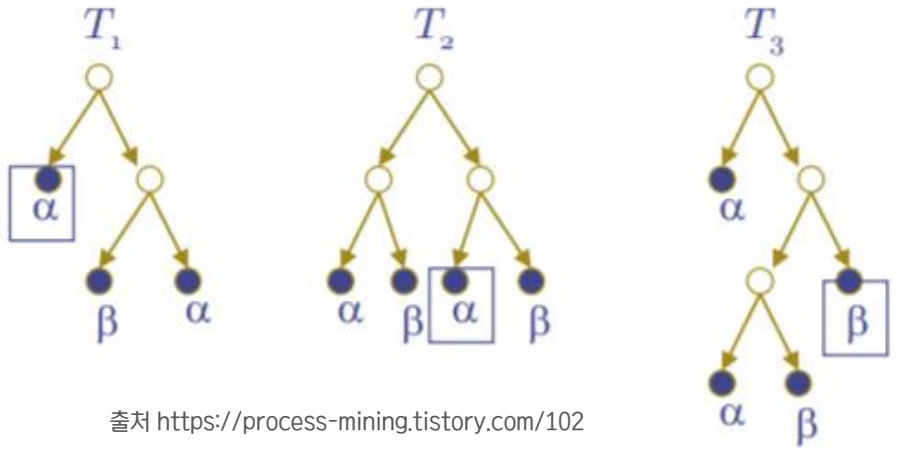
랜덤 포레스트 형성과정

1. 많은 Decision tree 형성

2. 트리들에 Randomness 주입
   -Bootstrap sampling process
   -Random attribute selection

3. Ensemble

```python
from sklearn.ensemble import RandomForestClassifier

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

#n_estimators 사용할 트리 수
#max_leaf_nodes
#n_jobs=-1 가능한 모든 cpu 코어 사용

y_pred_rf=rnd_clf.predict(X_test)
```



출처 https://process-mining.tistory.com/102

4. 랜덤포레스트

랜덤 포레스트 장단점

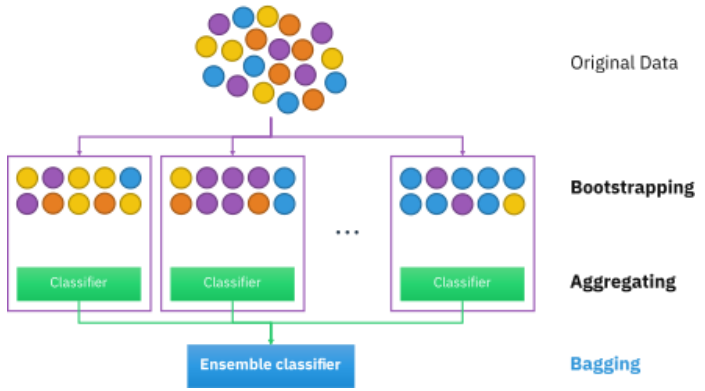
장점
-Decistion tree의 overfitting 문제 예방
-알고리즘이 굉장히 간단
-빠른 training 속도

단점
-Memory 사용량이 매우 많다.
-training data의 양이 증가해도 급격한 성능의 향상 X

4. 랜덤포레스트

Bagging vs Random Forest

Bagging
Boostrap Aggregation
복원추출을 사용한
표본추출방법(Boostrap)+
통합(Aggregation)



If
하나의 강력한 독립변수가 존재,

→ 첫 분할 변수만 고려될 것.
그 결과 배깅된 트리들은 서로 유사하게 보일 것
→ 배깅된 트리들에서 얻은 예측치들의 높은 상관성 초래

Random Forest
Bagging의 일종
'설명변수'도 무작위 선택
→ 트리의 다양성 확보
→ 상관관계 줄여
예)
1st sampling:변수 a,b,c,d,e
2nd sampling: 변수 c,w,z,
3rd sampling: 변수 a,u,v,b
각각의 변수를 기반으로
의사결정나무 형성

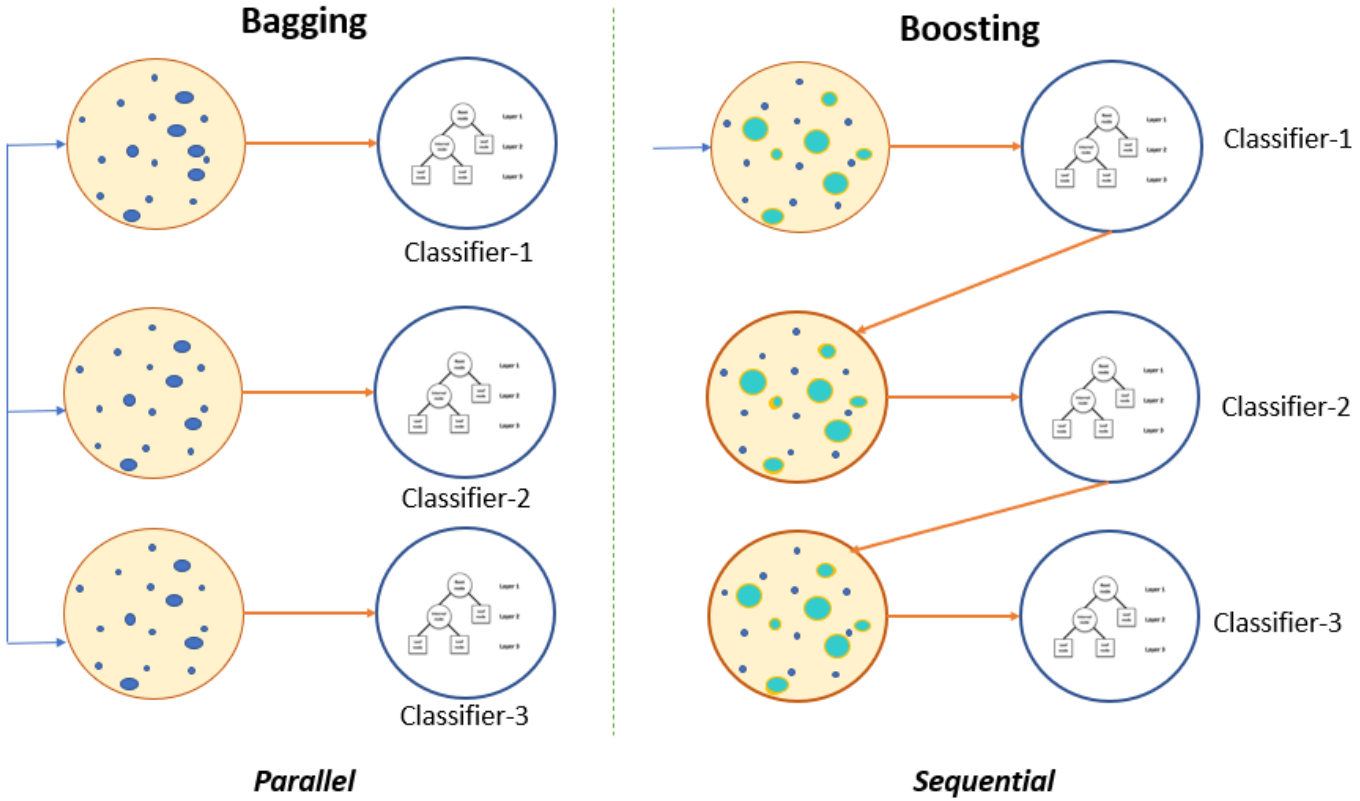
출처: https://www.analyticsvidhya.com/blog/2020/02/what-is-bootstrap-sampling-in-statistics-and-machine-learning/

5. 부스팅

Boosting : 잘못된 데이터, 더 큰 가중치

부스팅은 약한 학습기를 여러 개 연결하여 강한 학습기를 만드는 앙상블 방법
앞의 모델을 보완해나가면서 일련의 예측기를 학습시켜

- Ada Boost 알고리즘
- SAMME 알고리즘

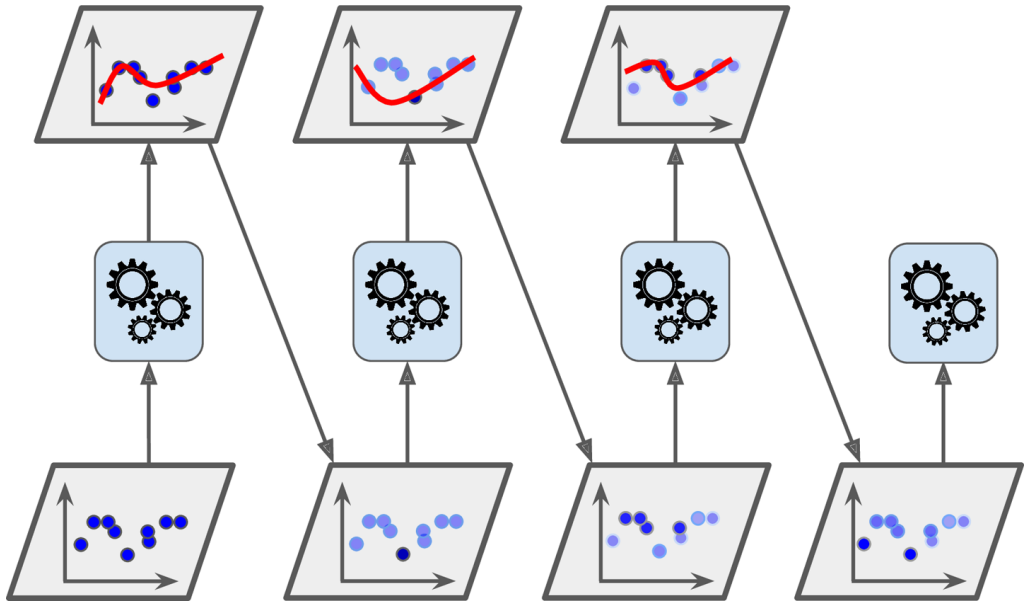


출처: https://www.pluralsight.com/guides/ensemble-methods:-bagging-versus-boosting

5.1 에이다부스트

AdaBoost 알고리즘

이전 모델이 과소적합했던 훈련 샘플의 가중치 높이는 것
단점: 병렬화(분화) 불가, 배깅이나 페이스팅 보다 확장성 떨어져



출처: Hands-On Machine Learning with Scikit-Learn & Tensorflow

5.1 에이다부스트

J번째 예측기, i번째 샘플

- J번째 예측기의 가중치가 적용된 에러율

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

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

- 가중치 업데이트 규칙

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

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

- 에이다 부스트 예측

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

5.1 에이다부스트

SAMME 알고리즘

에이다 부스트의 다중 클래스 버전
(클래스가 2개이면 Ada Boost와 일치)
예측값 대신 클래스 확률에 기반하여 성능이 좋아

```
from sklearn.ensemble import AdaBoostClassifier

ada_clf=AdaBoostClasifier(
  DecisionTreeClassifier(max_depth=1), n_estimators=200,
  algorithm="SAMME.R", learnign_rate=0.5)
ada_clf.fit(X_train, y_train)
```

5.2 그레이디언트 부스팅

그레이디언트 부스팅 gradient boosting:

이전 예측기가 만든 잔여 오차에 새로운 예측기를 학습시키는 알고리즘.
즉, 경사 하강법(Gradient Descent)를 이용해 가중치 업데이트를 수행하는 기법

5.2 그레이디언트 부스팅

그레이디언트 부스팅 gradient boosting:

손실함수 $L(y, F(x)) = (y - F(x))^2/2$

잔차 = 미분의 음수값 $y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$

목적함수 $J = \sum_i L(y_i, F(x_i))$

손실함수 최소화 위해 경사하강법 사용

$$F_t(x_i) = F_{t-1}(x_i) - \eta \underbrace{\frac{\partial J}{\partial F_{t-1}(x_i)}}_{F_{t-1}(x_i) - y_i}, \; t = 1, 2, \ldots$$

$$F_t(x_i) = F_{t-1}(x_i) + \eta \cdot \underbrace{(y_i - F_{t-1}(x_i))}_{\text{잔차}}$$

미분

$$\begin{aligned} \frac{\partial J}{\partial F(X_i)} &= \frac{\sum_i L(y_i, F(x_i))}{\partial F(x_i)} \\ &= \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \\ &= \underline{F(x_i) - y_i} \end{aligned}$$

→ t번째 예측값 $Ft(xi)$는
(t-1번째 예측값 +학습률*잔차)로 업데이트 됨.

5.2 그레이디언트 부스팅

그레이디언트 부스티드 회귀 트리(GBRT)

```
from sklearn.tree import DecisionTreeRegressor

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

```
DecisionTreeRegressor(max_depth=2, random_state=42)
```

DecisionTreeRegressor를 훈련세트에 학습시킴

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

```
DecisionTreeRegressor(max_depth=2, random_state=42)
```

첫 번째 예측기에서 생긴 잔여오차에 두 번째 회귀 모델 훈련시킴

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

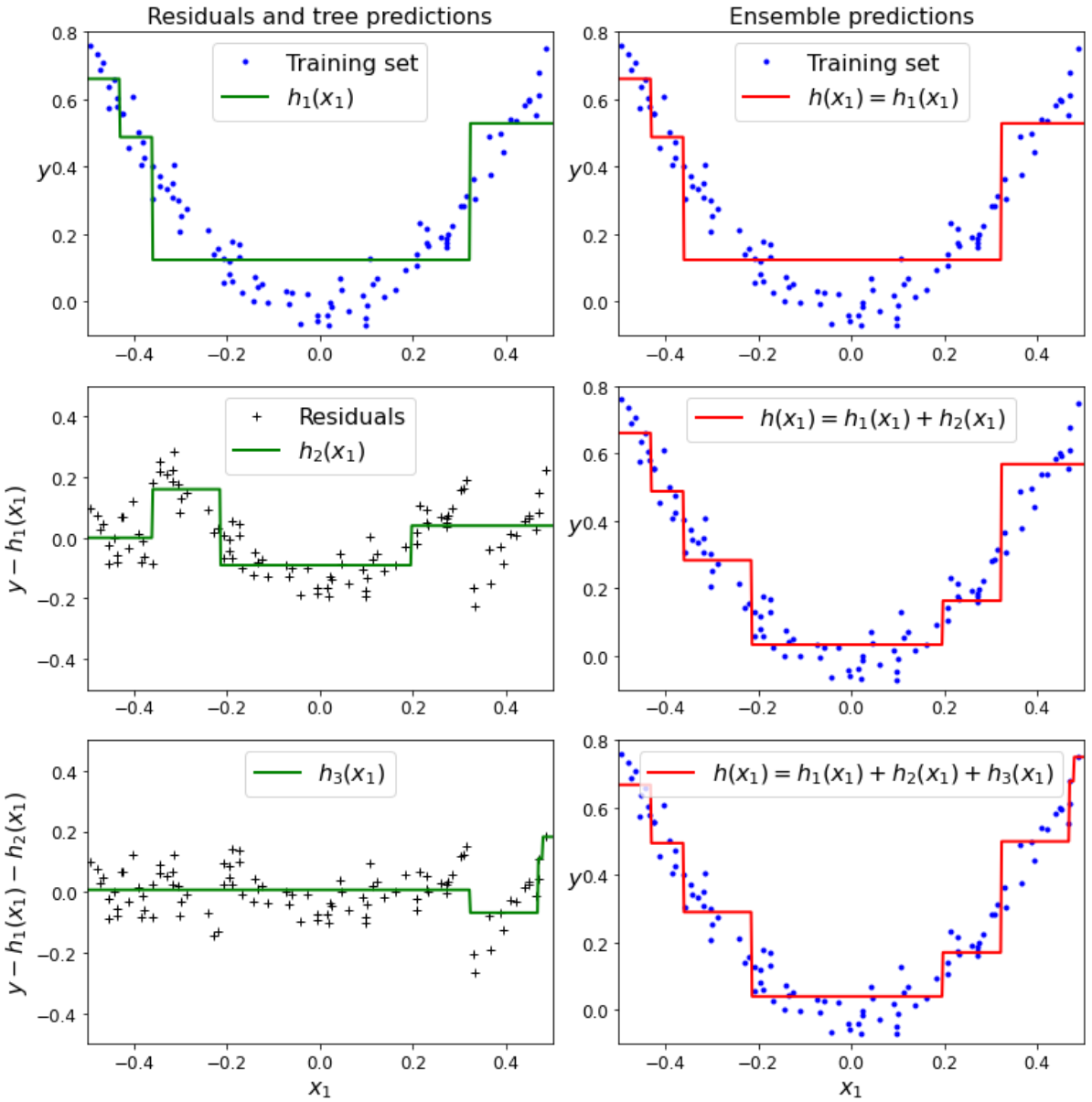
```
DecisionTreeRegressor(max_depth=2, random_state=42)
```

두 번째 예측기에서 생긴 잔여오차에 세 번째 회귀 모델 훈련시킴

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

모든 트리의 예측을 더해 새로운 샘플에 대한 예측 만듦

5.2 그레이디언트 부스팅



➔ 트리가 앙상블에 추가될수록 앙상블의 예측이 좋아짐

5.2 그레이디언트 부스팅

그레이디언트 부스티드 회귀 트리(GBRT)

사이킷런의 GradientBoostingRegressor 사용

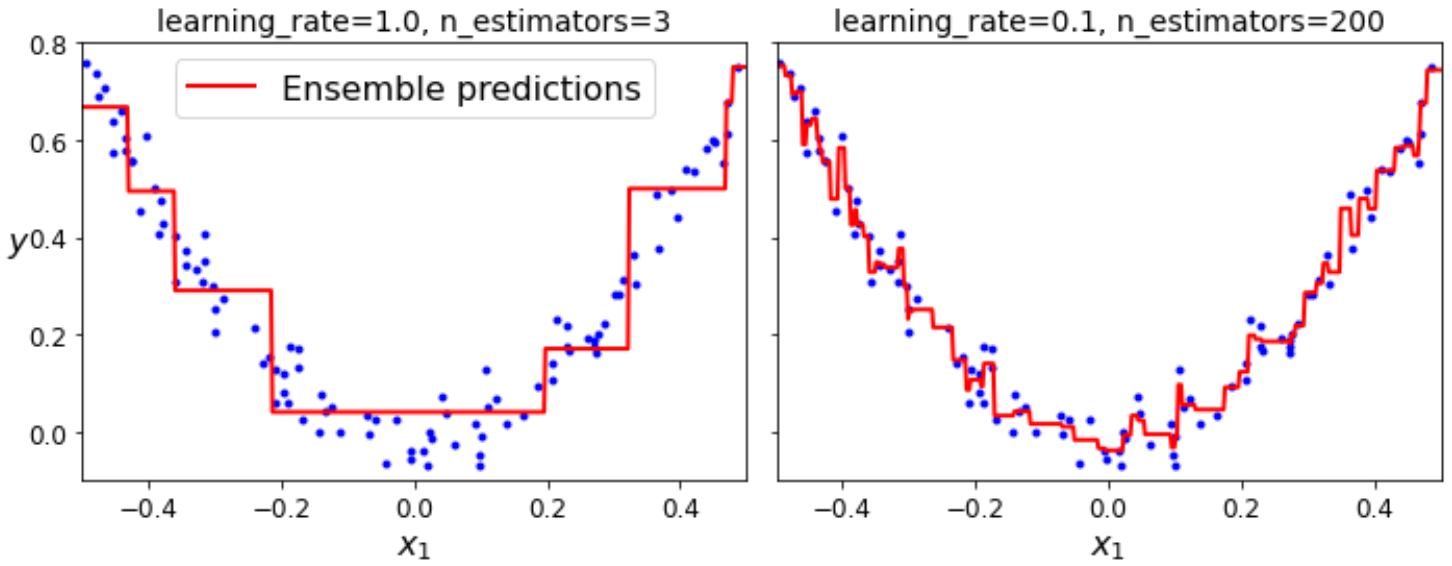
```
from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0, random_state=42)
gbrt.fit(X, y)
```

```
GradientBoostingRegressor(learning_rate=1.0, max_depth=2, n_estimators=3,
                          random_state=42)
```

5.2 그레이디언트 부스팅

learning_rate: 각 트리의 기여 정도 조절하는 매개변수:

약한 학습기가 순차적으로 오류 값을 보정해 나가는 데 적용하는 계수로 0 ~ 1 값 지정 가능 (디폴트는 0.1)



큰 값 지정 → 빠른 수행 가능 but 예측 성능 저하

작은 값 지정 → 예측 성능 높아짐 ➔ 축소 shrinkage (but 과대적합의 위험 있음)

5.2 그레이디언트 부스팅

최적의 트리 수? **조기 종료 기법**

조기 종료 기법: 이전 epoch와 비교해서 오차가 증가했다면 학습을 중단하는 방법

(∵ epoch 수 많을수록 오차 줄어들지만,
어느 시점부터 오차 점점 커짐(오버피팅 의미))

5.2 그레이디언트 부스팅

최적의 트리 수? 조기 종료 기법

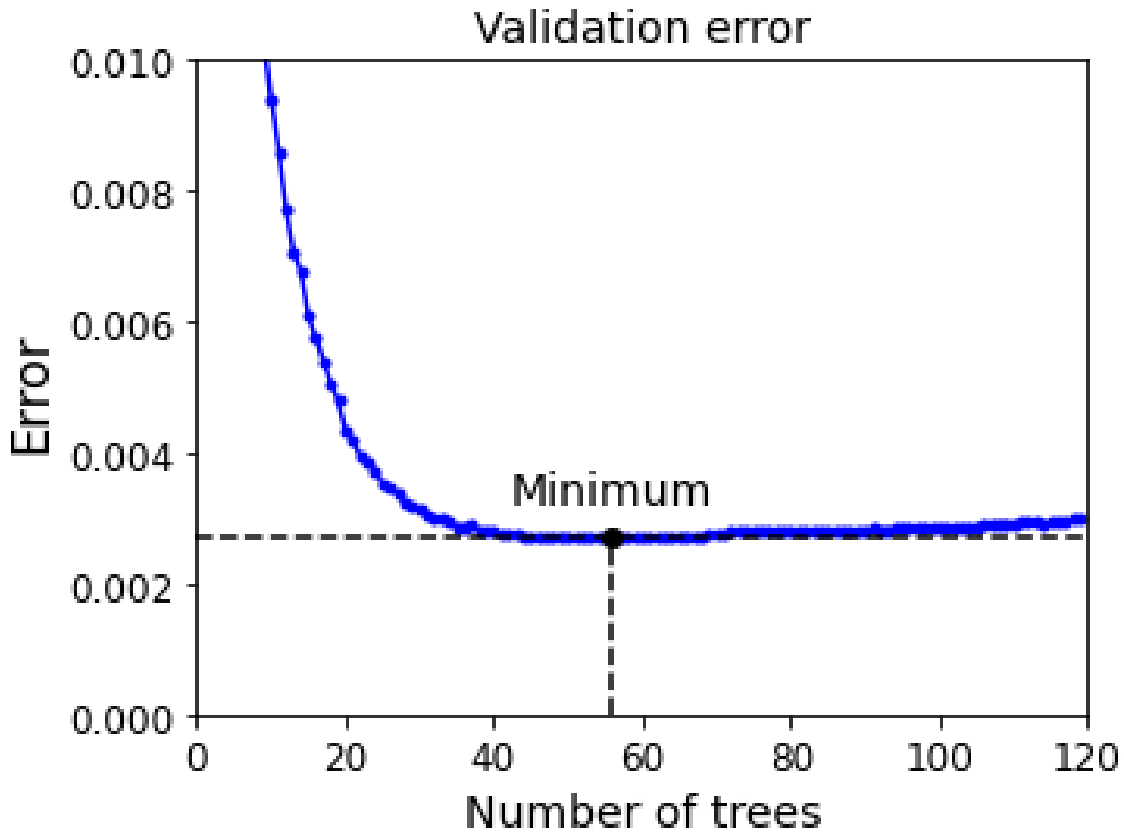
staged_predict():훈련의 각 단계에서 앙상블에 의해 만들어진 예측기를 순회하는 반복자 반환.

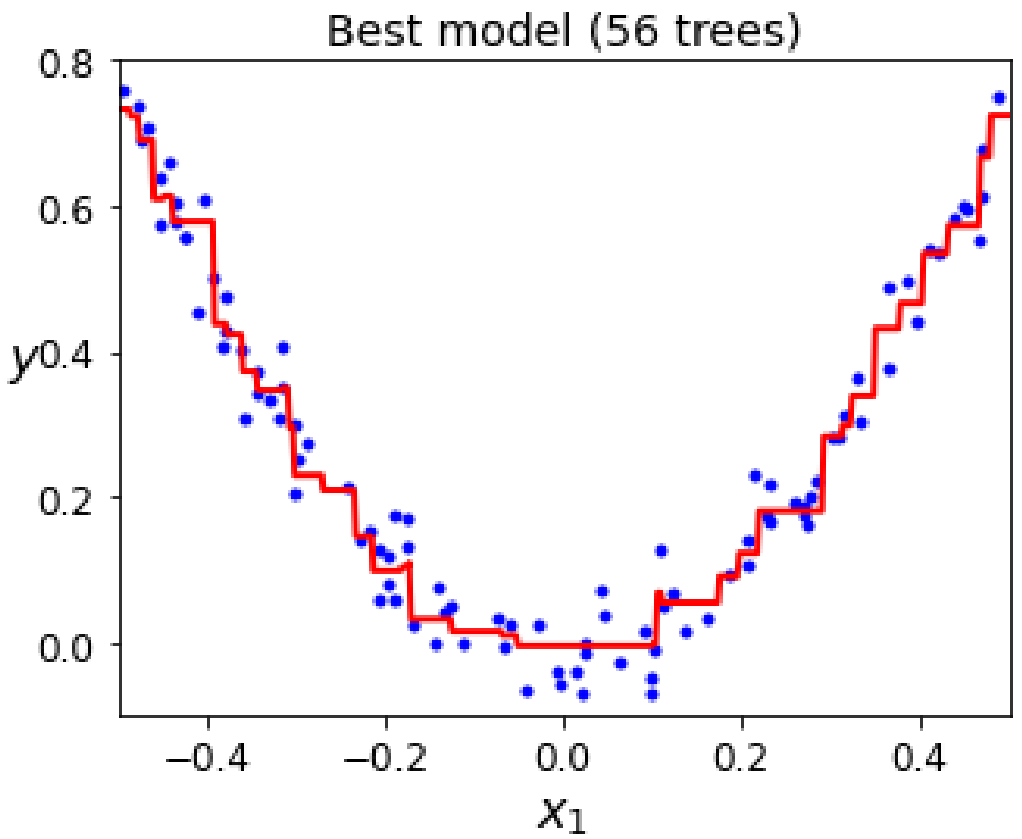
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

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)

errors = [mean_squared_error(y_val, y_pred)
          for y_pred in gbrt.staged_predict(X_val)]
bst_n_estimators = np.argmin(errors) + 1

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

np.argim(array) : array에서 최솟값의 인덱스 반환
즉, 에러가 최소값을 갖게 하는 인덱스로
최적 트리의 수 판단

```
GradientBoostingRegressor(max_depth=2, n_estimators=56, random_state=42)
```

5.2 그레이디언트 부스팅





5.2 그레이디언트 부스팅

실제로 훈련을 중지하는 방법으로 조기 종료 구현.

```
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: #연속해서 5번 MSE가 기존보다 크면
            break  # early stopping조기종료
```

Fit() 메서드가 호출될 때
기존 트리를 유지하고
훈련을 추가할 수 있도록 해줌.

5.2 그레이디언트 부스팅

장점

- (랜덤포레스트 보다) 뛰어난 성능
- 조기 종료를 통한 과적합 방지 가능
- 이진특성이나 연속적인 특성에서 잘 동작

단점

- 느린 수행 시간
  (병렬처리 지원되지 않아 대용량 데이터의 경우 학습에 매우 많은 시간 필요)

→병렬처리 불가능한 GBM의 단점 보완해주고 최고의 성능 보여주는

그레이디언트 부스팅 기반의 패키지 XGBoost와 LightGBM 등장

XGBoost

XGBoost: 병렬 처리 가능한 최적화된 그레이디언트 부스팅 구현하는 파이썬 라이브러리

특징 및 장점

- 분류와 회귀 모두 가능
- 그레이디언트 부스팅과 마찬가지로 가중치 업데이트를 경사하강법 사용
- 그레이디언트 부스팅보다 훨씬 빠른 속도
- 자체적으로 과적합 규제 기능 가짐

XGBoost

```
try:
    import xgboost
except ImportError as ex:
    print("에러: xgboost 라이브러리 설치되지 않았습니다.")
    xgboost = None
```

```
if xgboost is not None:  # 책에 없음
    xgb_reg = xgboost.XGBRegressor(random_state=42)
    xgb_reg.fit(X_train, y_train)
    y_pred = xgb_reg.predict(X_val)
    val_error = mean_squared_error(y_val, y_pred)
    print("Validation MSE:", val_error)
```

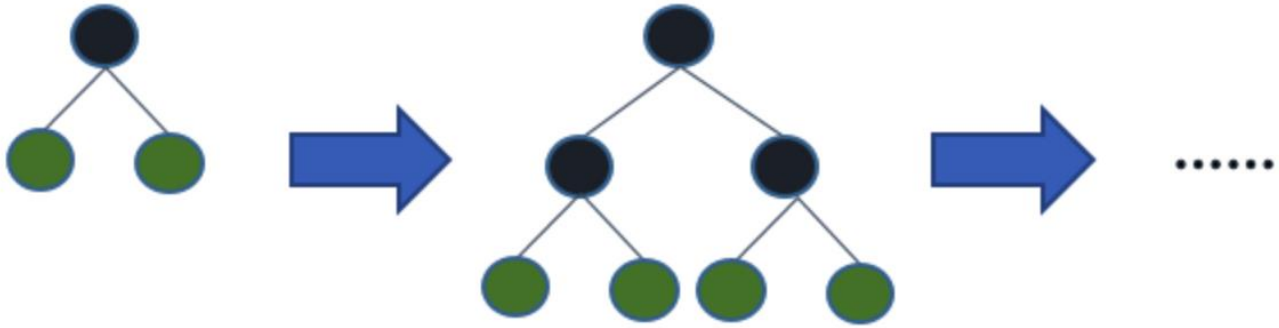
```
Validation MSE: 0.004000408205406276
```

```
if xgboost is not None:  # 책에 없음
    xgb_reg.fit(X_train, y_train,
                eval_set=[(X_val, y_val)], early_stopping_rounds=2) #자동조기종료 기능 제공
    y_pred = xgb_reg.predict(X_val)
    val_error = mean_squared_error(y_val, y_pred)
    print("Validation MSE:", val_error)
```

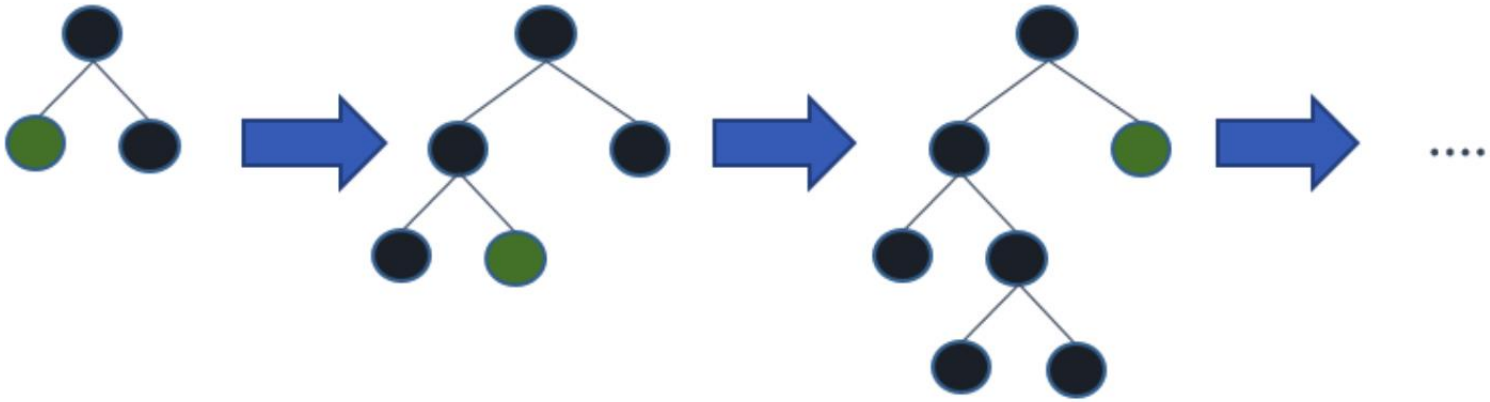
자체 조기종료 기능: Fit() 메서드에 eval_set 추가, early_stopping_rounds: 최대한 몇 개의 트리를 완성해볼 것인지, valid loss에 더 이상 진전이 없으면 멈춤.

LightGBM

LightGBM: 리프 중심 트리 분할 방식을 사용하는 그레이디언트 부스팅 방식의 알고리즘



Level-wise tree growth



Leaf-wise tree growth

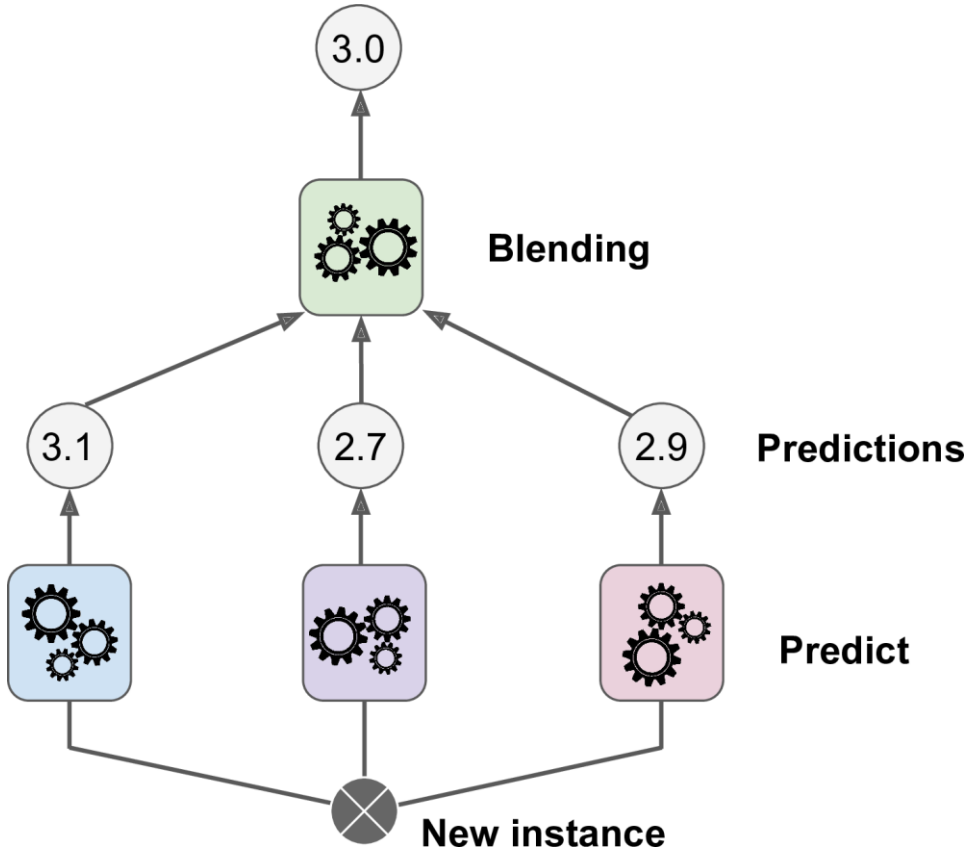
LightGBM

LightGBM: 리프 중심 트리 분할 방식을 사용하는 그레이디언트 부스팅 방식의 알고리즘

장점	단점
- 학습하는데 걸리는 시간이 적음 - 메모리 사용량이 상대적으로 적음 - 카테고리형 피처들의 자동 변환과 최적 분할	- 적은 데이터를 세트를 적용할 경우 과적합 가능성이 큼

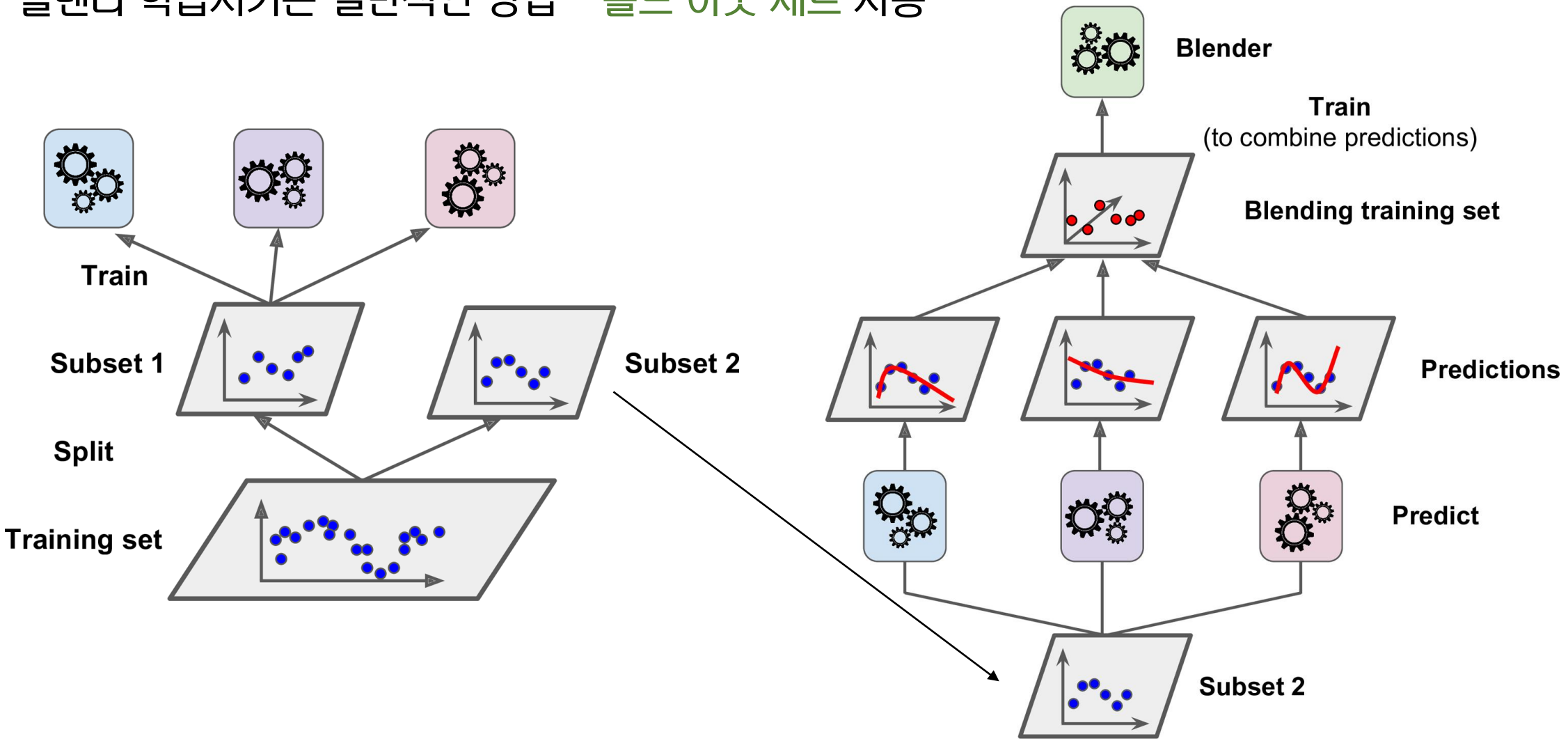
6. 스태킹

스태킹 stacking:
여러 가지 모델들의 예측값을 최종 모델의 학습 데이터로 사용해 최종 예측을 만드는 방법
→ 예측을 취합하는 모델을 훈련시킴



6. 스태킹

블렌더 학습시키는 일반적인 방법 - 홀드 아웃 세트 사용



감사합니다

Q&A