

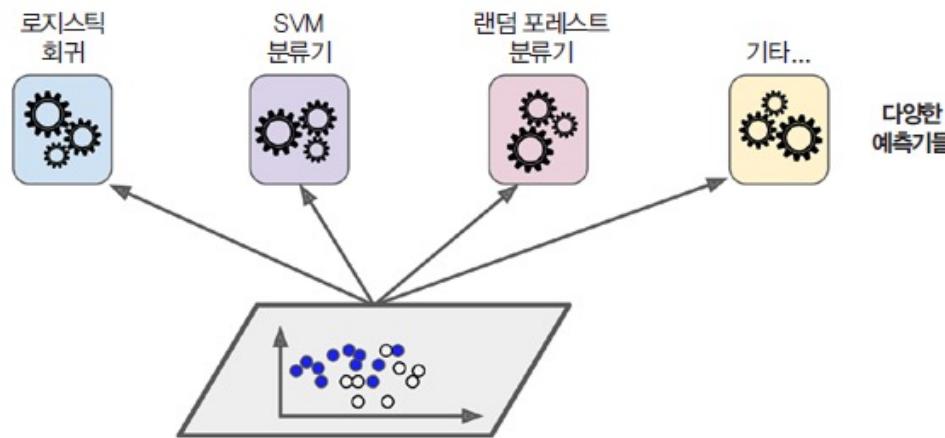
# 앙상블 Ensemble 1.

- 앙상블의 개념
- 보팅(Voting)
- 배깅(Bagging)
  - 랜덤포레스트(RandomForest)
- 부스팅(Boosting)
  - XGBoosting
  - LGBM
  - CatBoosting

# 앙상블 개요

## ■ 앙상블 기법 ensemble methods

- 다양한 분석 방법론을 조합해 하나의 예측 모형을 만드는 지도학습 방법론
- 앙상블은 다양한 분석 방법론을 조합하기 때문에 해석이 어렵지만, 보다 우수한 예측 성능을 갖는다는 장점을 가짐
- 이는 실생활에서 중요한 결정을 앞두고 여러 가지 의견을 모으고 종합해 최종 결정을 하는 것과 유사함



※ 출처: 핸즈온머신러닝, 한빛미디어, 2021, p246

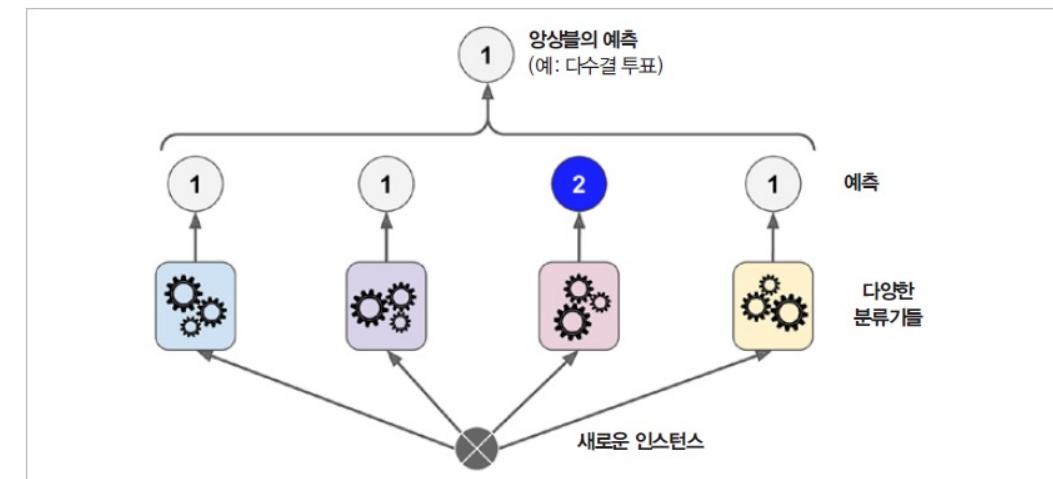


그림 7-2 직접 투표 분류기의 예측

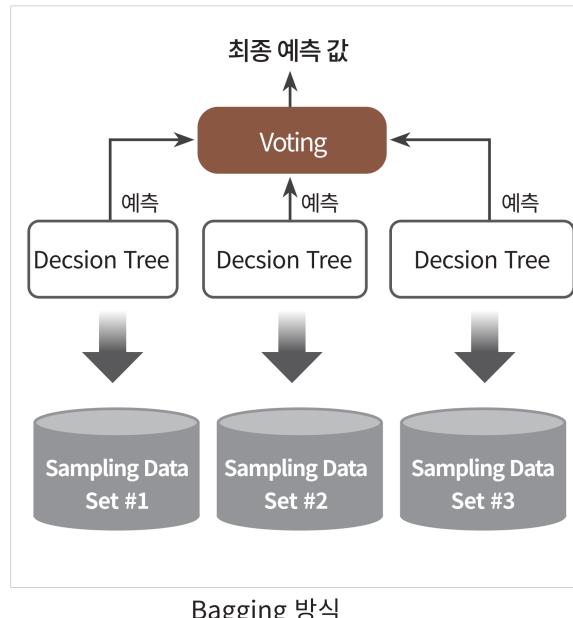
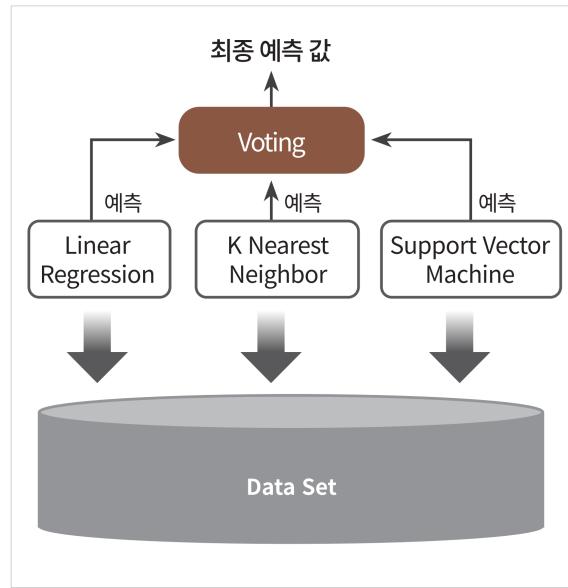
# 앙상블의 특징

---

- 단일 모델의 약점을 **다수의 모델들을 결합**하여 보완
- 성능이 떨어지더라도 서로 다른 유형의 모델을 섞는 것이 오히려 전체 성능에 도움이 될 수 있음
  - 예측기가 가능한 한 서로 독립적일 때 최고의 성능을 발휘함
  - 서로 다른 알고리즘으로 학습시키면 서로 다른 종류의 오차를 만들 가능성이 높기 때문에 정확도를 향상시킴
- 랜덤포레스트 및 뛰어난 알고리즘들은 모두 결정 트리 알고리즘 기반 알고리즘으로 적용함
- 결정트리의 단점인 과적합을 수십~수천개의 많은 분류기를 결합해 보완하고 장점인 직관적인 분류 기준은 강화됨
- 정형 데이터를 다루는데 가장 뛰어난 알고리즘(비정형데이터: 신경망)
- 랜덤포레스트, 그래디언트 부스팅 알고리즘
- XGBoost, LightGBM

# 앙상블 방식, 유형

보팅(Voting), 배깅(Bagging), 부스팅(Boosting), 스태킹(Stacking) 등

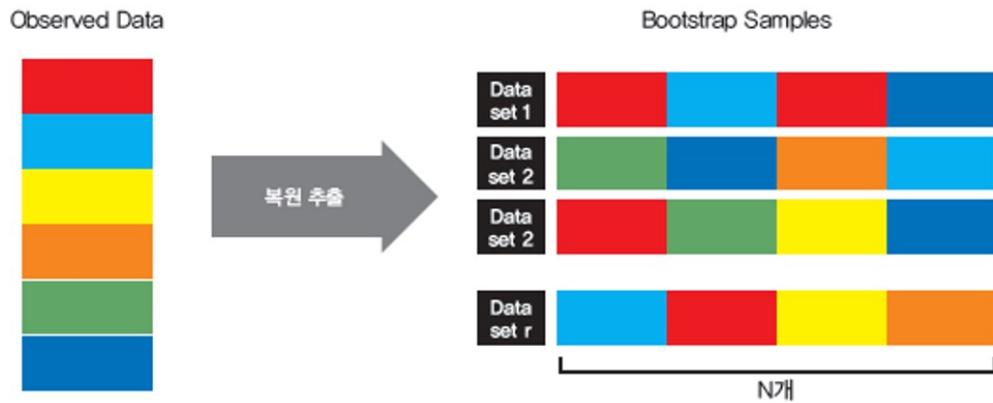


	보팅 Voting	배깅 Bagging
학습데이터셋	동일 데이터	서로 다른 데이터 샘플링, 중첩을 허용
알고리즘	서로 다른 알고리즘	같은 알고리즘
최종 결정 방식	보팅	보팅(동일)

※ 출처: 파이썬 머신러닝 완벽가이드, 위키북스, p212

※ **부트스트랩 BootStrap 분할 방식**: 원본 학습 데이터에서 데이터를 샘플링해 추출하는 방식, 중복을 허용

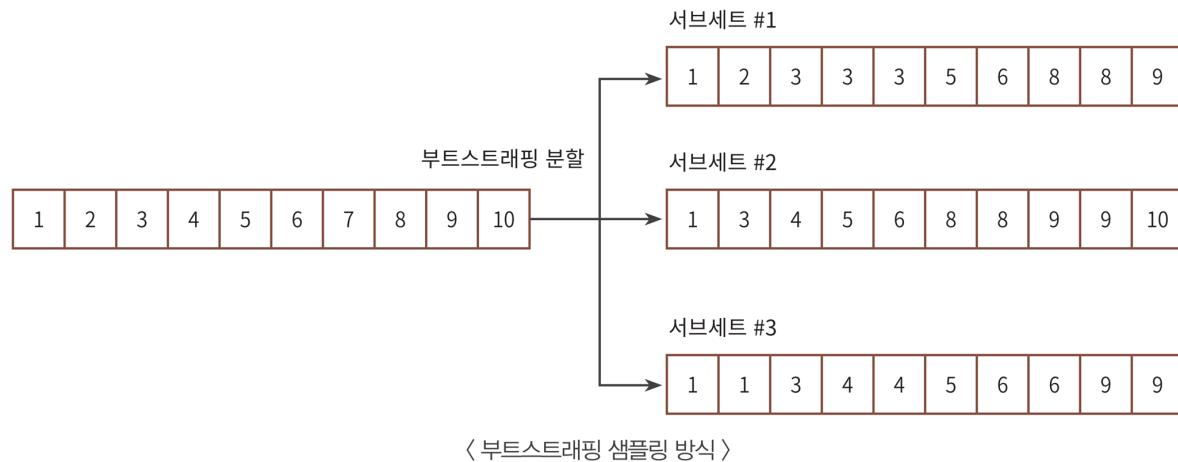
# Bootstrap 표본 추출 sampling 방법



Bootstrap 표본 추출 sampling 방법

- **부트스트래핑(bootstrapping)**이라고도 하는 표본 추출 방법
- 주어진 데이터에 대해 사전 설정한 비율에 따라 **복원 추출(sampling with replacement)**, 표본을 뽑고 다시 넣음)하는 방법
- 가령 100개의 관측치가 있을 때 첫 번째 추출로 선택된 관측치가 두 번째 표본 추출 시에도 포함될 수 있음.( 즉, 각 관측치에 대한 표본 추출 확률은 모두  $0.01 = 1/100$ 로 동일)
- 부트스트랩 표본 추출 방법을 이용하면 전체 데이터를 이용해 하나의 모델을 만드는 것보다 안정적인 예측 또는 분류 결과를 얻을 수 있음

# Bootstrap 표본 추출 sampling 방법



- 서브세트를 3으로 지정한 경우:
  - 서브세트의 데이터 건수는 전체 데이터 건수와 동일하지만, 개별 데이터가 중첩되어 만들어짐

## ■ OOB<sup>Out Of Bag</sup> 평가

- Bagging으로 샘플링할 때 선택되지 않은 샘플(일반적으로 37%)을 검증 데이터로 사용함
- 예측기마다 남겨진 샘플이 모두 다름
- 앙상블의 경우, 각 예측기의 oob 평가를 평균하여 사용
- sklearn의 경우 객체 생성시 oob\_score=True로 설정하여 학습 후 자동으로 oob평가를 수행시킬 수 있음

<※그림 출처: 머신러닝 완벽가이드, 위키북스, 2021, p214>

# 앙상블 기법1) - 보팅 Voting

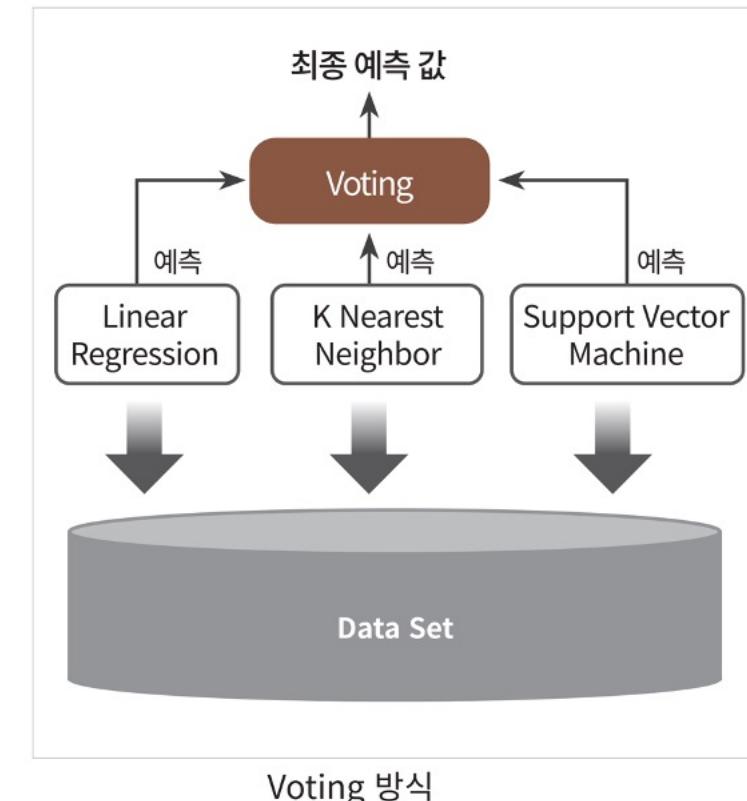
이 자료는 저작권자의 사전 서면 승인 없이 외부에 배포하기 위해 그 일부를 배포 인용 또는 복제 할 수 없습니다.

© Copyright

# 양상블 기법 1) – 보팅(Voting)

## [보팅(Voting)]

- 보팅과 배깅은 여러 개의 분류기가 **투표를 통해** 최종 결과를 예측
- 서로 다른 알고리즘을 사용하는 여러 개의 분류모델 사용
- 모두 같은 데이터를 사용하여 학습

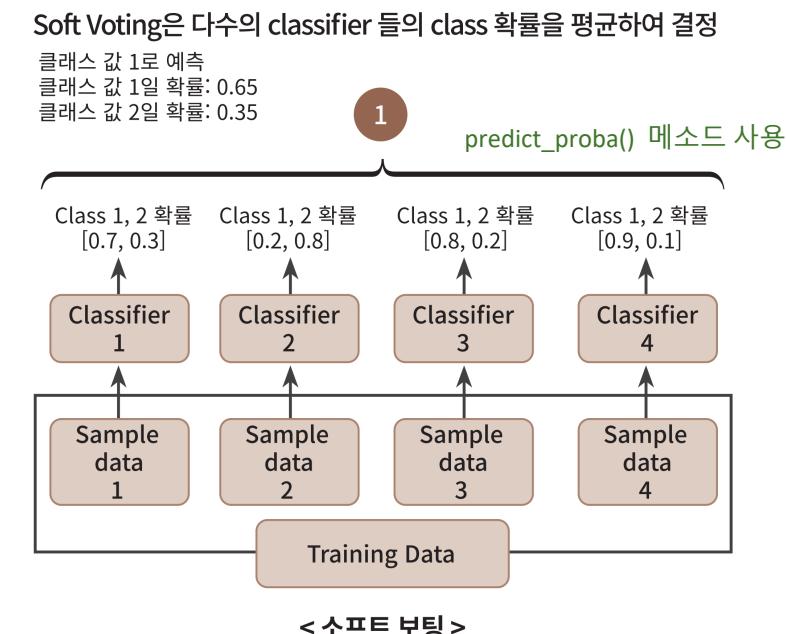
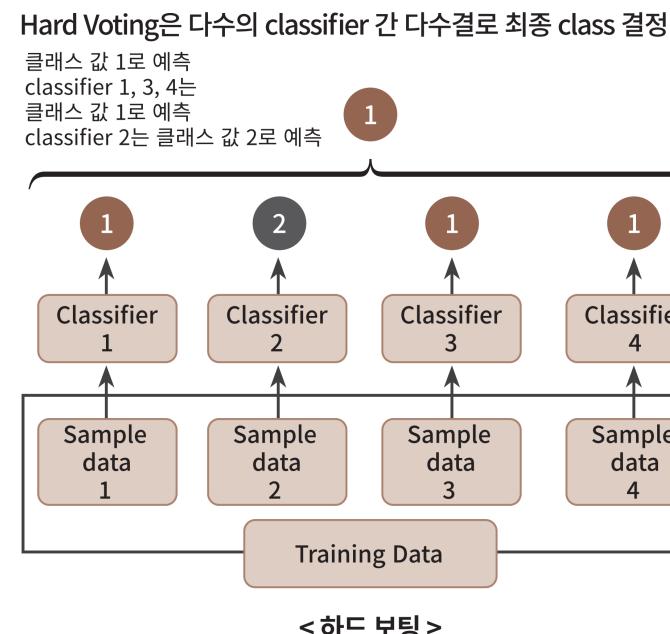


<이미지 출처: 파이썬 머신러닝 완벽가이드, 위키북스, 2021, p212>

# 양상블 기법 1) – 보팅(Voting)

보팅(Voting)의 유형 : 보팅은 하드 보팅과 소프트 보팅으로 나누어짐

- **하드 보팅(직접 투표)** : 각 분류기가 예측한 결과를 다수결로 최종 class로 결정
- **소프트 보팅(간접 투표)**: 각 분류기가 예측한 확률을 평균하여 결정하는 방식
- 일반적으로는 확률이 더 높은 투표에 비중을 더 두는 소프트 보팅이 하드 보팅보다 예측 성능이 상대적으로 우수
- 사이킷런은 VotingClassifier 클래스를 통해 보팅을 지원



출처:머신러닝 완벽가이드, 위키북스, 2021, p214

# 보팅 분류기(Voting Classifier)

실습 – 위스콘신 유방암 데이터 세트

```
vo_clf = VotingClassifier(...)
```

Parameters:

**estimators : list of (str, estimator) tuples**

Invoking the `fit` method on the `VotingClassifier` will fit clones of those original estimators that will be stored in the class attribute `self.estimators_`. An estimator can be set to 'drop' using `set_params`.

*Changed in version 0.21:* 'drop' is accepted. Using None was deprecated in 0.22 and support was removed in 0.24.

**voting : {'hard', 'soft'}, default='hard'**

If 'hard', uses predicted class labels for majority rule voting. Else if 'soft', predicts the class label based on the argmax of the sums of the predicted probabilities, which is recommended for an ensemble of well-calibrated classifiers.

# [실습] Hard Voting, Soft Voting

---

## [실습1]

- 위스콘신 유방암 데이터세트

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import VotingClassifier

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

b_cancer = load_breast_cancer()

b_cancer.keys()

print(b_cancer.DESCR)
```

## [실습] Hard Voting, Soft Voting

```
# 개별 분류기 모델 생성
lr_clf = LogisticRegression(max_iter=10000)
knn_clf = KNeighborsClassifier(n_neighbors=8)
dt_clf = DecisionTreeClassifier(max_depth=3)

# 양상별 분류기(보팅)
vo_clf = VotingClassifier(estimators=[('LR', lr_clf), ('KNN', knn_clf), ('DT', dt_clf)], \
                           voting='soft')

# 학습, 예측, 평가
vo_clf.fit(X_train, y_train)

y_pred_test = vo_clf.predict(X_test)
test_score = accuracy_score(y_test, y_pred_test)
print("voting 양상별 test score: ", test_score)
```

# 앙상블 기법 2) - 배깅 Bagging

이 자료는 저작권자의 사전 서면 승인 없이 외부에 배포하기 위해 그 일부를 배포 인용 또는 복제 할 수 없습니다.

© Copyright

# 앙상블 기법2) - 배깅(bagging)

---

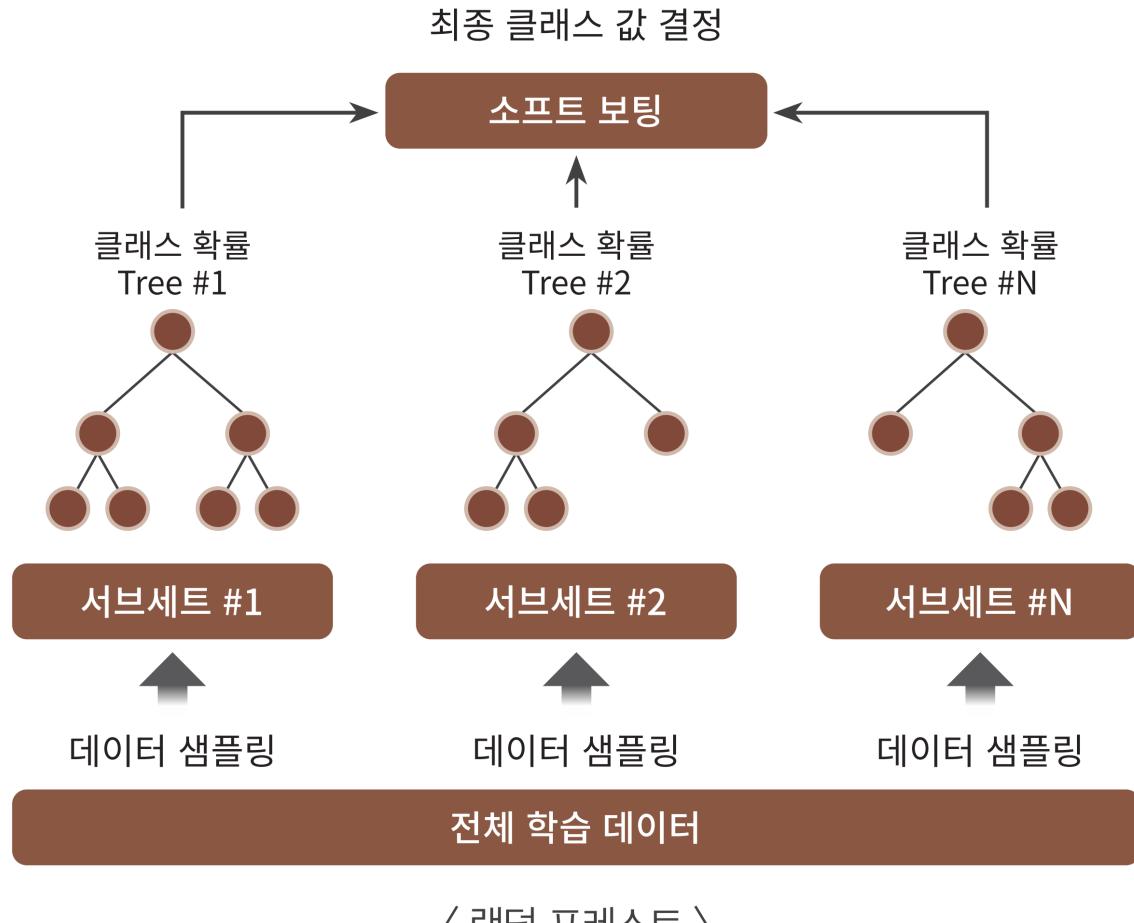
## 배깅(bagging; Bootstrap AGGREGATlNG)

- Bootstrap 샘플링을 통한 voting을 통한 앙상블 예측기
- 최종 예측을 하는 점은 voting과 동일
- 학습할 데이터를 추출하는 방식에서 차이: 부트스트랩(bootstrap)이란 표본 추출 방법(복원 추출)을 이용해 각 모형의 다양성을 높여 앙상블 하는 방법
- 같은 종류의 알고리즘 모델을 여러 개 결합하여 예측
- 각각의 모델에 전체 학습 데이터 중 서로 다른 데이터를 샘플링하여 학습하는 점이 보팅과 차이
- 앙상블 모형으로 주로 결정트리 모형이 많이 쓰임.
- sklearn.ensemble.BaggingClassifier

## 랜덤 포레스트 RandomForest

- 결정 트리를 기반으로 한 대표적인 배깅 방식
- 앙상블 알고리즘 중 비교적 빠른 수행속도
- 다양한 영역에서 높은 예측 성능을 보임
- 연속형 및 범주형 모두에 사용 가능
- sklearn.ensemble.RandomForestClassifier(Regressor)

# 랜덤 포레스트



<이미지 출처: 파이썬 머신러닝 완벽가이드, 위키북스, 2021, p217>

- 서브세트를 3으로 지정한 경우( $n\_estimators=3$ ) 서브세트의 데이터 건수는 전체 데이터 건수와 동일하지만, 개별 데이터가 부트스트랩 방식으로 임의로(random) 중첩되어 만들어짐
- 여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 각자의 데이터를 샘플링해 개별적으로 학습을 수행한 뒤 최종적으로 모든 분류기가 보팅을 통해 예측 결정을 수행
- 트리 모형이 가지 분할 할 때마다(알고리즘의 각 단계마다) 임의로(Random) 결정된 전체 변수들의 부분 집합( $p$ 개)에 한정되고 해당 집합 내에서만 가지 분할 변수를 고려
- 랜덤하게 선택한 샘플과 특성을 사용하므로 훈련 세트에 과대적합되는 것을 막아줌.
- 기본 트리수( $n\_estimators = 100$ )
- 기본 특성수: 전체 특성갯수의 제곱근(분류일 경우)

# [실습] 랜덤 포레스트 실습: 와인 데이터 세트

## 실습: 와인 데이터 세트

```
import numpy as np

wine_kaggle = pd.read_csv('../data/wine_dataset.csv')

wine = wine_kaggle[['alcohol', 'residual_sugar', 'pH']]
wine.columns = ['alcohol', 'sugar', 'pH']
wine_kaggle['style'] = wine_kaggle['style'].replace('red', 0)
wine_kaggle['style'] = wine_kaggle['style'].replace('white', 1)

X = wine.to_numpy()
y = wine_kaggle['style'].to_numpy()
```

```
# 데이터 분할 : test_size=0.2, random_state=42

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,#
                                                    test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape)

(5197, 3) (1300, 3)
```

## [실습] 랜덤 포레스트 실습: 와인 데이터 세트

```
# RandomForestClassifier 모델 구축, 학습 및 평가

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

rf_clf = RandomForestClassifier(n_jobs=-1, random_state=42)
rf_clf.fit(X_train, y_train)

y_pred_train = rf_clf.predict(X_train)
train_score = accuracy_score(y_train, y_pred_train)

y_pred_test = rf_clf.predict(X_test)
test_score = accuracy_score(y_test, y_pred_test)

print("train score: ", train_score)
print("test score: ", test_score)
```

```
train score: 0.996921300750433
test score: 0.8892307692307693
```

## [실습] 랜덤 포레스트 실습: 와인 데이터 세트

---

### 하이퍼 파라미터

**n\_estimators:** 모형(weak learner)의 개수, 순차적으로 오류를 보정해 수가 많으면 성능이 일정 수준까지 높아질 수 있으나, 수행 시간이 오래 걸린다는 단점이 있음(디폴트는 100)

**min\_samples\_leaf:** 말단 리프 노드의 최소한의 샘플 데이터 수, 디폴트 1

**max\_depth:** 트리의 최대 깊이, 디폴트 3

**max\_features:** 디폴트는 auto, If "auto", then  $\text{max\_features} = \sqrt{\text{n\_features}}$  즉, 피처가 4개면 분할을 위해 2개 참조

최적 하이퍼 파라미터:

```
{'max_depth': 10, 'min_samples_leaf': 8, 'min_samples_split': 20, 'n_estimators': 100}
```

최고 예측 정확도: 0.8722

train score: 0.9028285549355397

test score: 0.8646153846153846

## [실습] 랜덤 포레스트: 타이타닉 데이터 세트

---

타이타닉 데이터 세트를 활용하여 랜덤 포레스트 모델과 결정 트리 모델을 동일하게 학습시키고 두 모델의 성능 차이를 비교하라

1. 기본 모델 작성 및 평가
2. 하이퍼파라미터 튜닝 및 평가
3. 특성 중요도 시각화

## [실습] 랜덤 포레스트: 사용자 행동 데이터 세트

---

1. 기본 모델 작성 및 평가
2. 하이퍼파라미터 튜닝 및 평가
3. 특성 중요도 시각화

## [실습] 랜덤 포레스트: 대출 상환 예측

---

## [실습] 랜덤 포레스트: MNIST 데이터 세트

---

1. 기본 모델 작성 및 평가
2. 하이퍼파라미터 튜닝 및 평가
3. 특성 중요도 시각화

## 용어 정리

---

**앙상블 ensemble**: 여러 모델의 집합을 이용해서 하나의 예측을 이끌어내는 방식(유의어: 모델 평균화 Model averaging)

**배깅 bagging**: 데이터를 부트스트랩핑해서 여러 모델을 만드는 일반적인 방법(유의어: 부트스트랩 종합 bootstrap aggregation)

**랜덤포레스트 random forest**: 결정 트리 모델에 기반을 둔 배깅 추정 모델(유의어: 배깅 의사 결정 트리)

## 양상블 기법 3) - 부스팅 Boosting

- \_ XGBoosting
- \_ LightGBM
- \_ CatBoosting

## 양상블 기법 3) 부스팅(Boosting)

---

여러 개의 약한 학습기를 순차적으로 학습

이전 모델이 잘못 예측한 데이터에 대한 예측 오차를 줄일 수 있는 방향으로 다음 모델을 계속 업데이트하면서 연속적으로 생성함.

여러 모델을 동시에 학습하지 않고 순서대로 학습하는 점에서 배깅과 차이

배깅과 마찬가지로 결정트리를 가장 많이 사용

배깅은 상대적으로 튜닝 필요성이 낮고 부스팅은 튜닝 개선 가능성이 높다

대표적 모델:

1. AdaBoost: 잔차에 따라 데이터의 가중치를 조절하는 부스팅의 초기 버전
2. GBM(Gradient Boost Machine) : 비용함수를 최소화하는 방향으로 부스팅을 활용하는 좀 더 일반적인 형태
3. 확률적 그래디언트 부스팅(Stochastic GB): 각 라운드마다 레코드와 열을 재표본추출
4. XGBoost 모델: SGB 중 하나
  1. kaggle, Dacon 등 경진대회에서 가장 많이 사용되는 알고리즘.
  2. 모델 학습 속도가 빠르고 예측력이 상당히 좋은 편
5. LightGBM

# 부스팅 알고리즘 작동원리

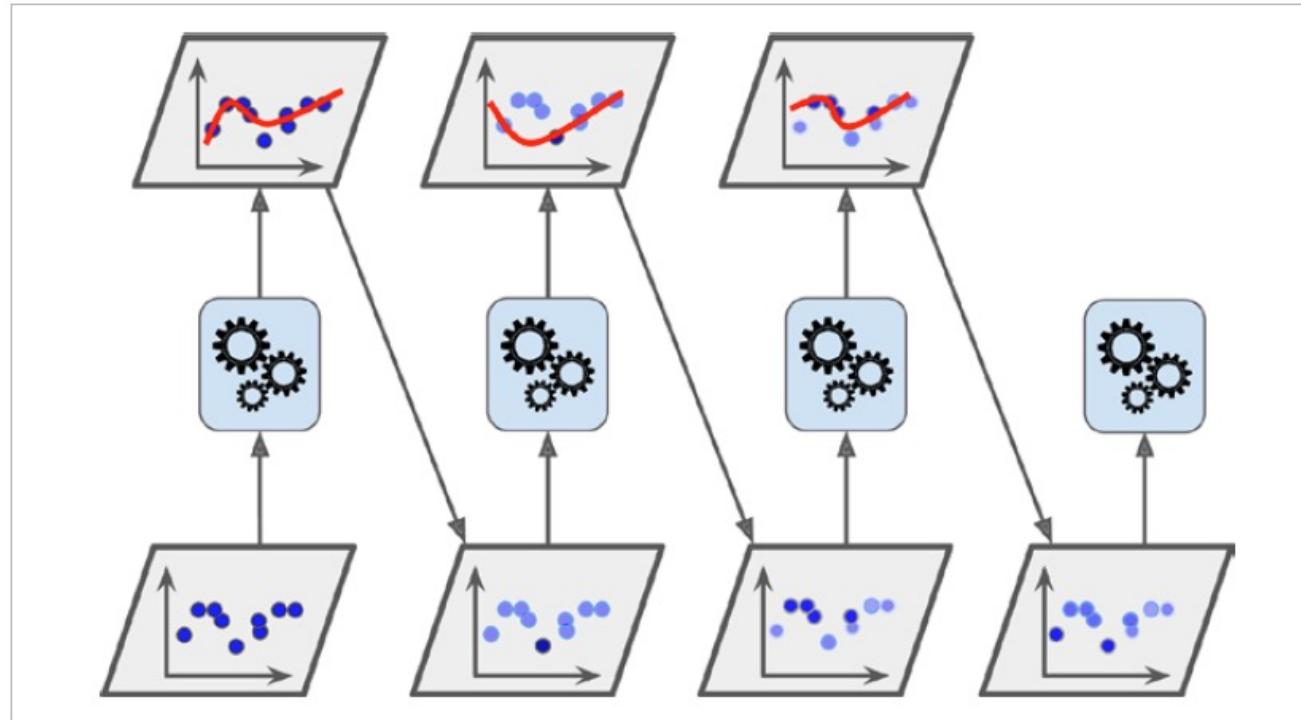
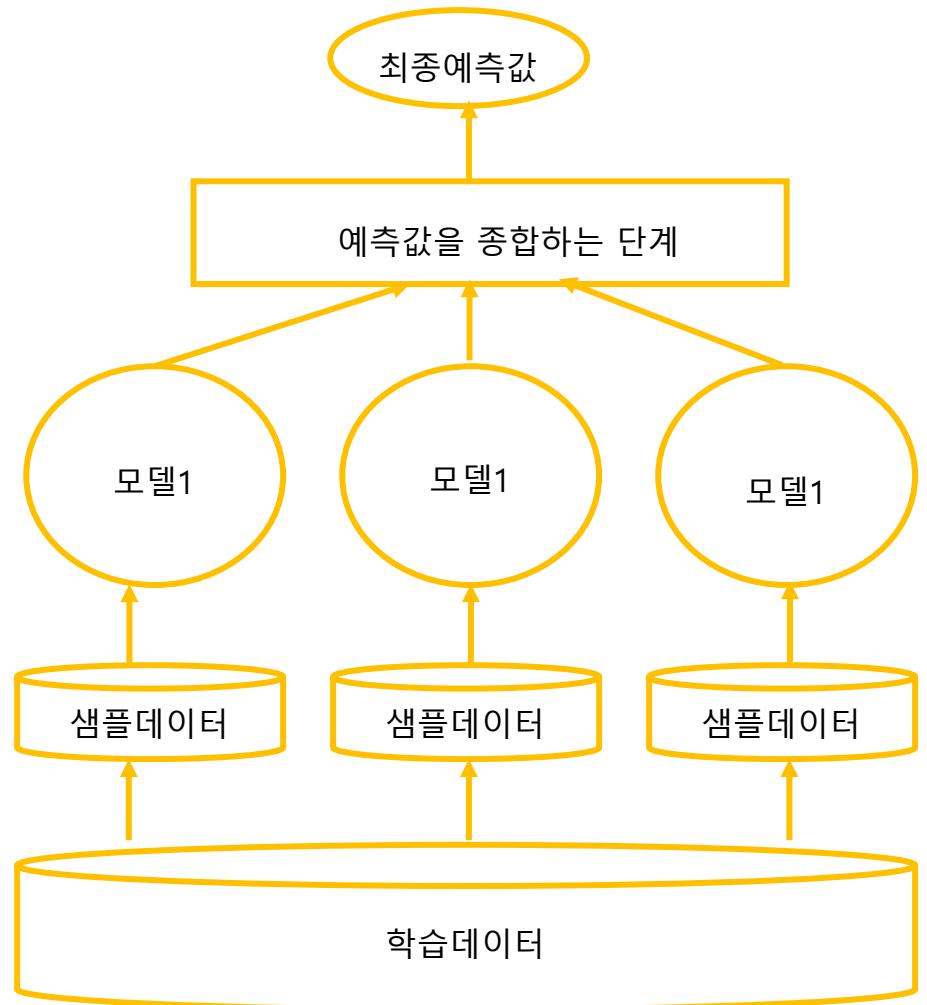


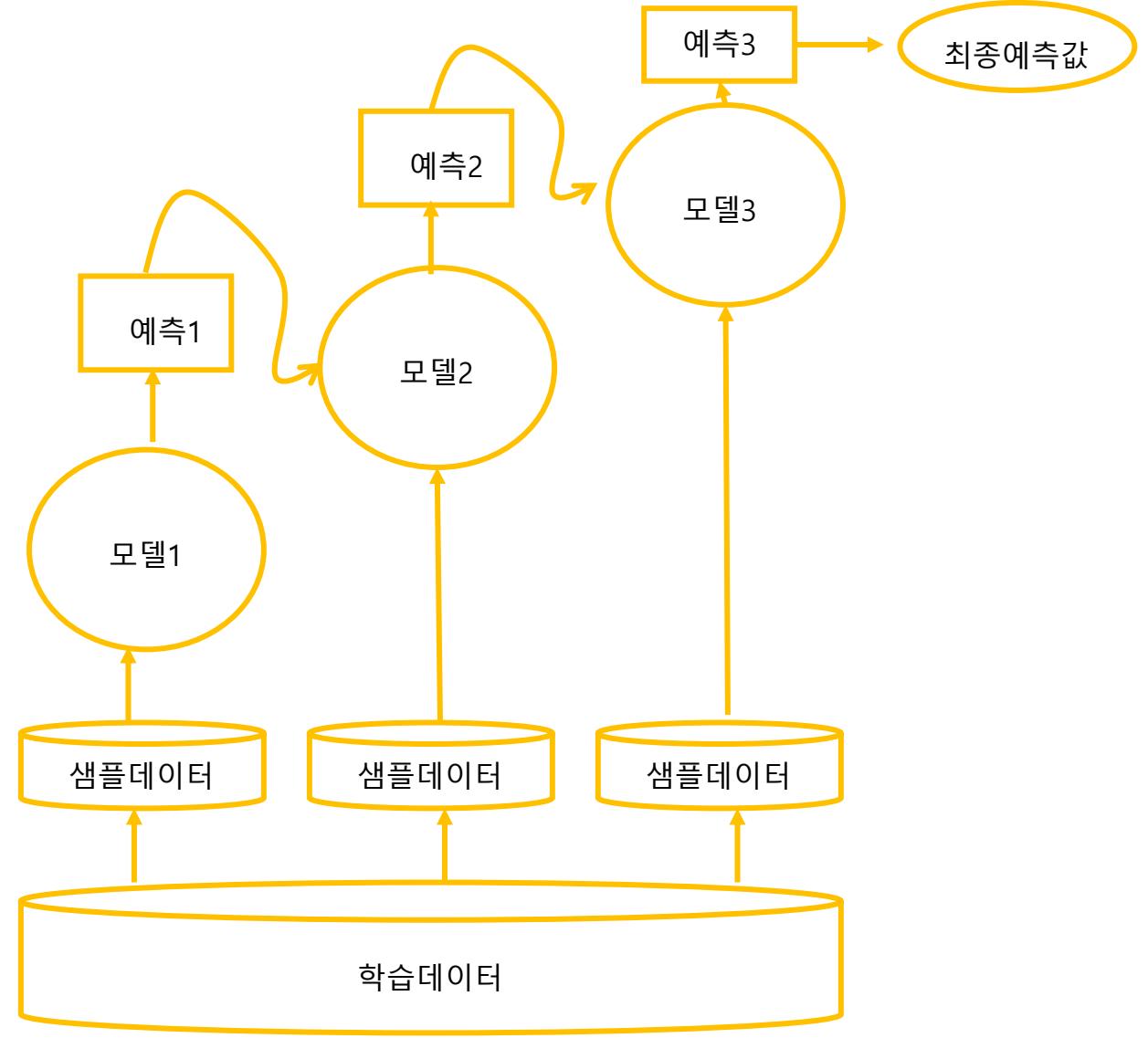
그림 7-7 샘플의 가중치를 업데이트하면서 순차적으로 학습하는 에이다부스트

※ 그림 출처: 핸즈온 머신러닝, p258, 한빛미디어

# Bagging과 Boosting 차이



<배깅(Bagging)>



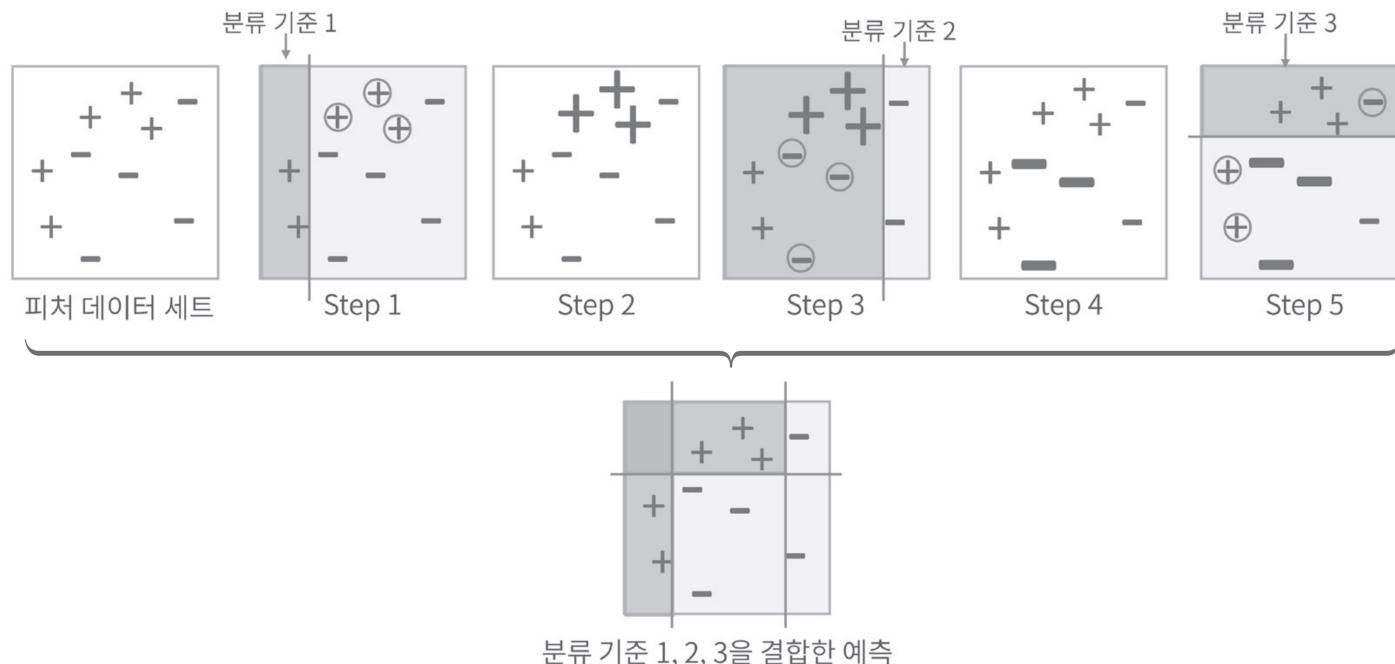
<부스팅(Boosting)>

# AdaBoosting

이전 모델의 오류 데이터에 가중치를 부여하여 오류를 개선해 나가면서 학습

아주 얇은 결정 트리 기반(max\_depth=1, 부모노드 1, 자식노드 1)

이전 모델이 학습하고 평가된 후에 다음 예측기 학습이 가능하기 때문에 병렬화할 수 없어 확장성이 높지 않고 느린다.



Step1 ~ Step5

- 약한 학습기가 순차적으로 오류값에 대해 가중치를 부여한 후 예측 결정 기준을 모두 결합해 예측을 수행

- 마지막 예측에서는 개별 약한 학습기보다 정확도가 훨씬 높아짐

<이미지 출처: 파이썬 머신러닝 완벽가이드, 위키북스, 2021, p217>

# [실습] AdaBoosting 타이타닉 데이터셋

## 1. 데이터 준비

```
# AdaBoosting

df = pd.read_csv('../data/df_titanic.csv')
# feature_columns에서 괄호 빼지면 안됨
feature_columns = (df.columns.difference(['Survived']))

X = df[feature_columns]
y = df['Survived']

# 테스트 분할 : test_size=0.2, random_state=42

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape)
```

# [실습] AdaBoosting 타이타닉 데이터셋

## 2. 기본 모델 구축

```
from sklearn.ensemble import AdaBoostClassifier  
  
dt = DecisionTreeClassifier(random_state=42, max_depth=1)  
  
ada_clf = AdaBoostClassifier(dt, n_estimators=200, #  
                             algorithm='SAMME.R', learning_rate=0.5)  
ada_clf.fit(X_train, y_train)  
  
train_score = accuracy_score(y_train, ada_clf.predict(X_train))  
test_score = accuracy_score(y_test, ada_clf.predict(X_test))  
print(train_score)  
print(test_score)
```

## [실습] AdaBoosting 타이타닉 데이터셋

---

- cross\_validate( ) 함수를 사용하여 파라미터 튜닝 없이 Adaboost 학습
- 하이퍼 파라미터 설정

**base\_estimator:** 예측할 모델, 디폴트는 None(Decision Tree Classifier)

**n\_estimators:** 모형(week learner)의 개수, 순차적으로 오류를 보정해 수가 많으면 성능이 일정 수준까지 높아질 수 있으나 수행 시간이 오래 걸린다는 단점이 있음(디폴트는 50)

**learning\_rate:** 학습률, 0~1 사이의 값을 지정. 너무 작은 값인 경우 최소점을 찾아 예측 성능이 높지만 학습에 오래 걸리고 너무 큰 값인 경우 최소점을 찾지 못해 예측 성능이 떨어질 확률이 높아서 n\_estimators와 상호 호환 필요(디폴트는 1.0)

**AdaBoost - GridSearchCV를 활용한 최적의 파라미터 찾기**

# [실습] AdaBoosting 타이타닉 데이터셋

## 3. 모델 성능 개선 – 하이퍼 파라미터 튜닝

```
ada_param = {
    'n_estimators': grid_n_estimator,
    'learning_rate': grid_learn,
    'random_state': grid_seed
}

# n_jobs = -1 학습시 모든 CPU 코어 사용
# AdaBoosting에 기본 추정기: 결정 트리(max_depth=1)
grid_ada = GridSearchCV(AdaBoostClassifier(), param_grid=ada_param,#
                        return_train_score=True, n_jobs = -1, cv=5,#
                        scoring='accuracy')
grid_ada.fit(X, y)

GridSearchCV(cv=5, estimator=AdaBoostClassifier(), n_jobs=-1,
            param_grid={'learning_rate': [0.0001, 0.001, 0.01, 0.03, 0.05, 0.1,
                                         0.2, 0.25, 0.5, 1.0],
                        'n_estimators': [10, 50, 100, 300, 500],
                        'random_state': [0]},#
            return_train_score=True, scoring='accuracy')
```

# [실습] AdaBoosting 타이타닉 데이터셋

## 3. 모델 성능 개선 – 하이퍼 파라미터 튜닝

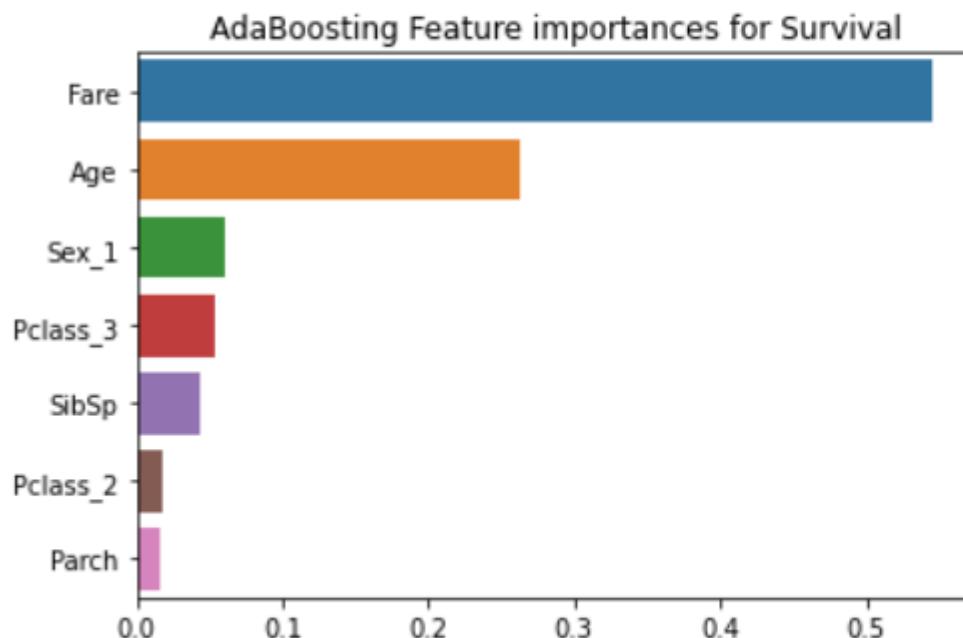
```
df_grid_ada = pd.DataFrame(grid_ada.cv_results_)
df_grid_ada[['param_learning_rate', 'param_n_estimators', '#',
             'params', 'mean_train_score',
             'mean_test_score', 'rank_test_score']].sort_values(['rank_test_score']).head(5)
```

	param_learning_rate	param_n_estimators	params	mean_train_score	mean_test_score	rank_test_score
39	0.25	500	{'learning_rate': 0.25, 'n_estimators': 500, '...}	0.872971	0.855369	1
47	1.0	100	{'learning_rate': 1.0, 'n_estimators': 100, 'r...}	0.873932	0.853446	2
43	0.5	300	{'learning_rate': 0.5, 'n_estimators': 300, 'r...}	0.874889	0.853441	3
26	0.1	50	{'learning_rate': 0.1, 'n_estimators': 50, 'ra...}	0.851402	0.850635	4
22	0.05	100	{'learning_rate': 0.05, 'n_estimators': 100, '...}	0.851162	0.850635	4

# [실습] AdaBoosting 타이타닉 데이터셋

## 4. 특성 중요도

```
# 가장 중요한 변수  
feature_names = list(X.columns)  
f_importance = pd.Series(best_grid_ada.feature_importances_,  
                         index=feature_names).sort_values(ascending=False)  
plt.title("AdaBoosting Feature importances for Survival")  
sns.barplot(x=f_importance.values, y=f_importance.index);
```



# Gradient Boosting(GBM)

---

에이다부스트와 유사: 양상블에 이전까지의 오차를 보정하면서 예측기를 순차적으로 추가  
다른 점: 반복 시 샘플의 가중치를 추가하는 대신 이전 예측기가 만든 잔여 오차에 새로운 예측기를 학습 시  
킴. 가중치 업데이트를 경사 하강법으로 이용

오류값: 실제값 – 예측값 (잔여오차)

$y$  : 분류의 실제 결과값

$X$  :  $x_1, x_2, x_3, \dots, x_n$

$F(x)$ : 피처에 기반한 예측함수

오류식  $H(x) = y - F(x)$

이 오류식이 최소화되도록 방향성을 가지고 반복적으로 가중치 값을 업데이트하는 방식

일반적으로 GBM이 랜덤 포레스트보다 예측 성능이 뛰어난 경우가 많지만, 수행 시간이 오래 걸리고 하이퍼파라미터 튜닝 노력도 더 필요함

Gradient Boosting Classifier

# Gradient Boosting(GBM)

```
# 동작원리
```

```
from sklearn.tree import DecisionTreeRegressor

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

y2 = y - tree_reg1.predict(X)

tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)

y3 = y2 - tree_reg2.predict(X)

tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)

# 이렇게 세 개의 트리를 포함하는 양상을 구성
# 양상을 모델의 예측: 모든 트리의 예측을 더함
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

GradientBoosting 분류기에서 각 트리가 훈련할 때 사용하는 샘플의 비율을 지정할 수 있는데(subsample), 이 때 지정된 비율에 따라 무작위로 훈련 샘플을 선택하도록 발전시킨 알고리즘이 **확률적 그래디언트 부스팅(Stochastic Gradient Boosting)** 기법이다.

# [실습] Gradient Boosting – 타이타닉

---

1. 데이터 준비
2. 기본 모델 구축 및 평가
3. 성능 개선을 위한 하이퍼파라미터 튜닝
4. 특성 중요도 파악

# [실습] Gradient Boosting – 와인

---

1. 데이터 준비
2. 기본 모델 구축 및 평가
3. 성능 개선을 위한 하이퍼파라미터 튜닝
4. 특성 중요도 파악



XGBoost

# XGBoost(eXtra Gradient Boost)

---

트리 기반 앙상블 학습에서 가장 각광받고 있는 알고리즘

분류에 있어서 다른 머신 러닝보다 뛰어난 예측 성능을 나타냄

GBM에 기반하고 있지만, GBM의 단점인 느린 수행 시간 및 과적합 규제 방법의 부재 등의 문제 해결

병렬 CPU 환경에서 병렬 학습이 가능하여 기존 GBM보다 빠르게 학습 완료

항목	설명
뛰어난 예측 성능	일반적으로 분류와 회귀 영역에서 뛰어난 예측 성능을 발휘합니다.
GBM 대비	일반적인 GBM은 순차적으로 Weak learner가 가중치를 증감하는 방법으로 학습하기 때문에 전반적으로 속도가 느립니다. 하지만 XGBoost는 병렬 수행 및 다양한 기능으로 GBM에 비해 빠른 수행
빠른 수행 시간	성능을 보장합니다. 아쉽게도 XGBoost가 일반적인 GBM에 비해 수행 시간이 빠르다는 것이지, 다른 머신러닝 알고리즘(예를 들어 랜덤 포레스트)에 비해서 빠르다는 의미는 아닙니다.

# XGBoost(eXtra Gradient Boost)

---

과적합 규제 (Regularization)	표준 GBM의 경우 과적합 규제 기능이 없으나 XGBoost는 자체에 과적합 규제 기능으로 과적합에 좀 더 강한 내구성을 가질 수 있습니다.
Tree pruning (나무 가지치기)	일반적으로 GBM은 분할 시 부정 손실이 발생하면 분할을 더 이상 수행하지 않지만, 이러한 방식도 자칫 지나치게 많은 분할을 발생할 수 있습니다. 다른 GBM과 마찬가지로 XGBoost도 <code>max_depth</code> 파라미터로 분할 깊이를 조정하기도 하지만, tree pruning으로 더 이상 긍정 이득이 없는 분할을 가지치기 해서 분할 수를 더 줄이는 추가적인 장점을 가지고 있습니다.
자체 내장된 교차 검증	XGBoost는 반복 수행 시마다 내부적으로 학습 데이터 세트와 평가 데이터 세트에 대한 교차 검증을 수행해 최적화된 반복 수행 횟수를 가질 수 있습니다.
결손값 자체 처리	지정된 반복 횟수가 아니라 교차 검증을 통해 평가 데이터 세트의 평가 값이 최적화 되면 반복을 중간에 멈출 수 있는 조기 중단 기능이 있습니다.
	XGBoost는 결손값을 자체 처리할 수 있는 기능을 가지고 있습니다.

# XGBoost 파라미터 : Booster parameters

파라미터명 [기본값]	설명
<b>learning_rate (eta) [0.3]</b>	<ul style="list-style-type: none"><li>- learning rate와 동일. XGBClassifier를 이용할 경우, learning_rate 인자로 사용.</li><li>- 각 스텝 별로 weight를 감소시키는 정도. 클수록 빨리 감소시킴.</li><li>- 0과 1 사이의 값 지정 가능. 일반적으로 [0.01, 0.2] 정도로 설정</li></ul>
<b>min_child_weight [default=1]</b>	<ul style="list-style-type: none"><li>- Child의 weights의 최소 합</li><li>- 더 높은 값을 설정할수록, overfitting을 방지함.</li><li>- 하지만 너무 높은 값으로 설정하면 under-fitting 발생</li></ul>
<b>max_depth [default=6]</b>	트리의 최대 Depth Typical values: [3, 10]
<b>min_split_loss(gamma) [default=0]</b>	<ul style="list-style-type: none"><li>- 트리의 리프 노드를 추가적으로 확장(분할)할 것인지 결정할 최소 손실 감소 값.</li><li>- 해당 값보다 큰 손실(loss)이 감소된 경우, 리프 노드 분리</li><li>- 값이 클수록 과적합 방지.</li></ul>
<b>sub_sample(subsample) [default=1]</b>	<ul style="list-style-type: none"><li>- 트리 확장 시, 고려되는 샘플의 비율</li><li>- Overfitting 발생 시, 이 값을 낮추면 overfitting을 방지할 수 있다.</li><li>- 너무 낮은 값은 under fitting 유발할 가능성이 있음.</li></ul>
<b>colsample_bytree[default=1]</b>	<ul style="list-style-type: none"><li>- Max_features와 동일</li><li>- 보통 0.5~1 설정</li></ul>
<b>reg_lambda(lambda) [default=1]</b>	<ul style="list-style-type: none"><li>- L2 regularization 적용값</li><li>- 피처 개수가 많을 경우 검토. 자주 사용되진 않음.</li><li>- 과적합 감소 효과</li></ul>
<b>reg_alpha(alpha) [default=0]</b>	<ul style="list-style-type: none"><li>- L1 regularization 적용값</li><li>- 피처 개수가 많을 경우 검토.</li><li>- 과적합 감소 효과</li></ul>

# XGBoost 파라미터 : 학습 parameters

---

파라미터명	설명
<b>objective</b> [default=reg:squarederror]	손실 함수(Loss function)을 정의. 주로 이진/다중 분류 여부에 따라 선택. <ul style="list-style-type: none"><li>- <b>reg:squarederror</b>: regression with squared loss</li><li>- <b>binary:logistic</b> –logistic regression for binary classification, returns predicted probability (not class)</li><li>- <b>multi:softmax</b> –multiclass classification using the softmax objective, returns predicted class (not probabilities)</li><li>- <b>multi:softprob</b> –same as softmax, but returns predicted probability of each data point belonging to each class.</li><li>- ....</li></ul>
<b>eval_metric</b> [ default: rmse for regression error for classification ]	검증에 사용되는 함수를 정의. <ul style="list-style-type: none"><li>- <b>rmse</b>: root mean square error</li><li>- <b>mae</b>: mean absolute error</li><li>- <b>logloss</b>: negative log-likelihood</li><li>- <b>error</b>: Binary classification error rate (0.5 threshold)</li><li>- <b>merror</b>: Multiclass classification error rate</li><li>- <b>mlogloss</b>: Multiclass logloss</li><li>- <b>auc</b>: Area under the curve</li><li>- ....</li></ul>
<b>seed</b> [default=0]	랜덤 값 생성을 위한 Seed값

## [실습] XGBoost : 위스콘신 유방암 데이터셋

---

```
import xgboost as xgb
from xgboost import plot_importance
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

b_cancer = load_breast_cancer()
X = b_cancer.data
y = b_cancer.target

cancer_df = pd.DataFrame(data=X, columns=b_cancer.feature_names)
cancer_df['target'] = y
cancer_df.head(3)
```

## [실습] XGBoost : 위스콘신 유방암 데이터셋

```
print(b_cancer.target_names)
print(cancer_df['target'].value_counts())

# 전체 데이터 중 80%는 학습용 데이터, 20%는 테스트용 데이터 추출
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, random_state=156)
print(X_train.shape, X_test.shape)
```

```
# 사이킷런 래퍼 XGBoost 클래스인 XGBClassifier 임포트
# gpu 사용 옵션 tree_method
from xgboost import XGBClassifier

xgb = XGBClassifier(n_estimators=400, learning_rate=0.1,#
                     max_depth=3, tree_method='gpu_hist')

# 초기 중단옵션을 사용하지 않은 경우
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
y_pred_proba = xgb.predict_proba(X_test)[:, 1]
print(y_pred)
print(y_pred_proba)
```

## [실습] XGBoost : 위스콘신 유방암 데이터셋

```
confusion = confusion_matrix( y_test, pred)
accuracy = accuracy_score(y_test , pred)
precision = precision_score(y_test , pred)
recall = recall_score(y_test , pred)
f1 = f1_score(y_test,pred)

roc_auc = roc_auc_score(y_test, pred_proba)
print('오차 행렬')
print(confusion)

print('정확도: {:.4f}, 정밀도: {:.4f}, 재현율: {:.4f}, F1: {:.4f}, AUC:{:.4f}'.format(accuracy, precision, recall, f1, roc_auc))
```

# XGBoost의 과적합 개선 팁

---

- \* 조기 중단 - 수행속도를 향상시키기 위한 XGBoost 기능, fit()에 사용

```
# 조기 중단 옵션을 사용한 경우
evals = [(X_test, y_test)]
xgb_wrapper.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="logloss",
                 eval_set=evals, verbose=True)
```

```
# early_stopping_rounds를 10으로 설정하고 재 학습.
xgb_wrapper.fit(X_train, y_train, early_stopping_rounds=10,
                 eval_metric="logloss", eval_set=evals, verbose=True)
|
```

# XGBoost의 과적합 개선 팁

---

과적합을 방지하는 방법은 학습 모델을 단순화하는 방향으로 하이퍼 파라미터 튜닝

learning\_rate은 낮추고 n\_estimators는 높임

max\_depth 낮춤

min\_child\_weight: 높임

min\_split\_loss : 높임

sub\_samples : 낮춤

colsample\_bytree: 낮춤

# XGBoost 시각화

```
# 시각화  
# f1스코어를 기반으로 피처 중요도 계산  
  
from xgboost import plot_importance  
import matplotlib.pyplot as plt  
%matplotlib inline  
  
fig, ax = plt.subplots(figsize=(10, 12))  
plot_importance(xgb_wrapper, ax=ax)
```

```
# feature_importances_ 사용하는 경우  
f_imp = pd.Series(xgb_wrapper.feature_importances_,  
                   index=b_cancer.feature_names).sort_values(ascending=False)  
  
import seaborn as sns  
plt.figure(figsize=(10, 12))  
sns.barplot(x=f_imp.values, y=f_imp.index);
```

# XGBoost에서 GPU 사용하기

---

구글 코랩 – Runtime 옵션 변경

stand alone PC:

**XGBoost (tree\_method='gpu\_hist')**

참조: <https://www.kaggle.com/general/236940>

# [실습] XGBoost

---

1. 데이터 준비
2. 기본 모델 구축 및 평가
3. 성능 개선을 위한 하이퍼파라미터 튜닝
4. 특성 중요도 파악



LightGBM

# LightGBM

---

장점:

XGBoost보다 학습에 걸리는 시간이 짧음.

예측 성능은 비슷, 더 작은 메모리 사용량

보다 다양한 기능 제공

단점:

데이터 세트가 적은 경우(10000) 과적합이 발생하기 쉬움

작동방식:

일반 GBM방법

균형트리분할(Level Wise) 방식, 트리 깊이 최소화, 과적합 방지, 균형을 맞추기 위한 시간 필요

LightGBM

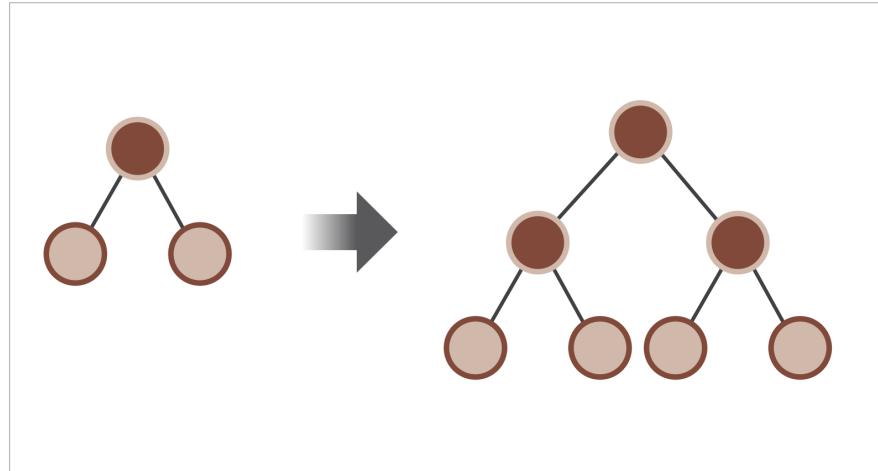
리프 중심 트리 분할(Leaf Wise) 방식: 트리의 균형을 맞추지 않고, 최대 손실값(max delta loss)을 가지는 리프 노드를 지속적으로 분할하면서 트리의 깊이가 깊어지고 비대칭적인 규칙 트리를 생성.

최대 손실값을 가지는 리프 노드를 지속적으로 분할하여 생성된 규칙 트리는 학습을 반복할수록 결국 균형 트리 분할방식보다 예측 오류 손실을 최소화할 수 있음.

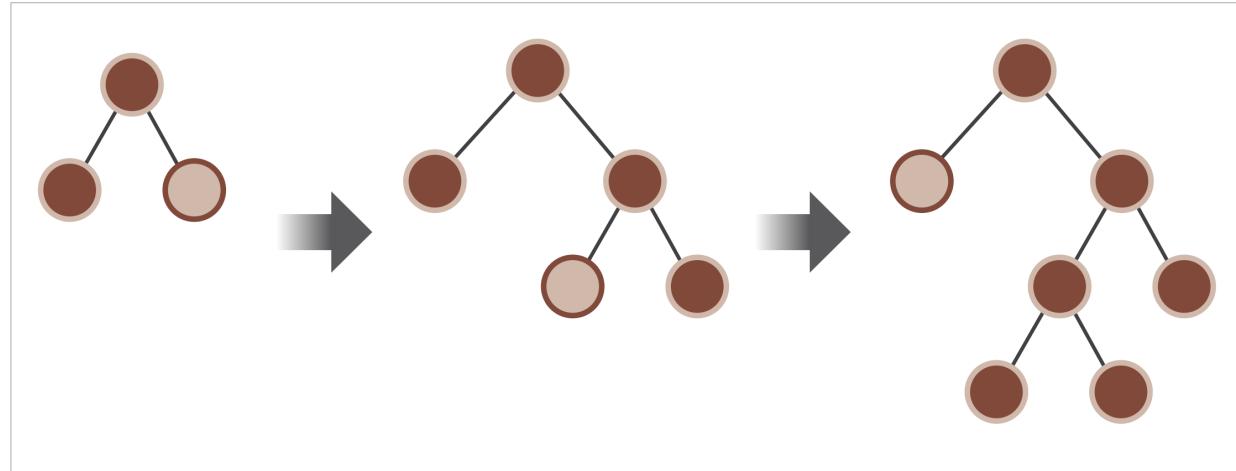
# LightGBM

---

균형 트리 분할(Level Wise)



리프 중심 트리 분할(Leaf Wise)



## [실습] LGBM – 위스콘신 유방암 데이터셋

```
#!pip install lightgbm  
# LightGBM을 윈도우에 설치할 경우에는 Visual Studio Build tool 2015이상이 먼저 설치되어야 함
```

```
# LightGBM의 파이썬 패키지인 lightgbm에서 LGBMClassifier 임포트  
from lightgbm import LGBMClassifier  
  
import pandas as pd  
import numpy as np  
from sklearn.datasets import load_breast_cancer  
from sklearn.model_selection import train_test_split  
import warnings  
warnings.filterwarnings('ignore')  
  
dataset = load_breast_cancer()  
ftr = dataset.data  
target = dataset.target  
  
# 전체 데이터 중 80%는 학습용 데이터, 20%는 테스트용 데이터 추출  
X_train, X_test, y_train, y_test = train_test_split(ftr, target, test_size=0.2, random_state=156)
```

## [실습] LGBM – 위스콘신 유방암 데이터셋

---

```
# 앞서 XGBoost와 동일하게 n_estimators는 400 설정.  
lgbm = LGBMClassifier(n_estimators=400, device="gpu")  
  
# LightGBM도 XGBoost와 동일하게 조기 중단 수행 가능.  
evals = [(X_test, y_test)]  
lgbm_wrapper.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="logloss",  
                  eval_set=evals, verbose=True)  
y_pred = lgbm.predict(X_test)  
y_pred_proba = lgbm.predict_proba(X_test)[:, 1]
```

## [실습] LGBM – 위스콘신 유방암 데이터셋

---

```
# plot_importance( )를 이용하여 feature 중요도 시각화
from lightgbm import plot_importance
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax = plt.subplots(figsize=(10, 12))
plot_importance(lgbm_wrapper, ax=ax)
```

# LightGBM에서 GPU 사용하기

---

구글 코랩 – Runtime 옵션 변경

stand alone PC:

```
LGBMClassifier (device='gpu_hist')
```

참조: <https://www.kaggle.com/general/236940>



CATBoost

# CATBoost

---