5th Meet

□ 오늘 공부할 것은

04

분류

01. 분류(Classification)의 개요 181

02. 결정 트리 183

결정 트리 모델의 특징 185

결정 트리 파라미터 186

결정 트리 모델의 시각화 187

결정 트리 과적합(Overfitting) 198

결정 트리 실습 - 사용자 행동 인식 데이터 세트 200

03. 앙상블 학습 210

앙상블 학습 개요 210

보팅 유형 - 하드 보팅(Hard Voting)과 소프트 보팅(Soft Voting) 212

보팅 분류기(Voting Classifier) 213

04. 랜덤 포레스트 216

랜덤 포레스트의 개요 및 실습 216

랜덤 포레스트 하이퍼 파라미터 및 튜닝 218

GBM의 개요 및 실습 221

05. GBM(Gradient Boosting Machine) 221

GBM 하이퍼 파라미터 소개 224

XGBoost 개요 225

06. XGBoost(eXtra Gradient Boost) 225

XGBoost 설치하기 227

파이썬 래퍼 XGBoost 하이퍼 파라미터 228

파이썬 래퍼 XGBoost 적용 - 위스콘신 유방암 예측 232

사이킷런 래퍼 XGBoost의 개요 및 적용 240

Now, It's your turn !!!

40mīn ~ 50mīn

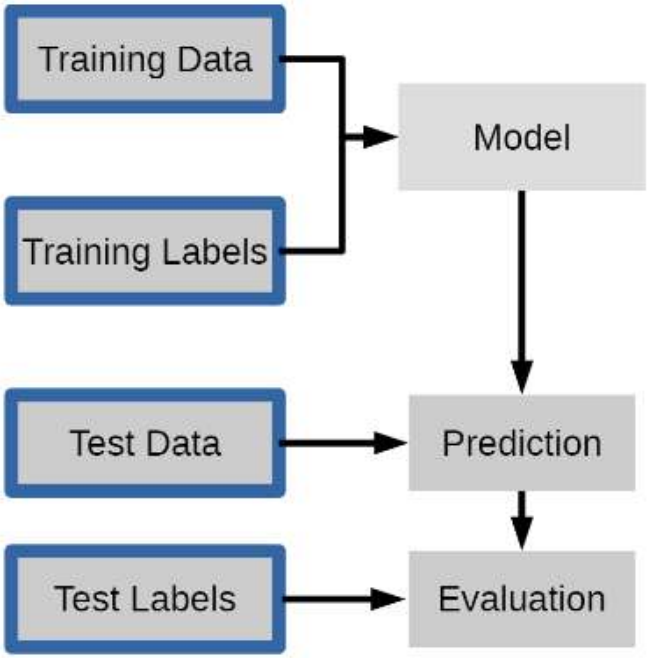
Chapter 04

분류 (Classification)

[04-01]
분류(Classification)의 개요

☐ Supervised Machine Learning

```
clf = RandomForestClassifier()

clf.fit(X_train, y_train)


y_pred = clf.predict(X_test)

clf.score(X_test, y_test)
```

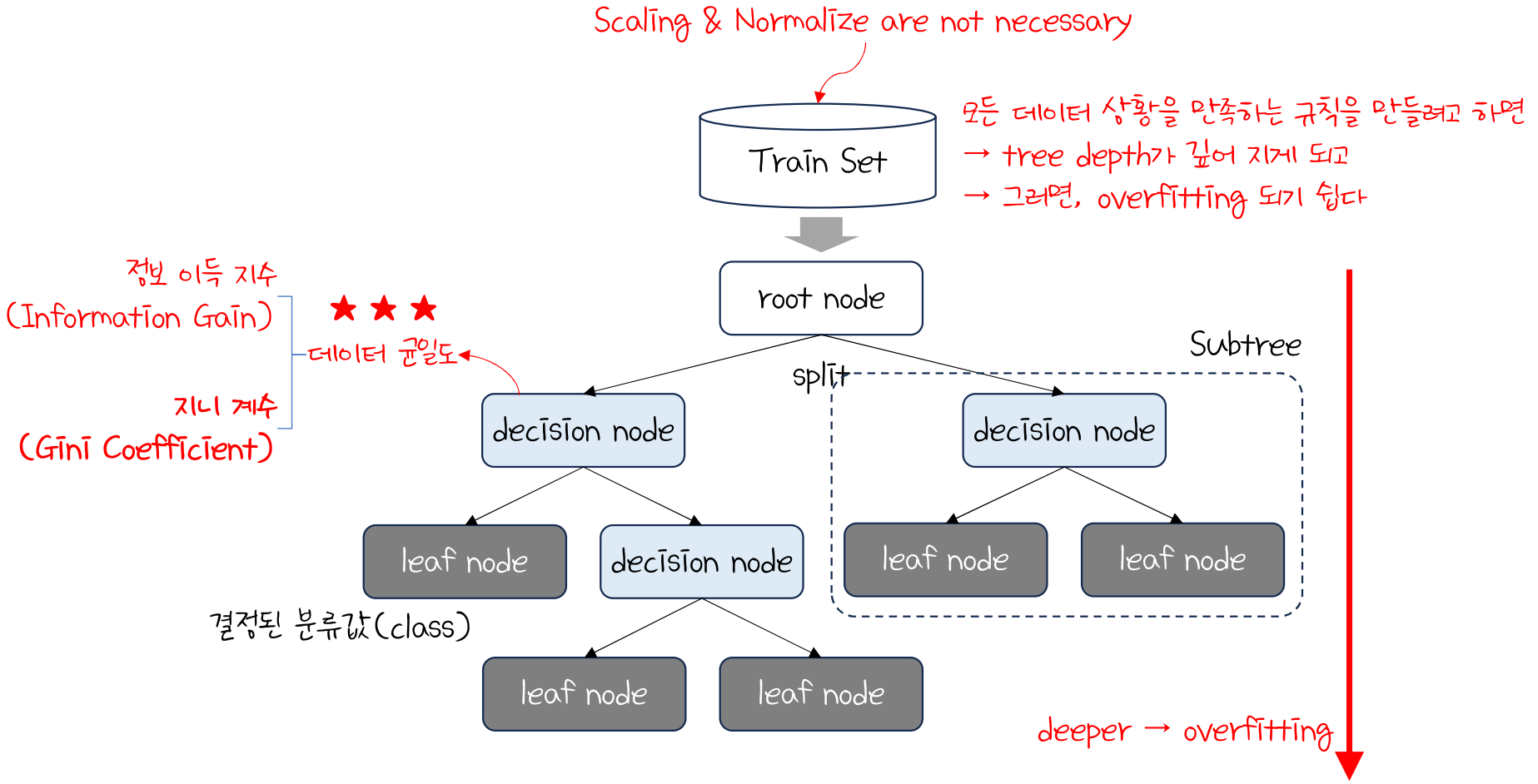


※ 출처 : https://github.com/amueller/odscon-2015

분류 (Classification)	나이브 베이즈 (Naïve Bayes)	베이즈(Bayes) 통계와 생성 모델 기반
	로지스틱 회귀 (Logistic Regression)	독립변수와 종속변수의 선형 관계성 기반
	결정 트리 (Decision Tree)	데이터 균일도에 따른 규칙 기반
	서포트 벡터 머신 (Support Vector Machine)	개별 클래스 간의 최대 분류 마진을 효과적으로 찾기
	최소 근접 알고리즘 (Nearest Neighbor)	근접 거리 기준
	신경망 (Neural Network)	심층 연결 기반
	앙상블 (Ensemble)	서로 다른(또는 같은) 머신러닝 알고리즘을 결합

앙상블 (Ensemble)	배깅 (Bagging)	**랜덤 포레스트 (Random Forest)**
	부스팅 (Boosting)	**그래디언트 부스팅 (Gradient Boosting Machine)**
		XGBoost (eXtra Gradient Boosting)
		LightGBM (Light Gradient Boosting Machine)
	스태킹 (Stacking)	

[04-02]
결정 트리 (Decision Tree)


Scaling & Normalize are not necessary
Train Set
모든 데이터 상황을 만족하는 규칙을 만들려고 하면
→ tree depth가 깊어 지게 되고
→ 그러면, overfitting 되기 쉽다
정보 이득 지수
(Information Gain)
지니 계수
(Gini Coefficient)
★ ★ ★
데이터 균일도
root node
Subtree
split
decision node
decision node
leaf node
decision node
leaf node
leaf node
결정된 분류값(class)
leaf node
leaf node
deeper → overfitting

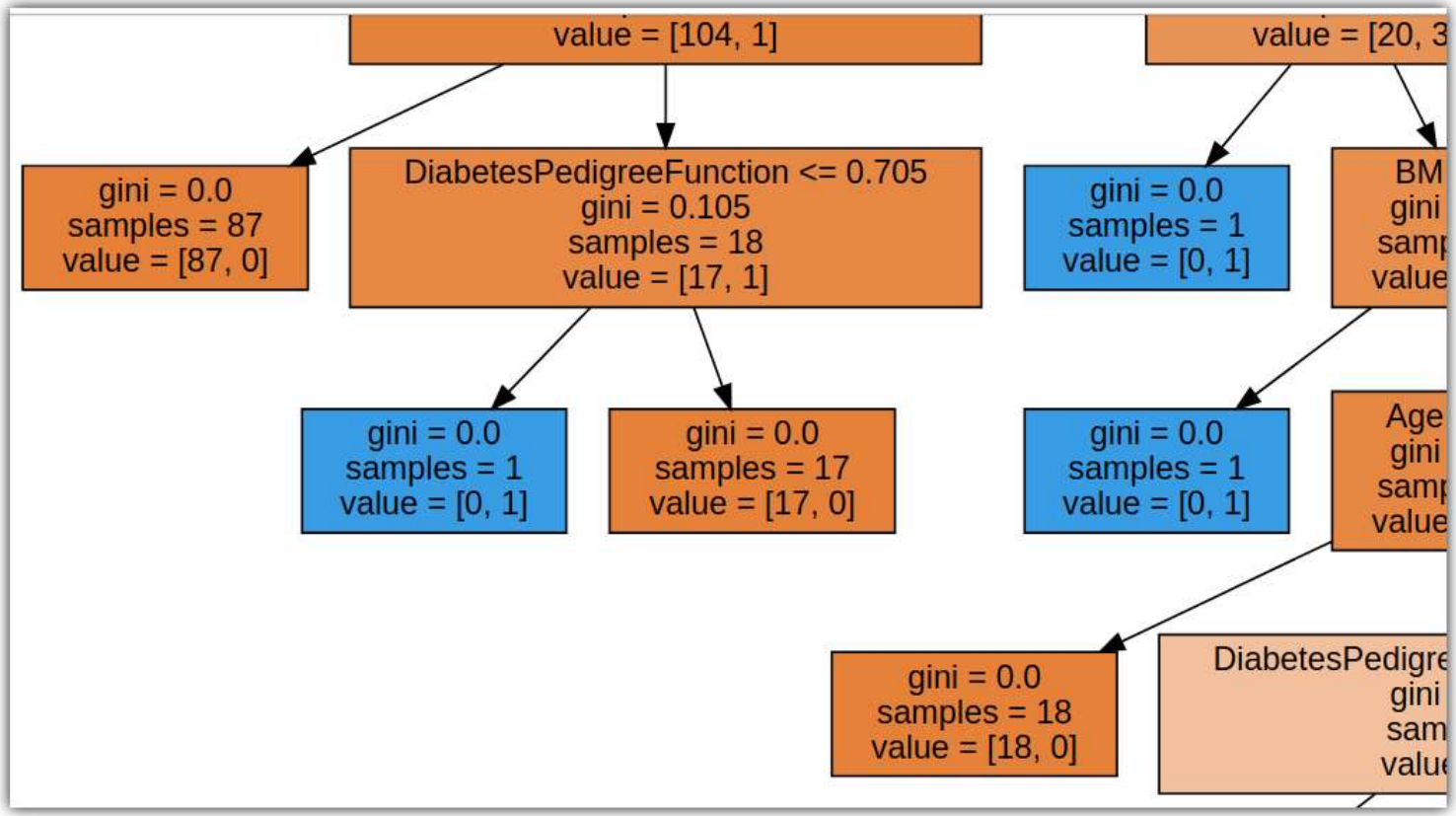
```
[1]: from sklearn.tree import DecisionTreeClassifier

[2]: DecisionTreeClassifier?
```

```
Init signature:
DecisionTreeClassifier(
    *,
    criterion='gini',
    splitter='best',
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features=None,
    random_state=None,
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    class_weight=None,
    ccp_alpha=0.0,
    monotonic_cst=None,
)
```

파라미터 명	설명
max_depth (Default = None) 트리의 최대 깊이	- None : 완벽하게 클래스 결정 값이 될 때까지 깊이를 계속 키우며 분할 - min_samples_split 설정대로 최대 분할하여 과적합 할 수 있음
min_samples_split (Default = 2) Split 최소 샘플 수	- 작게 설정할수록 분할되는 노드가 많아져서 과적합 가능성 증가
min_samples_leaf (Default = 1) leaf의 최소 샘플 수	- 왼쪽과 오른쪽의 브랜치 노드에서 가져야 할 최소한의 샘플 데이터 수 - 큰 값은 조건 만족 어려워 노드 분할을 상대적으로 덜 수행 함 - 과적합 제어 용도 - 비대칭적 데이터 경우 특정 클래스 데이터가 극도로 작을 수 있으므로 이 경우는 작게 설정 필요
max_features (Default = None) 최대 피처 개수	- None : 데이터 세트의 모든 피처를 사용해 분할 수행 - int형 : 대상 피처의 개수 - float형 : 전체 피처 중 대상 피처의 퍼센트 - auto / sqrt : 전체 피처 중 sqrt(전체 피처 개수)만큼 선정 - log : 전체 피처 중 log2(전체 피처 개수) 선정
max_leaf_nodes (Default = None)	- 말단 노드(leaf)의 최대 개수

☐ Graph

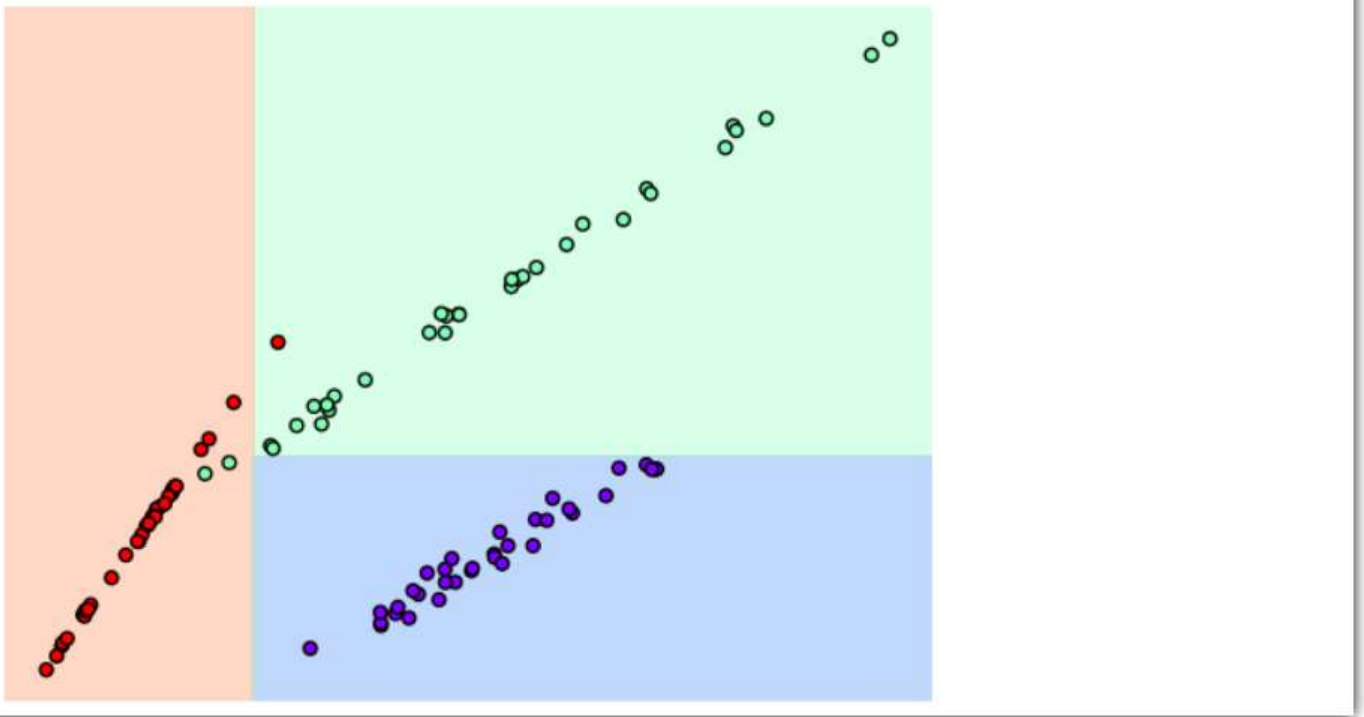
https://github.com/whatwant-school/python-ml/blob/main/05-week/05-week_02_graph.ipynb



☐ Overfitting

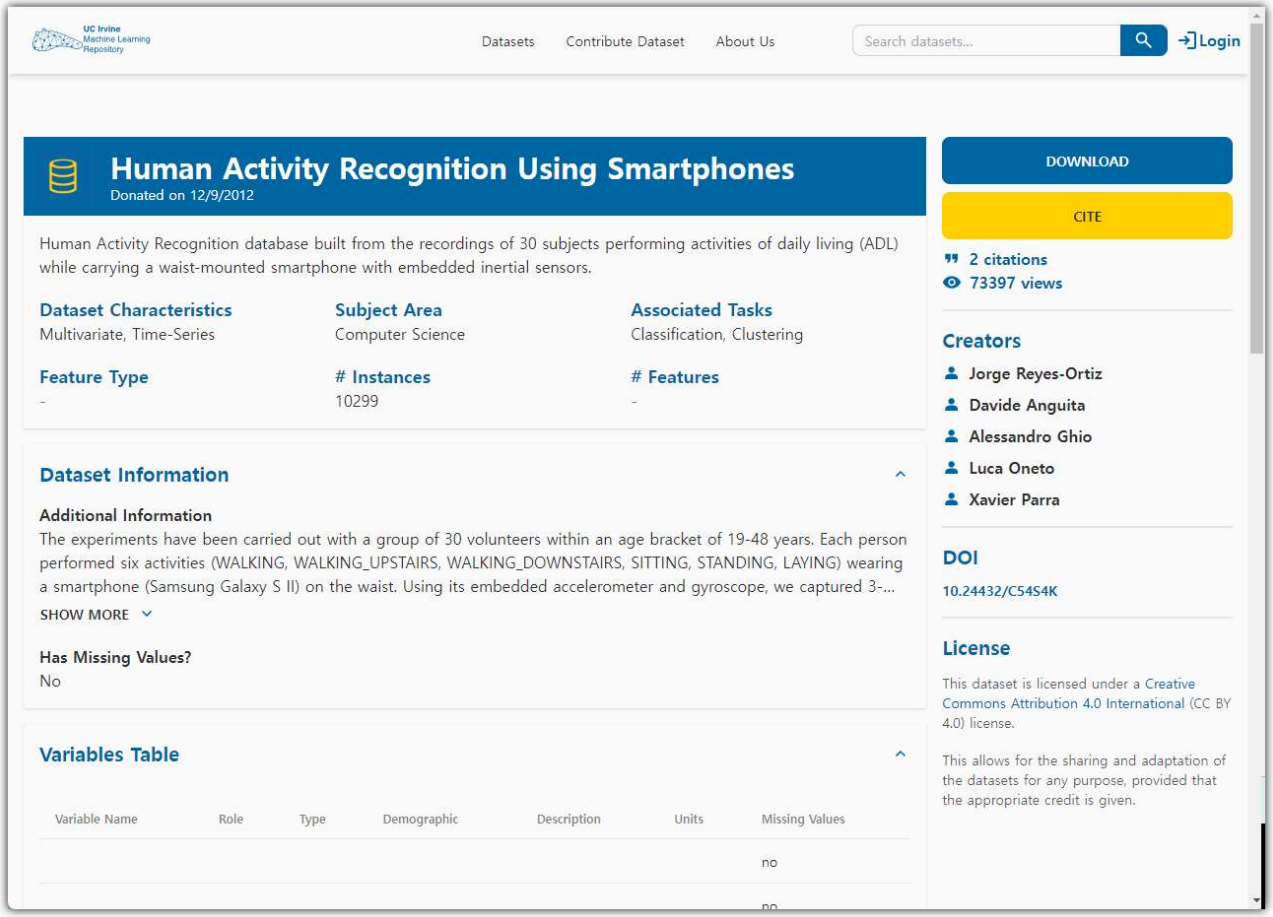
https://github.com/whatwant-school/python-ml/blob/main/05-week/05-week_03_overfit.ipynb

```
[7]: dt_clf = DecisionTreeClassifier(min_samples_leaf=6, random_state=156).fit(train_X, train_y)
     visualize_boundary(dt_clf, train_X, train_y)
```



☐ Human Activity Recognition Using Smartphones

https://archive.ics.uci.edu/dataset/240/human+activity+recognition+using+smartphones

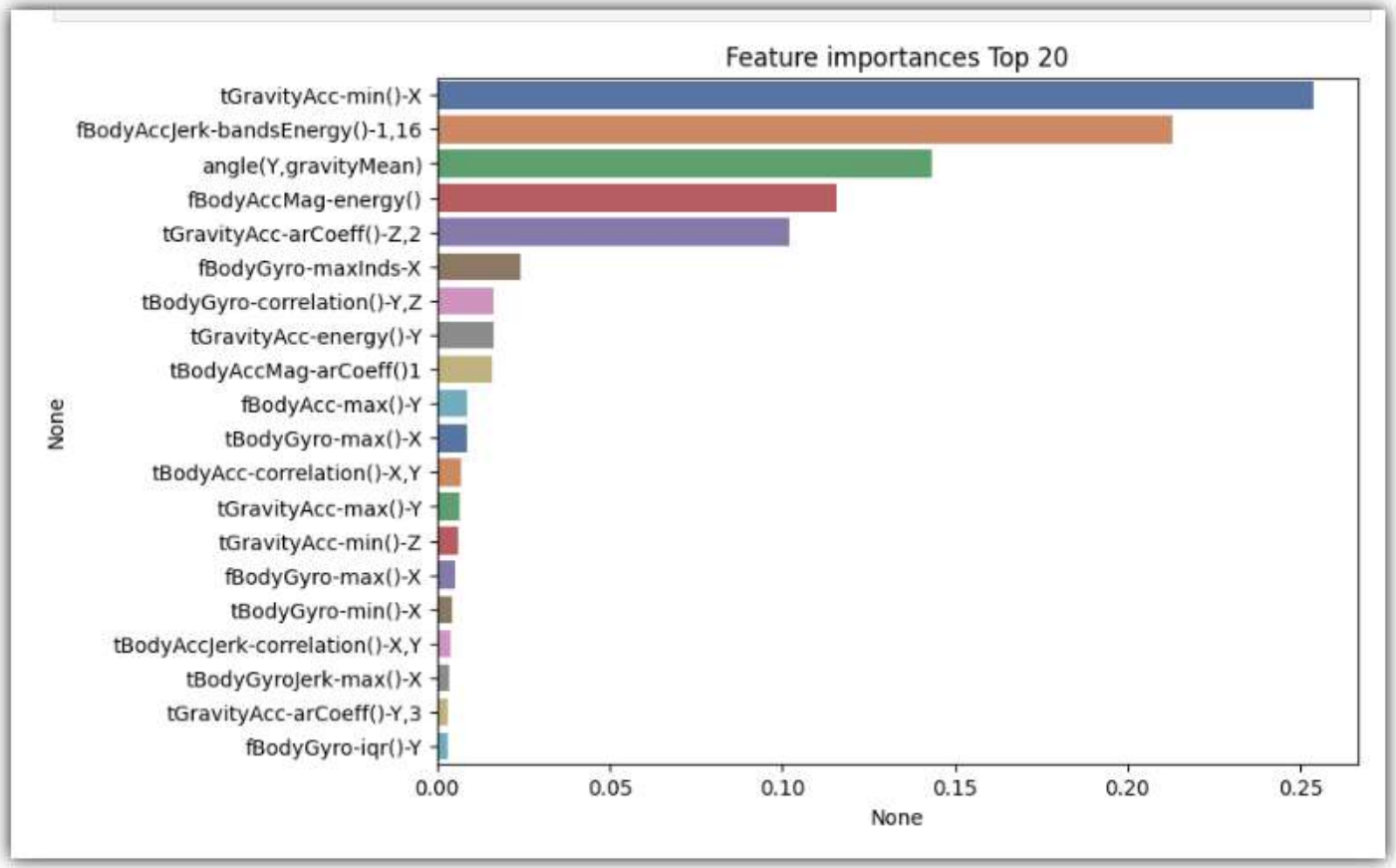


스마트폰을 이용한 사용자 행동 인식

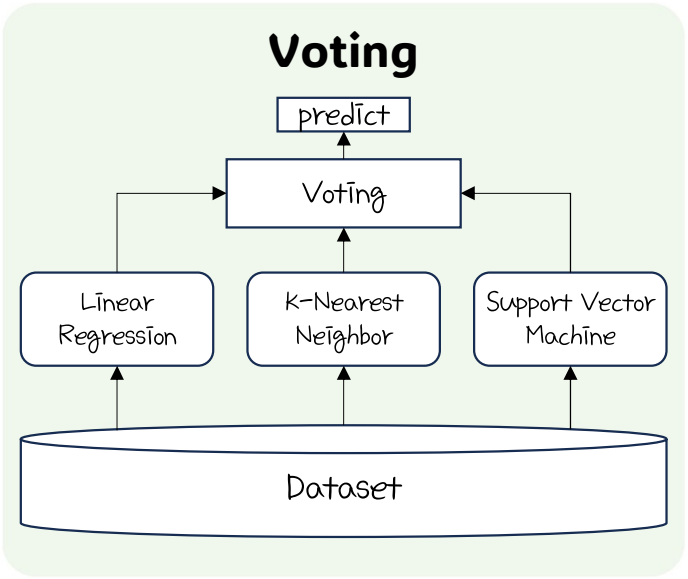
출처: https://sakshamchecker.medium.com/human-activity-recognition-7abaa9a1cf34

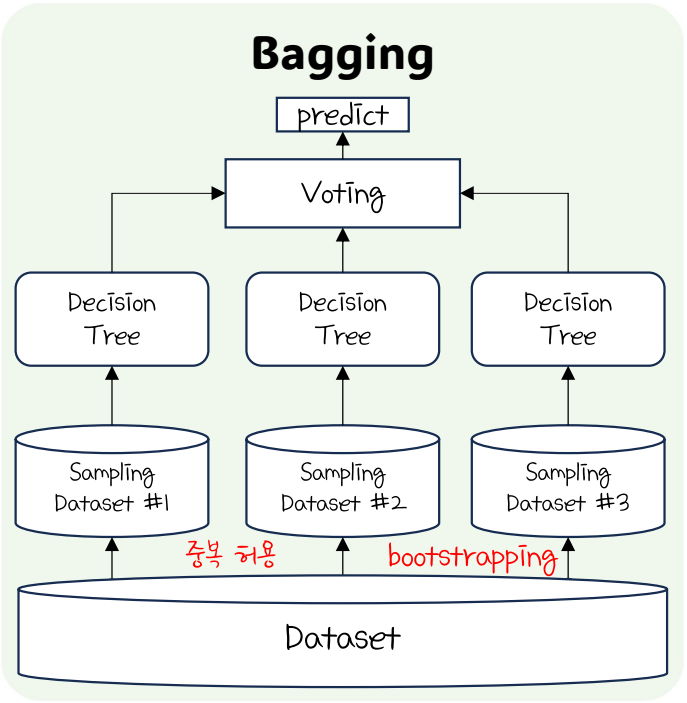
☐ Human Activity Recognition Using Smartphones

https://github.com/whatwant-school/python-ml/blob/main/05-week/05-week_04-HAR.ipynb

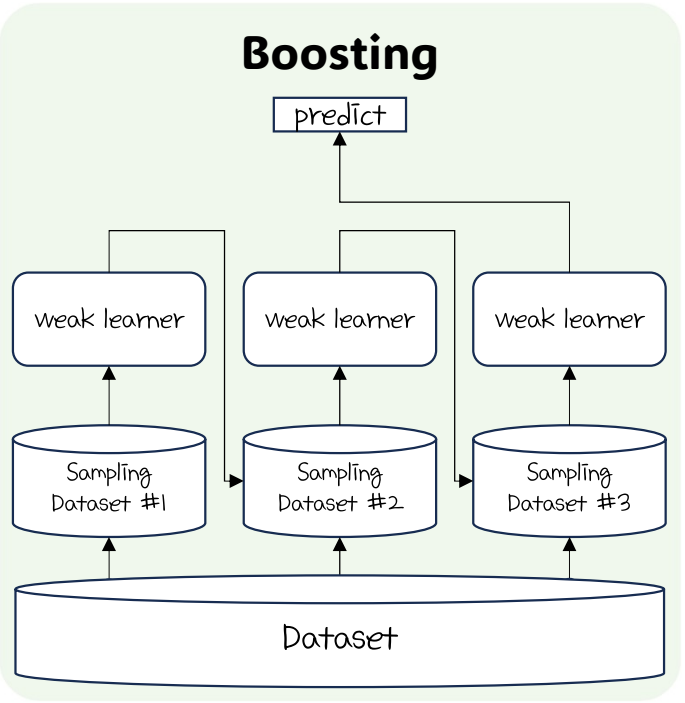


[04-03]
앙상블 학습 (Ensemble Learning)



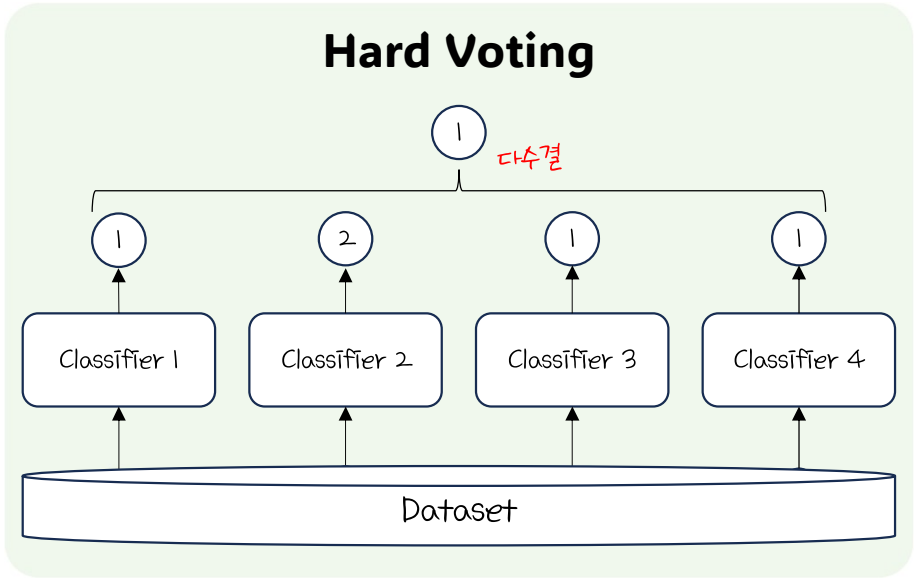


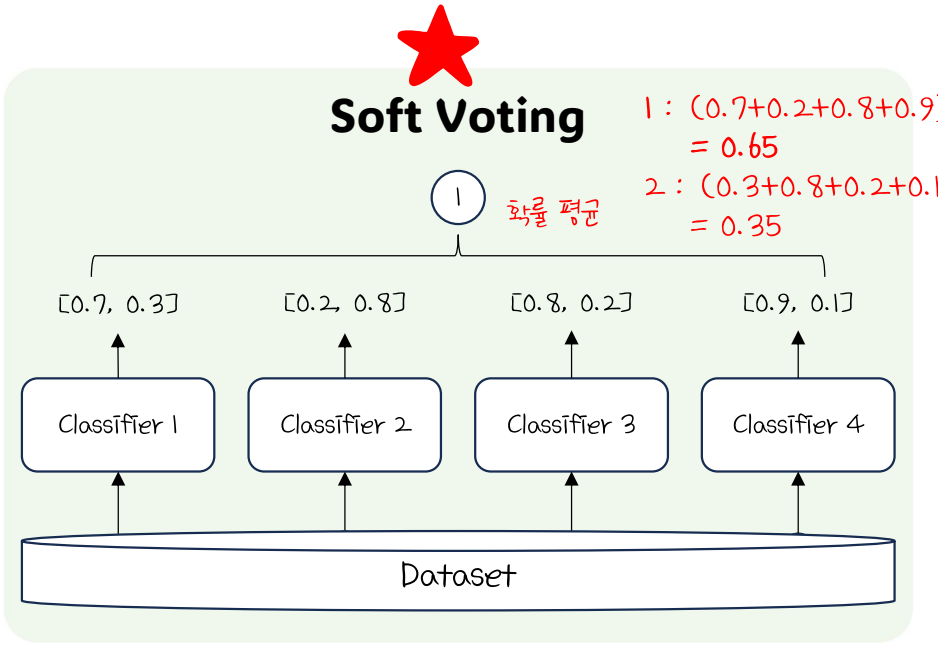
RandomForest



GBM, XGBoost, LightGBM

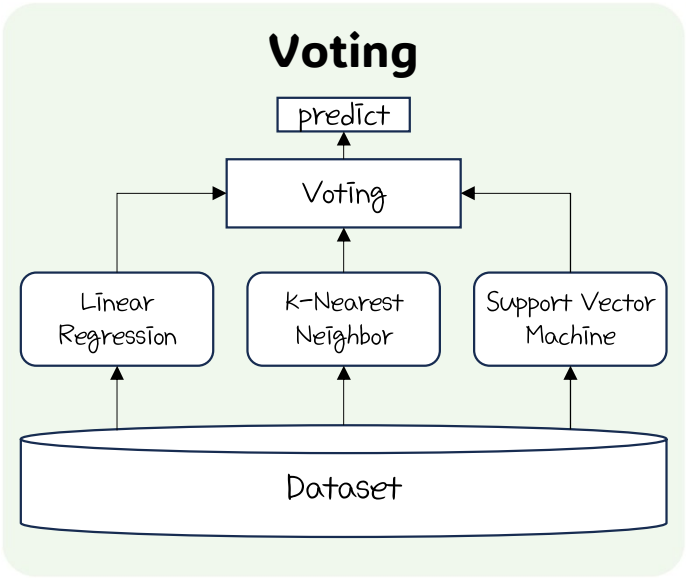
☐ Hard Voting vs. Soft Voting

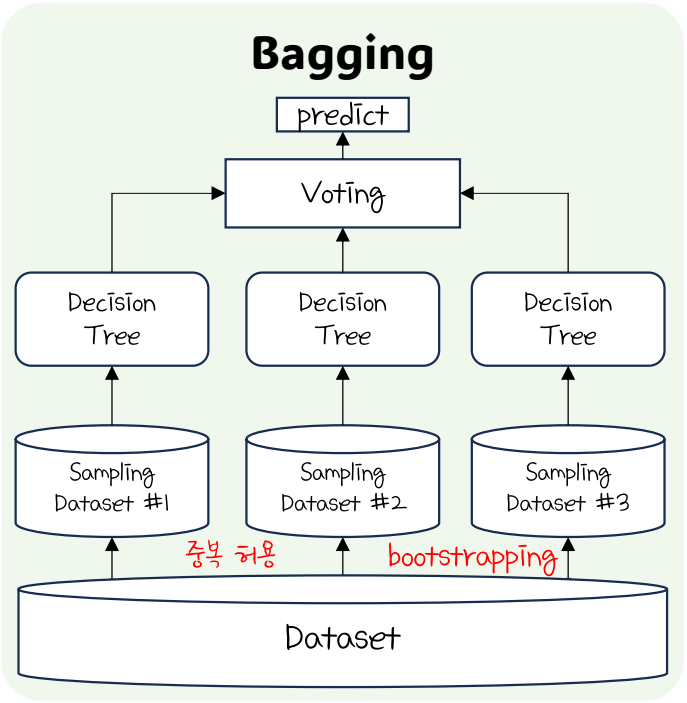




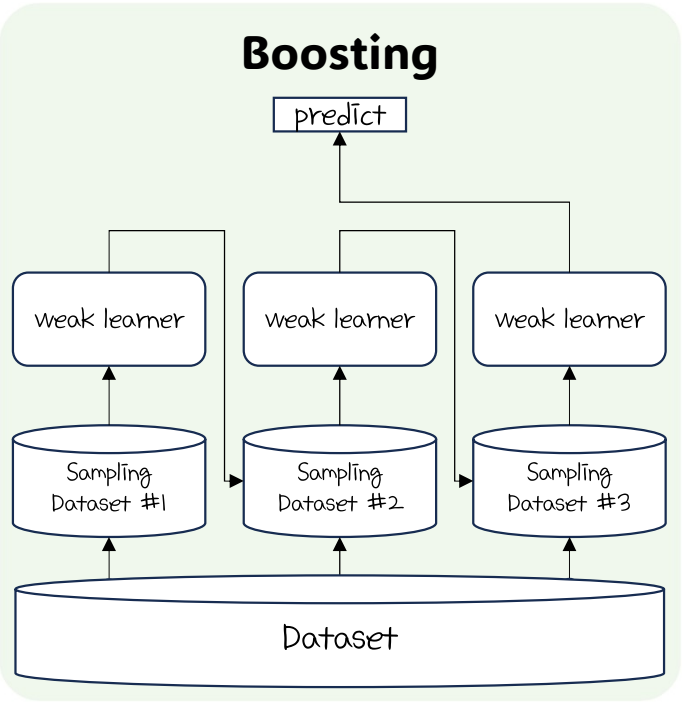
https://github.com/whatwant-school/python-ml/blob/main/05-week/05-week_05-Voting.ipynb

[04-04]
랜덤 포레스트 (Random Forest)



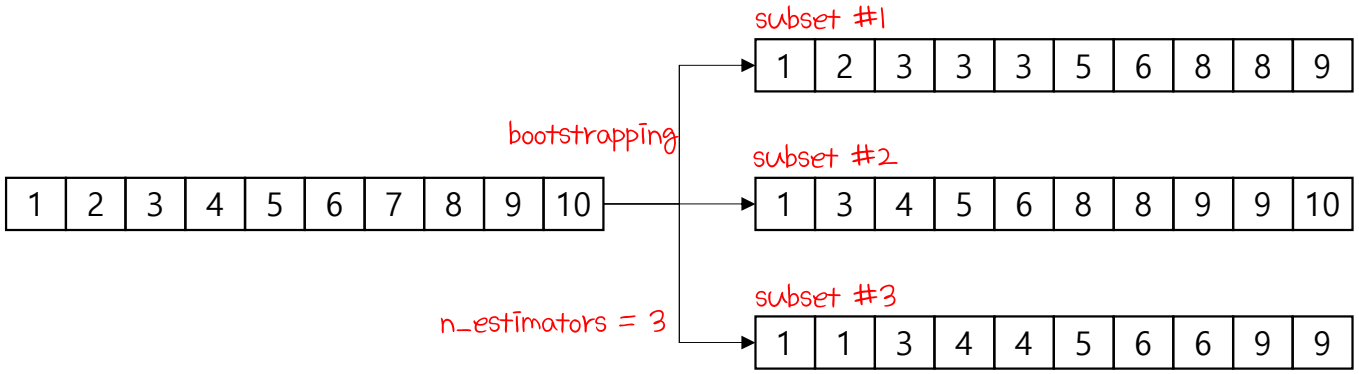


RandomForest



GBM, XGBoost, LightGBM

☐ Bagging = Bootstrap aggregating



```
[4]: from sklearn.ensemble import RandomForestClassifier

     RandomForestClassifier?
```

```
RandomForestClassifier(
    n_estimators=100,
    *,
    criterion='gini',
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features='sqrt',
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    bootstrap=True,
    oob_score=False,
    n_jobs=None,
    random_state=None,
    verbose=0,
    warm_start=False,
    class_weight=None,
```

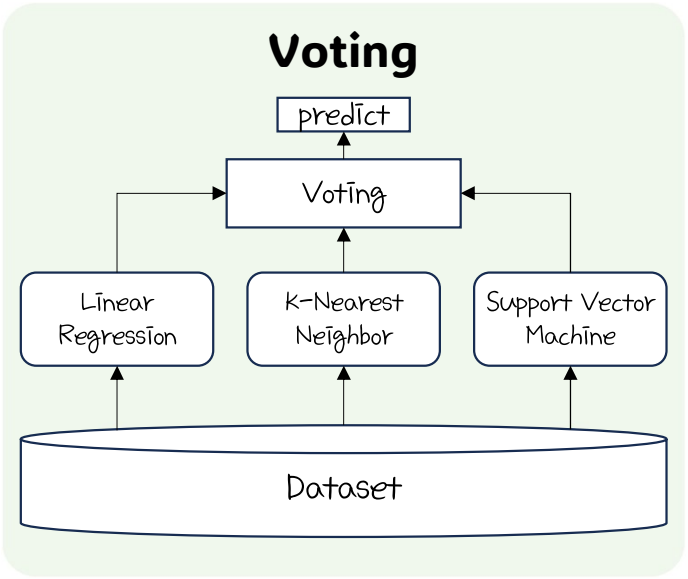
파라미터 명	설명
n_estimators (Default = 10) DecisionTree 개수	- 많이 설정할수록 좋은 성능 → 수행 시간 길어짐 - 높은 수치라고 무조건 성능 향상되지는 않음
max_features (Default = auto/sqrt) 최대 피처 개수	- DecisionTree에서는 default=None → 모든 피처 사용 - RandomForest에서는 default=sqrt → 전체 feature=16 이라면 4개 피처 사용

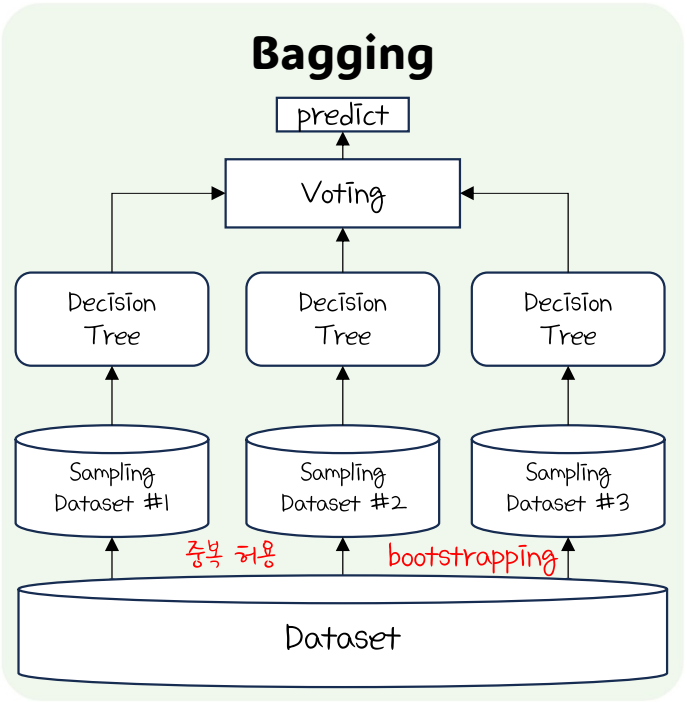
https://github.com/whatwant-school/python-ml/blob/main/05-week/05-week_06-RandomForest.ipynb

[12]:
```
rf_clf1 = RandomForestClassifier(max_depth=12,
                                 min_samples_leaf=8,
                                 min_samples_split=8,
                                 n_estimators=100,
                                 random_state=42)
rf_clf1.fit(train_X, train_y)

predict = rf_clf1.predict(test_X)
print('예측 정확도: {0:.4f}'.format(accuracy_score(test_y, predict)))
```

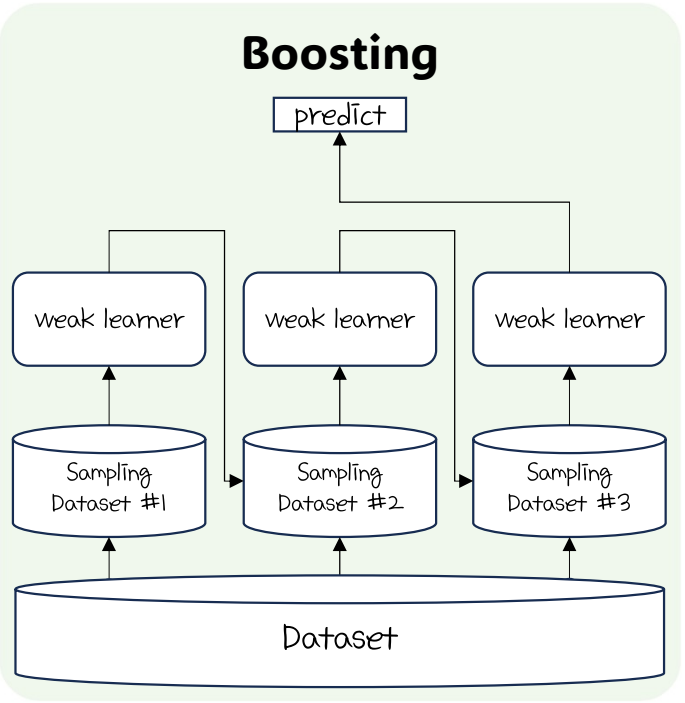
```
예측 정확도: 0.9291
```

[04-05]
그래디언트 부스팅
(GBM, Gradient Boosting Machine)

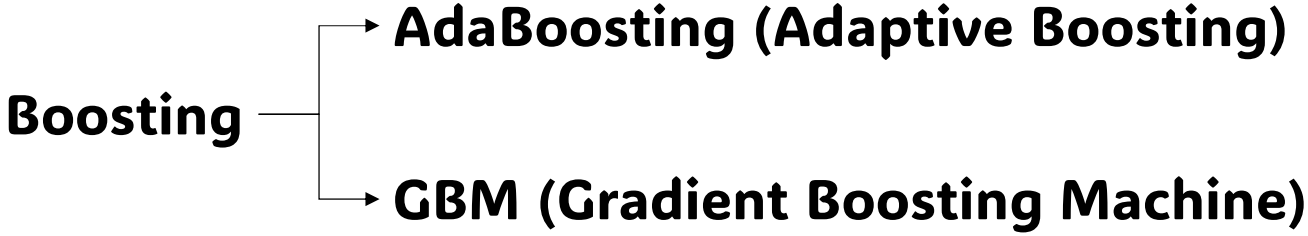




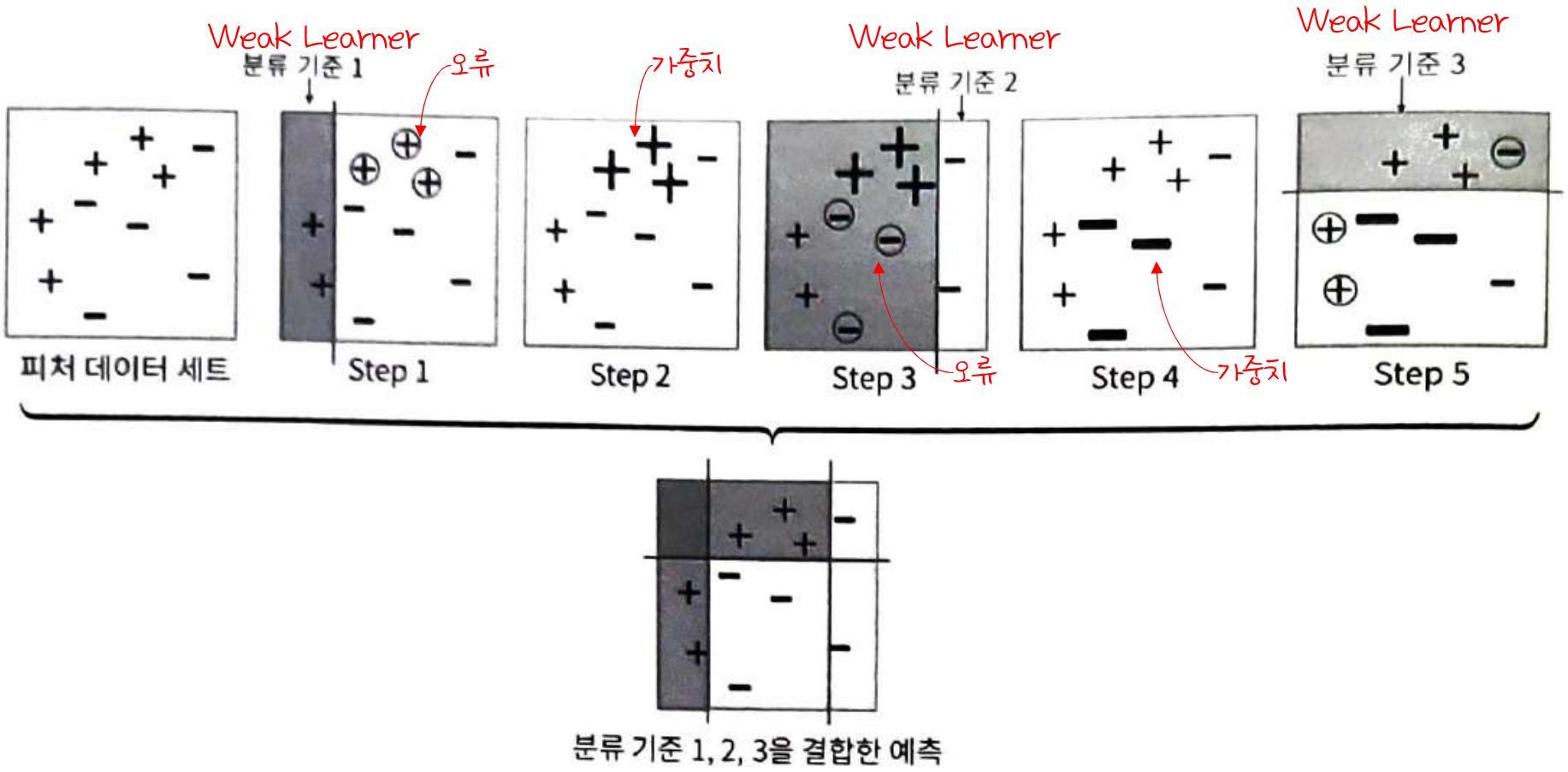
RandomForest



GBM, XGBoost, LightGBM


Boosting
AdaBoosting (Adaptive Boosting)
GBM (Gradient Boosting Machine)

☐ AdaBoosting (Adaptive Boosting)



□ GBM (Gradient Boosting Machine)

```
from sklearn.ensemble import GradientBoostingClassifier

GradientBoostingClassifier?
```

```
Init signature:
GradientBoostingClassifier(
    *,
    loss='log_loss',
    learning_rate=0.1,
    n_estimators=100,
    subsample=1.0,
    criterion='friedman_mse',
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_depth=3,
    min_impurity_decrease=0.0,
    init=None,
    random_state=None,
    max_features=None,
    verbose=0,
    max_leaf_nodes=None,
```

가중치 업데이트 방식
= 경사하강법 (Gradient Descent)

파라미터 명	설명
loss (Default = deviance) 비용 함수	- 보통 기본값 그대로 사용
learning_rate (Default = 0.1) 학습률	- 일반적으로 작은 값이 성능이 좋아지지만, 수행 시간이 오래 걸림
n_estimators (Default = 100) weak learner 개수	- 개수가 많을 수록 성능은 좋아지지만, 수행 시간이 오래 걸림
subsample (Default = 1) 데이터 샘플링 비율	- 0.5 = 50% - 과적합 우려가 있기 때문에 보통 1보다 작은 값 설정

https://github.com/whatwant-school/python-ml/blob/main/05-week/05-week_07-GBM.ipynb

[5]:
```
from sklearn.metrics import accuracy_score
import time

start_time = time.time()

gb_clf = GradientBoostingClassifier(random_state=42)
gb_clf.fit(train_X, train_y)

predict = gb_clf.predict(test_X)
accuracy = accuracy_score(test_y, predict)

print('GBM 정확도: {0:.4f}'.format(accuracy))
print("GBM 수행 시간: {0:.1f} 초 ".format(time.time() - start_time))
```

```
GBM 정확도: 0.9389
GBM 수행 시간: 764.3 초
```

[04-06]
XGBoost (eXtra Gradient Boost)

항목	설명
뛰어난 예측 성능	일반적으로 분류와 회귀 영역에서 뛰어난 예측 성능을 발휘합니다.
GBM 대비 빠른 수행 시간	일반적인 GBM은 순차적으로 Weak learner가 가중치를 증감하는 방법으로 학습하기 때문에 전반적으로 속도가 느립니다. 하지만 XGBoost는 병렬 수행 및 다양한 기능으로 GBM에 비해 빠른 수행 성능을 보장합니다. 아쉽게도 XGBoost가 일반적인 GBM에 비해 수행 시간이 빠르다는 것이지, 다른 머신러닝 알고리즘(예를 들어 랜덤 포레스트)에 비해서 빠르다는 의미는 아닙니다.
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 max_depth 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. 지정된 반복 횟수가 아니라 교차 검증을 통해 평가 데이터 세트의 평가 값이 최적화 되면 반복을 중간에 멈출 수 있는 조기 중단 기능이 있습니다.
결손값 자체 처리	XGBoost는 결손값을 자체 처리할 수 있는 기능을 가지고 있습니다.

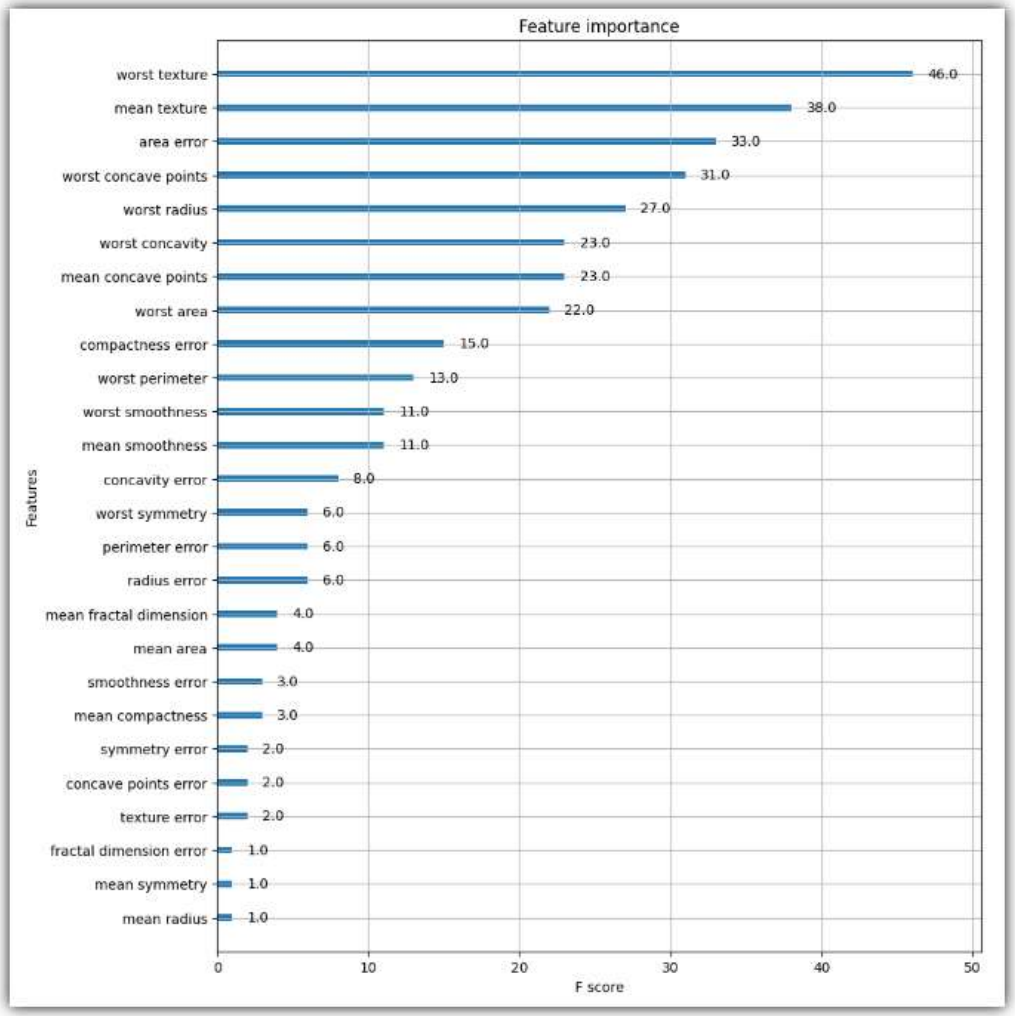
구분	파라미터 명	Default	설명
일반	**booster**	gbtree	- gbtree (tree model) / gblinear (linear model)
	silent	0	- 출력 메시지를 나타내고 싶지 않을 경우 0 설정
	nthread	전체	- 전체 CPU를 사용하지 않고 일부만 사용하고 싶을 경우 변경
부스터	**eta [alias: learning_rate]**	0.3	- 보통은 0.01 ~ 0.2 사이의 값을 선호
	num_boost_rounds		- GBM의 n_estimators
	min_child_weight	1	- 브랜칭 결정을 위한 데이터들의 weight 총합. 클수록 분할 자제.
	gamma [alias: min_split_loss]	0	- leaf node를 추가로 나눌지를 결정할 최소 손실 감소 값
	max_depth	6	- 0을 지정하면 depth 제한 없음. 보통 3~10 사이의 값 선호
	sub_sample	1	- 데이터를 샘플링하는 비율을 지정. 보통 0.5~1 사이의 값 선호
	colsample_bytree	1	- GBM의 max_features와 유사
	lambda [alias: reg_lambda]	1	- L2 Regularization 적용 값. 클수록 과적합 감소 효과
	alpha [alias: reg_alpha]	0	- L1 Regularization 적용 값. 클수록 과적합 감소 효과
	scale_pos_weight	1	- 비대칭 클래스로 구성된 데이터셋의 균형을 유지하기 위한 파라미터

구분	파라미터 명	Default	설명
학습	**objective**		- 손실함수 정의 . binary:logistic : 이진 분류일 때 적용 . multi:softmax : 다중 분류일 때 적용 . multi:softprob : 개별 레이블 클래스의 예측 확률을 반환
	eval_metric	회귀: rmse 분류: error	- 검증 함수 정의 . rmse : Root Mean Square Error . mae : Mean Absolute Error . logloss : Negative log-likelihood . error : Binary classification error rate (0.5 threshold) . merror : Multiclass classification error rate . mlogloss : Multiclass logloss . auc : Area under the curve

□ Overfitting

- eta 값을 낮춥니다(0.01 ~ 0.1) → num_round/n_estimators 값은 높여주어야 합니다
- max_depth 값을 낮춥니다.
- min_child_weight 값을 높입니다.
- gamma 값을 높입니다.
- subsample & colsample_bytree 값을 조정하는 것도 도움이 될 수 있습니다.

https://github.com/whatwant-school/python-ml/blob/main/05-week/05-week_08-XGBoost.ipynb



You've really worked hard today

Next Week ~ ?

06. XGBoost(eXtra Gradient Boost) 225
XGBoost 설치하기 227
파이썬 래퍼 XGBoost 하이퍼 파라미터 228
파이썬 래퍼 XGBoost 적용 – 위스콘신 유방암 예측 232
사이킷런 래퍼 XGBoost의 개요 및 적용 240

07.LightGBM 244
LightGBM 설치 246
LightGBM 하이퍼 파라미터 247
하이퍼 파라미터 튜닝 방안 248
파이썬 래퍼 LightGBM과 사이킷런 래퍼 XGBoost, LightGBM 하이퍼 파라미터 비교 249
LightGBM 적용 – 위스콘신 유방암 예측 250

08. 베이지안 최적화 기반의 HyperOpt를 이용한 하이퍼 파라미터 튜닝 253
베이지안 최적화 개요 254
HyperOpt 사용하기 256
HyperOpt를 이용한 XGBoost 하이퍼 파라미터 최적화 262

09. 분류 실습 – 캐글 산탄데르 고객 만족 예측 267
데이터 전처리 268
XGBoost 모델 학습과 하이퍼 파라미터 튜닝 271
LightGBM 모델 학습과 하이퍼 파라미터 튜닝 276

10. 분류 실습 – 캐글 신용카드 사기 검출 279
언더 샘플링과 오버 샘플링의 이해 279
데이터 일차 가공 및 모델 학습/예측/평가 281
데이터 분포도 변환 후 모델 학습/예측/평가 285
이상치 데이터 제거 후 모델 학습/예측/평가 288
SMOTE 오버 샘플링 적용 후 모델 학습/예측/평가 292

11. 스태킹 앙상블 295
기본 스태킹 모델 297
CV 세트 기반의 스태킹 300

12. 정리 306

05
회귀

01. 회귀 소개 308

02. 단순 선형 회귀를 통한 회귀 이해 310

03. 비용 최소화하기 – 경사 하강법(Gradient Descent) 소개 312

04. 사이킷런 LinearRegression을 이용한 보스턴 주택 가격 예측 321
LinearRegression 클래스 – Ordinary Least Squares 321
회귀 평가 지표 322
LinearRegression을 이용해 보스턴 주택 가격 회귀 구현 324

05. 다항 회귀와 과(대)적합/과소적합 이해 329
다항 회귀 이해 329
다항 회귀를 이용한 과소적합 및 과적합 이해 332
편향-분산 트레이드오프(Bias-Variance Trade off) 336

06. 규제 선형 모델 – 릿지, 라쏘, 엘라스틱넷 337
규제 선형 모델의 개요 337
릿지 회귀 339
라쏘 회귀 342
엘라스틱넷 회귀 345
선형 회귀 모델을 위한 데이터 변환 347

07. 로지스틱 회귀 350

Who ~ ?

Can I share your materials?

See you Next Weekend ~ ?

See you Next Weekend ~ ?