



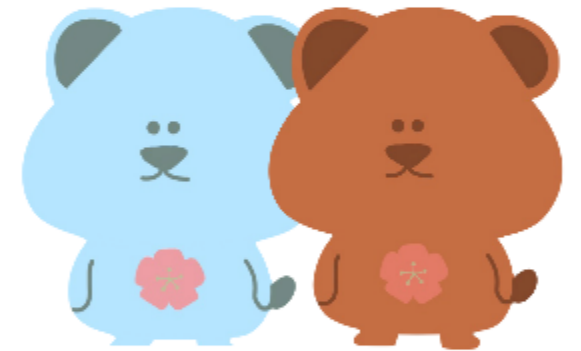
4장 02 결정 트리 실습 ~

03 앙상블 학습

강정인

결정 트리 실습

-사용자 행동 인식 데이터 세트



#사용자 행동 인식 데이터 세트

- 스마트폰 센서를 장착한 뒤 사람의 동작과 관련된 여러 가지 피처를 수집한 데이터
- [Human Activity Recognition Using Smartphones - UCI Machine Learning Repository](https://archive.ics.uci.edu/dataset/240/human+activity+recognition+using+smartphones)에 접속 후 Data set zip Download

The screenshot displays the UCI Machine Learning Repository website for the 'Human Activity Recognition Using Smartphones' dataset. The page features a blue header with the dataset name and a 'Donated on 12/9/2012' note. Below the header, there is a description of the dataset: 'Human Activity Recognition database built from the recordings of 30 subjects performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors.' To the right of the description, there are buttons for 'DOWNLOAD' and 'CITE', and a note indicating '2 citations'. A file explorer window is overlaid on the page, showing the downloaded files: 'UCI HAR Dataset' (59,570KB) and 'UCI HAR Dataset.names' (7KB).

이름	유형	압축된 크기	암호 사용	크기	비율	수정한 날짜
UCI HAR Dataset	압축(ZIP) 폴더	59,570KB	아니요	59,570KB	0%	2023-05-22 오후
UCI HAR Dataset.names	NAMES 파일	7KB	아니요	7KB	0%	2023-05-22 오후

#Data set 파일 구성

이름	수정한 날짜	유형
test	2023-09-17 오전 12:05	파일 폴더
train	2023-09-17 오전 12:05	파일 폴더
._DS_Store	2023-09-17 오전 12:05	DS_STORE 파일
._activity_labels	2023-09-17 오전 12:05	텍스트 문서
._features	2023-09-17 오전 12:05	텍스트 문서
._features_info	2023-09-17 오전 12:05	텍스트 문서
._README	2023-09-17 오전 12:05	텍스트 문서
._test	2023-09-17 오전 12:05	_TEST 파일
._train	2023-09-17 오전 12:05	_TRAIN 파일

Inertial Signals	2023-09-17 오전 12:05	파일 폴더	
._Inertial Signals	2023-09-17 오전 12:05	파일	1KB
._subject_test	2023-09-17 오전 12:05	텍스트 문서	1KB
._X_test	2023-09-17 오전 12:05	텍스트 문서	1KB
._y_test	2023-09-17 오전 12:05	텍스트 문서	1KB

Inertial Signals	2023-09-17 오전 12:05	파일 폴더	
._Inertial Signals	2023-09-17 오전 12:05	파일	1KB
._subject_train	2023-09-17 오전 12:05	텍스트 문서	1KB
._X_train	2023-09-17 오전 12:05	텍스트 문서	1KB
._y_train	2023-09-17 오전 12:05	텍스트 문서	1KB

1KB

1KB

1KB

#파일 DataFrame으로 로딩

- features.txt 파일을 DataFrame으로 로딩하기

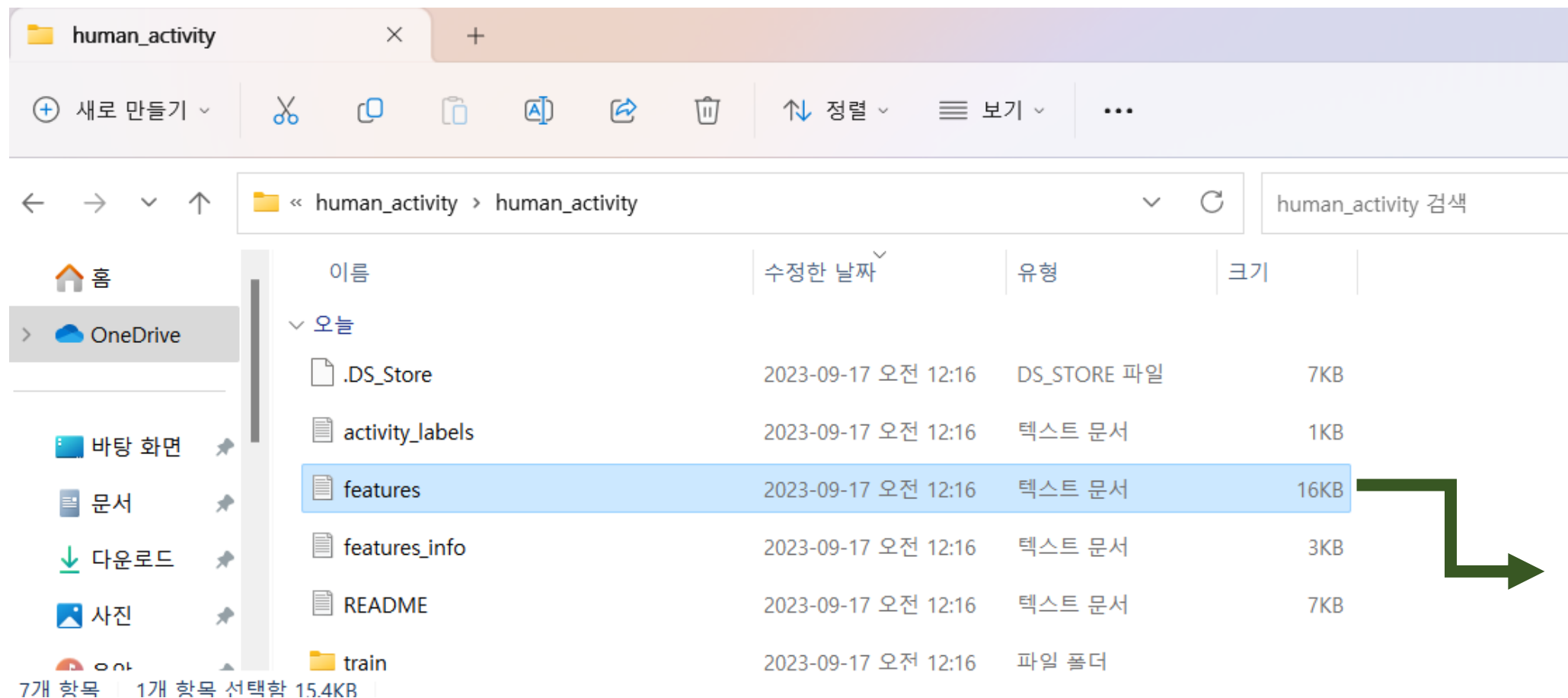
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

# features.txt 파일에는 피쳐 이름 index와 피쳐명이 공백으로 분리되어 있음. 이를 DataFrame으로 로드.
feature_name_df = pd.read_csv('./human_activity/features.txt', sep='\\s+',
                               header=None, names=['column_index', 'column_name'])

# 피쳐명 index를 제거하고, 피쳐명만 리스트 객체로 생성한 뒤 샘플로 10개만 추출
feature_name = feature_name_df.iloc[:, 1].values.tolist()
print('전체 피쳐명에서 10개만 추출:', feature_name[:10])
```

전체 피쳐명에서 10개만 추출: ['tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y', 'tBodyAcc-mean()-Z', 'tBodyAcc-std()-X', 'tBodyAcc-std()-Y', 'tBodyAcc-std()-Z', 'tBodyAcc-mad()-X', 'tBodyAcc-mad()-Y',

#중복된 피쳐명



이름	수정한 날짜	유형	크기
.DS_Store	2023-09-17 오전 12:16	DS_STORE 파일	7KB
activity_labels	2023-09-17 오전 12:16	텍스트 문서	1KB
features	2023-09-17 오전 12:16	텍스트 문서	16KB
features_info	2023-09-17 오전 12:16	텍스트 문서	3KB
README	2023-09-17 오전 12:16	텍스트 문서	7KB
train	2023-09-17 오전 12:16	파일 폴더	

```
1 tBodyAcc-mean()-X
2 tBodyAcc-mean()-Y
3 tBodyAcc-mean()-Z
4 tBodyAcc-std()-X
5 tBodyAcc-std()-Y
6 tBodyAcc-std()-Z
7 tBodyAcc-mad()-X
8 tBodyAcc-mad()-Y
9 tBodyAcc-mad()-Z
10 tBodyAcc-max()-X
11 tBodyAcc-max()-Y
12 tBodyAcc-max()-Z
13 tBodyAcc-min()-X
14 tBodyAcc-min()-Y
15 tBodyAcc-min()-Z
16 tBodyAcc-sma()
17 tBodyAcc-energy()-X
18 tBodyAcc-energy()-Y
19 tBodyAcc-energy()-Z
20 tBodyAcc-iqr()-X
21 tBodyAcc-iqr()-Y
22 tBodyAcc-iqr()-Z
23 tBodyAcc-entropy()-X
24 tBodyAcc-entropy()-Y
25 tBodyAcc-entropy()-Z
26 tBodyAcc-arCoeff()-X,1
27 tBodyAcc-arCoeff()-X,2
28 tBodyAcc-arCoeff()-X,3
29 tBodyAcc-arCoeff()-X,4
30 tBodyAcc-arCoeff()-Y,1
31 tBodyAcc-arCoeff()-Y,2
32 tBodyAcc-arCoeff()-Y,3
33 tBodyAcc-arCoeff()-Y,4
34 tBodyAcc-arCoeff()-Z,1
35 tBodyAcc-arCoeff()-Z,2
36 tBodyAcc-arCoeff()-Z,3
37 tBodyAcc-arCoeff()-Z,4
38 tBodyAcc-correlation()-X,Y
39 tBodyAcc-correlation()-X,Z
40 tBodyAcc-correlation()-Y,Z
```

#중복된 피처명

```
feature_dup_df = feature_name_df.groupby('column_name').count()
print(feature_dup_df[feature_dup_df['column_index'] > 1].count())
feature_dup_df[feature_dup_df['column_index'] > 1].head()
```

column_index 42
dtype: int64

중복된 피처명 42개

column_index

column_name

fBodyAcc-bandsEnergy0-1,16	3
fBodyAcc-bandsEnergy0-1,24	3
fBodyAcc-bandsEnergy0-1,8	3
fBodyAcc-bandsEnergy0-17,24	3
fBodyAcc-bandsEnergy0-17,32	3



#중복된 피처명 처리

- 중복된 피처명에 대해서 원본 피처명에 _1 또는 _2를 추가
- 새로운 피처명을 가지는 DataFrame을 반환하는 함수인 get_new_feature_name_df()를 생성

```
def get_new_feature_name_df(old_feature_name_df):  
    feature_dup_df = pd.DataFrame(data=old_feature_name_df.groupby('column_name').cumcount(),  
                                  columns=['dup_cnt'])  
    feature_dup_df = feature_dup_df.reset_index()  
    new_feature_name_df = pd.merge(old_feature_name_df.reset_index(), feature_dup_df, how='outer')  
    new_feature_name_df['column_name'] = new_feature_name_df[['column_name', 'dup_cnt']].apply(lambda x : x[0]+'_'+str(x[1])  
                                                    if x[1] >0 else x[0] , axis=1)  
    new_feature_name_df = new_feature_name_df.drop(['index'], axis=1)  
    return new_feature_name_df
```


#학습,테스트용 피쳐 파일 로딩

```
import pandas as pd

def get_human_dataset( ):

    # 각 데이터 파일들은 공백으로 분리되어 있으므로 read_csv에서 공백 문자를 sep으로 할당.
    feature_name_df = pd.read_csv('./human_activity/features.txt', sep='\s+',
                                   header=None, names=['column_index', 'column_name'])

    # 중복된 피쳐명을 수정하는 get_new_feature_name_df()를 이용, 신규 피쳐명 DataFrame 생성.
    new_feature_name_df = get_new_feature_name_df(feature_name_df)

    # DataFrame에 피쳐명을 컬럼으로 부여하기 위해 리스트 객체로 다시 변환
    feature_name = new_feature_name_df.iloc[:, 1].values.tolist()

    # 학습 피쳐 데이터 셋과 테스트 피쳐 데이터를 DataFrame으로 로딩. 컬럼명은 feature_name 적용
    X_train = pd.read_csv('./human_activity/train/X_train.txt', sep='\s+', names=feature_name )
    X_test = pd.read_csv('./human_activity/test/X_test.txt', sep='\s+', names=feature_name)

    # 학습 레이블과 테스트 레이블 데이터를 DataFrame으로 로딩하고 컬럼명은 action으로 부여
    y_train = pd.read_csv('./human_activity/train/y_train.txt', sep='\s+', header=None, names=['action'])
    y_test = pd.read_csv('./human_activity/test/y_test.txt', sep='\s+', header=None, names=['action'])

    # 로드된 학습/테스트용 DataFrame을 모두 반환
    return X_train, X_test, y_train, y_test
```

```
X_train, X_test, y_train, y_test = get_human_dataset()
```

#학습용 피쳐 데이터 세트 분석

```
▶ print('## 학습 피쳐 데이터셋 info()')  
print(X_train.info())
```

**7352개의 레코드
561개의 피쳐**

```
↳ ## 학습 피쳐 데이터셋 info()  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 7352 entries, 0 to 7351  
Columns: 561 entries, tBodyAcc-mean()-X to angle(Z,gravityMean)  
dtypes: float64(561)  
memory usage: 31.5 MB  
None
```

```
[20] print(y_train['action'].value_counts())
```

```
6    1407  
5    1374  
4    1286  
1    1226  
2    1073  
3     986  
Name: action, dtype: int64
```

**레이블값 1,2,3,4,5,6 개
분포도는 고르게 분포**

#동작 예측 분류 수행

- 사이킷런의 `DecisionTreeClassifier`를 이용해 동작 예측 분류 수행
- 하이퍼 파라미터의 값 추출

```
[▶] # @title 기본 제목 텍스트
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# 예제 반복 시마다 동일한 예측 결과 도출을 위해 random_state 설정
dt_clf = DecisionTreeClassifier(random_state=156)
dt_clf.fit(X_train , y_train)
pred = dt_clf.predict(X_test)
accuracy = accuracy_score(y_test , pred)
print('결정 트리 예측 정확도: {0:.4f}'.format(accuracy))

# DecisionTreeClassifier의 하이퍼 파라미터 추출
print('DecisionTreeClassifier 기본 하이퍼 파라미터:\n', dt_clf.get_params())
```

```
☞ 결정 트리 예측 정확도: 0.8548
DecisionTreeClassifier 기본 하이퍼 파라미터:
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None,
```

#예측 정확도

- 결정 트리의 깊이(Tree Depth)가 예측 정확도에 주는 영향
- 결정 트리의 경우 리프 노드가 될 수 있는 적합한 수준이 될 때까지 트리의 분할 수행
- GridSearchCV를 이용해 하이퍼 파라미터 max_depth의 값을 변화시킴
- Min_samples_split는 16으로 고정 후 max_depth를 6,8,10,12,16,20,24로 계속 늘리면서 예측 성능 측정

```
▶ from sklearn.model_selection import GridSearchCV

params = {
    'max_depth' : [ 6, 8 ,10, 12, 16 ,20, 24]
}

grid_cv = GridSearchCV(dt_clf, param_grid=params, scoring='accuracy', cv=5, verbose=1 )
grid_cv.fit(X_train , y_train)
print('GridSearchCV 최고 평균 정확도 수치:{0:.4f}'.format(grid_cv.best_score_))
print('GridSearchCV 최적 하이퍼 파라미터:', grid_cv.best_params_)
```

Fitting 5 folds for each of 7 candidates, totalling 35 fits
GridSearchCV 최고 평균 정확도 수치:0.8549
GridSearchCV 최적 하이퍼 파라미터: {'max_depth': 8}

#예측 성능 변화 관찰



```
# GridSearchCV객체의 cv_results_ 속성을 DataFrame으로 생성.
```

```
cv_results_df = pd.DataFrame(grid_cv.cv_results_)
```

```
# max_depth 파라미터 값과 그때의 테스트(Evaluation)셋, 학습 데이터 셋의 정확도 수치 추출
```

```
cv_results_df[['param_max_depth', 'mean_test_score']]
```

	param_max_depth	mean_test_score
0	6	0.847662
1	8	0.854879
2	10	0.852705
3	12	0.845768
4	16	0.847127
5	20	0.848624
6	24	0.848624



#결정 트리의 정확도 측정



```
max_depths = [ 6, 8 ,10, 12, 16 ,20, 24]
# max_depth 값을 변화 시키면서 그때마다 학습과 테스트 셋에서의 예측 성능 측정
for depth in max_depths:
    dt_clf = DecisionTreeClassifier(max_depth=depth, min_samples_split=16, random_state=156)
    dt_clf.fit(X_train , y_train)
    pred = dt_clf.predict(X_test)
    accuracy = accuracy_score(y_test , pred)
    print('max_depth = {0} 정확도: {1:.4f}'.format(depth , accuracy))
```

max_depth = 6 정확도: 0.8551

max_depth = 8 정확도: 0.8717

max_depth = 10 정확도: 0.8599

max_depth = 12 정확도: 0.8571

max_depth = 16 정확도: 0.8599

max_depth = 20 정확도: 0.8565

max_depth = 24 정확도: 0.8565

#정확도 성능 튜닝

- max_depth와 min_samples_split 둘 다 변경

```
▶ params = {  
    'max_depth' : [ 8 , 12, 16 ,20],  
    'min_samples_split' : [16, 24],  
}  
  
grid_cv = GridSearchCV(dt_clf, param_grid=params, scoring='accuracy', cv=5, verbose=1 )  
grid_cv.fit(X_train , y_train)  
print('GridSearchCV 최고 평균 정확도 수치: {0:.4f}'.format(grid_cv.best_score_))  
print('GridSearchCV 최적 하이퍼 파라미터:', grid_cv.best_params_)
```

Fitting 5 folds for each of 8 candidates, totalling 40 fits

GridSearchCV 최고 평균 정확도 수치: 0.8549

GridSearchCV 최적 하이퍼 파라미터: {'max_depth': 8, 'min_samples_split': 16}

max_depth가 8, min_samples_split이 16일 때 정확도 85.49%

#결정 트리의 예측 정확도

- `best_estimator_` :
최적 하이퍼 파라미터인 `max_depth8`, `min_samples_split 16`으로 학습이 완료된 Estimator 객체



```
best_df_clf = grid_cv.best_estimator_  
pred1 = best_df_clf.predict(X_test)  
accuracy = accuracy_score(y_test , pred1)  
print('결정 트리 예측 정확도:{0:.4f}'.format(accuracy))
```

결정 트리 예측 정확도:0.8717

`max_depth 8`, `min_samples_split 16`일 때 테스트 데이터 세트의 예측 정확도는 약 87.17%

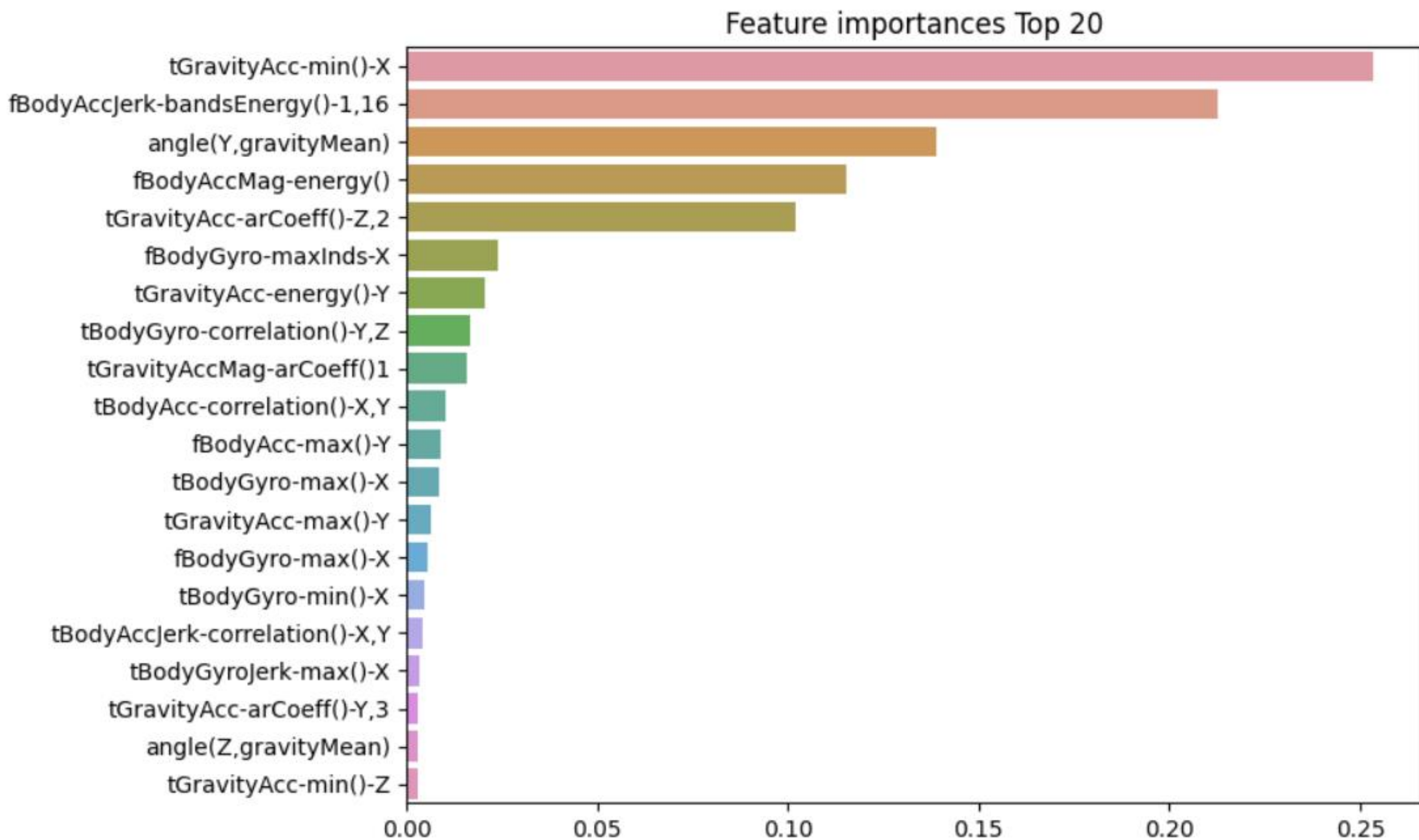
#피처의 중요도

- Feature의 중요도를 feature_importances_ 속성을 이용해 알아보기
- 중요도 높은 순으로 Top 20 피처를 막대그래프로 표현

```
import seaborn as sns

ftr_importances_values = best_df_clf.feature_importances_
# Top 중요도로 정렬을 쉽게 하고, 시본(Seaborn)의 막대그래프로 쉽게 표현하기 위해 Series변환
ftr_importances = pd.Series(ftr_importances_values, index=X_train.columns )
# 중요도값 순으로 Series를 정렬
ftr_top20 = ftr_importances.sort_values(ascending=False)[:20]
plt.figure(figsize=(8,6))
plt.title('Feature importances Top 20')
sns.barplot(x=ftr_top20 , y = ftr_top20.index)
plt.show()
```

#중요도 높은 순으로 막대 그래프로 표현



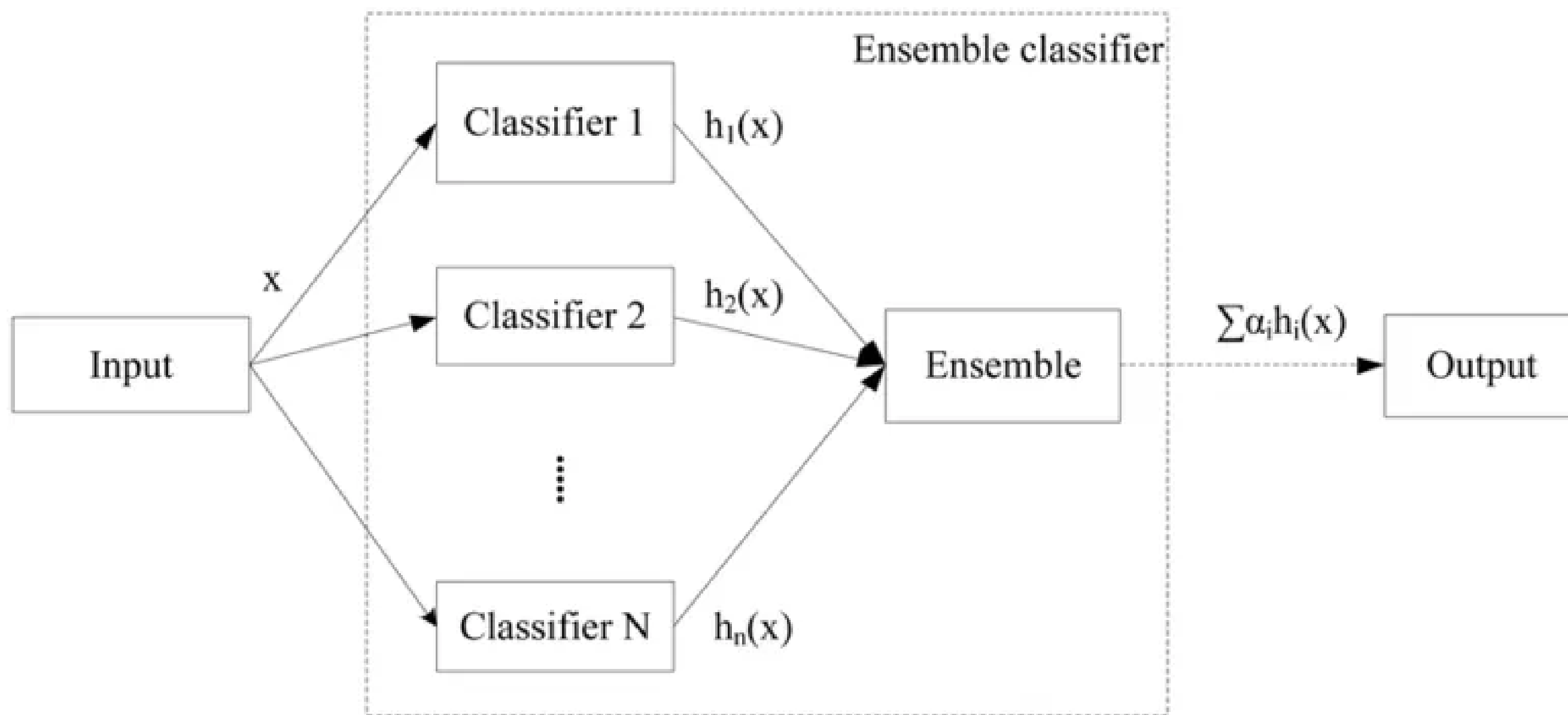
03 앙상블 학습



#앙상블 학습의 개요

여러 개의 분류기를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측 도출

-단일 분류기보다 신뢰성의 높은 예측값을 얻음



#앙상블 학습의 유형

- 일반적으로 보팅(Voting), 배깅(Bagging), 부스팅(Boosting)으로 구분하며 이외에 스택킹 기법 등이 있음
- 배깅은 랜덤 포레스트(Random Forest) 알고리즘이 대표적
- 부스팅은 에이다 부스팅, 그래디언트 부스팅, XGBoost, LightGBM 등이 있음
- 정형 데이터의 분류나 회귀에서는 GBM 부스팅 계열의 앙상블이 높은 예측 성능을 나타냄

#앙상블 학습의 유형_보팅과 배깅

■ 보팅과 배깅

- 여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정하는 방식

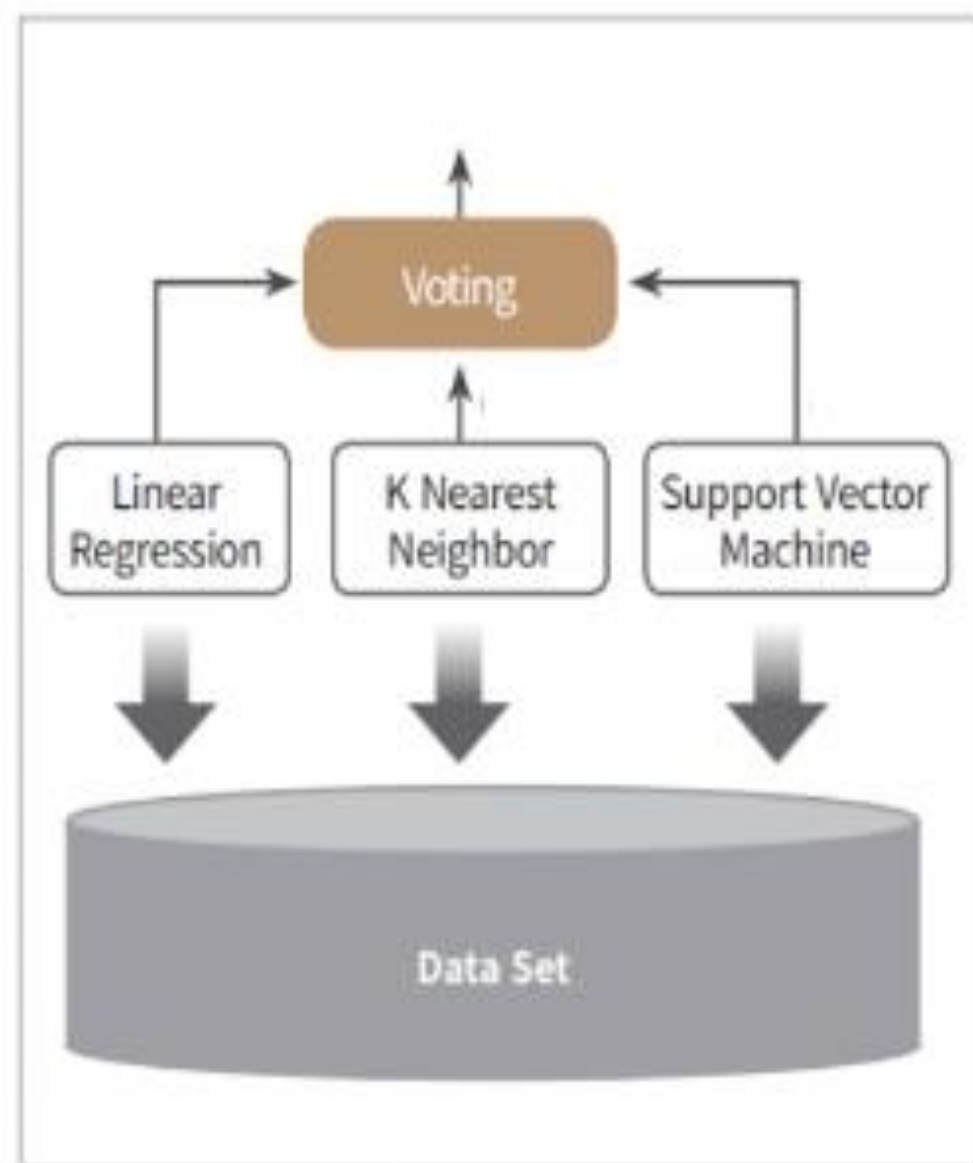
■ 보팅

- 서로 다른 알고리즘을 가진 분류기 결합

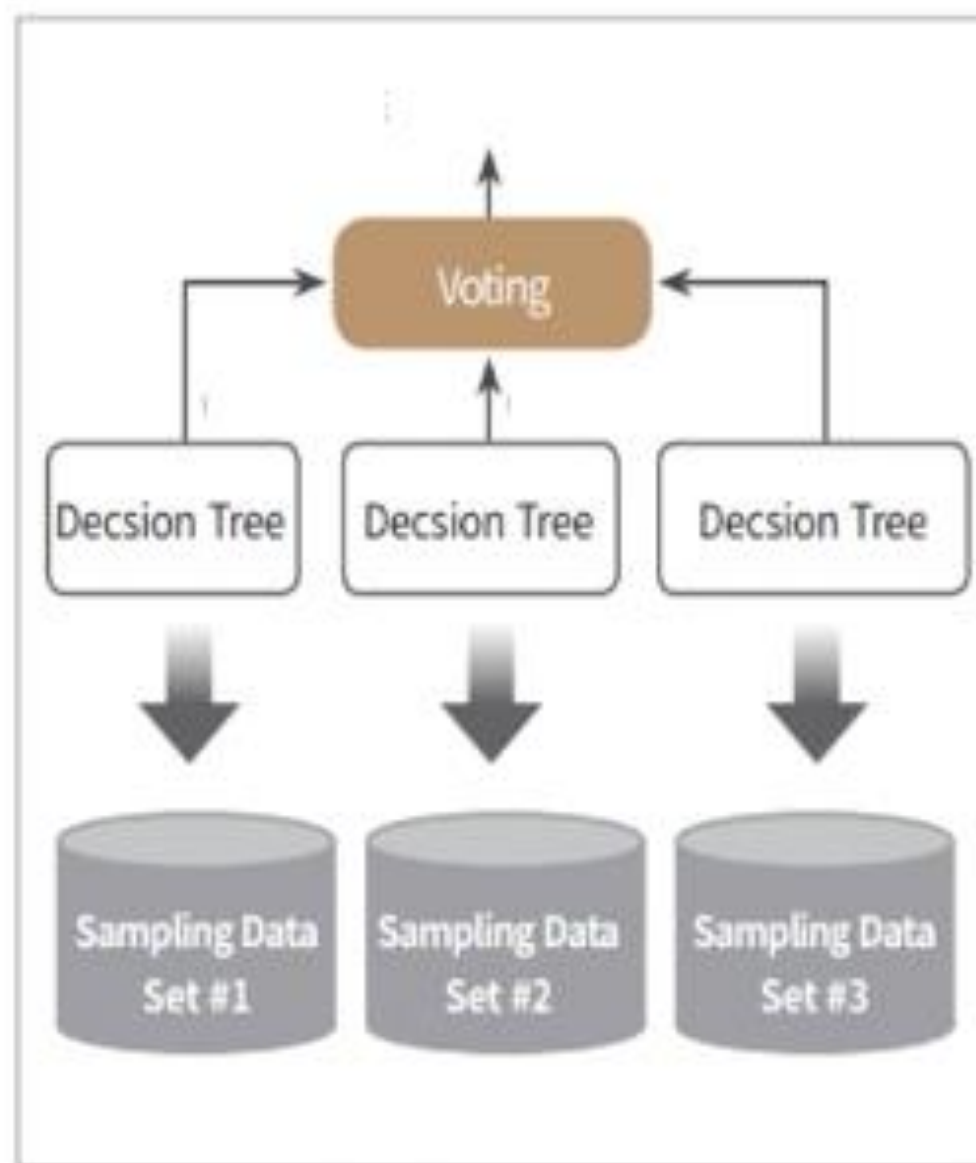
■ 배깅

- 각각의 분류기가 모두 같은 유형의 알고리즘 기반
- 데이터 샘플링을 서로 다르게 가져가면서 학습 수행
- 랜덤 포레스트 알고리즘

#분류기 도식화



Voting



Bagging

보팅 분류기

‘선형 회귀, K 최근접 이웃, 서포트 벡터 머신’이라는 3개의 ML알고리즘이 같은 데이터 세트에 대해 학습하고 예측한 결과를 보팅을 통해 최종 예측 결과 산정

배깅 분류기

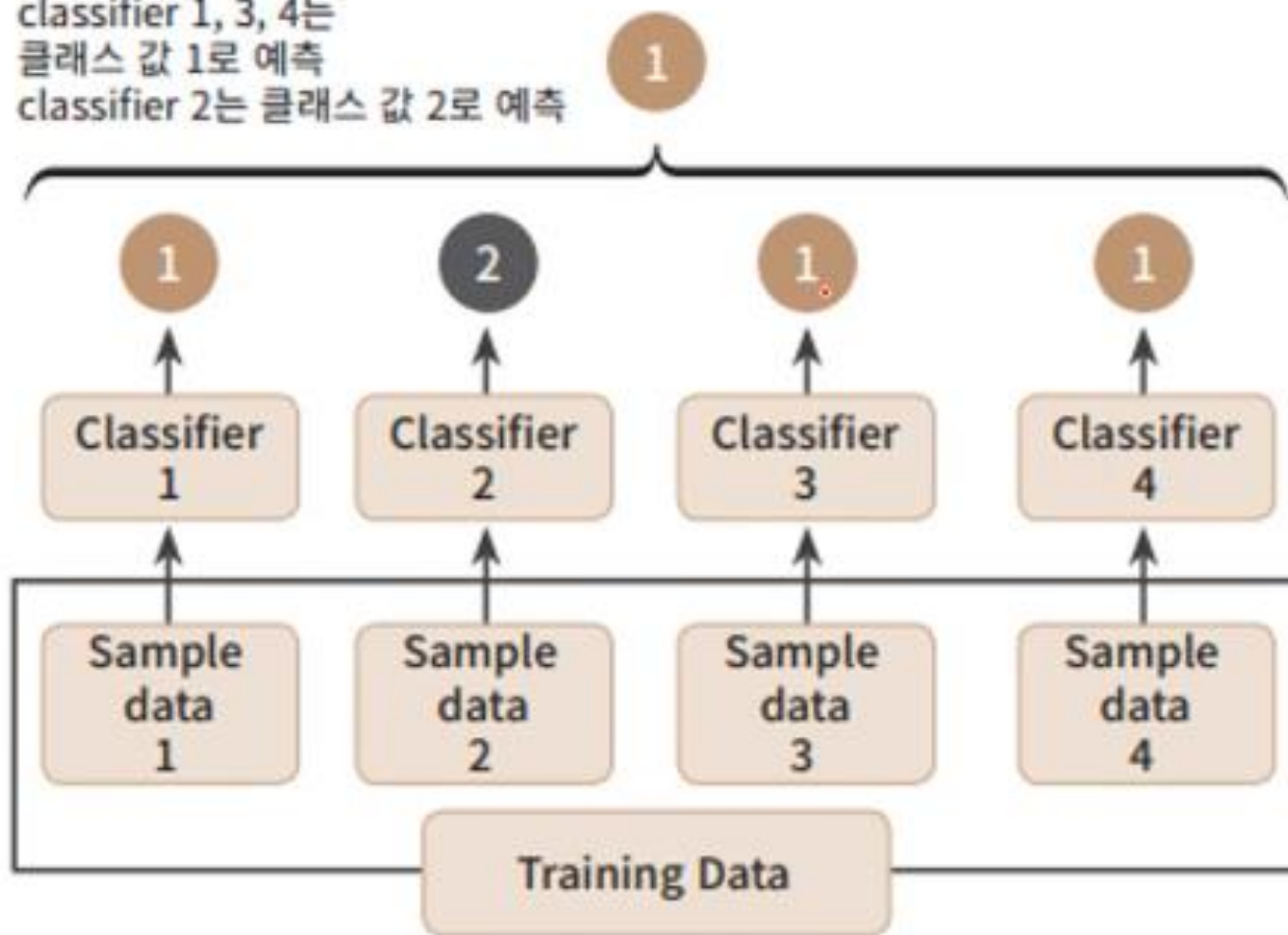
개별 분류기가 부트 스트래핑 방식으로 샘플링된 데이터 세트에 대해 학습을 통해 개별적인 예측을 수행한 결과를 보팅을 통해 최종 예측 결과 선정

#보팅 유형

- 하드 보팅 - 다수결의 원칙
- 소프트 보팅 - 확률을 평균하여 결정

Hard Voting은 다수의 classifier 간 다수결로 최종 class 결정

클래스 값 1로 예측
classifier 1, 3, 4는
클래스 값 1로 예측
classifier 2는 클래스 값 2로 예측

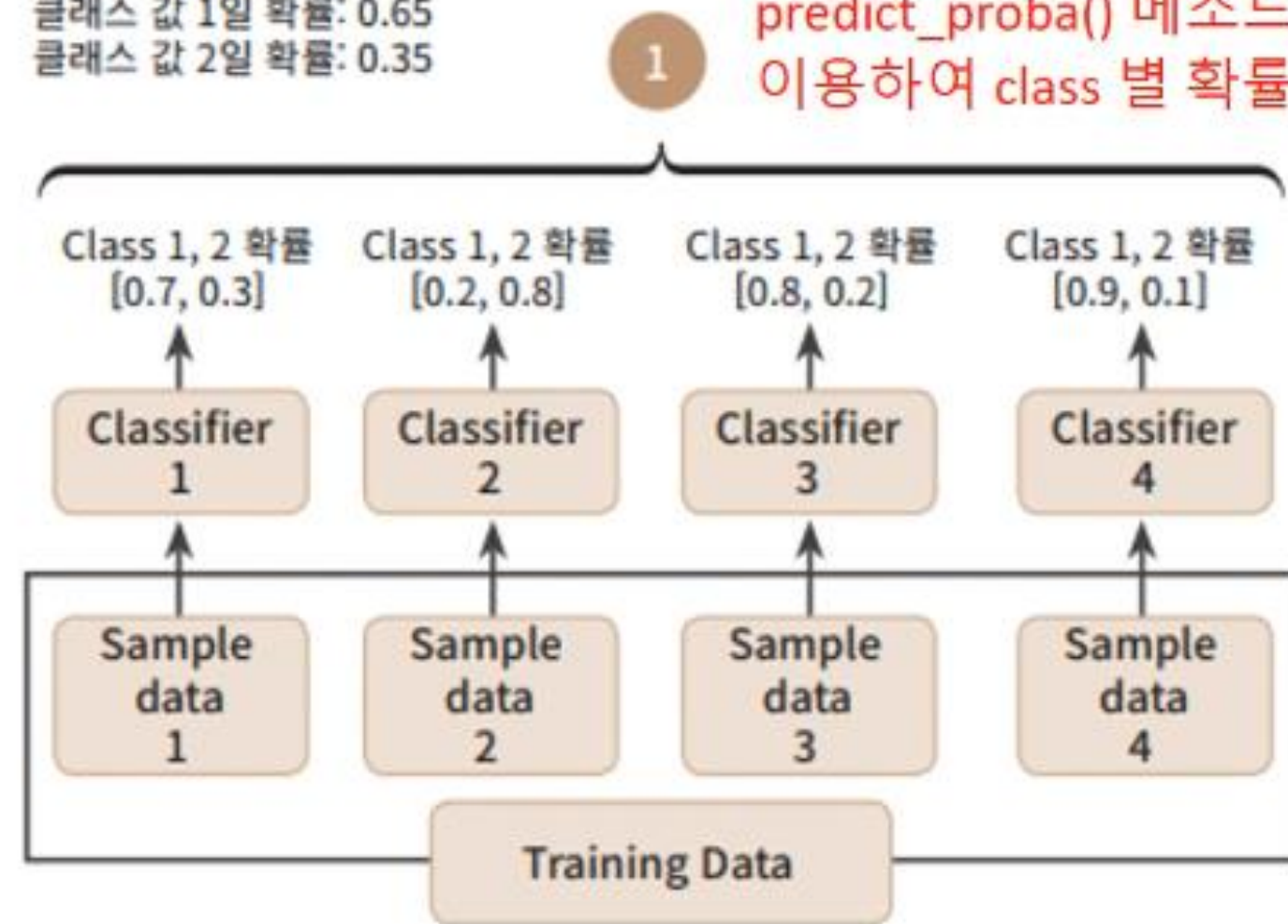


<하드 보팅>

Soft Voting은 다수의 classifier 들의 class 확률을 평균하여 결정

클래스 값 1로 예측
클래스 값 1일 확률: 0.65
클래스 값 2일 확률: 0.35

**predict_proba() 메소드를
이용하여 class 별 확률 결정**



<소프트 보팅>

#보팅 분류기

사이킷런은 보팅 방식의 앙상블을 구현한 VotingClassifier 클래스 제공

위스콘신 유방암 데이터 세트 예측 분석

- 유방암의 악성종양, 양성종양 여부를 결정하는 이진 분류 데이터 세트
- 종양의 크기, 모양 등의 형태와 관련한 피쳐
- 로지스틱 회귀와 KNN 기반으로 보팅 분류기 생성

#위스콘신 데이터 세트 예측 분석

```
import pandas as pd

from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()

data_df = pd.DataFrame(cancer.data, columns=cancer.feature_names)
data_df.head(3)
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter
0	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.6
1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.8
2	19.69	21.25	130.0	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.5

3 rows x 30 columns

#소프트 방식의 보팅 분류기 생성

- *VotingClassifier* 클래스는 주요 생성 인자로 *estimators*와 *voting* 값 입력
- *Estimators*는 리스트 값으로 보팅에 사용될 여러 개의 *Classifier* 객체들을 튜플 형식으로 입력

```
[4] # 개별 모델은 로지스틱 회귀와 KNN 임.  
lr_clf = LogisticRegression(solver='liblinear')  
knn_clf = KNeighborsClassifier(n_neighbors=8)  
  
# 개별 모델을 소프트 보팅 기반의 앙상블 모델로 구현한 분류기  
vo_clf = VotingClassifier( estimators=[('LR',lr_clf),('KNN',knn_clf)] , voting='soft' )  
  
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,  
                                                    test_size=0.2 , random_state= 156)  
  
# VotingClassifier 학습/예측/평가.  
vo_clf.fit(X_train , y_train)  
pred = vo_clf.predict(X_test)  
print('Voting 분류기 정확도: {0:.4f}'.format(accuracy_score(y_test , pred)))  
  
# 개별 모델의 학습/예측/평가.  
classifiers = [lr_clf, knn_clf]  
for classifier in classifiers:  
    classifier.fit(X_train , y_train)  
    pred = classifier.predict(X_test)  
    class_name= classifier.__class__.__name__  
    print('{0} 정확도: {1:.4f}'.format(class_name, accuracy_score(y_test , pred)))
```

```
Voting 분류기 정확도: 0.9561  
LogisticRegression 정확도: 0.9474  
KNeighborsClassifier 정확도: 0.9386
```

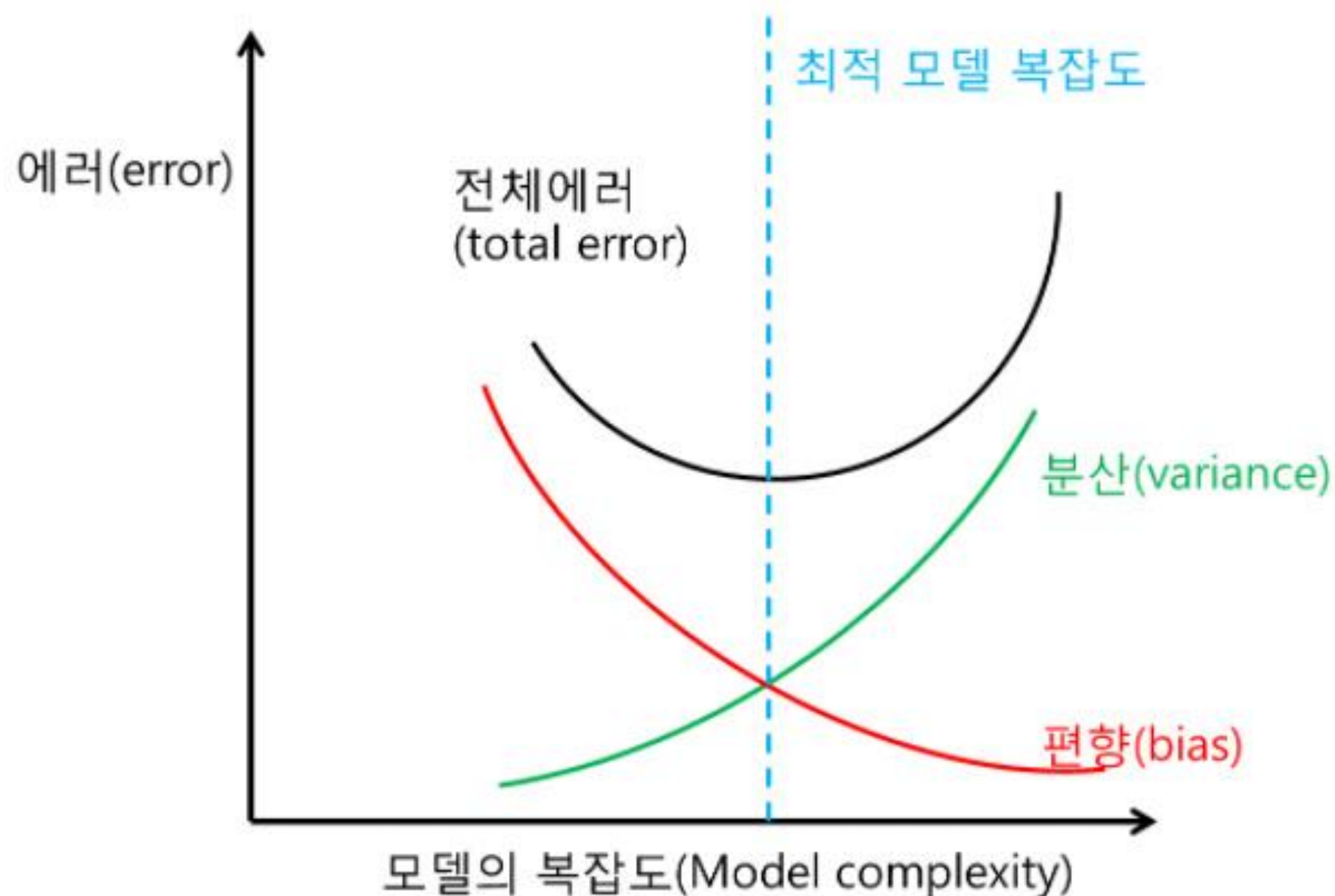
#ML 모델의 성능

- 보팅과 배깅 등의 앙상블 방법은 다른 단일 ML 알고리즘보다 예측 성능이 뛰어남
- 현실 세계는 다양한 변수와 예측이 어려운 규칙으로 구성
- 다양한 관점을 가진 알고리즘이 서로 결합하면 더 나은 성능을 실제 환경에서 끌어낼 수 있음

#ML 모델의 성능

- ML 모델은 어떻게 높은 유연성을 가지고 현실에 대처할 수 있는가가 중요한 평가요소

· '편향-분산 트레이드오프' 극복이 중요



#ML 모델의 성능

- 배깅과 부스팅

- 결정 트리 알고리즘 기반
- 예외 상황에 집착해 과적합 발생 가능성이 있음



- 앙상블 학습에서는 매우 많은 분류기를 결합해 다양한 상황을 학습함으로써 극복
- 결정 트리의 단점인 과적합을 수십~수천개의 많은 분류기를 결합해 보완하고 장점인 직관적인 분류 기준은 강화됨