파머완 4장 - 분류

01. 분류(Classification)의 개요

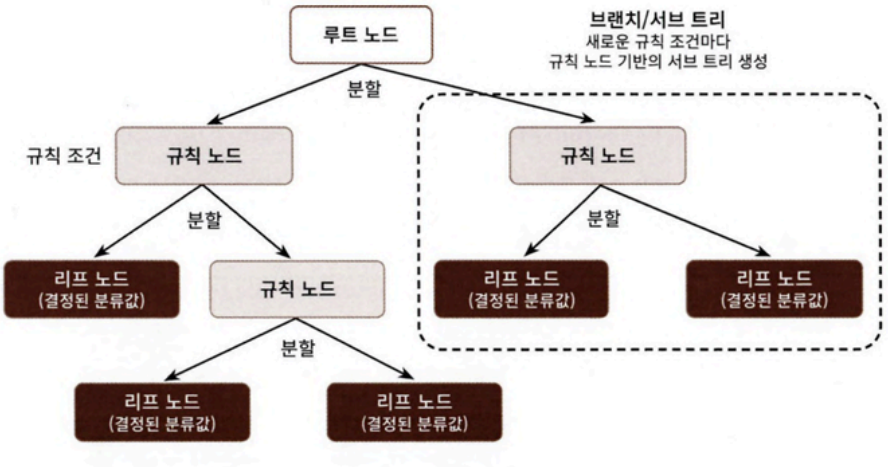
- 지도 학습 : 명시적인 정답이 있는 데이터가 주어진 상태에서 학습하는 머신러닝 방식 (Label)
- 분류(Classification) : 다양한 머신러닝 알고리즘 구현 (베이즈, 로지스틱 회귀, 결정 트리, 벡터 머신, 최소 근접 알고리즘, 신경망, 앙상블 등

앙상블 방법 (Ensemble Method) : 정형 데이터 예측 분석 영역

- Bagging (ex-random forest) , Boosting (ex-gradient boosting)
- XGBoost, LightGBM, Stacking
- 결정트리 - 사전 가공의 영향이 적음, 과적합 (overfitting) 발생 가능성

02. 결정 트리

- 데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 트리 기반의 분류 규칙을 만드는 것
- 규칙 노드 : 규칙 조건, 리프 노드 : 결정된 클래스 값, 새로운 규칙 조건마다 서브 트리 생성
- 규칙이 많을수록 과적합으로 이어지기 쉽고, 트리의 깊이가 깊어질수록 결정 트리의 예측 성능이 저하될 가능성 높음



- 가능한 한 적은 결정 노드로 높은 예측 정확도를 가지려면 데이터를 분류할 때 최대한 많은 데이터 세트가 해당 분류에 속할 수 있도록 결정 노드의 규칙이 정해져야 함 → 최대한 균일한 데이터 세트를 구성하도록 분할(split)하는 것이 중요
- 결정 노드는 정보 균일도가 높은 데이터 세트를 먼저 선택할 수 있도록 규칙 조건을 만듦 : 데이터 세트를 쪼개 서브 데이트 세트 생성 → 자식 데이터 세트
- 정보 이득 (Information Gain) : 엔트로피(정보 혼잡도)를 이용

→ 정보 이득 지수 = 1 - 엔트로피 지수

- 지니 계수 : 머신러닝에서 지니 계수가 낮을수록 데이터 균일도가 높음 → 지니 계수가 낮은 속성을 기준으로 분할
- DecisionTreeClassifier : 지니 계수를 이용해 데이터 세트 분할
- 결정 트리 - 데이터 세트를 분할하는 데 가장 좋은 조건 (정보 이득이 높거나 지니 계수가 낮음)

[결정 트리]

- 균일도를 기반으로 하고 있어 알고리즘이 쉽고 직관적, but 과적합 문제
- 복잡한 학습 모델을 만들게 되면 실제 상황에는 유연하게 대처 X, 예측 성능이 떨어질 수밖에 없음

[결정 트리 파라미터]

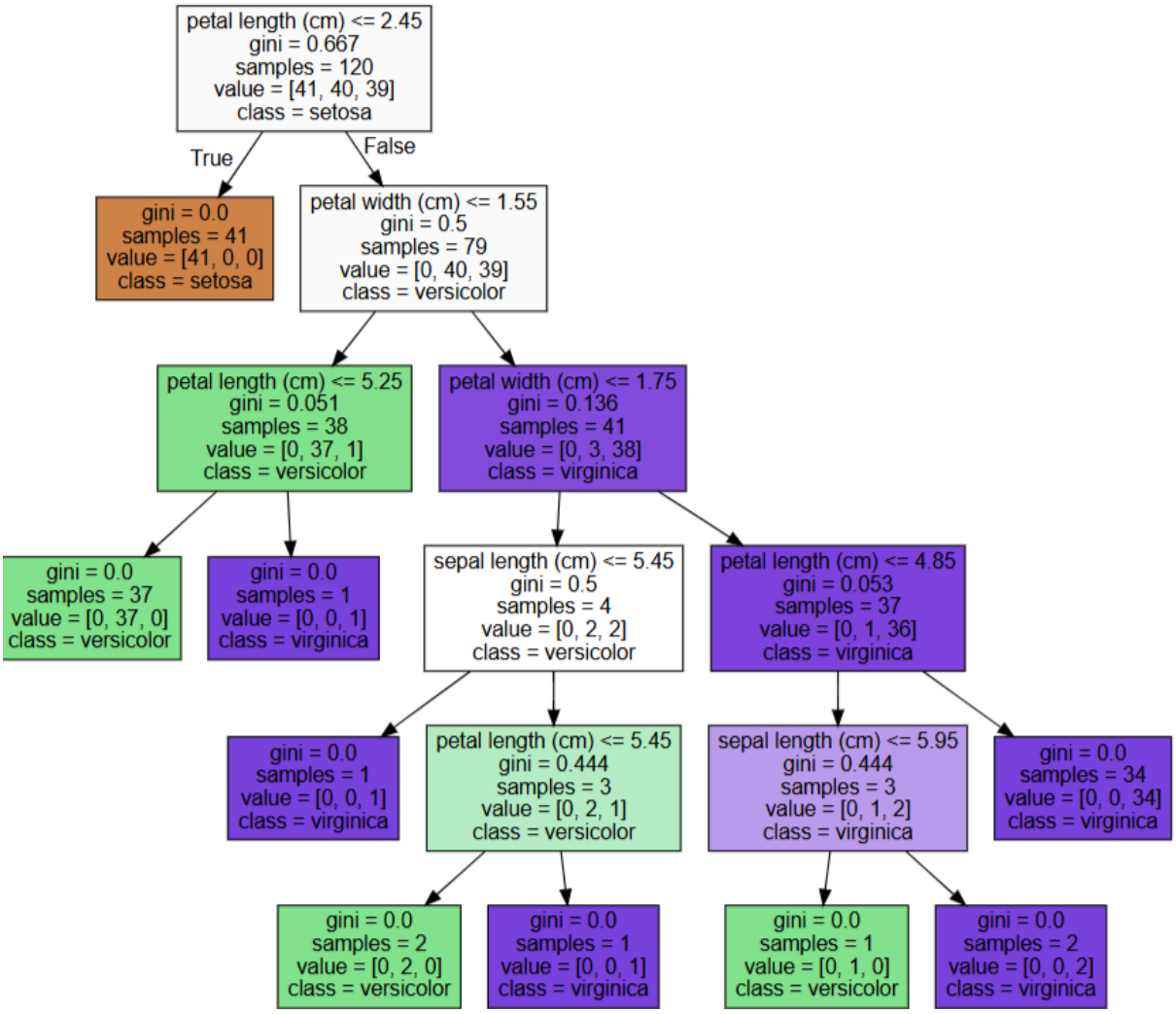
- 사이킷런의 결정트리 구현은 CART 알고리즘 기반

min_samples_split	노드를 분할하기 위한 최소한의 샘플 데이터 수로 과적합을 제어하는 데 사용됨
min_samples_leaf	분할될 경우, 왼쪽과 오른쪽의 브랜치 노드에서 가져야할 최소한의 샘플 데이터 수
max_features	최적의 분할을 위해 고려할 최대 피처 개수 int, sqrt, auto, log, None
max_depth	트리의 최대 깊이
max_leaf_nodes	말단 노드의 최대 개수

결정 트리 모델의 시각화 ; Graphviz

- Graphviz : 그래프 기반의 dot 파일로 기술된 다양한 이미지를 쉽게 시각화할 수 있는 패키지
- export_graphviz () API : Estimator, 피처의 이름 리스트, 레이블 이름 리스트 입력 → 학습된 결정 트리 규칙을 실제 트리 형태로 시각화
- Graphviz 설치

- export_graphviz() : 그래프 형태로 시각화할 수 있는 출력 파일을 생성

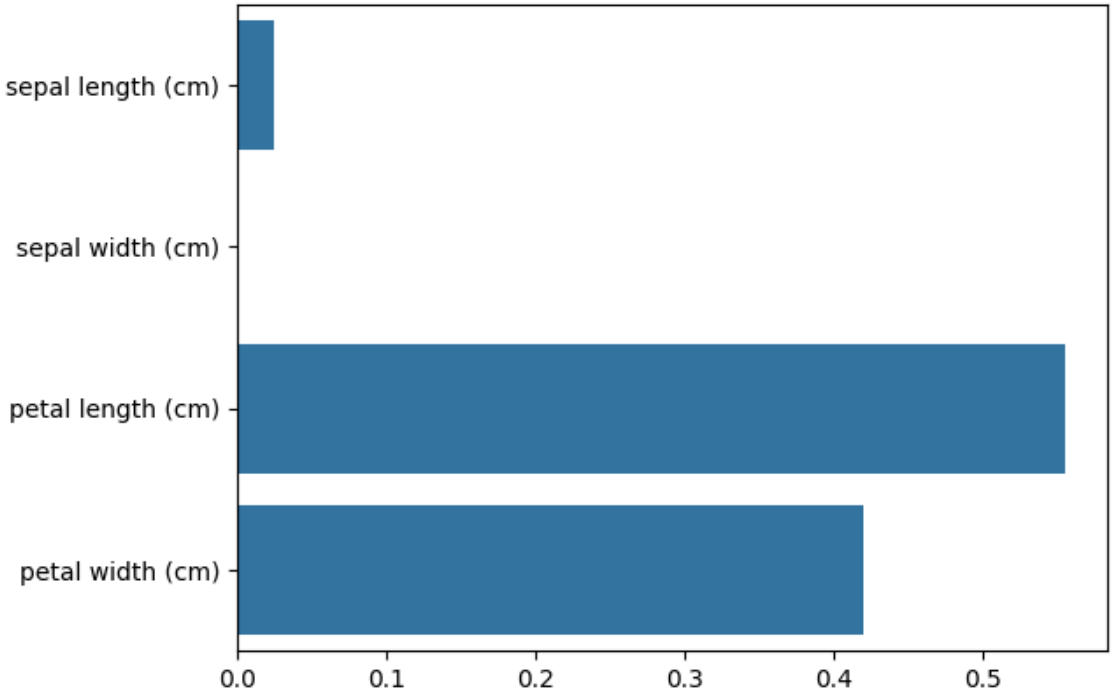


- 리프 노드 : 더 이상 자식 노드 X, 최종 클래스 값이 결정되는 노드
- 브랜치 노드 : 자식 노드 O, 자식 노드를 만들기 위한 분할 규칙 조건 있음

petal length(cm)	피처의 조건이 있는 것은 자식 노드를 만들기 위한 규칙 조건
gini	다음의 valued]로 주어진 데이터 분포에서의 지니 계수
samples	현 규칙에 해당하는 데이터 건수
value = [ ]	클래스 값 기반의 데이터 건수

- max_depth : 결정 트리의 최대 트리 깊이를 제어
- min_samples_leaf : 분할될 경우 왼쪽과 오른쪽 자식 노드 각각이 가지게 될 최소 데이터 건수를 지정
- feature_importances : 피처의 중요한 역할 지표, ndarray 형태로 값을 반환하며 피처 순서대로 값이 할당 → 일반적으로 값이 높을수록 해당 피처의 중요도가 높음

```
import seaborn as sns
import numpy as np
%matplotlib inline

#feature importance 추출
print("Feature importances:\n{0}".format(np.round(dt_clf.feature_importances

#feature별 importance 매핑
for name, value in zip(iris_data.feature_names, dt_clf.feature_importances_):
    print('{0}:{1:.3f}'.format(name, value))

#feature importance를 column별로 시각화하기
sns.barplot(x=dt_clf.feature_importances_, y = iris_data.feature_names)
```

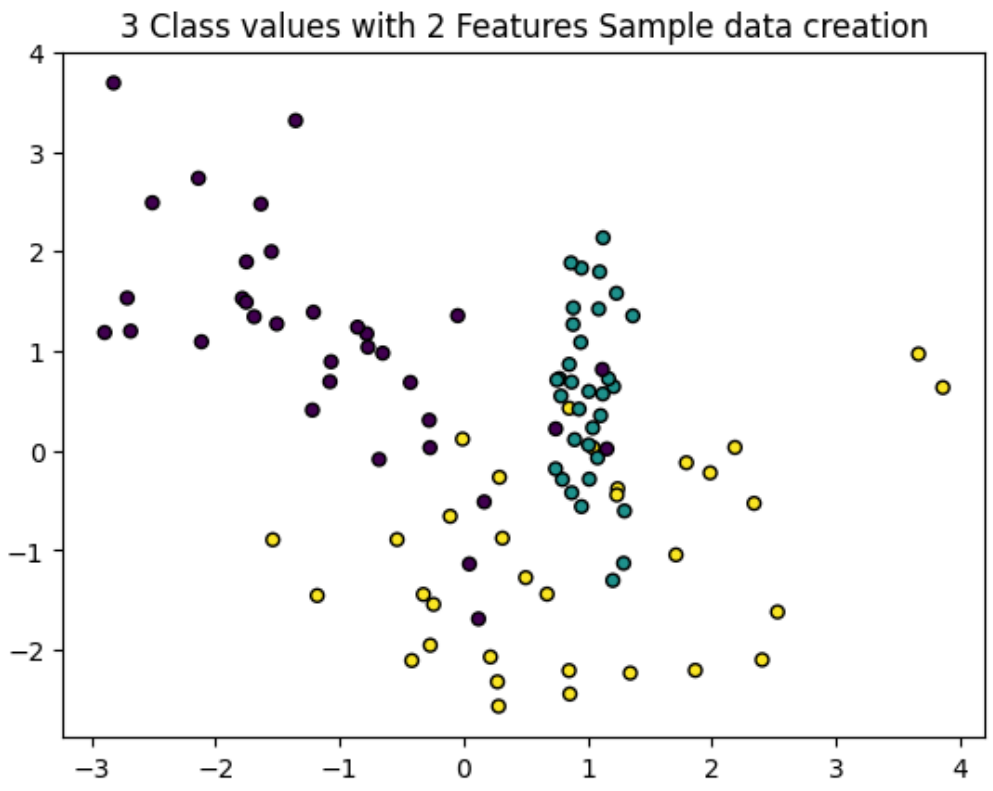


→ petal_length가 가장 피처 중요도가 높음

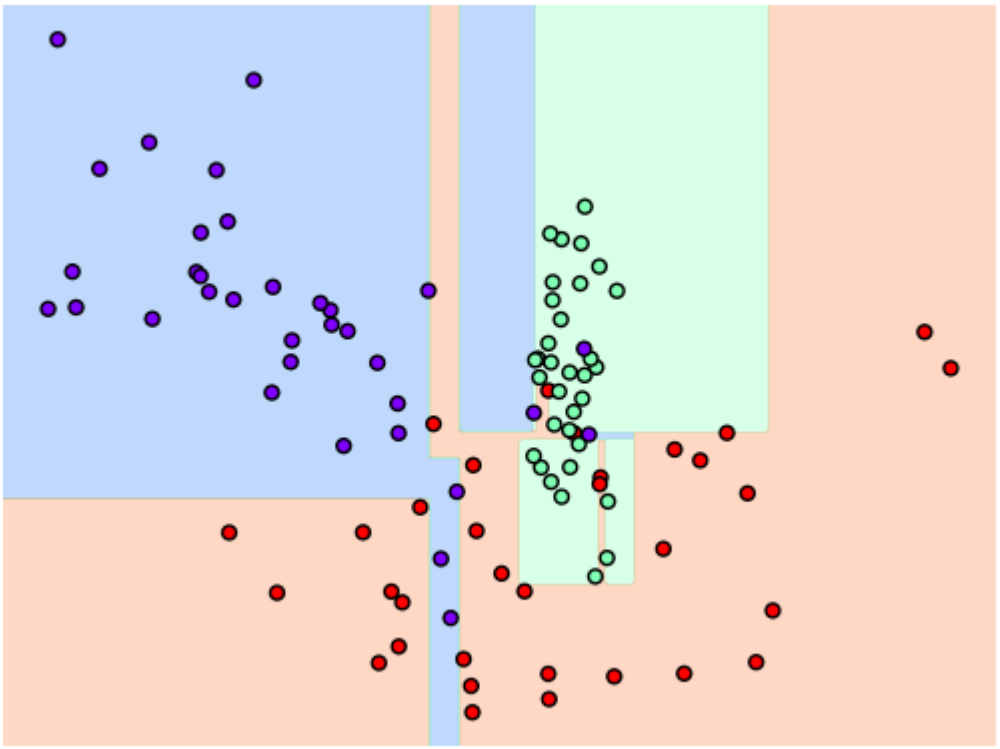
결정 트리 과적합(Overfitting)

- make_classification()

데이터 세트 시각화



- visualize_boundary() : 머신러닝 모델이 클래스 값을 예측하는 결정 기준을 색상과 경계로 나타냄



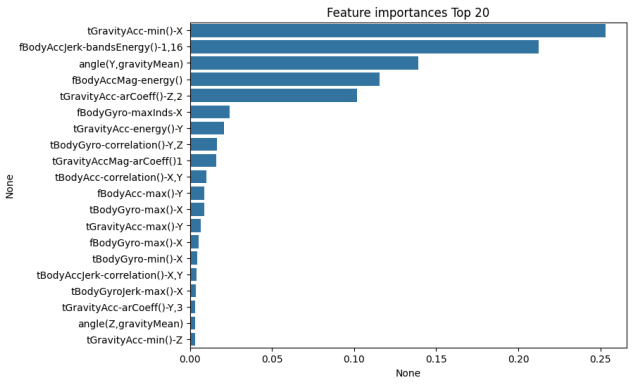
결정 트리 실습 - 사용자 행동 인식 데이터 세트

→ UCI HAR DATASET 이용

- 중복 피처명 처리
- 동작 예측 분류 수행

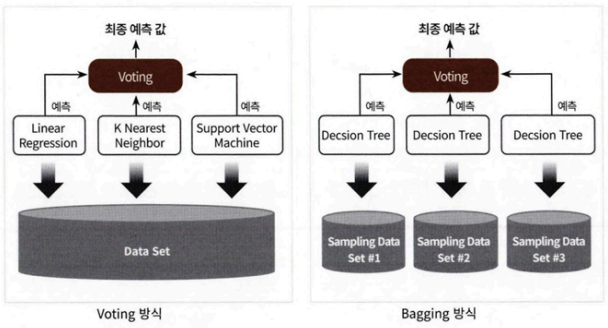
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

#예제 반복 시마다 동일한 예측결과 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 트리의 트리 깊이(Tree Depth)가 예측 정확도에 영향을 줌
- GridSearchCV： max_depth 변화시키며 예측 성능 확인
- cv_results_： CV세트에 하이퍼 파라미터를 순차적으로 입력했을 때의 성능 수치
- 깊이가 깊어질수록 과적합의 영향력이 커지므로 하이퍼 파라미터를 이용해 깊이를 제어할 수 있어야 함
- 정확도 성능 튜닝 : max_depth, min_samples_split 변화
- feature_importances : 결정트리에서 각 피처의 중요도



03. 앙상블 학습

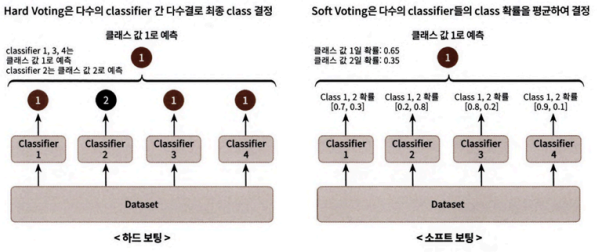
- 앙상블 학습 (Ensemble Learning)을 통한 분류 : 여러 개의 Classifier을 생성하고 그 예측을 결합함으로써 보다 정확하나 최종 예측을 도출하는 기법
- 목표 : 다양한 분류기의 예측 결과를 결합 → 단일 분류기보다 신뢰성이 높은 예측값을 얻는 것
- 정형 데이터 분류 시에 뛰어남
- XGboost, LightBGM
- 앙상블 학습의 유형 : Voting, Bagging, Boosting
- 보팅&배깅 : 여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정 (보팅 : 서로 다른 알고리즘을 가진 분류기를 결합, 배깅 : 모두 같은 유형의 알고리즘 기반이며 데이터 샘플링이 다름)



- Bootstrapping 분할 방식 : 개별 Classifier에게 데이터를 샘플링해서 추출
- 배깅 앙상블 방식 : 개별 분류기가 부트
  스트래핑 방식으로 샘플링된 데이터 세트 → 학습을 통해 개별적인 예측을 수행한 결과를 보팅을 통해서 최종 예측 결과를 선정
- 부스팅 : 여러 개의 분류기가 순차적으로 학습을 수행하되, 앞에서 학습한 분류기가 예측이 틀린 데이터에 대해서는 올바르게 예측할 수 있도록 다음 분류기에는 가중치(weight)를 부여하며 학습과 예측을 진행하는 것
- 대표적인 부스팅 모듈 : 그래디언트 부스트, XGBoost, LightBGM

보팅 유형 - 하드 보팅과 소프트 보팅

- 하드 보팅 : 예측한 결괏값들 중 다수의 분류기가 결정한 예측값을 최종 보팅 결괏값으로 선정
- 소프트 보팅 : 분류기들의 레이블 값 결정 확률을 모두 더하고 이를 평균해서 이들 중 확률이 가장 높은 레이블 값을 최종 보팅 결괏값으로 선정



보팅 분류기 (Voting Classifier) :

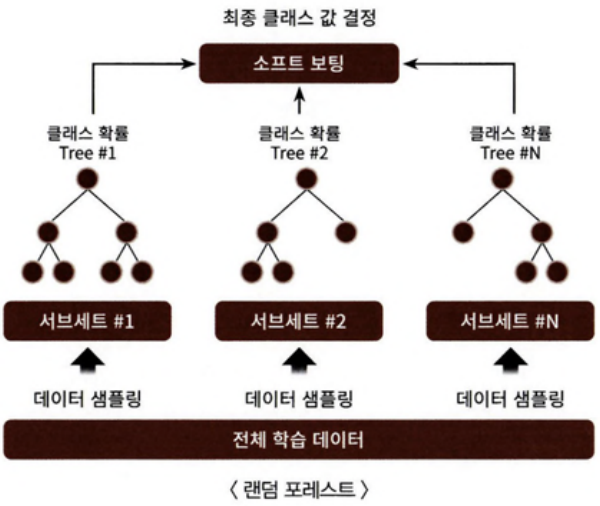
- 보팅 방식의 앙상블을 구현

```
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)
```

- estimators와 voting값을 입력받음
- voting은 ‘hard’ 시 하드보팅, ‘soft’시 소프트 보팅 적용
- 보팅으로 여러 개의 기반 분류기를 결합한다고 해서 무조건 기반 분류기보다 예측 성능이 향상되지는 X
- 편향-분산 트레이드 오프
- 결정 트리 알고리즘의 단점 (과적합) → 앙상블 학습 : 수십~수천 개의 매우 많은 분류기를 결합해 다양한 상황을 학습하게 함으로써 극복 → 편향-분산 트레이드오프의 효과 극대화

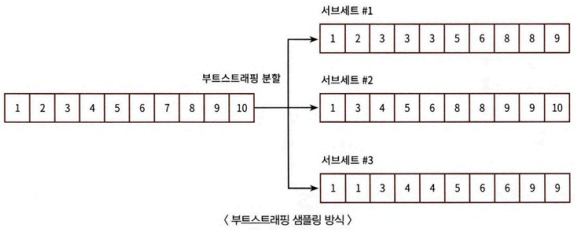
04. 랜덤 포레스트

- 배깅(Bagging) : 같은 알고리즘으로 여러 개의 분류기를 만들어서 보팅으로 최종 결정하는 알고리즘 (ex-랜덤 포레스트)
- 랜덤 포레스트 : 여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 각자의 데이터를 샘플링해 개별적으로 학습을 수행한 뒤 최종적으로 모든 분류기가 보팅을 통해 예측 결정



〈 랜덤 포레스트 〉

- 개별 트리가 학습하는 데이터 세트는 전체 데이터에서 일부가 중첩되게 샘플링된 데이터 세트
- 부트스트래핑(bootstrapping) 분할 방식 : 여러 개의 데이터 세트를 중첩되게 분리하는 것 → Subset 만듦



〈부트스트래핑 샘플링 방식〉

- 데이터가 중첩된 개별 데이터 세트에 결정 트리 분류기를 각각 적용
- 사이킷런 _ RandomForestClassifier 클래스 : 랜덤 포레스트 기반의 분류를 지원

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

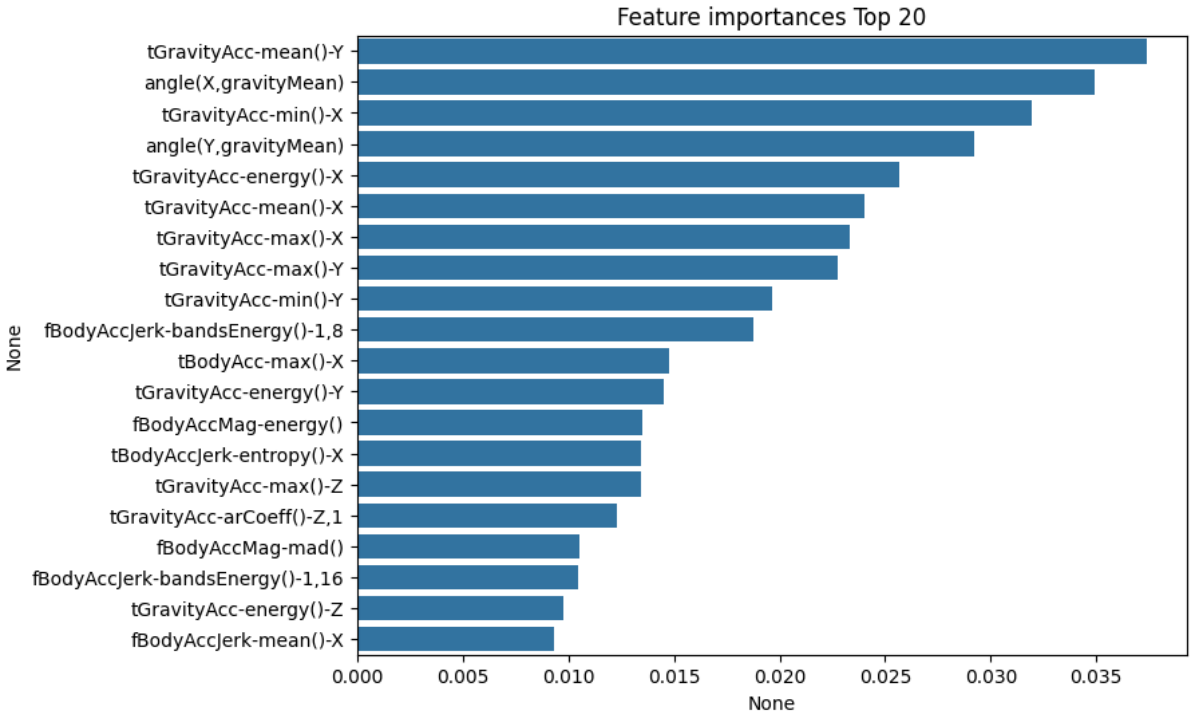
```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

#결정 트리에서 사용한 get_human_dataset()를 이용해 학습/테스트용 DataFrame 반
X_train, X_test, y_train, y_test = get_human_dataset()

#랜덤 포레스트 학습 및 별도의 테스트 세트로 예측 성능 평가
rf_clf = RandomForestClassifier(random_state = 0, max_depth = 8)
rf_clf.fit(X_train,y_train)
pred = rf_clf.predict(X_test)
accuracy = accuracy_score(y_test, pred)
print('랜덤 포레스트 정확도:{0:.4f}'.format(accuracy))
```

랜덤 포레스트 하이퍼 파라미터 및 튜닝

- 트리 기반의 앙상블 알고리즘의 단점 : 하이퍼 파라미터가 너무 많아 튜닝을 위한 시간이 많이 소모됨

n_estimators	랜덤 포레스트에서 결정 트리의 개수를 지정
max_features	결정트리에서의 파라미터와 동일, but 기본 max_features는 auto,sqrt 과 동일

- GridSearchCV를 이용한 랜덤 포레스트의 하이퍼 파라미터 튜닝
- 피처 중요도 막대 그래프 시각화


Feature importances Top 20
tGravityAcc-mean()-Y
angle(X,gravityMean)
tGravityAcc-min()-X
angle(Y,gravityMean)
tGravityAcc-energy()-X
tGravityAcc-mean()-X
tGravityAcc-max()-X
tGravityAcc-max()-Y
tGravityAcc-min()-Y
fBodyAccJerk-bandsEnergy()-1,8
tBodyAcc-max()-X
tGravityAcc-energy()-Y
fBodyAccMag-energy()
tBodyAccJerk-entropy()-X
tGravityAcc-max()-Z
tGravityAcc-arCoeff()-Z,1
fBodyAccMag-mad()
fBodyAccJerk-bandsEnergy()-1,16
tGravityAcc-energy()-Z
fBodyAccJerk-mean()-X
None
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
None