# Week3_예습과제_우정연

## 4.1 분류(Classification)의 개요

- 지도학습
  - 레이블(Label), 명시적인 정답이 있는 데이터가 주어진 상태에서 학습하는 머신러닝 방식
  - 분류
    - 학습 데이터로 주어진 데이터의 피처 값과 레이블 값(결정값, 클래스값)을 머신러닝 알고리즘으로 학습해 모델 생성
    - 생성된 모델에 새로운 데이터 값이 주어졌을 때 미지의 레이블 값을 예측
    - 기존 데이터가 어떤 레이블에 속하는지 패턴을 알고리즘으로 인지 → 새롭게 관측된 데이터에 대한 레이블 판별
    - 다양한 머신러닝 알고리즘으로 구현 가능

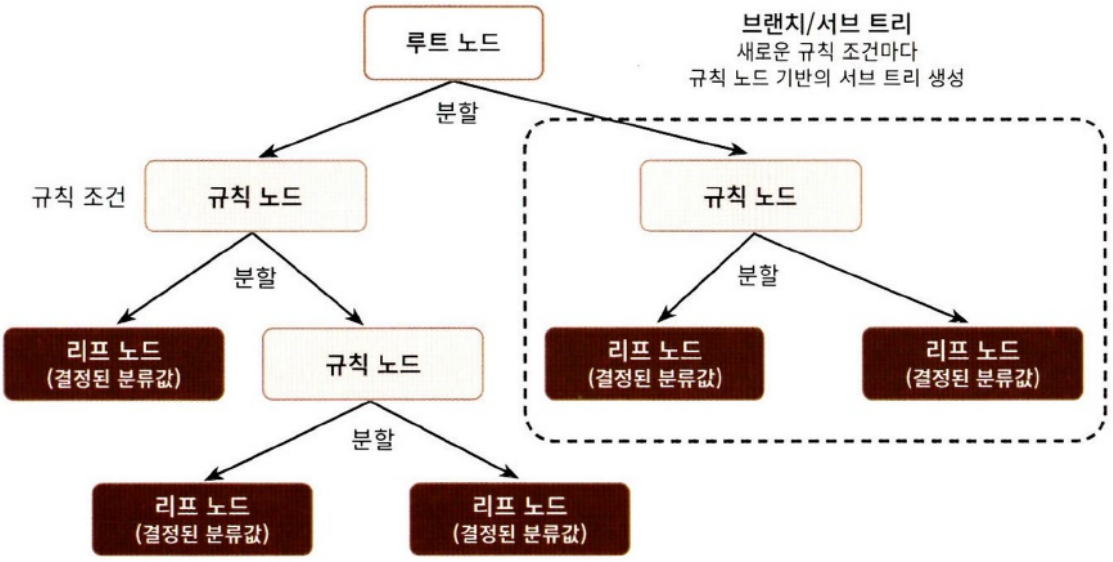
      - 베이즈(Bayes) 통계와 생성 모델에 기반한 나이브 베이즈(Naïve Bayes)
      - 독립변수와 종속변수의 선형 관계성에 기반한 로지스틱 회귀(Logistic Regression)
      - 데이터 균일도에 따른 규칙 기반의 결정 트리(Decision Tree)
      - 개별 클래스 간의 최대 분류 마진을 효과적으로 찾아주는 서포트 벡터 머신(Support Vector Machine)
      - 근접 거리를 기준으로 하는 최소 근접(Nearest Neighbor) 알고리즘
      - 심층 연결 기반의 신경망(Neural Network)
      - 서로 다른(또는 같은) 머신러닝 알고리즘을 결합한 앙상블(Ensemble)

- 앙상블 방법(Ensemble Method)
  - 정형 데이터의 예측 분석 영역에서 매우 높은 예측 성능으로 인해 많은 분석가와 데이터 과학자들에게 애용
  - 배깅(Bagging)/부스팅(Boosting) 방식
  - 근래 앙상블 방법은 부스팅 방식으로 지속적 발전
  - 그래디언트 부스팅(Gradient Boosting)
    - 뛰어난 예측 성능
    - 수행 시간이 오래 걸리는 단점으로 최적화 모델 튜닝이 어려운

- XgBoost(eXtra Gradient Boost)/ LightGBM
  - 기존 그래디언트 부스팅의 예측 성능을 한 단계 발전 + 수행 시간 단축 → 활용도 높음
- 대부분 동일한 알고리즘 결합 - 일반적으로 기본 알고리즘으로 결정 트리 사용
  - 결정 트리
    - 쉽고 유연하게 적용 가능
    - 데이터 스케일링이나 정규화 등의 사전 가공의 영향이 매우 적음
    - 예측 성능을 향상시키기 위해 복잡한 규칙 구조를 가져야 함 → 과적합(overfitting)이 발생해 예측 성능이 저하될 수 있음 → 앙상블 기법에서는 오히려 장점으로 작용
  - 앙상블은 많은 여러개의 약한 학습기를 결합해 확률적 보완과 오류가 발생한 부분에 대한 가중치를 계속 업데이트하며 예측 성능 향상 → 결정 트리가 좋은 약간 학습기가 됨

## 4.2 결정 트리

- 결정 트리(Decision Tree)
  - ML 알고리즘 중 직관적으로 이해하기 쉬운 알고리즘
  - 데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 트리(Tree) 기반의 분류 규칙을 만듬
  - 프로그램에 적용되는 if, else를 자동으로 찾아내 예측을 위한 규칙을 만드는 알고리즘
  - 데이터의 어떤 기준을 바탕으로 규칙을 만들어야 가장 효율적인 분류가 될 것인가 → 알고리즘의 성능 크게 좌우
- 결정 트리의 구조



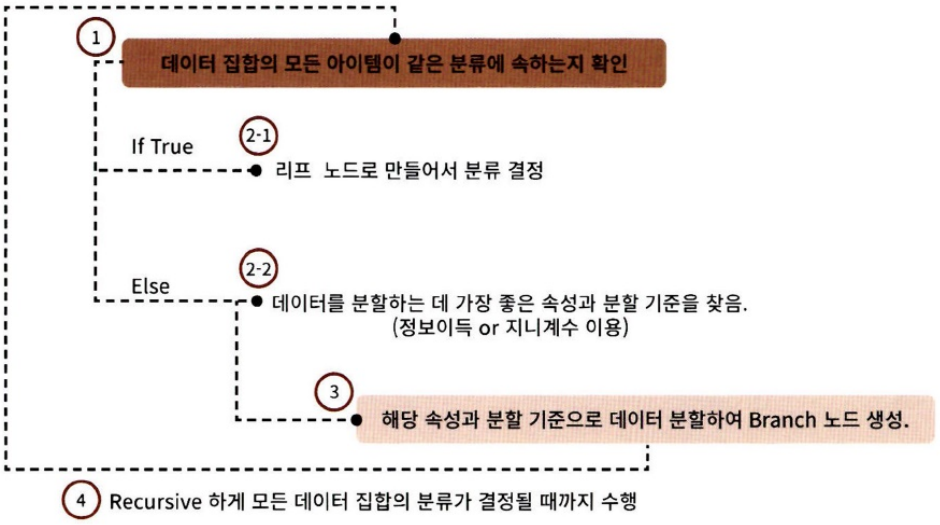

- 규칙 노드 (Decision Node) : 규칙 조건이 됨
- 리프 노트 (Leaf Node) : 결정된 클래스 값
- 새로운 규칙 조건마다 서브 트리 (Sub tree)가 생성됨
- 데이터 세트의 피처가 결합해 규칙 조건을 만들 때마다 규칙 노드가 만들어짐
  - 많은 규칙 → 분류 결정 방식이 복잡 → 과적합으로 이어지기 쉬움
- 트리의 깊이 (depth)가 깊어질수로 결정 트리의 예측 성능이 저하될 가능성이 높음
- 데이터 분류 시 최대한 많은 데이터 세트가 해당 분류에 속할 수 있도록 결정 노드의 규칙이 정해져야 함 → 가능한 한 적은 결정 노드로 높은 예측 정확도

- 정보 균일도
  - 데이터 세트의 균일도는 데이터를 구분하는 데 필요한 정보의 양에 영향
  - 결정 노드: 정보 균일도가 높은 데이터 세트를 먼저 선택할 수 있도록 규칙 조건을 만듬.
    - 정보 균일도가 데이터 세트로 쪼개질 수 있도록 조건을 찾아 서브 데이터 생성

    → 이 서브 데이터에서 균일도가 높은 자식 데이터 세트 쪼개는 방식을 자식 트리로 내려가면서 반복하며 데이터 값 예측
  - 정보 균일도를 측정하는 대표적인 방법
    - 엔트로피를 이용한 정보 이득(Information Gain) 지수
    - 지니 계수

- 정보 이득은 엔트로피라는 개념을 기반으로 합니다. 엔트로피는 주어진 데이터 집합의 혼잡도를 의미하는데, 서로 다른 값이 섞여 있으면 엔트로피가 높고, 같은 값이 섞여 있으면 엔트로피가 낮습니다. 정보 이득 지수는 1에서 엔트로피 지수를 뺀 값입니다. 즉, 1-엔트로피 지수입니다. 결정 트리는 이 정보 이득 지수로 분할 기준을 정합니다. 즉, 정보 이득이 높은 속성을 기준으로 분할합니다.
- 지니 계수는 원래 경제학에서 불평등 지수를 나타낼 때 사용하는 계수입니다. 경제학자인 코라도 지니(Corrado Gini)의 이름에서 딴 계수로서 0이 가장 평등하고 1로 갈수록 불평등합니다. 머신러닝에 적용될 때는 지니 계수가 낮을수록 데이터 균일도가 높은 것으로 해석해 지니 계수가 낮은 속성을 기준으로 분할합니다.

- DeicisionTreeClassifier - 결정 트리 알고리즘을 사이킷런에서 구현
  - 기본으로 지니 계수를 이용해 데이터 세트를 분할
  - 정보 이득이 높거나 지니 계수가 낮은 조건을 찾아 자식 트리 노드에 걸쳐 반복적으로 분할한 뒤, 데이터가 모두 특정 분류에 속하게 되면 분할을 멈추고 분류 결정




## [결정 트리 모델의 특징]

- 장점
  - 정보의 '균일도'라는 룰을 기반으로 하고 있어 알고리즘이 쉽고 직관적
  - 룰이 매우 명확
  - 이에 기반해 규칙 노드와 리프 노드가 만들어지는 방법을 알 수 있음
  - 시각화로 표현 가능
  - 정보의 균일도만 신경 쓰면 되므로 보통 각 피처의 스케일링과 정규화 같은 전처리 작업 불필요
- 단점
  - 과적합으로 정확도가 떨어짐

- 서브 트리 계속 생성 → 피처가 많고 균일도가 다양하게 존재할수록 트리의 깊이가 커지고 복잡해짐
- 복잡한 학습 모델 → 실제 상황(test data set)에 유연하게 대처 불가 → 예측 성능 떨어짐
- 보완
  - 트리의 크기를 사전에 제한하는 것이 성능 튜닝에 도움이 됨

## [결정 트리 파라미터]

- DecisionTreeClassifier
  - 사이킷런이 제공하는 결정 트리 알고리즘 - 분류를 위한 클래스
- DecisionTreeRegressor
  - 회귀를 위한 클래스
- 사이킷런의 결정 트리 구현은 CART(Classification And Regression Trees)알고리즘 기반
  - 분류, 회귀에서 사용 가능한 트리 알고리즘
- 파라미터

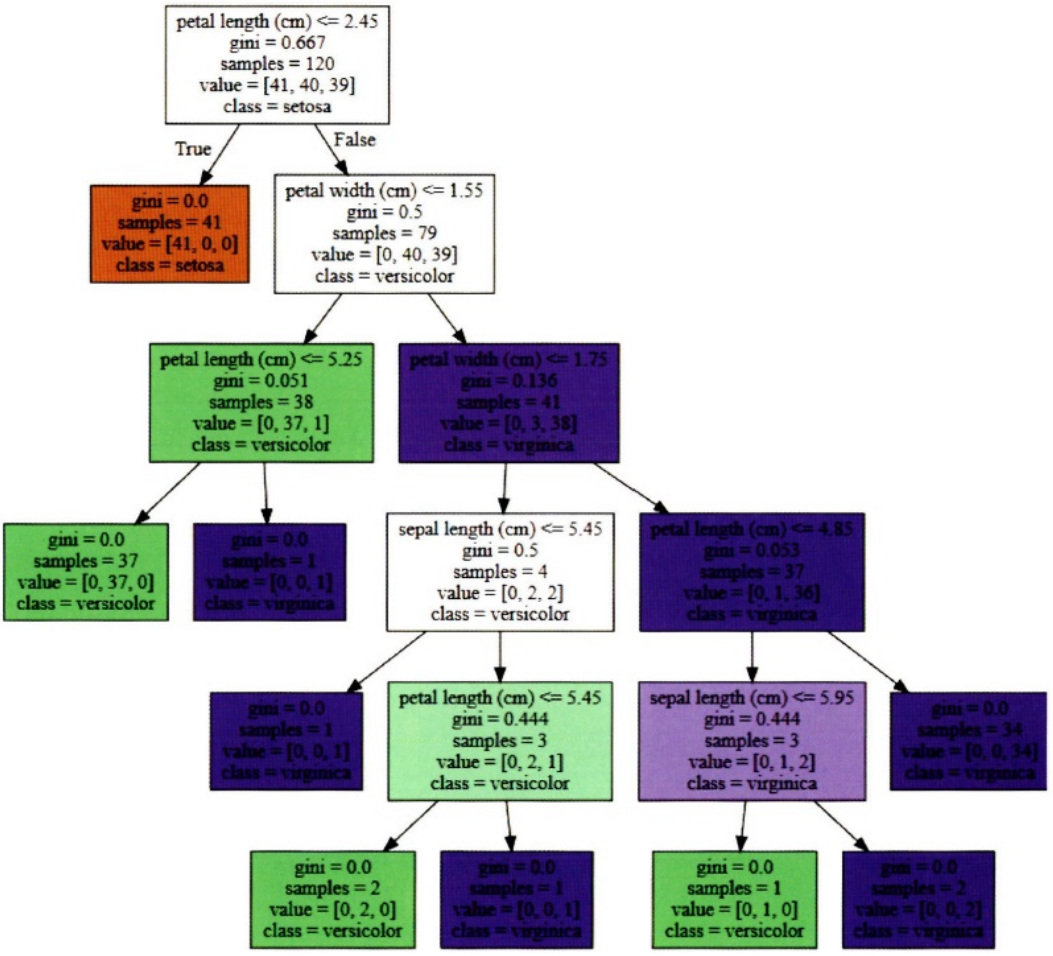
| 파라미터 명 | 설명 |
| --- | --- |
| min_samples_split | • 노드를 분할하기 위한 최소한의 샘플 데이터 수로 과적합을 제어하는 데 사용됨.<br>• 디폴트는 2이고 작게 설정할수록 분할되는 노드가 많아져서 과적합 가능성 증가<br>• 과적합을 제어. 1로 설정할 경우 분할되는 노드가 많아져서 과적합 가능성 증가 |
| min_samples_leaf | • 말단 노드(Leaf)가 되기 위한 최소한의 샘플 데이터 수<br>• Min_samples_split와 유사하게 과적합 제어 용도. 그러나 비대칭적(imbalanced) 데이터의 경우 특정 클래스의 데이터가 극도로 작을 수 있으므로 이 경우는 작게 설정 필요. |

| 파라미터 명 | 설명 |
| --- | --- |
| max_features | • 최적의 분할을 위해 고려할 최대 피처 개수. 디폴트는 None으로 데이터 세트의 모든 피처를 사용해 분할 수행.<br>• int 형으로 지정하면 대상 피처의 개수. float 형으로 지정하면 전체 피처 중 대상 피처의 퍼센트임<br>• 'sqrt'는 전체 피처 중 sqrt(전체 피처 개수). 즉 $\sqrt{\text{전체 피처 개수}}$만큼 선정<br>• 'auto'로 지정하면 sqrt와 동일<br>• 'log'는 전체 피처 중 log2(전체 피처 개수) 선정<br>• 'None'은 전체 피처 선정 |
| max_depth | • 트리의 최대 깊이를 규정.<br>• 디폴트는 None. None으로 설정하면 완벽하게 클래스 결정 값이 될 때까지 깊이를 계속 키우며 분할하거나 노드가 가지는 데이터 개수가 min_samples_split보다 작아질 때까지 계속 깊이를 증가시킴.<br>• 깊이가 깊어지면 min_samples_split 설정대로 최대 분할하여 과적합할 수 있으므로 적절한 값으로 제어 필요. |
| max_leaf_nodes | • 말단 노드(Leaf)의 최대 개수 |

## [결정 트리 모델의 시각화]

- Graphviz 패키지
  - 결정 트리 알고리즘이 어떠한 규칙을 가지고 트리를 생성하는지 시각적으로 보여줌
  - 그래프 기반의 dot 파일로 기술된 다양한 이미지를 쉽게 시각화할 수 있는 패키지
- export_graphviz( )
  - Graphviz 패키지와 쉽게 인터페이스할 수 있도록 사이킷런이 제공하는 API
  - 파라미터
    - 학습이 완료된 estimator, output 파일 명, 피처의 이름 리스트, 레이블 이름(결정 클래스의 명칭) 리스트
  - 파라미터 입력시 학습된 결정 트리 규칙을 실제 트리 형태로 시각화해 보여줌
  - Graphviz가 읽어 들여서 그래프 형태로 시각화할 수 있는 출력 파일 생성
- tree.dot
  - Graphviz의 파이썬 래퍼 모듈을 호출해 결정 트리의 규칙 시각적 표현 가능
- C/C++로 운영 체제에 포팅된 패키지 → 파이썬 기반의 모듈과 인터페이스 위해 Graphviz 설지 뒤 파이썬 Wrapper모듈 별도 설치 필요
  - 붓꽃 데이터 세트 DecisionTreeClassifier 예시

```
import graphviz
# 위에서 생성된 tree.dot 파일을 Graphviz가 읽어서 주피터 노트북상에서 시각화
with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```




- petal length(cm) <= 2.45와 같이 피처의 조건이 있는 것은 자식 노드를 만들기 위한 규칙 조건입니다. 이 조건이 없으면 리프 노드입니다.
- gini는 다음의 value=[ ]로 주어진 데이터 분포에서의 지니 계수입니다.
- samples는 현 규칙에 해당하는 데이터 건수입니다.
- value = [ ]는 클래스 값 기반의 데이터 건수입니다. 붓꽃 데이터 세트는 클래스 값으로 0, 1, 2를 가지고 있으며, 0 : Setosa, 1: Versicolor, 2: Virginica 품종을 가리킵니다. 만일 Value = [41, 40, 39]라면 클래스 값의 순서로 Setosa 41개, Vesicolor 40개, Virginica 39개로 데이터가 구성돼 있다는 의미입니다.

- 규칙에 따라 트리의 브랜치 노드와 말단 리프 노드의 구성을 한눈에 파악 가능
- 리프 노드:
  - 더 이상 자식 노드가 없는 노드

        - 최종 클래스(레이블) 값이 결정되는 노드
        - 하나의 클래스 값으로 최종 데이터가 구성되거나 리프 노드가 될 수 있는 하이퍼 파라미터 조건을 충족하면 됨
    - 브랜치 노드:
        - 자식 노드가 있는 노드
        - 자식 노드를 만들기 위한 분할 규칙 조건을 가짐
    - 각 노드의 색: 붓꽃 데이터의 레이블 값
        - 색이 짙어질수록 지니 계수가 낮고 해당 레이블에 속하는 샘플 데이터가 많음을 뜻함
- 결정 트리는 규칙 생성 로직을 미리 제어하지 않으면 완벽한 클래스 값 구별을 위해 트리 노드를 계속 만듬 → 매우 복잡한 규칙 트리 생성 → 모델 과적합 문제

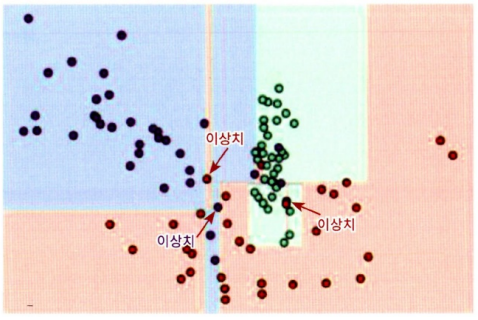
  ⇒ 결정 트리 알고리즘을 제어하는 대부분 하이퍼 파라미터는 복잡한 트리 생성을 막기 위한 용도
- max_depth
    - 결정 트리의 최대 트리 깊이 제어
- min_samples_splits
    - 자식 규칙 노드를 분할해 만들기 위한 최소한의 샘플 데이터 개수
    - 최소 샘플 데이터 개수보다 적은 경우 더 이상 자식 규칙 노드를 위한 분할을 하지 않고 리프 노드가 됨
- min_samples_leaf
    - 리프 노드가 될 수 있는 샘플 데이터 건수의 최솟값 지정
    - 값을 키우면 더 이상 분할하지 않고 리프 노드가 될 수 있는 조건 완화
        - min_samples_leaf ≤ 지정값
            - 기준 만족시 리프 노드가 될 수 있음
- 결정 트리는 균일도에 기반해 어떠한 속성을 규칙 조건으로 선택하느냐가 중요한 요건
    - 중요한 일부 피처가 명확한 규칙 트리를 만드는 데 크게 기여, 모델을 더 간결하고 이상치(Outlier)에 강한 모델을 만들 수 있음
- feature_importances_ 속성
    - 학습을 통해 규칙을 정하는 데 있어 피처의 중요한 역할 지표

- ndarray 형태로 값 반환, 피처 순서대로 값이 할당됨
- 값이 높을수록 해당 피처의 중요도가 높음
- 예시

```
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_, 3)))

# 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 column 별로 시각화하기
sns.barplot(x=dt_clf.feature_importances_, y=iris_data.feature_names)
```

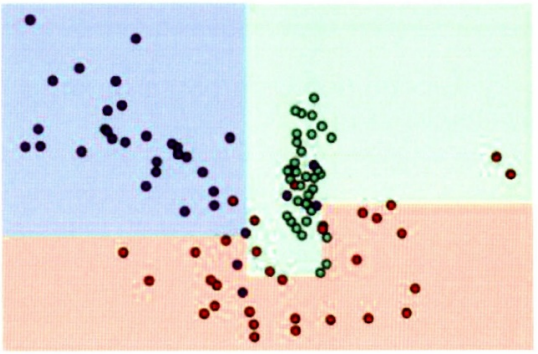
## [결정 트리 과적합(Overfitting)

- make_classification( )
  - 사이킷런에서 제공하는 분류를 위한 테스트용 데이터 생성 함수
  - 반환 객체: 피처 데이터 세트와 클래스 레이블 데이터 세트




- 일부 이상치 데이터까지 분류하기 위해 분할이 자주 일어나 결정 기준 경계가 매우 많아짐
- 결정 트리의 기본 하이퍼 파라미터 설정 - 리프 노드 내 데이터가 모두 균일하거나 하나만 존재해야 하는 엄격한 분할 기준 → 결정 기준 경계가 많아지고 복잡해짐
  - 학습 데이터 특성과 약간만 다른 형태의 데이터 세트를 예측하면 예측 정확도가 떨어짐
- 리프 노드 생성 규칙 완화 후 하이퍼 파라미터 변경시

```
# min_samples_leaf=6으로 트리 생성 조건을 제약한 결정 경계 시각화
dt_clf = DecisionTreeClassifier(min_samples_leaf=6).fit(X_features, y_labels)
visualize_boundary(dt_clf, X_features, y_labels)
```



- 이상치에 크게 반응하지 않으면서 좀 더 일반화된 분류 규칙에 따라 분류됨
- 트리 생성 조건을 제약한 모델의 예측 성능이 더 뛰어날 가능성이 있음
  - 학습 데이터에만 지나치게 최적화된 분류 기준은 오히려 테스트 데이터 세트에서 정확도를 떨어뜨릴 수 있기 때문

### [결정 트리 실습 - 사용자 행동 인식 데이터 세트]

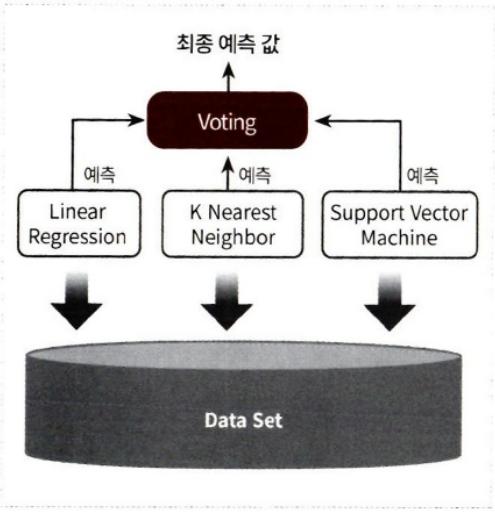
- max_depth 값에 따른 예측 성능 변화
  - cv_results_ 속성
    - GridSearchCV의 CV세트에 하이퍼 파라미터를 순차적으로 입력했을 때의 성능 수치
  - 깊어진 트리는 검증 데이터 세트에서는 오히려 과적합으로 인한 성능 저하를 유발
  - 복잡한 모델보다도 트리 깊이를 낮춘 단순한 모델이 더욱 효과적인 결과를 가져올 수 있음

## 4.3 앙상블 학습

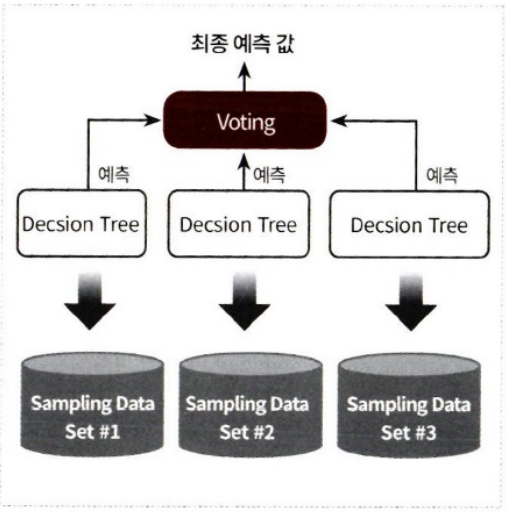
### [앙상블 학습 개요]

- 앙상블 학습(Ensemble Learning)을 통한 분류
  - 여러 개의 분류기(Classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법
  - 다양한 분류기의 예측 결과를 결합하여 단일 분류기보다 신뢰성이 높은 예측값 획득 목표
  - 이미지, 영상, 음성 등의 비정형 데이터 분류 - 딥러닝

    - 대부분의 정형 데이터 분류 시 앙상블이 뛰어난 성능 나타냄
    - 쉽고 편하면서도 강력한 성능 보유
- 보티(Voting), 배깅(Bagging)
    - 여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정하는 방식
    - 보팅: 일반적으로 서로 다른 알고리즘을 가진 분류기 결합
    - 배깅: 각각의 분류기가 모두 같은 유형의 알고리즘 기반이지만, 데이터 샘플링을 서로 다르게 가져가면서 학습을 수행해 보팅을 수행하는 것
        - 대표적인 배깅 방식 - 랜덤 포레스트 알고리즘



Voting 방식



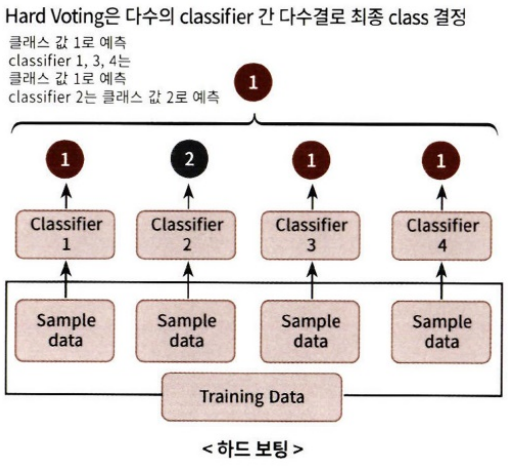
Bagging 방식

- 배깅 앙상블 방식
    - 부트스트래핑 분할 방식 (Bootstrapping)
        - 개별 Classifier에게 데이터를 샘플링해서 추출하는 방식
    - 개별 분류기가 부트스트래핑 방식으로 샘플링된 데이터 세트에 대해 학습을 통해 개별적인 예측을 수행한 결과를 보팅을 통해 최종 예측 결과를 선정하는 방식
    - 데이터 세트 간의 중첩을 허용
- 부스팅
    - 여러 개의 분류기가 순차적으로 학습을 수행하되, 앞에서 학습한 분류기가 예측이 틀린 데이터에 대해서는 올바르게 예측할 수 있도록 다음 분류기에게는 가중치(weight)를 부여하면서 학습과 예측을 진행하는 것
    - 대표적인 부스팅 모듈

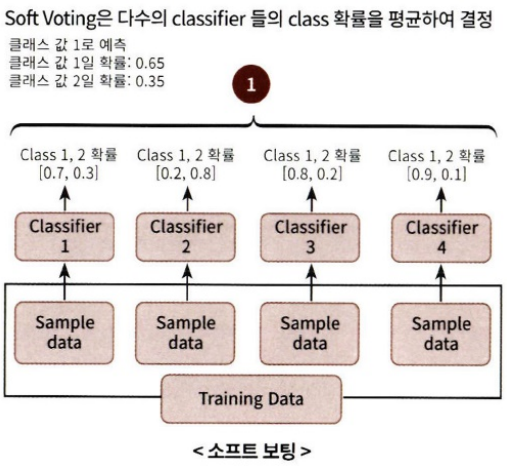
        - 그래디언트 부스트, XGBoost(eXtrai Gradient Boost), LightGBM(Light Gradient Boost)
- 스태킹
    - 여러가지 다른 모델의 예측 결과값을 다시 학습 데이터로 만들어 다른 모델(메타 모델)로 재학습시켜 결과를 예측하는 방법

## [보팅 유형 - 하드 보팅(Hard Voting)과 소프트 보팅(Soft Voting)]

- 하드 보팅
    - 예측한 결과값들 중 다수의 분류기가 결정한 예측값을 최종 보팅 결과값으로 선정하는 것
- 소프트 보팅
    - 분류기들의 레이블 값 결정 확률을 모두 더하고 이를 평균해서 확률이 가장 높은 레이블 값을 최종 보팅 결과값으로 선정
    - 일반적인 보팅 방법



< 하드 보팅 >



< 소프트 보팅 >

## [보팅 분류기(Voting Classifier)]

- VotingClassifier 클래스
    - 사이킷런이 제공하는 보팅 방식의 앙상블을 구현한 클래스
    - 보팅 분류기 생성 가능
    - 주요 인자
        - estimators: 리스트 값으로 보팅에 사용될 여러 Classifier 객체들을 튜플 형식으로 입력
        - voting: ‘hard’-하드보팅(default), ‘soft’-소프트보팅

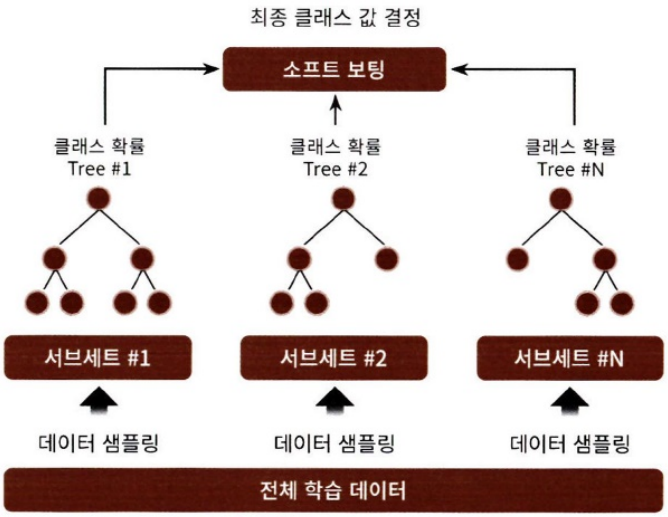
- load_breast_cancer( ) 함수
  - 위스콘신 유방암 데이터 세트 생성 함수
- 보팅으로 여러개의 기반 분류기를 결합한다고 해서 무조건 기반 분류기보다 예측 성능이 향상되는 것은 아님
  - 데이터 특성, 분포 등 다양한 요건에 따라 오히려 기반 분류기 중 가장 좋은 분류기의 성능이 보팅 시보다 나을 수 있음
  - but 보팅, 배깅, 부스팅 등의 앙상블 방법 - 전반적으로 다른 단일 ML 알고리즘보다 뛰어난 예측 성능을 가지는 경우가 많음
- ML 모델의 중요 평가요소
  - 어떻게 높은 유연성을 가지고 현실에 대처할 수 있는가
  - 편향 - 분산 트레이드오프는 ML이 극복해야 할 중요 과제
  - 배깅, 부스팅은 대부분 결정 트리 알고리즘 기반
    - 결정 트리 알고리즘 - 과적합 발생해 실제 테스트 데이터에서 예측 성능이 떨어지는 현상 다수 발생
    - 앙상블 학습 - 많은 분류기를 결합해 다양한 상황을 학습 → 결정 트리 단점 극복

      ⇒ 장점을 취하고 단점은 보완하여 편향 - 분산 트레이드오프 효과 극대화 가능

# 4.4 랜덤 포레스트

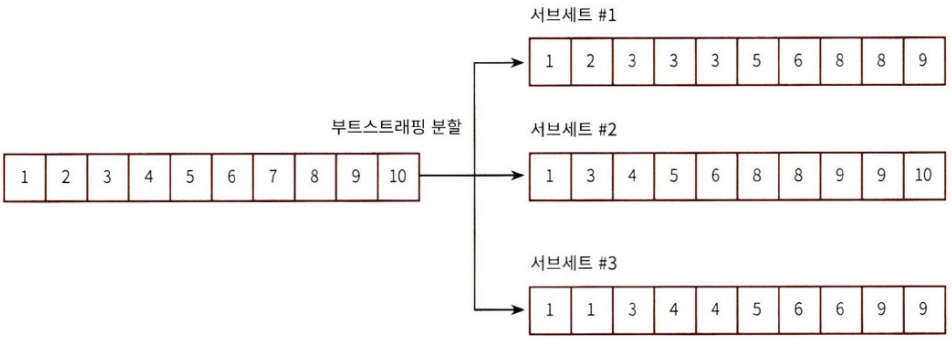
## [랜덤 포레스트의 개요 및 실습]

- 배깅(Bagging = Bootstrap aggregating)
  - 같은 알고리즘으로 여러 개의 분류기를 만들어 보팅으로 최종 결정하는 알고리즘
  - 대표적인 알고리즘: 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
  - 앙상블 알고리즘 중 비교적 빠른 수행 속도
  - 다양한 영역에서 높은 예측 성능
  - 결정 트리 알고리즘 기반 → 결정 트리의 쉽고 직관적인 장점 가짐




〈 랜덤 포레스트 〉

- 개별 트리가 학습하는 데이터 세트는 전체 데이터에서 일부가 중첩되게 샘플링된 데이터 세트 = 부트스트래핑(Bootstrapping)
  - 부트스트랩 : 여러개의 작은 데이터 세트를 임의로 만들어 개별 평균의 분포도를 측정하는 등의 목적을 위한 샘플링 방식
- 램덤 포레스트의 서브세트(Subset)
  - 부트스트래핑으로 데이터가 임의로 만들어짐
  - 전체 데이터 건수와 동일
  - 개별 데이터가 중첩되어 만들어짐




〈 부트스트래핑 샘플링 방식 〉

- RandomForestClassifier 클래스
  - 사이킷런에서 랜덤 포레스트 기반의 분류 지원

## [랜덤 포레스트 하이퍼 파라미터 및 튜닝]

- 트리 기반의 앙상블 알고리즘의 단점
  - 하이퍼 파라미터가 너무 많음 → 튜닝을 위한 시간이 많이 소모됨

- 많은 시간 소모 후에도 튜닝 후 예측 성능이 크게 향상되는 경우가 많지 않음

  - n_estimators: 랜덤 포레스트에서 결정 트리의 개수를 지정합니다. 디폴트는 10개입니다. 많이 설정할수록 좋은 성능을 기대할 수 있지만 계속 증가시킨다고 성능이 무조건 향상되는 것은 아닙니다. 또한 늘릴수록 학습 수행 시간이 오래 걸리는 것도 감안해야 합니다.
  - max_features는 결정 트리에 사용된 max_features 파라미터와 같습니다. 하지만 RandomForestClassifier의 기본 max_features는 'None'이 아니라 'auto', 즉 'sqrt'와 같습니다. 따라서 랜덤 포레스트의 트리를 분할하는 피처를 참조할 때 전체 피처가 아니라 sqrt(전체 피처 개수)만큼 참조합니다(전체 피처가 16개라면 분할을 위해 4개 참조).
  - max_depth나 min_samples_leaf와 같이 결정 트리에서 과적합을 개선하기 위해 사용되는 파라미터가 랜덤 포레스트에도 똑같이 적용될 수 있습니다.

- 랜덤 포레스트는 CPU 병렬 처리도 효과적으로 수행되어 빠른 학습이 가능 → 그래디언트 부스팅보다 예측 성능이 떨어지더라도 랜덤 포레스트로 일단 기반 모델을 구축하는 경우가 많음