결정 트리

Decision Tree

<1조> 황예은 / 정세웅 / 한형진

01
결정트리의 정의 및 개념

02
분기, 불순도, 정보 획득량

03
사전가지치기, 비용함수

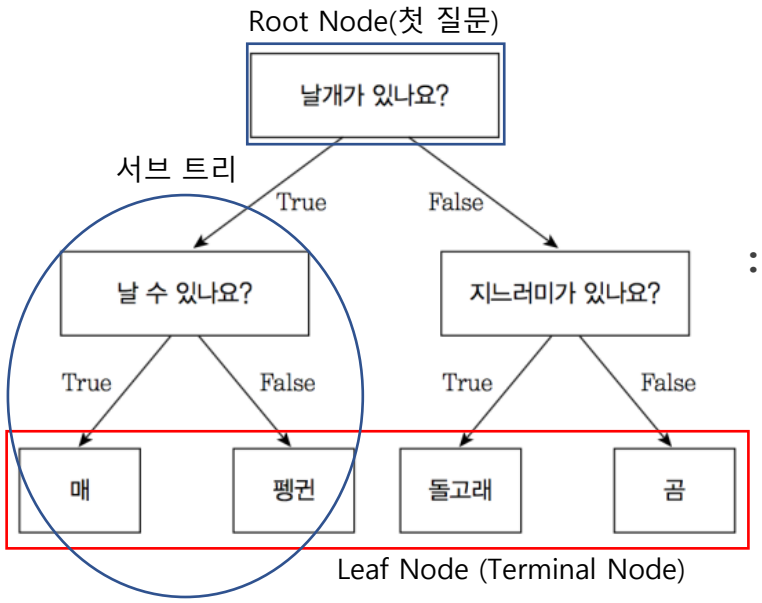
01
feature_importances

결정 트리

결정 트리 ?

분류와 회귀 모두 가능한 지도 학습의 한 방법으로 질문을 반복해 나가면서 학습

가장 적은 질문으로 가장 빨리 정답에 도달할 수 있도록 하는 그리디 알고리즘 (매 질문마다 최선의 질문 선택)



노드 Node

: 트리 내 질문 또는 정답

1. Root Node
2. Intermediate Node
3. Leaf Node

분류

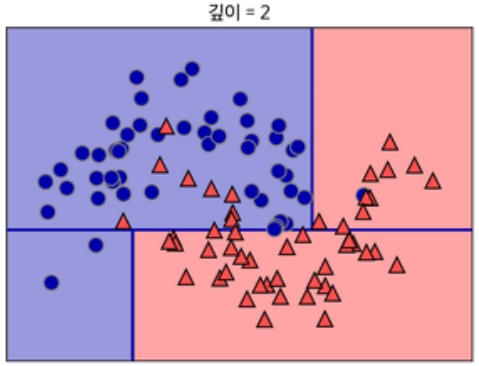
: 새로운 데이터 포인트가 분할된 리프 노드 중 어디에 속하는지

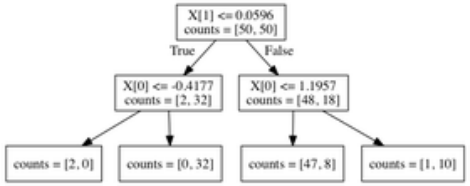
리프 노드내 주된(majority) 클래스를 예측값으로 return

회귀

: 새로운 데이터 포인트가 속한 리프 노드의 훈련 데이터의 평균 값

외삽 : 트리가 결정되면 리프 노드에 속한 훈련 데이터들도 정해지기 때문에 예측할 수 있는 회귀 값들은 훈련데이터 범위를 벗어날 수 없음



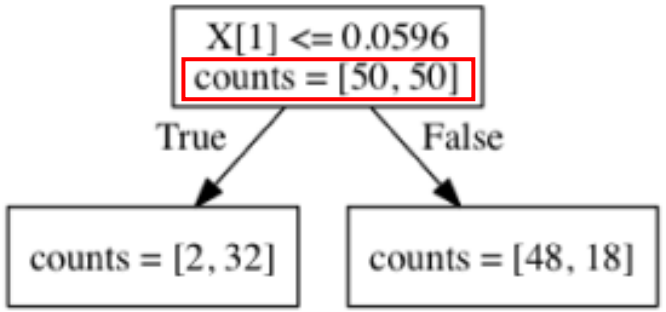


분기

: 질문에 따라 데이터 분할

매 분기마다 하나의 특성에 대해서만 이루어지므로 나누어진 영역은
항상 축에 평행

ex. 이진 분류(양성/음성)



+1 depth

- **불순도(Impurity)**

  : 분기 기준을 선택하기 위해 사용하는 개념으로 복잡성을 의미

  =해당 범주안에 서로 다른 데이터가 섞여 있는 정도

  (같은 비율로 존재할 경우 가장 불순도가 높음)

$$p_k : \frac{\text{클래스 } k \text{ 데이터 수}}{\text{노드 내 데이터 수}}$$

지니(Gini) 지수

$$I(A) = 1 - \sum_{k=1}^{m} p_k^2$$
$$= 1 - \left(\frac{6}{16}\right)^2 - \left(\frac{10}{16}\right)^2$$
$$\approx 0.47$$

- 준수한 성능에 빠른 계산

엔트로피

$$\text{Entropy} = -\sum_i (p_i) \log_2 (p_i)$$

- 지니보다 잘 분류하지만 log 계산으로 오래 걸림

- **정보 획득(Information gain)**

  : 현재(부모) 노드의 불순도 – [가중치 평균]자식 노드의 불순도

  - 분기 기준은 현재 노드의 불순도에 비해 자식 노드의 불순도가 감소되도록 설정

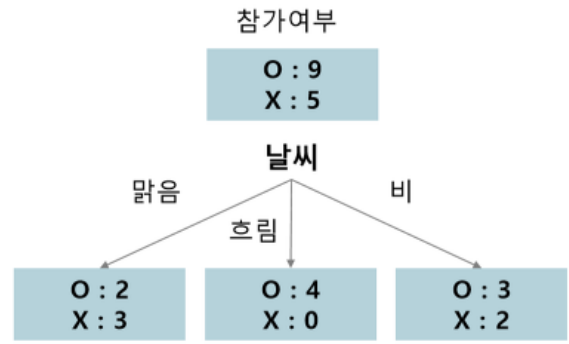
  - 현재 노드에서 정보 획득량이 가장 많은 방향으로 그리디(Greedy)하게 분기 기준 설정

  - [가중치 평균] :

  분기를 하면 부모 노드는 2개 이상의 자식 노드로 분할되기 때문

✓ 자식 노드N 가중치 : $\frac{\text{자식 노드 } N \text{으로 분할된 데이터 수}}{\text{부모 노드 데이터 수}}$

$$E(\text{경기}|\text{날씨}) = \frac{5}{14}\left(-\frac{2}{5}log_2\left(\frac{2}{5}\right) - \frac{3}{5}log_2\left(\frac{3}{5}\right)\right)$$
$$+\frac{4}{14}\left(-\frac{4}{4}log_2\left(\frac{4}{4}\right) - \frac{0}{4}log_2\left(\frac{0}{4}\right)\right)$$
$$+\frac{5}{14}\left(-\frac{3}{5}log_2\left(\frac{3}{5}\right) - \frac{2}{5}log_2\left(\frac{2}{5}\right)\right)$$
$$= 0.694$$



- **회귀 트리 분기는?**

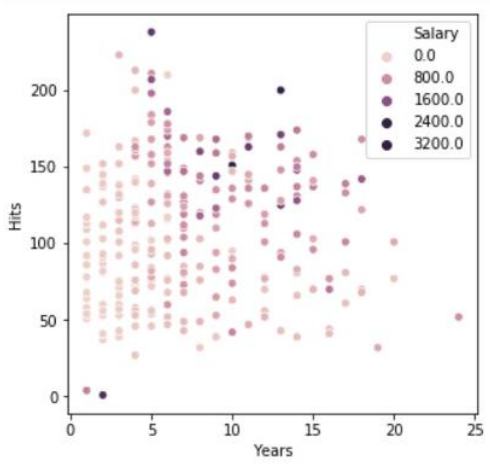
$$R_1(j,s) = \{X|X_j < s\} \; and \; R_2(j,s) = \{X|X_j \geq s\}$$

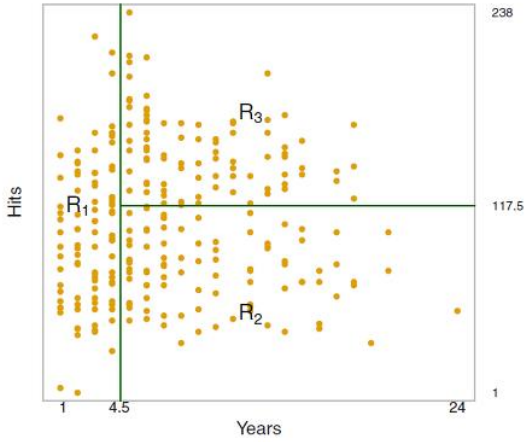
✓SSE (오차 제곱 합) 이 최소가 되는 j,s를 구한다

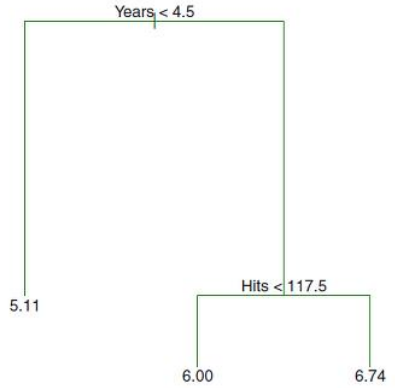
$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

y_i : 해당 노드에 속하는 y의 실제 값

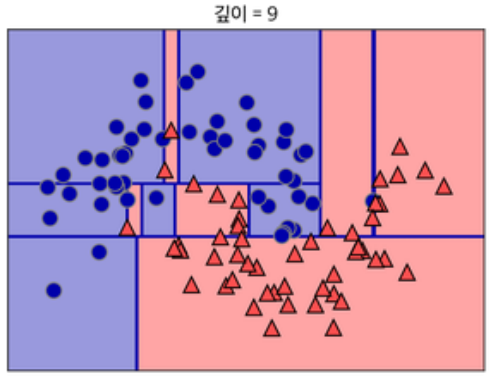
$\hat{y}_{R_1}$: 해당 노드에 속하는 y들의 평균값



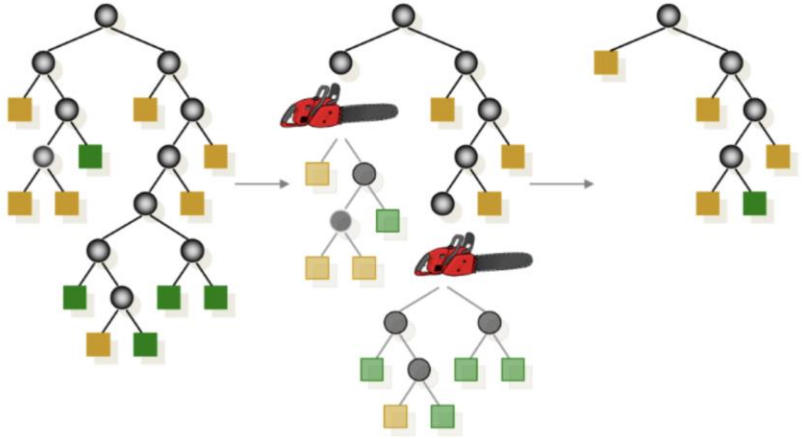




가지치기(Pruning)



Default 상태의 트리 모델은 모든 리프 노드가 순수 노드, 즉 불순도가 0인 노드가 될 때까지 분기
과적합이 일어나 일반화 성능이 떨어짐
→ 적절한 규제 필요 = **가지치기**



- 사전 가지치기
  : 트리 생성 과정에 제한을 둠으로써 일찍 중단

✓ max_depth (=tree의 최대 깊이를 제한, 질문 개수와 같다 / default : None)
✓ min_sample_leaf (=리프 노드가 되기 위한 최소 샘플 개수 / default : 2)
: 클수록 리프 노드가 되기 쉬워 과적합 방지
✓ max_leaf_nodes (=리프 노드의 최대 개수 / deault = None)
: 리프 노드가 많아지는 것을 막아줌
✓ min_sample_split (=분할하기 위한 최소 샘플 개수 / default : 2)
: 노드 샘플의 개수가 설정한 값 이하가 되면 분기를 멈춤
✓ max_features (=고려할 최대 feature 개수 / default : None)
: 'auto', 'sqrt', 'log2'

- 사후 가지치기
  : 트리 생성 후 데이터 포인트가 적은 노드 삭제 또는 병합

➢ 사이킷런에서는 사전가지치기만 지원했었음
➢ 사이킷런 0.22버전
: 비용 복잡도 기반의 사후 가지치기를 위한 ccp_alpha 매개변수 추가

사후 가지치기 – CCP(Cost Complexity Pruning)

- 결정 트리 - 회귀
  → 성능 측도 : 오차 제곱 합(SSE)
- 결정 트리 – 분류
  → 성능 측도 : 오분류율(ERR)

1. 가장 낮은 불순도율 방향으로 트리 생성
   → 훈련 데이터에 과적합
2. 사후 가지치기
   → 오분류 비용이 가장 낮은 곳을 가지치기
   → 하부 트리(sub tree)의 비용 복잡도가 더 큰 경우만

(오분류 비용 = 오분류율 * 노드t까지 오는 확률)

- **가지치기 비용함수**

✓ **CC(T) = ERR(T) + α × L(T)**

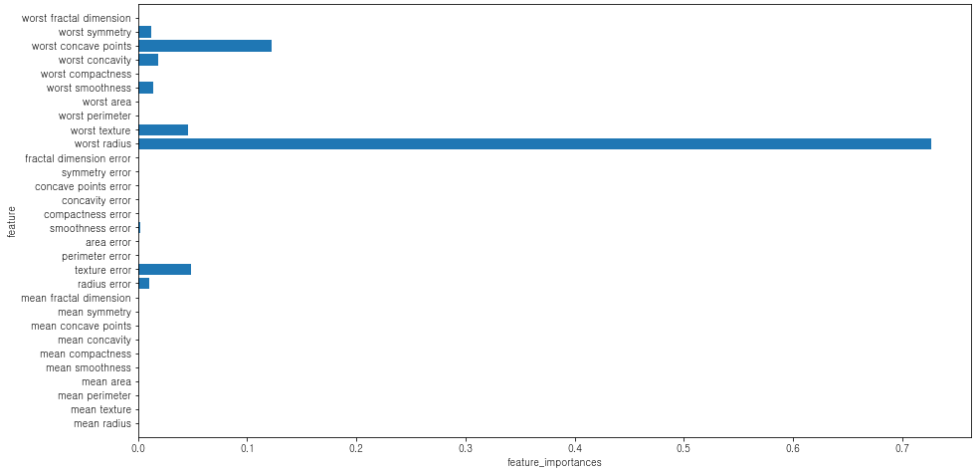
- **CC(T)** (= Cost Complexity)
  - 의사결정나무의 비용 복잡도
  - 오류가 적으면서 leaf node 수가 적은 단순한 모델일 수록 작은 값
- **ERR(T)**
  - 검증데이터에 대한 오분류율(≒불순도)
- **L(T)**
  - leaf node의 수
  - 구조의 복잡도
- **Alpha**
  - ERR(T)와 L(T)를 결합하는 가중치
  - 사용자에 의해 부여됨, 보통 0.01~0.1의 값을 씀

✓ CC(T)를 최소화 = 오분류율 최소화 + 트리 복잡도 최소화

특성 중요도 (feature importances)

- feature_importnaces

  : 트리를 만드는 결정에 각 특성이 얼마나 중요한지를 평가하는 값(0~1)



- 항상 양수이며 모든 특성들의 중요도를 합치면 1이다.
- 특성이 어떤 클래스를 지지하는지 알 수 없다.
- 시각화가 가능하여 직관적이다.
- feature selection에 사용될 수 있다.

- feature_importance base on
  - gini
  - entropy
  - weight

- feature_importance의 단점 : bias될 가능성이 높음

  ex) A변수 : 0/1

  B변수 : 1/2/3/4/5

  A보다 B변수의 활용도가 더 높음(cardinality 따라 bias됨)

  → permutation importance : 변수의 순서를 무작위로 섞은 뒤, 판단하려고 하는 변수를 noise 처리.

  모델이 해당 변수에 대한 의존도가 높을 수록 설명력은 감소

결정트리의 장단점

•장점

① 데이터의 전처리 (정규화, 결측치, 이상치 등) 를 하지 않아도 된다.

② 수치형과 범주형 변수를 한꺼번에 다룰 수 있다.

③ 모델을 쉽게 시각화 할 수 있어서 이해가 쉽다.

• 단점

① 한 번에 하나의 변수만을 고려하므로 변수간 상호작용을 파악하기가 어렵다.

② 일반적인 Greedy 방식의 알고리즘이 그렇듯이 이 방식은 최적의 해를 보장하지는 못한다.

③ 약간의 차이에 따라 트리의 모양이 많이 달라질 수 있다.

- 두 변수가 비슷한 수준의 정보력을 갖는다고 했을 때, 약간의 차이에 의해 다른 변수가 선택되면 이 후의 트리 구성이 크게 달라질 수 있다.

✓ 이러한 결정 트리의 단점을 보완하는 Bagging, Boost 방법의 트리 알고리즘이 나옴
ex) RandomForset, Extra Tree, Gradient Boost(LightGBM, XGBoost, CatBoost) . . .

실습 코드

1) 결정트리 학습 & 예측

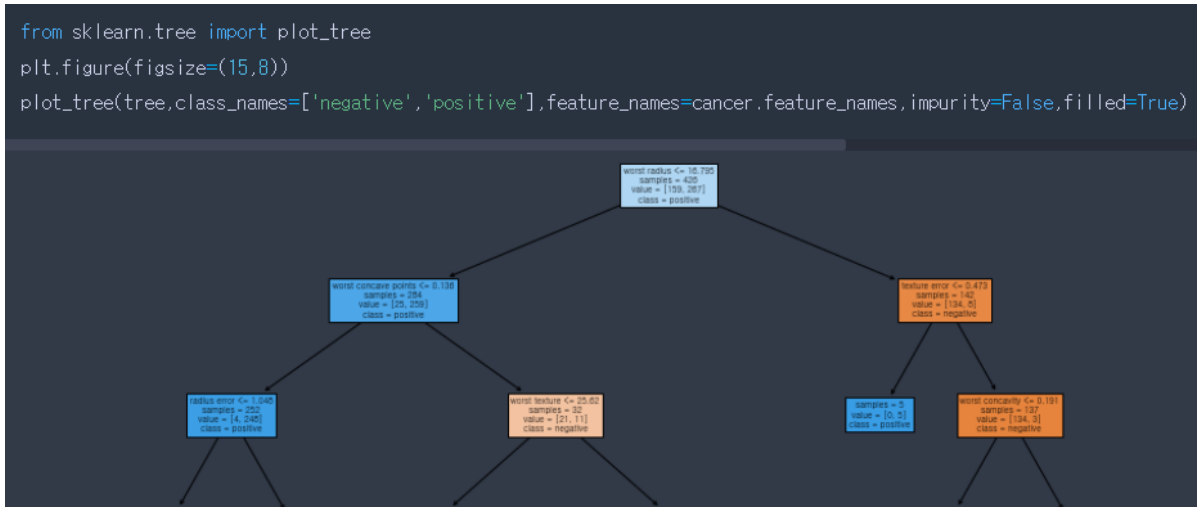
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer=load_breast_cancer()
X_train,X_test,y_train,y_test=train_test_split(cancer.data,cancer.target,stratify=cancer.target)
tree=DecisionTreeClassifier(random_state=0).fit(X_train,y_train)
print('훈련 세트 점수 : {:.2f}'.format(tree.score(X_train,y_train)))
print('테스트 세트 점수 : {:.2f}'.format(tree.score(X_test,y_test)))
```

```
훈련 세트 점수 : 1.00
테스트 세트 점수 : 0.94
```

2) 생성된 트리 Plot

```
from sklearn.tree import plot_tree
plt.figure(figsize=(15,8))
plot_tree(tree,class_names=['negative','positive'],feature_names=cancer.feature_names,impurity=False,filled=True)
```



3) feature importance plot

```
plt.figure(figsize=(15,8))
n_features=cancer.data.shape[1]
plt.barh(np.arange(n_features),tree.feature_importances_,align='center') #align default ='center'/ 'edge'-> 눈금 끝에
plt.yticks(np.arange(n_features),cancer.feature_names)
plt.xlabel('feature_importances')
plt.ylabel('feature')
plt.ylim(-1,n_features) #-1인 이유 : 첫번째 특성인 mean radius가 맨 바닥이 있지 않게 하기 위해
```