의사결정트리

- **01** 의사결정트리의 이해
- **02** 의사결정트리 알고리즘
- **03** 의사결정트리의 확장
- **04** 의사결정트리 알고리즘의 다양한 변형
- **05** 의사결정트리의 구현

학습목표

1. 의사결정트리와 엔트로피의 개념이해
2. 정보이득에 대해 학습하고, ID3 알고리즘을 활용하여 의사결정트리 표현
3. C4.5 알고리즘과 지니 지수 학습
4. 트리 가지치기, 연속형 변수 나누기, 회귀 트리 등 의사결정트리 알고리즘의 다양한 변형에 대해 학습
5. 의사결정트리 구현

01
의사결정트리의 이해

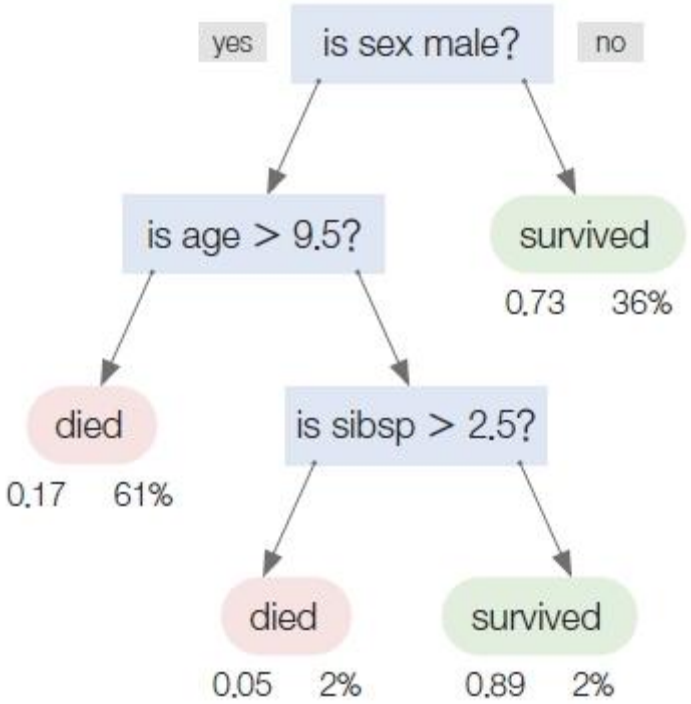
1. 의사결정트리의 개념

- 의사결정트리(decision tree) : 어떤 규칙을 하나의 트리(tree) 형태로 표현한 후 이를 바탕으로 분류나 회귀 문제를 해결
  - 규칙은 'if-else' 문으로 표현이 가능
  - 트리는 일종의 경로를 표현하는 것
  - 트리 구조의 마지막 노드에는 분류 문제에서 클래스, 회귀 문제에서는 예측치가 들어감

01 의사결정트리의 이해


```
if age > 30:
    return True
else:
    return False
```

(a) if-else문의 예



(b) if-else문의 경로 표현

그림 12-1 의사결정트리의 이해

01 의사결정트리의 이해

그림 12-2 아키네이터(akinator) 게임 © https://kr.akinator.com/

01 의사결정트리의 이해

- 의사결정트리는 딥러닝 기반을 제외한 전통적인 통계 기반의 머신러닝 모델 중 효과와 실용성이 가장 좋음
  - 테이블형 데이터에 있어 설명력과 성능의 측면에서 딥러닝 모델들과 대등하게 경쟁
  - 앙상블(ensemble) 모델이나 부스팅(boosting) 같은 새로운 기법들이 모델들의 성능을 대폭 향상시키고 있음

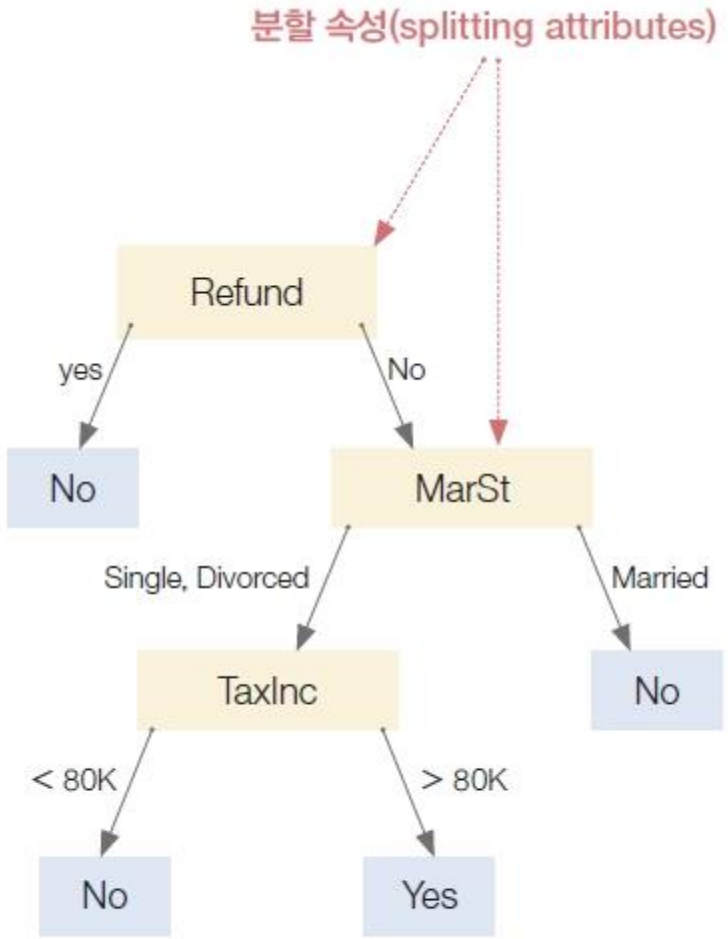
2. 의사결정트리 분류기

- 의사결정트리의 노드(node) 구성이 가장 중요
- 마지막 노드에 클래스나 예측치를 기입하고 상위의 부모 노드들에는 if-else문의 조건에 해당하는 정보 기입
- 분할 속성(splitting attributes) : 부모 노드에 들어가는 if-else문의 조건들
  - 어떤 분할 속성이 가장 모호성을 줄일 것인지 파악
    - 예시: 1부터 100까지의 숫자 중 하나를 맞추는 '숫자 예측 게임'
  - '재귀적 지역 최적화 방법' : 첫 문제로 분할 속성을 설정하고, 그 다음 남은 데이터 속에서 최적의 분할 속성을 찾아냄

01 의사결정트리의 이해

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

(a) 훈련 데이터(training data)



(b) 의사결정트리(decision tree)

그림 12-3 의사결정트리 만들기

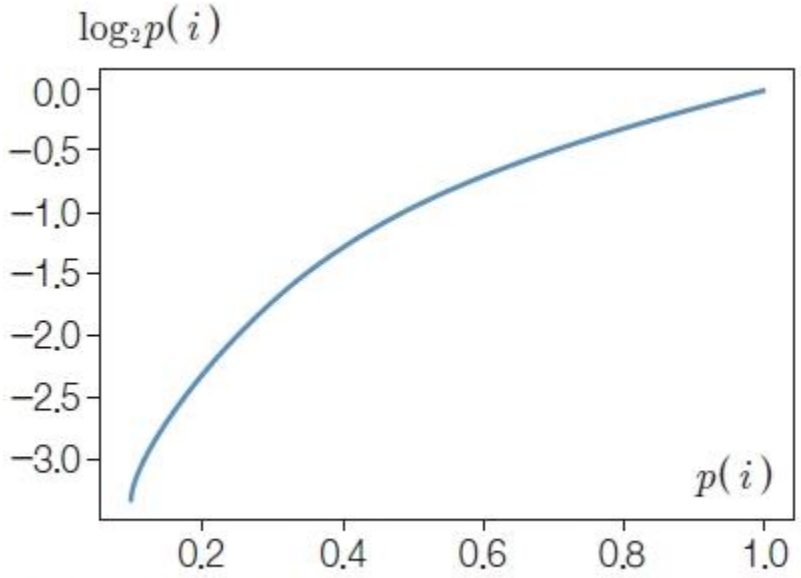
3. 엔트로피의 이해

- 엔트로피 : 어떤 목적 달성을 위한 경우의 수를 정량적으로 표현하는 수치
  - 현재의 정보 제공 상태를 측정
  - 어떤 분할 속성을 선택하였을 때 정보를 제공하는 기준 값을 정하고, 그 값을 최소화 또는 최대화하는 방향으로 알고리즘 실행
- 낮은 엔트로피 = 경우의 수가 적음 = 낮은 불확실성
- 높은 엔트로피 = 경우의 수가 높음 = 높은 불확실성

01 의사결정트리의 이해

- 엔트로피를 측정하는 방법 : 샤논(Shannon, Claude Elwood)이라는 공식을 사용

$$h(D) = -\sum_{i=1}^{m} p_i \log_2(p_i) \qquad \text{where} \begin{cases} D & \text{Data set} \\ p_i & \text{Probability of label } i \end{cases}$$



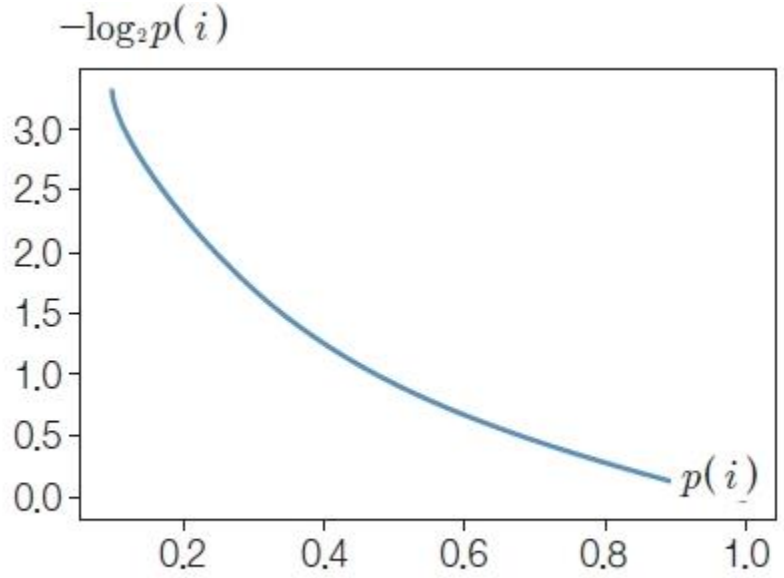


그림 12-4 샤논의 공식을 그래프로 표현

4. 엔트로피의 예시

표 12-1 예시 데이터

No	age	income	student	credit_rating	class_buys_computer
0	youth	high	no	fair	no
1	youth	high	no	excellent	no
2	middle_aged	high	no	fair	yes
3	senior	medium	no	fair	yes
4	senior	low	yes	fair	yes
5	senior	low	yes	excellent	no
6	middle_aged	low	yes	excellent	yes
7	youth	medium	no	fair	no
8	youth	low	yes	fair	yes
9	senior	medium	yes	fair	yes
10	youth	medium	yes	excellent	yes
11	middle_aged	medium	no	excellent	yes
12	middle_aged	high	yes	fair	yes
13	senior	medium	no	excellent	no

01 의사결정트리의 이해

- 사람의 나이, 소득, 학생 여부, 신용 등급 등을 고려하여 컴퓨터를 구매할지 구매하지 않을지 의사결정
- 컴퓨터를 구매할 확률 $\frac{9}{14}$, 구매하지 않을 확률 $\frac{5}{14}$

$$h(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

$$h(d) = -\frac{9}{14}\log_2\frac{9}{14} + \left(-\frac{5}{14}\log_2\frac{5}{14}\right) = 0.940$$

02
의사결정트리 알고리즘

1. 정보 이득

- 정보 이득(information gain) : 엔트로피를 사용하여 속성별 분류 시 데이터가 얼마나 순수한지(impurity)를 측정하는 지표
  - 각 속성을 기준으로 데이터를 분류했을 때 엔트로피를 측정

전체 엔트로피 - 속성별 엔트로피 = 속성별 정보 이득
❶ ❷ ❸

① 전체 엔트로피

$$Info(D) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

② 속성별 엔트로피

$$Info_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

- 속성 A로 데이터를 분류했을 때 속성 A가 가진 모든 클래스의 각 엔트로피를 계산한 후, 데이터의 개수만큼 가중치를 줌

③ 속성별 정보 이득 :

$$Gain(A) = Info(D) - Info_A(D)$$

- 정보 이득이 크면 클수록 A를 기준으로 데이터를 분류했을 때 얻을 수 있는 정보량이 많다는 뜻
- A를 기준으로 데이터를 나눌 때 엔트로피가 작다면 해당 속성을 기준으로 데이터를 나누기 좋다고 볼 수 있음

2. ID3 알고리즘을 활용한 의사결정트리의 성장

- 성장(grow) : 일반적으로 의사결정트리를 생성하는 방법을 성장이라고 부름. 트리(나무)를 성장시키는 개념
- ID3(Iterative Dichotomiser 3) : 반복적으로(iteratively) 데이터를 나누는(divides) 알고리즘
  - 톱다운(top-down) 방식으로 데이터를 나누면서 탐욕적(greedy)으로 현재 상태에서 최적화를 추진하는 방법을 선택

02 의사결정트리 알고리즘

- 기본적인 ID3 알고리즘

```
if 데이터 집합에 있는 모든 항목이 같은 레벨임 :
  분류 항목 표시를 반환함(ex. buy_yes)
else :
  Find Best Split_branch_attribute(ex, attribute-age)
  해당 속성(attribute)을 기준으로 데이터셋 분할
  가지 노드(branch node) 생성
    for each branch
      branch_node.add(Recursive branch split)
  return branch node
```

- [표 12-2]의 컴퓨터 구매 데이터로 의사결정트리 생성

02 의사결정트리 알고리즘

표 12-2 예시 데이터

No	age	income	student	credit_rating	class_buys_computer
0	youth	high	no	fair	no
1	youth	high	no	excellent	no
2	middle_aged	high	no	fair	yes
3	senior	medium	no	fair	yes
4	senior	low	yes	fair	yes
5	senior	low	yes	excellent	no
6	middle_aged	low	yes	excellent	yes
7	youth	medium	no	fair	no
8	youth	low	yes	fair	yes
9	senior	medium	yes	fair	yes
10	youth	medium	yes	excellent	yes
11	middle_aged	medium	no	excellent	yes
12	middle_aged	high	yes	fair	yes
13	senior	medium	no	excellent	no

02 의사결정트리 알고리즘

- 모든 데이터가 동일한 클래스가 아님
- 최적 분류 기준이 되는 속성을 선정하기 위해, 정보 이득을 기반으로 속성별 데이터 분류의 기준을 정함

age	$Gain(age) = Info(D) - Info_{age}(D)$
credit	$Gain(credit) = Info(D) - Info_{credit}(D)$
income	$Gain(income) = Info(D) - Info_{income}(D)$
student	$Gain(student) = Info(D) - Info_{student}(D)$

$$Info(D) = -\sum_{i=1}^{n} p_i \log_2(p_i) = \left(-\frac{9}{14}\right)\log_2\frac{9}{14} + \left(-\frac{5}{14}\right)\log_2\frac{5}{14} = 0.940$$

2.1 age 기준의 정보 이득

$$Info_{age}(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

- j는 age 속성의 클래스인 youth, middle_age, senior

$$Info_{age}(D) = \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} \right)$$

$$+ \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4} \right)$$

$$+ \frac{4}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} \right)$$

- $Gain(age)$는 0.24674

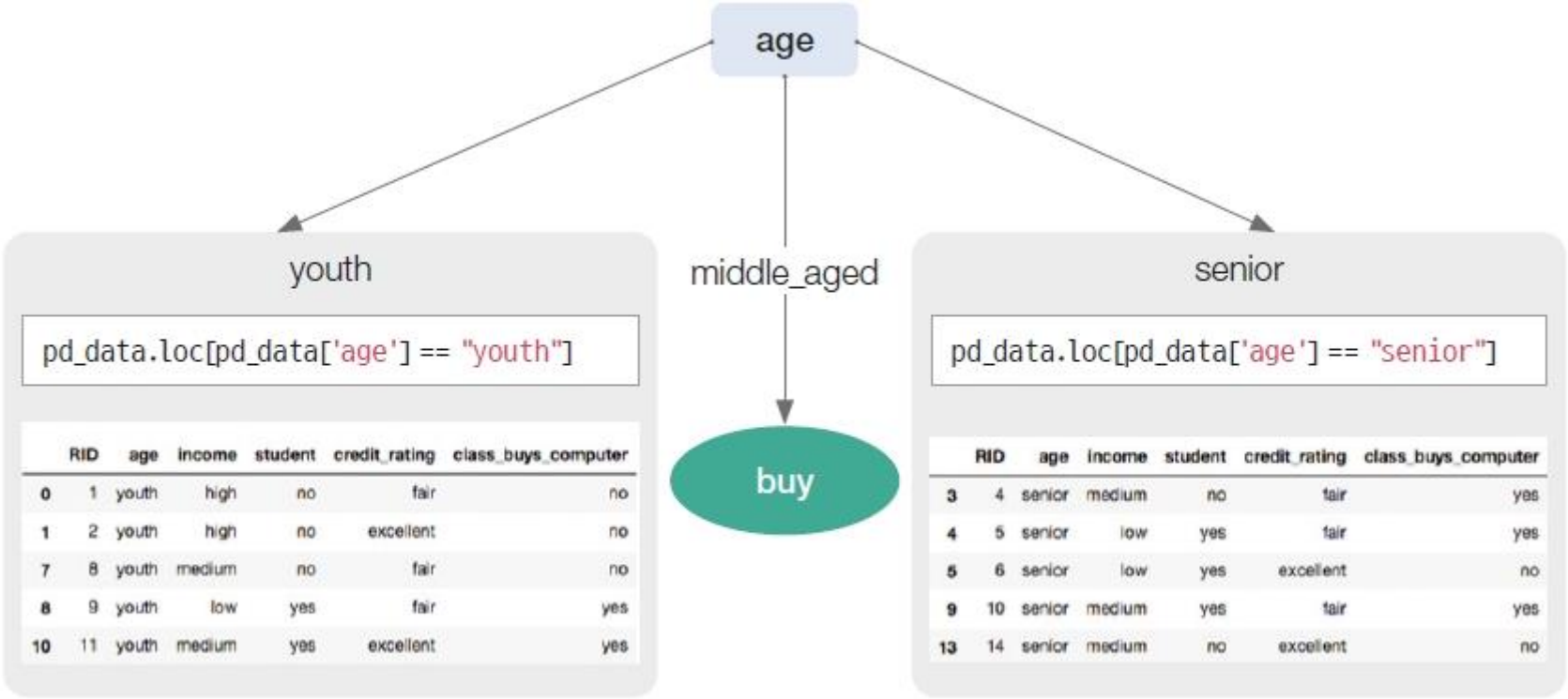
2.2 다른 속성들의 정보 이득과 가지 속성

age	$Gain(age) = Info(D) - Info_{age}(D) = 0.24674$
credit	$Gain(credit) = Info(D) - Info_{credit}(D) = 0.02922$
income	$Gain(income) = Info(D) - Info_{income}(D) = 0.15183$
student	$Gain(student) = Info(D) - Info_{student}(D) = 0.04812$

- ID3 알고리즘의 순서에 따라 가장 많은 정보를 주는 age 속성을 첫 번째 가지(branch) 속성이라고 함

02 의사결정트리 알고리즘

- age 속성 기준으로 데이터를 나누어 새로운 트리 생성



youth

	RID	age	income	student	credit_rating	class_buys_computer
0	1	youth	high	no	fair	no
1	2	youth	high	no	excellent	no
7	8	youth	medium	no	fair	no
8	9	youth	low	yes	fair	yes
10	11	youth	medium	yes	excellent	yes

senior

	RID	age	income	student	credit_rating	class_buys_computer
3	4	senior	medium	no	fair	yes
4	5	senior	low	yes	fair	yes
5	6	senior	low	yes	excellent	no
9	10	senior	medium	yes	fair	yes
13	14	senior	medium	no	excellent	no

그림 12-5 age 속성을 기준으로 한 트리(tree) 생성

- youth와 senior는 3:2 비율로 컴퓨터 구매 여부가 나누어짐
- middle_aged의 경우 모두 컴퓨터를 구입한다는 데이터이므로 더 이상 데이터를 분류할 필요가 없음

- age가 youth로 분류된 5개의 데이터에 대한 정보 이득

`credit`	$Gain(credit) = Info(D_{youth}) - Info_{credit}(D_{youth}) = -1.580$
`income`	$Gain(income) = Info(D_{youth}) - Info_{income}(D_{youth}) = -1.5270$
`student`	$Gain(student) = Info(D_{youth}) - Info_{student}(D_{youth}) = -1.2367$

02 의사결정트리 알고리즘

- student를 기준으로 데이터를 분리했을 때 가장 많은 정보를 획득함

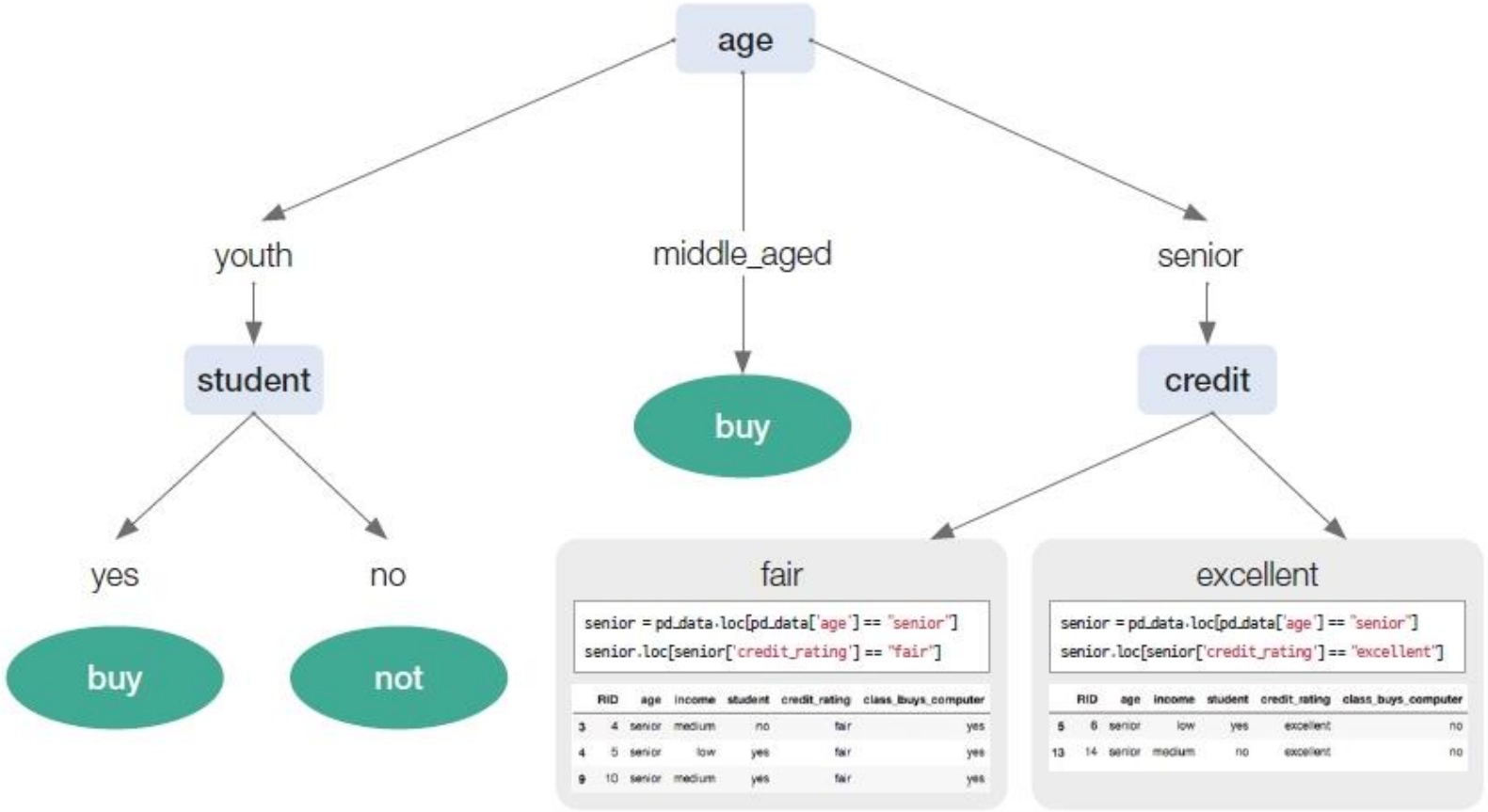


그림 12-6 데이터 재분류

02 의사결정트리 알고리즘

- 의사결정트리 완성

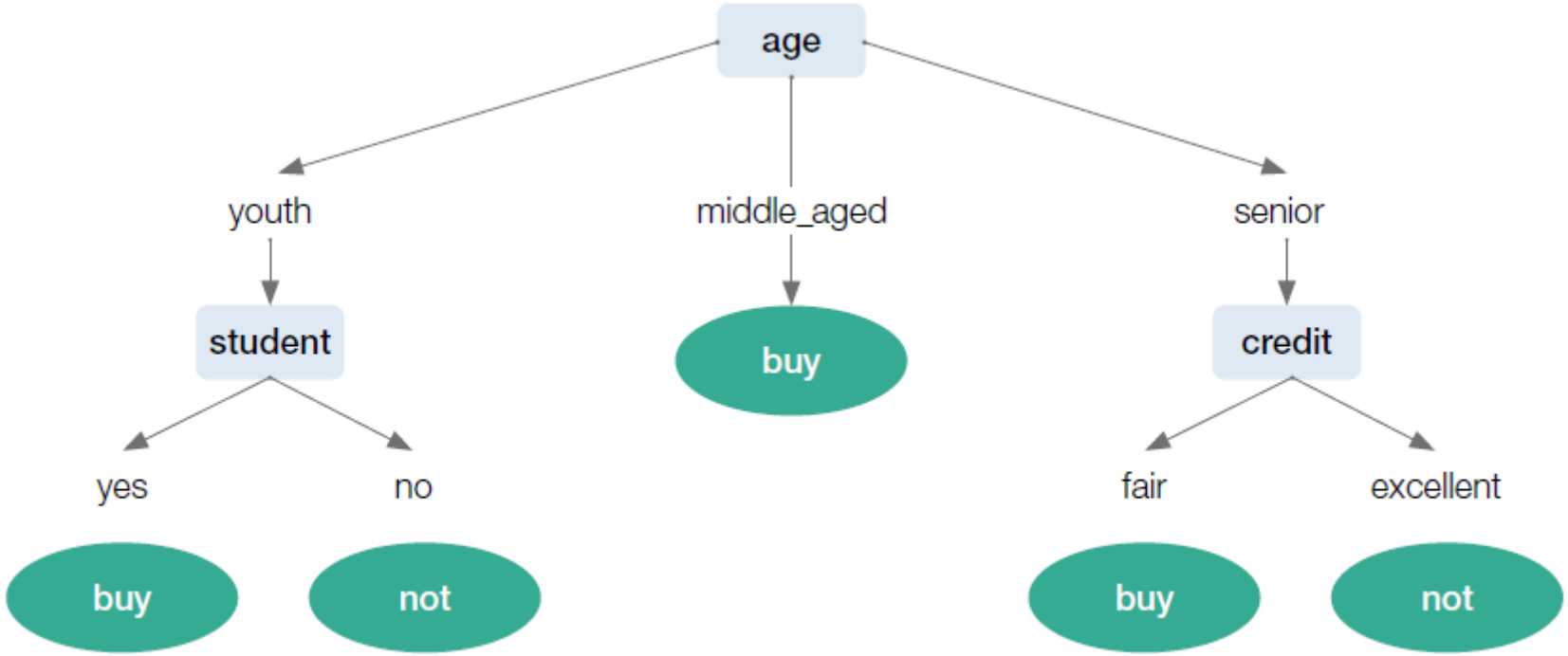


그림 12-7 의사결정트리 완성

3. 의사결정트리 알고리즘의 특징

① 재귀적 작동 : 가지가 되는 속성을 선택한 후 해당 가지로 데이터를 나누면, 이전에 적용되었던 알고리즘이 남은 데이터에 적용됨

- 남은 데이터에서만 최적의 모델을 찾는 방법으로 작동

② 속성 기준으로 가지치기 수행 : 가장 불확실성이 적은 속성을 기준으로 가지치기를 수행

③ 중요한 속성 정보 제공 : 처음 분리 대상이 되는 속성이 가장 중요한 속성

- 이 특징을 '해석 가능한 머신러닝'이라고 부름

[하나 더 알기] 의사결정트리의 장점

- 불필요한 속성 값에 대한 스케일링 : 전처리 단계 없이 바로 사용할 수 있다
- 강건(robust)한 이상치(outlier) : 관측치의 절대값이 아닌 순서가 중요하기 때문에 필요 이상으로 엄청 큰 값이나 작은 값에 대해서도 분류 성능이 크게 떨어지지 않는다.
- 자동적인 변수 선택 : 알고리즘에 의해 중요한 변수들이 우선적으로 선택되어 조금 더 손쉽게 중요한 속성을 확인할 수 있다. 의사결정트리 계열의 알고리즘이 가지고 있는 가장 큰 장점 중 하나이다.

03
의사결정트리의 확장

1. 정보 이득의 문제점

- 수식의 특성상 속성의 값이 다양할수록 선택의 확률이 높아지는 문제가 발생

$$Info_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

  - 데이터가 매우 많고 속성이 다양할 때 위 수식의 $\frac{|D_j|}{|D|}$ 값이 작아짐. 해당 속성의 엔트로피가 낮아져 단순히 속성 안에 있는 값의 종류를 늘리는 것만으로 정보 이득이 높아짐

2. C4.5 알고리즘

- C4.5 : 정보 이득을 측정하는 방식을 좀 더 평준화시켜 단순한 정보 값을 대신 사용
  - 기존 정보 이득의 분모에 평준화 함수 SplitInfo 추가

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)} = \frac{Info(D) - Info_a(D)}{SplitInfo_A(D)}$$

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \left(\log_2 \frac{|D_j|}{|D|} \right)$$

03 의사결정트리의 확장

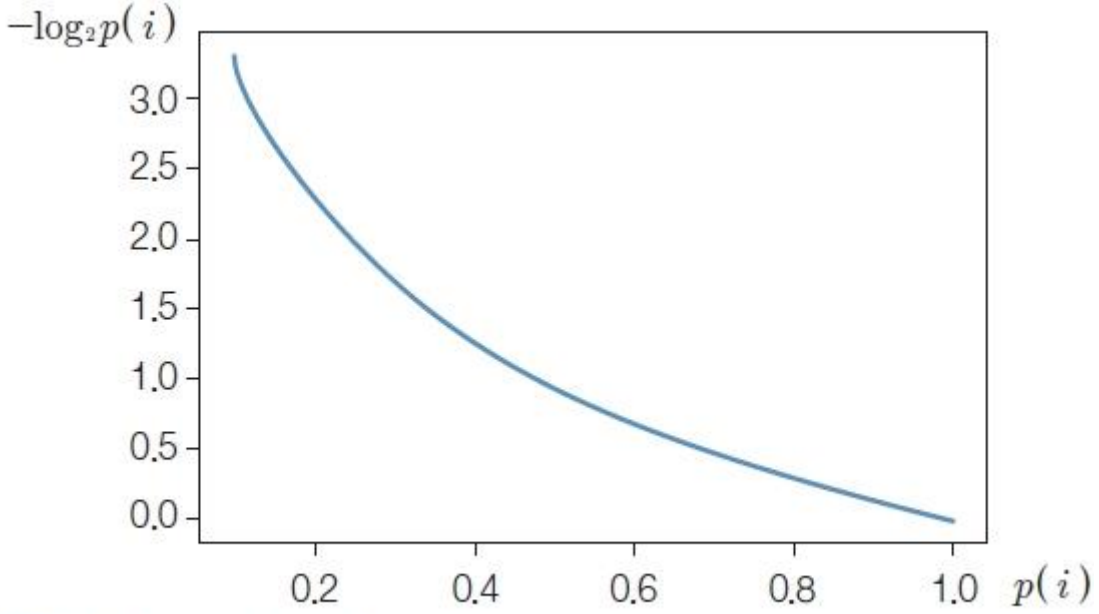


그림 12-8 C4.5 알고리즘

- 클래스가 많을수록 $\frac{|D_j|}{|D|}$값이 작아지고 $-\log_2 \frac{|D_j|}{|D|}$값은 커져 정규화됨
- SplitInfo 함수 값이 분모에 들어가면서 클래스 불균형에 의해 생기는 불합리한 속성 분류를 보정

3. 지니 지수

- 경제학에서 소득의 불평등도를 측정할 때 사용하는 지표인데, 의사결정트리에서 각 속성의 불순도를 측정하는 방법으로 사용

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2 = 1 - \sum_{i=1}^{m} \frac{|C_{i,D}|}{|D|} \text{ where } C_i \text{ is a class}$$

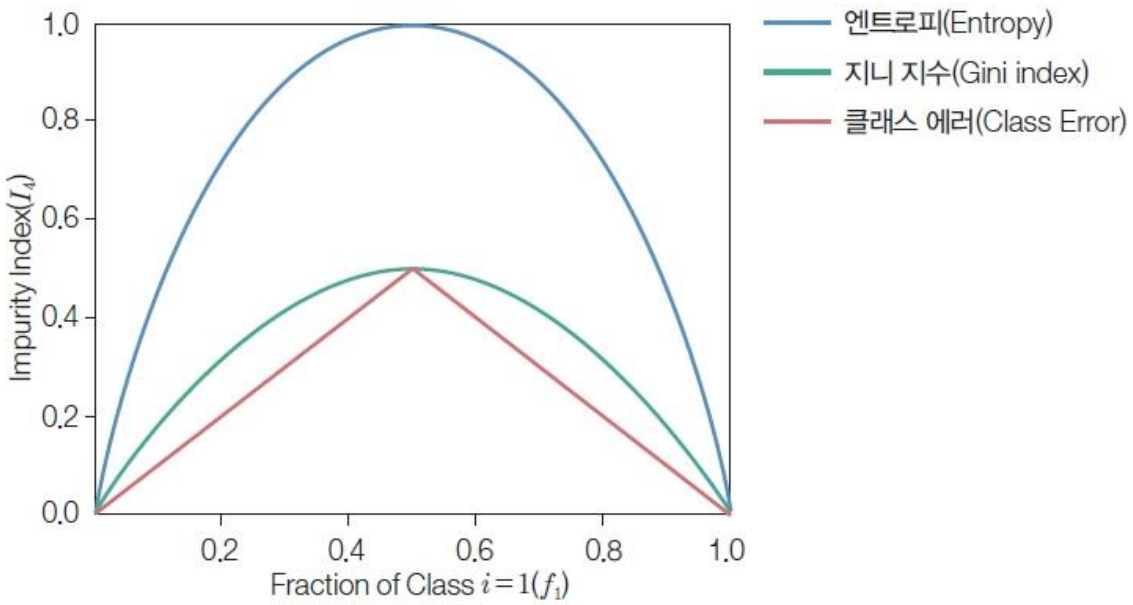


그림 12-9 지니 지수로 작성된 그래프

3.1 이진 분할

- CART 알고리즘의 핵심은 불확실성을 측정하는 기준 값이 엔트로피에서 지니 지수로 바뀐 것
- 구현 측면에서 가장 큰 차이점은 이진 분할을 실시한다는 것
- k가 속성 내에 있는 데이터의 개수일 때, $2^{k-1}-1$개의 분할이 생성됨
- 각 속성별 지니 지수 정보

$$Gini_A(D)=\frac{|D_1|}{|D|}Gini(D_1)+\frac{|D_2|}{|D|}Gini(D_2)$$

03 의사결정트리의 확장

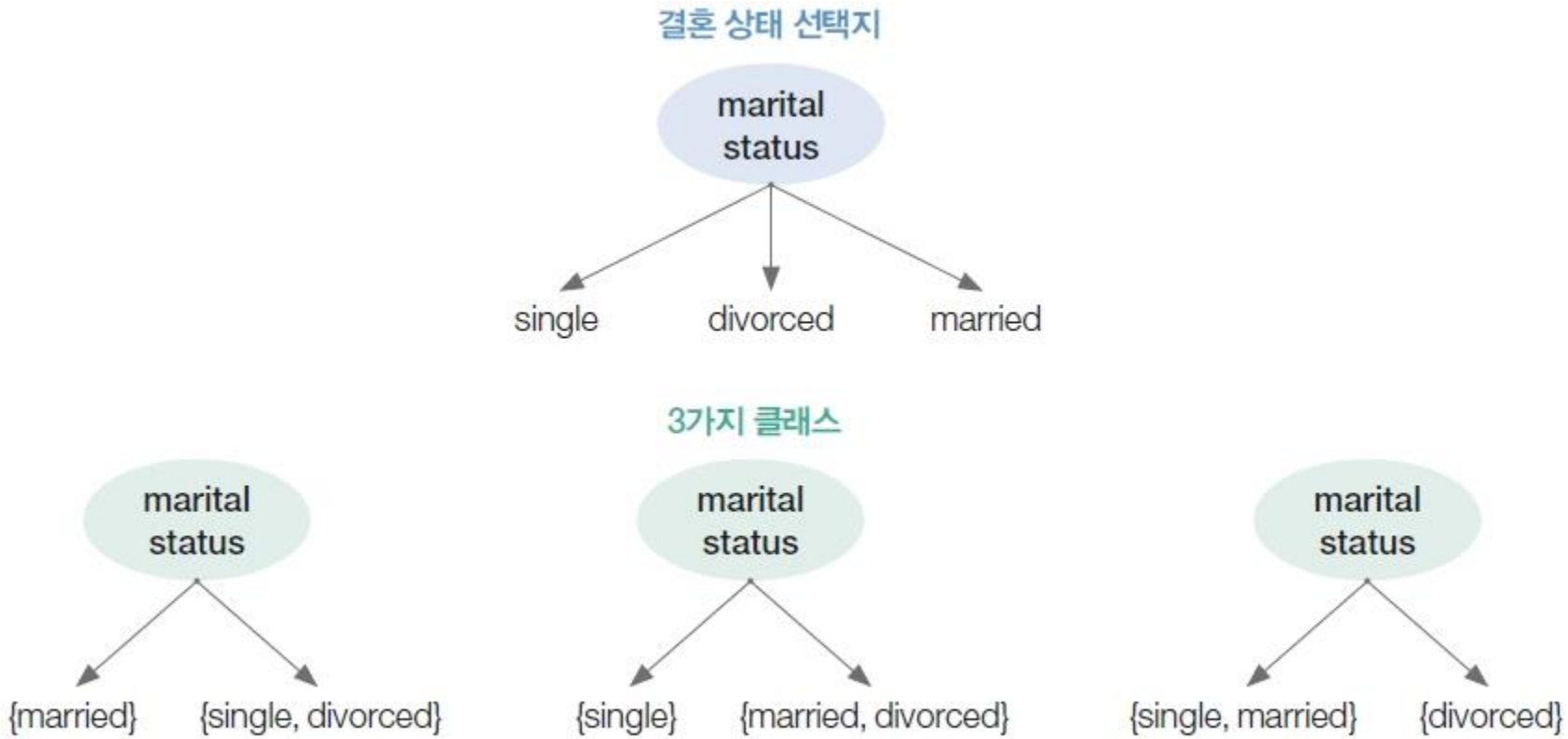


그림 12-10 결혼 상태(martial status)의 3가지 클래스

3.2 실제 데이터에 지니 지수를 적용하여 의사결정트리 만들기

표 12-3 예시 데이터

No	age	income	student	credit_rating	class_buys_computer
0	youth	high	no	fair	no
1	youth	high	no	excellent	no
2	middle_aged	high	no	fair	yes
3	senior	medium	no	fair	yes
4	senior	low	yes	fair	yes
5	senior	low	yes	excellent	no
6	middle_aged	low	yes	excellent	yes
7	youth	medium	no	fair	no
8	youth	low	yes	fair	yes
9	senior	medium	yes	fair	yes
10	youth	medium	yes	excellent	yes
11	middle_aged	medium	no	excellent	yes
12	middle_aged	high	yes	fair	yes
13	senior	medium	no	excellent	no

03 의사결정트리의 확장

- age, credit, income, student의 4가지 속성들을 정의

$$Gini_{age}(D) \qquad Gini_{credit}(D) \qquad Gini_{income}(D) \qquad Gini_{student}(D)$$

- age 속성에 3가지 클래스가 존재하기 때문에 3가지 종류의 이진 분할 경우의 수로 나눌 수 있음

$$age \in \{\texttt{youth, middle\_aged, senior}\}$$

$$age\_1 \in \{youth\} = \{middle\_aged, senior\}$$
$$age\_2 \in \{middle\_aged\} = \{youth, senior\}$$
$$age\_3 \in \{senior\} = \{youth, middle\_aged\}$$

03 의사결정트리의 확장

표 12-4 age가 youth인 경우 데이터

No	age	income	student	credit_rating	class_buys_computer
0	youth	high	no	fair	no
1	youth	high	no	excellent	no
7	youth	medium	no	fair	no
8	youth	low	yes	fair	yes
10	youth	medium	yes	excellent	yes

표 12-5 age가 youth가 아닌 경우 데이터

No	age	income	student	credit_rating	class_buys_computer
2	middle_aged	high	no	fair	yes
3	senior	medium	no	fair	yes
4	senior	low	yes	fair	yes
5	senior	low	yes	excellent	no
6	middle_aged	low	yes	excellent	yes
9	senior	medium	yes	fair	yes
11	middle_aged	medium	no	excellent	yes
12	middle_aged	high	yes	fair	yes
13	senior	medium	no	excellent	no

03 의사결정트리의 확장

$$Gini_{age_1}(D) = \frac{5}{14}Gini(D_1) + \frac{9}{14}Gini(D_2) = 0.393$$

$$Gini(D_1) = 1 - \left(\frac{3}{5}\right)^2 - \left(\frac{2}{5}\right)^2$$

$$Gini(D_2) = 1 - \left(\frac{7}{9}\right)^2 - \left(\frac{2}{9}\right)^2$$

- age_1, age_2, age_3 각각 연산하면 0.393, 0.357, 0.457
- age 속성의 지니 지수는 이 중 가장 작은 값인 0.357

$$Min(Gini_{age_i}) = 0.357$$

$$Min(Gini_{income_i}) = 0.443$$

$$Min(Gini_{credit}) = 0.429$$

$$Min(Gini_{student}) = 0.367$$

04
의사결정트리 알고리즘의 다양한 변형

1. 트리 가지치기

- 클래스의 마지막 노드인 잎 노드(leaf node)의 개수를 개발자가 직접 결정
  - 1개로 이루어진 잎 노드가 많을 경우 과대적합되어 있는 상태
  - 잎 노드의 개수와 관계 없이 해당 가지에 불확실성이 너무 높을 경우 의사결정트리의 성능에 문제를 줄 수 있음
- 트리 가지치기(tree pruning) : 의사결정트리의 마지막 노드의 개수를 지정하여 트리의 깊이를 조정하는 방법

- 사전 가지치기(pre-pruning) : 처음 트리를 만들 때 트리의 깊이나 마지막 노드의 최소 개수 등을 사전에 결정하여 입력
  - 데이터 분석가가 하이퍼 매개변수로 모든 값을 입력해야 하는 점이 어려움
  - 계산 효율이 좋고 작은 데이터셋에서도 쉽게 작동
  - 사용자가 중요한 속성 값을 놓치거나 과소적합 문제 발생할 수 있다

- 사후 가지치기(post-pruning) : 트리를 먼저 생성한 후 실험적으로 하이퍼 매개변수를 조정
  - 하나의 지표를 정해두고 실험적으로 다양한 하이퍼 매개변수를 조정하며 최적의 값을 찾음
  - '최종 노드의 개수', '트리의 깊이', 또는 '선택되는 속성의 개수' 등을 하이퍼 매개변수로 보고 조정하며 성능을 비교
  - 먼저 전체 데이터를 훈련셋, 검증셋, 테스트셋으로 분류하고, 훈련셋과 테스트셋의 성능을 비교

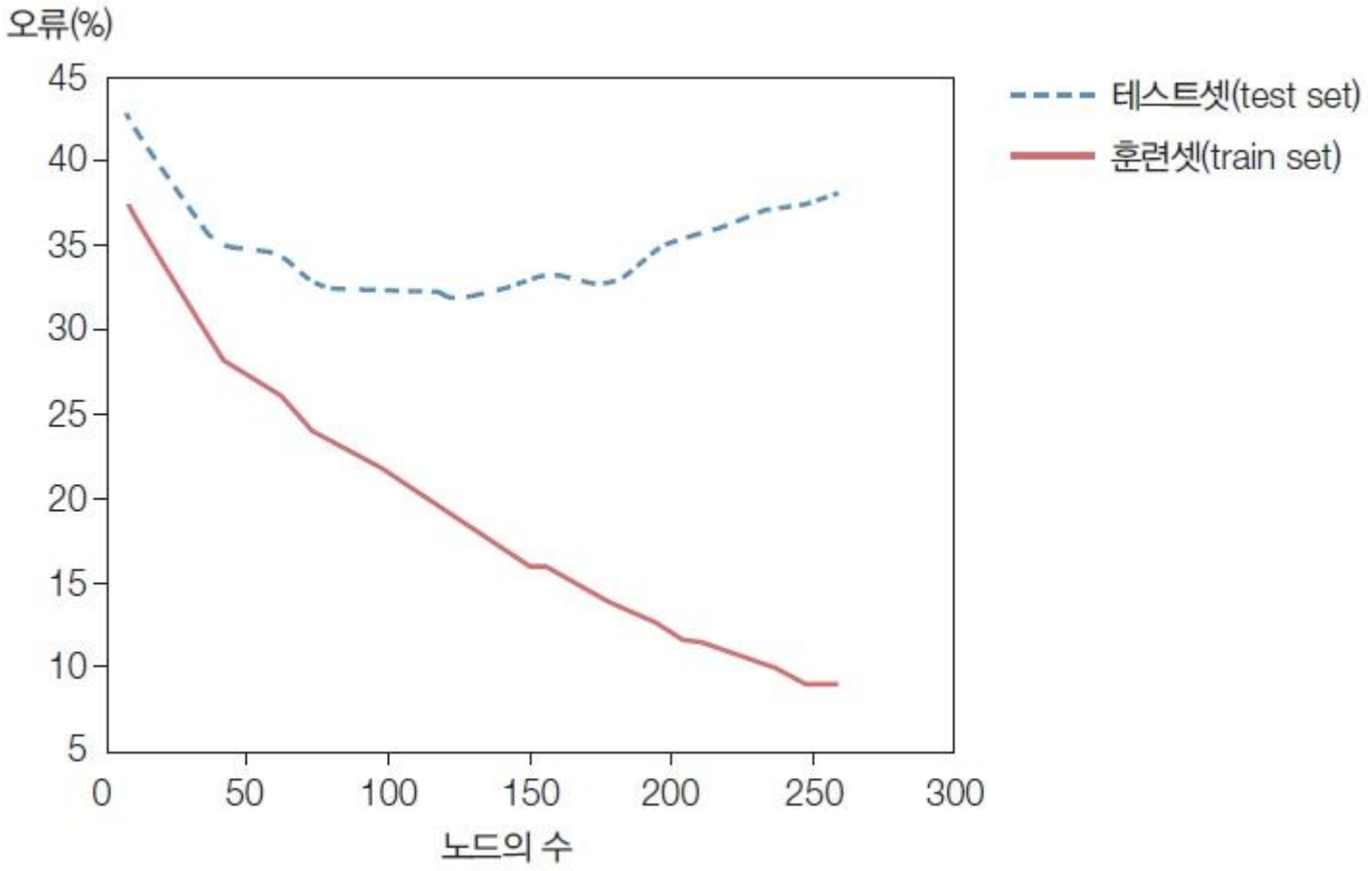


그림 12-11 사후 가지치기에서 훈련셋과 테스트셋의 성능 비교

- 전체 노드 개수를 조정하면서 훈련셋과 테스트셋 성능 비교

2. 연속형 변수 나누기

표 12-6 샘플 데이터(vegeterianl_dataset.csv)

	ID	STREAM	SLOPE	ELEVATION	VEGETATION
0	1	False	steep	3900	chapparal
1	2	True	no	300	riparian
2	3	True	steep	1500	riparian
3	4	False	steep	1200	chapparal
4	5	False	flat	4450	conifer
5	6	True	steep	5000	conifer
6	7	True	steep	3000	chapparal

- 연속형 데이터를 나누는 기준
  - 모든 데이터를 기준점으로 하여 데이터를 나누기 : 너무 많은 기준점이 생겨 과대적합 문제가 발생하거나 분류의 정확도가 떨어짐
  - 통계적 수치로 중위값이나 4분위수를 기준점으로 나누기 : 25%씩 데이터를 나눠서 분류 기준을 변경. 과소적합 문제가 발생하여 분류의 성능을 떨어뜨릴 수 있음
  - 가장 많이 쓰는 방법으로, Y 클래스의 값을 기준으로 해당 값이 변할 때를 기준점으로 삼아 분기

In [1]:
```
import pandas as pd
import numpy as np

pd_data = pd.read_csv(
    'c:/source/ch12/vegeterianl_dataset.csv',
delimiter=r"\s+")
pd_data.drop("ID",axis=1)
```

Out [1]:

	ID	STREAM	SLOPE	ELEVATION	VEGETATION
1	2	True	moderate	300	riparian
3	4	False	steep	1200	chapparal
2	3	True	steep	1500	riparian
6	7	True	steep	3000	chapparal
0	1	False	steep	3900	chapparal
4	5	False	flat	4450	conifer
5	6	True	steep	5000	conifer

- 분류 대상이 되는 ELEVATION 속성의 데이터를 정렬

In [2]:
```
pd_data.sort_values("ELEVATION")
```

Out [2]:

	STREAM	SLOPE	ELEVATION	VEGETATION
0	False	steep	3900	chapparal
1	True	moderate	300	riparian
2	True	steep	1500	riparian
3	False	steep	1200	chapparal
4	False	flat	4450	conifer
5	True	steep	5000	conifer
6	True	steep	3000	chapparal

- 데이터 중 분류 대상이 되는 기준점 찾기

	ID	STREAM	SLOPE	ELEVATION	VEGETATION
1	2	True	moderate	300	riparian
3	4	False	steep	1200	chapparal
2	3	True	steep	1500	riparian
6	7	True	steep	3000	chapparal
0	1	False	steep	3900	chapparal
4	5	False	flat	4450	conifer
5	6	True	steep	5000	conifer

(1) (2) (3) (4)

그림 12-12 pd_data.sort_values("ELEVATION")

- 300, 1200, 1500, 3900에서 각각 Y 데이터의 라벨이 달라졌다

- 데이터를 자르는 기준값 정하기 : 구간별 경계 평균값

	ID	STREAM	SLOPE	ELEVATION	VEGETATION
1	2	True	moderate	300	riparian
					750
3	4	False	steep	1200	chapparal
					1350
2	3	True	steep	1500	riparian
					2250
6	7	True	steep	3000	chapparal
0	1	False	steep	3900	chapparal
					4175
4	5	False	flat	4450	conifer
5	6	True	steep	5000	conifer

그림 12-13 데이터를 자르는 기준값 선정

- 구간별 경계값을 기준으로 엔트로피를 산출
  - 4개의 기준점 각각의 정보 이득을 구했을 때 가장 큰 값이 ELEVATION의 대표 정보 이득이 되어 다른 속성값 정보 이득과 비교하여 최종적으로 분기가 일어나는 속성으로 선택됨

$$Gain(elev_{750}) = Info(D) - Info_{elev_{750}}(D)$$

$$Gain(elev_{1350}) = Info(D) - Info_{elev_{1350}}(D)$$

$$Gain(elev_{2250}) = Info(D) - Info_{elev_{2250}}(D)$$

$$Gain(elev_{4175}) = Info(D) - Info_{elev_{4175}}(D)$$

  - 위 계산 결과는 각각 0.3059, 0.1813, 0.5916, 0.8631이고 STREAM, SLOPE 속성의 정보 이득은 각각 0.3059, 0.5774로 가장 먼저 ELEVATION 이 4175인 값을 기준으로 트리를 분기

3. 회귀 트리

- 회귀 트리(regression tree) : Y 데이터의 값이 연속형일 때의 의사결정트리 생성 방법

연속형 속성 분기의 특징

명목 속성과는 달리 여러 번 재사용이 가능하다. 경계값을 기준으로 여러 번의 분기를 할 수 있다.
처음에는 ELEVATION 4175의 값을 기준으로 분기를 했다면 동일하게 맨 마지막에는 2250의 값으로 분기를 할 수 있다.

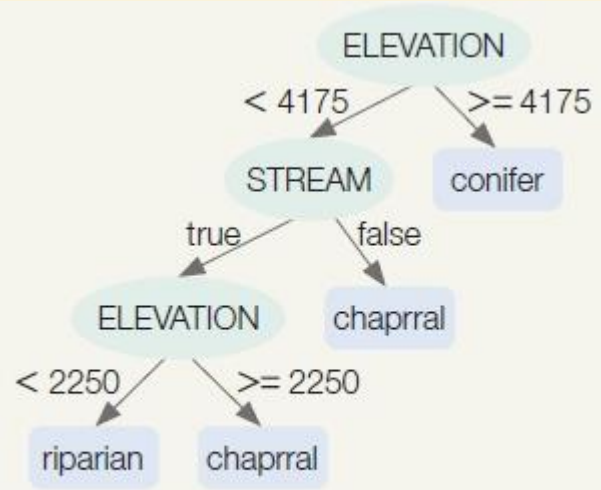


그림 12-14 연속형 속성 분기의 특징

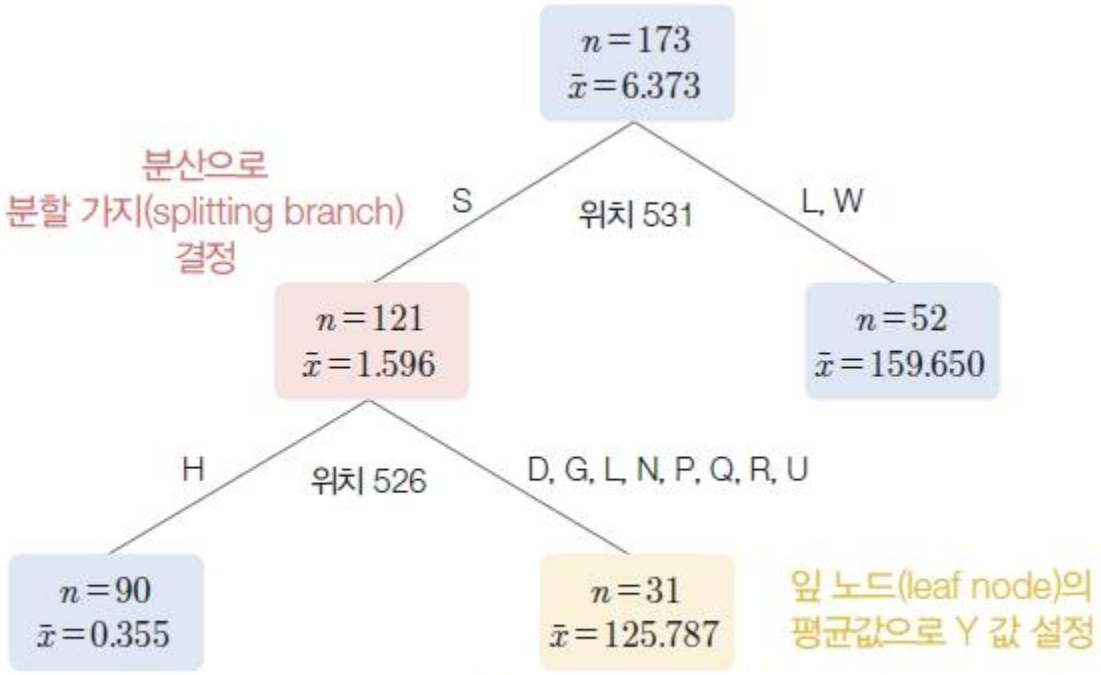


그림 12-15 연속형 속성 분기의 특징

- Y 값의 각 속성별 분산이 얼마나 작은지를 측정
- 최종 결과 노드에서는 결과 노드들의 Y 평균값으로 최종 예상치를 반환

$$var(D) = \frac{\sum_{i=1}^{n}(y^{(i)} - \bar{y})^2}{n-1} \qquad where\ \bar{y} = \frac{\sum_{i=1}^{n} y^{(i)}}{n}$$

- 속성별 분산 구하기 : 각 클래스값들의 분산을 구하고 해당 클래스가 가진 데이터 개수의 비율만큼 곱함

$$\underset{attr \in data}{\operatorname{argmax}} \sum_{l \in levels(attr)} \frac{|D_{attr=l}|}{|D|} \times var(D_{attr} = l)$$

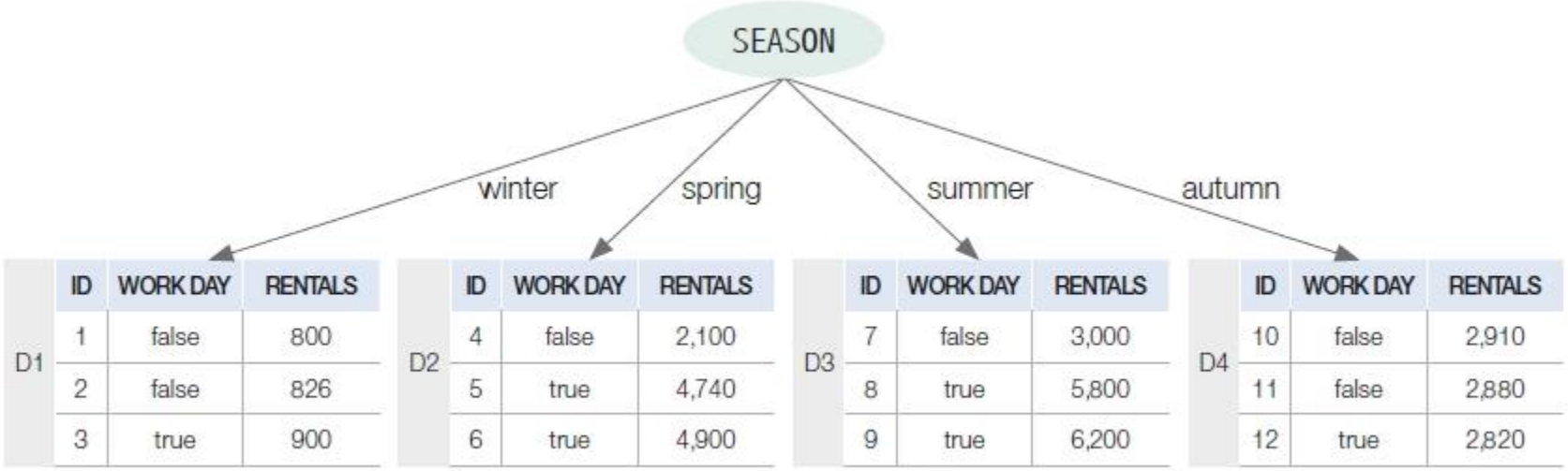
표 12-7 연속형 값 예측을 위한 데이터

	SEASON	WORK_DAY	RENTALS		SEASON	WORK_DAY	RENTALS
0	winter	False	800	6	summer	True	3000
1	winter	False	826	7	summer	True	5800
2	winter	True	900	8	autumn	False	6200
3	spring	False	2100	9	autumn	False	2910
4	spring	True	4740	10	autumn	True	2880
5	spring	True	4900	11	autumn	False	2820

- SEASON 속성이 WORK_DAY에 비해서 분산이 낮으므로 해당 값으로 분기가 일어남

$$Var_{SEASON} = \frac{3}{12} * 618133 + \frac{3}{12} * 698033 + \frac{3}{12} * 910308 + \frac{3}{12} * 673 = 2227148$$

$$Var_{WORK\_DAY} = \frac{6}{12} * 1974105 + \frac{6}{12} * 1593393 + \frac{3}{12} = 3567498$$



D1

ID	WORK DAY	RENTALS
1	false	800
2	false	826
3	true	900

D2

ID	WORK DAY	RENTALS
4	false	2,100
5	true	4,740
6	true	4,900

D3

ID	WORK DAY	RENTALS
7	false	3,000
8	true	5,800
9	true	6,200

D4

ID	WORK DAY	RENTALS
10	false	2,910
11	false	2,880
12	true	2,820

그림 12-16 SEASON을 기준으로 4개의 분기 발생

- 각 분기별로는 WORK_DAY로 데이터를 나누고, 나누어진 데이터는 평균값으로 최종 예측값이 반환됨
  - summer를 기준으로 분기가 일어난 후 WORK_DAY가 True인 분기에서 RENTALS 값이 5,800과 6,200이어서 평균값 6,000이 예측치로 반환됨

05
의사결정트리의 구현

1. 맛보기로 의사결정트리 구현하기

1.1 데이터 불러오기

- [표12-1] 데이터를 로딩하고 인덱스 번호인 RID 열 제거

In [1]:
```
import pandas as pd
import numpy as np

pd_data =
pd.read_csv('https://raw.githubusercontent.com/AugustLONG
/ML01/master/01decisiontree/AllElectronics.csv')
pd_data = pd_data.drop("RID",axis=1)
```

1.2 정보 이득 함수 만들기

- y 값의 클래스를 기준으로 엔트로피 연산을 실시
  - class_buys_computer가 yes인 경우와 no인 경우로 나눔

$$Info(D) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

In [2]:	def get_info(df):
	    buy = df.loc[df["class_buys_computer"]=="yes"]
	    not_buy = df.loc[df["class_buys_computer"]=="no"]
	    x = np.array([len(buy)/len(df),len(not_buy) /len(df)])
	    y = np.log2(x[x!=0])
	
	    info_all = - sum(x[x!=0] * y)
	    return info_all

In [3]:	get_info(pd_data)
Out [3]:	0.9402859586706311

05 의사결정트리의 구현

- 속성별 정보 이득률 구하기 : 각 속성들의 클래스를 기준으로 데이터를 추출한 후 정보 이득을 산출

In [4]:	youth = pd_data.loc[pd_data['age'] == "youth"] senior = pd_data.loc[pd_data['age'] == "senior"] middle_aged = pd_data.loc[pd_data['age'] == "middle_aged"]
In [5]:	print(get_info(youth))
Out [5]:	0.9709505944546686
In [6]:	print(get_info(senior))
Out [6]:	0.9709505944546686
In [7]:	print(get_info(middle_aged))
Out [7]:	-0.0

1.3 자동으로 속성별 정보 이득 연산하기

- 속성을 입력하면 자동으로 연산하는 함수

In [8]:	def get_attribute_info(df, attribute_name):
	    attribute_values = pd_data[attribute_name].unique()
	    get_infos = []
	    for value in attribute_values:
	        split_df = pd_data.loc[pd_data[attribute_name] == value]
	
	        get_infos.append((len(split_df) / len(df)) * get_info(split_df))
	
	    return sum(get_infos)

In [9]:	get_attribute_info(pd_data, "age")
Out [9]:	0.6935361388961918

1.4 정보 이득 계산하기

- 전체 데이터 대비 각각의 속성 데이터를 분리하여 계산

In [10]:	get_info(pd_data) - get_attribute_info(pd_data, "age")
Out [10]:	0.246749819774439 33
In [11]:	get_info(pd_data) - get_attribute_info(pd_data, "income")
Out [11]:	0.02922256565895487
In [12]:	get_info(pd_data) - get_attribute_info(pd_data, "student")
Out [12]:	0.15183550136234159
In [13]:	get_info(pd_data) - get_attribute_info(pd_data, "credit_rating")
Out [13]:	0.04812703040826949

05 의사결정트리의 구현

- 정보 이득이 가장 큰 age 속성값을 기준으로 의사결정 트리 가지를 생성

In [14]:	youth = pd_data.loc[pd_data['age'] == "youth"] get_info(youth) - get_attribute_info(youth, "income")
Out [14]:	-1.580026905978025
In [15]:	get_info(youth) - get_attribute_info(youth, "student")
Out [15]:	-1.2367106860085422
In [16]:	get_info(youth) - get_attribute_info(youth, "credit_rating")
Out [16]:	-1.5270944046799 44

- 가지 생성 알고리즘이 재귀적으로 일어남

2. 사이킷런으로 의사결정트리 구현하기

- 타이타닉 데이터셋 사용

2.1 데이터 불러오기

In [17]:
```
import pandas as pd

train_df = pd.read_csv("c:/source/ch12/train.csv")
test_df = pd.read_csv("c:/source/ch12/test.csv")

train_id = train_df["PassengerId"].values
test_id = test_df["PassengerId"].values

all_df = train_df.append(test_df).set_index('PassengerId')
```

2.2 데이터 전처리

- 데이터를 코드화시키고 결측치를 채우기

In [18]:
```
all_df["Sex"] =
all_df["Sex"].replace({"male":0,"female":1})

# 데이터 중 age 값의 빈칸의 값을 `class의 평균값으로 채운다.
all_df["Age"].fillna(
    all_df.groupby("Pclass")["Age"].transform("mean"),
inplace=True)
```

In [19]:
```
all_df["cabin_count"] = all_df["Cabin"].map(
   lambda x : len(x.split()) if type(x) == str else 0)
```

In [20]:
```
def transform_status(x):
    if "Mrs" in x or "Ms" in x:
        return "Mrs"
    elif "Mr" in x:
        return "Mr"
    elif "Miss" in x:
        return "Miss"
```

05 의사결정트리의 구현

```
    elif "Master" in x:
        return "Master"
    elif "Dr" in x:
        return "Dr"
    elif "Rev" in x:
        return "Rev"
    elif "Col" in x:
        return "Col"
    else:
        return "0"
all_df["social_status"] = all_df["Name"].map(lambda x :
transform_status(x))
```

In [21]:

```
all_df["social_status"].value_counts()
```

Out [21]:

```
Mr        758
Miss      258
Mrs       203
Master     61
0           9
Rev         8
Dr          8
Col         4
Name: social_status, dtype: int64
```

05 의사결정트리의 구현

- 사용하지 않을 데이터를 삭제

In [22]:
```
all_df[all_df["Embarked"].isnull()]
```

Out [22]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	cabin_count	social_status
62	1.0	1	Icard, Miss. Amelie	1	38.0	0	0	113572	80.0	B28	NaN	1	Miss
830	1.0	1	Stone, Mrs. George Nelson (Martha Evelyn)	1	62.0	0	0	113572	80.0	B28	NaN	1	Mrs

In [23]:
```
all_df = all_df.drop([62,830])
train_id =np.delete(train_id, [62-1,830-1])
```

In [24]:
```
p_v_cap_s = sum((np_data[:, 0] == 1) & (np_data[:, 1] == 1)) / len(np_data)
p_v_cap_s
```

Out [24]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	cabin_count	social_status
1044	NaN	3	Storey, Mr. Thomas	0	60.5	0	0	3701	NaN	NaN	S	0	Mr

05 의사결정트리의 구현

- 빈 데이터는 적당한 값을 채우기

In [25]:	all_df[all_df["Embarked"].isnull()]
Out [25]:	Pclass Sex 1 0 69.888385 1 109.826644 2 0 19.904946 1 23.234827 3 0 12.415462 1 15.324250 Name: Fare, dtype: float64
In [26]:	all_df.loc[all_df["Fare"].isnull(), "Fare"] = 12.415462

05 의사결정트리의 구현

- 짐의 형태에 따라 생존 여부가 달라질 수 있다고 가정하고 전처리 작업을 진행

In [27]:	`all_df["cabin_type"] = all_df["Cabin"].map(lambda x : x[0] if type(x) == str else "99")`

In [28]:	`del all_df["Cabin"]`
	`del all_df["Name"]`
	`del all_df["Ticket"]`

In [29]:	`y = all_df.loc[train_id, "Survived"].values` `del all_df["Survived"]`

- 일반적으로 전처리 작업을 진행할 때 가설을 먼저 생성한 후 해당 가설이 맞는지를 실험적으로 검증하여 적합한 변수를 알아낸다

2.3 원핫인코딩과 스케일링

- 데이터를 원핫으로 변경 후 넘파이 배열로 변경
- 데이터 스케일링

In [30]:	X_df = pd.get_dummies(all_df) X = X_df.values
In [31]:	from sklearn.preprocessing import MinMaxScaler minmax_scaler = MinMaxScaler() minmax_scaler.fit(X) X = minmax_scaler.transform(X)

2.4 학습 실행하기

In [32]:	X_train = X[:len(train_id)] X_test = X[len(train_id):]

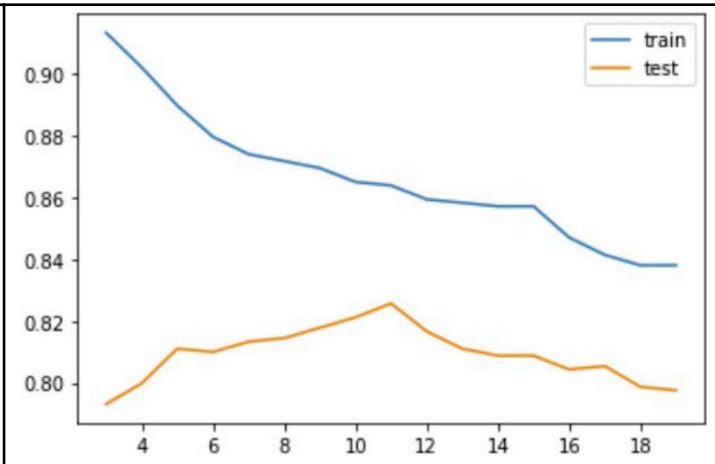
- DecisionTreeClassifier 객체를 생성
- 주요 하이퍼 매개변수
  - “criterion” : [“gini”, “entropy”] : 지니 지수를 기준으로 나눌지, 정보 이득을 기준으로 나눌지 지정
  - “max_depth” : int : 트리의 깊이를 지정
  - “min_samples_leaf” : int or float : 마지막 노드의 최소 데이터의 개수를 지정
    - int는 데이터의 개수, float는 전체 데이터에서의 비율

In [33]:

```
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score

test_accuracy = []
train_accuracy = []
for idx in range(3, 20):
    df = DecisionTreeClassifier(min_samples_leaf=idx)
    acc = cross_val_score(df, X_train, y,
scoring="accuracy", cv=5).mean()
    train_accuracy.append(
accuracy_score(df.fit(X_train, y).predict(X_train), y))
    test_accuracy.append(acc)

result = pd.DataFrame(train_accuracy, index=range(3,20),
columns=["train"])
result["test"] = test_accuracy

result.plot()
```

05 의사결정트리의 구현

Out [33]:



- test 데이터셋의 정확성은 11에서 가장 높았다가 계속 떨어짐
- 의사결정트리의 경우 마지막 노드의 데이터 개수가 적으면 적을수록 과대적합이 발생

05 의사결정트리의 구현

- 두 개 이상의 알고리즘과 하이퍼 매개변수 실험을 수행

In [34]:
```
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression

algorithmes = [LogisticRegression(),
DecisionTreeClassifier()]

c_params = [0.1, 5.0, 7.0, 10.0, 15.0, 20.0, 100.0]
params = []


params.append([{
    "solver" : ["saga"],
    "penalty" : ["l1"],
    "C" : c_params
    },{
    "solver" : ['liblinear'],
    "penalty" : ["l2"],
    "C" : c_params
```

```
    }
    ])
params.append({
    "criterion" : ["gini", "entropy"],
    "max_depth" : [10,8,7,6,5,4,3,2],
    "min_samples_leaf": [1,2,3,4,5,6,7,8,9]})
```

In [35]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report,
accuracy_score

scoring = ['accuracy']
estimator_results = []
for i, (estimator, params) in
enumerate(zip(algorithmes,params)):
    gs_estimator = GridSearchCV(
        refit="accuracy",estimator=estimator,
        param_grid=params, scoring=scoring, cv=5,
        verbose=1, n_jobs=4)

    gs_estimator.fit(X_train, y)
    estimator_results.append(gs_estimator)
```

05 의사결정트리의 구현

In [36]:	estimator_results[0].best_score_
Out [36]:	0.8282844768062269
In [37]:	estimator_results[1].best_score
Out [37]:	0.8361308141359614
In [38]:	import pandas as pd from pandas import DataFrame from collections import defaultdict result_df_dict = {} result_attributes = ["model", "accuracy", "penalty", "solver", "C", "criterion", "max_depth", "min_samples_leaf"] result_dict = defaultdict(list) algorithm_name= ["LogisticRegression", "DecisionTreeClassifier"]

```
    for i, estimators in enumerate(estimator_results):
        number_of_estimators =
len(estimators.cv_results_["mean_fit_time"])

    for idx_estimator in range(number_of_estimators):
        result_dict["model"].append(algorithm_name[i])

result_dict["accuracy"].append(estimators.cv_results_["me
an_test_accuracy"][idx_estimator])

    for param_value in estimators.cv_results_["params"]:
        for k,v in param_value.items():
            result_dict[k].append(v)
    for attr_name in result_attributes:
        if len(result_dict[attr_name])
< len(result_dict["accuracy"]):
            result_dict[attr_name].extend([None for i in
range(number_of_estimators)])
```

05 의사결정트리의 구현

- 최종 결과

In [39]:
```
result_df = DataFrame(result_dict,
columns=result_attributes)
result_df.sort_values("accuracy",ascending=False).head(n=
10)
```

Out [39]:

	model	accuracy	penalty	solver	C	criterion	max_depth	min_samples_leaf
137	DecisionTreeClassifier	0.836131	None	None	NaN	entropy	4.0	7.0
138	DecisionTreeClassifier	0.836131	None	None	NaN	entropy	4.0	8.0
135	DecisionTreeClassifier	0.835007	None	None	NaN	entropy	4.0	5.0
136	DecisionTreeClassifier	0.835007	None	None	NaN	entropy	4.0	6.0
139	DecisionTreeClassifier	0.835001	None	None	NaN	entropy	4.0	9.0
133	DecisionTreeClassifier	0.832760	None	None	NaN	entropy	4.0	3.0
102	DecisionTreeClassifier	0.831662	None	None	NaN	entropy	8.0	8.0
134	DecisionTreeClassifier	0.831636	None	None	NaN	entropy	4.0	4.0
131	DecisionTreeClassifier	0.831636	None	None	NaN	entropy	4.0	1.0
93	DecisionTreeClassifier	0.830538	None	None	NaN	entropy	10.0	8.0

- 엔트로피를 사용하여 가지가 생성되면서 max_depth가 4, min_samples_leaf가 7일 때 가장 좋은 성능이 나옴

2.5 의사결정트리 시각화하기

- 의사결정트리는 데이터 분석가에게 어떤 피쳐가 가장 중요한지를 보여줌

In [40]:	estimator_results[1].best_estimator_.feature_importances_
Out [40]:	array([0.13457412, 0.06268058, 0.04328291, 0. , 0. , 0.18776793, 0.01340804, 0. , 0. , 0. , 0. , 0. , 0.04353206, 0. , 0.51475435, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.])
In [41]:	X_df.columns
Out [41]:	Index(['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'cabin_count', 'Embarked_C', 'Embarked_Q', 'Embarked_S', 'social_status_0', 'social_status_Col', 'social_status_Dr',

Out [40]:	'social_status_Master', 'social_status_Miss', 'social_status_Mr', 'social_status_Mrs', 'social_status_Rev', 'cabin_type_99', 'cabin_type_A', 'cabin_type_B', 'cabin_type_C', 'cabin_type_D', 'cabin_type_E', 'cabin_type_F', 'cabin_type_G', 'cabin_type_T'], dtype='object')

- 인덱스를 정렬하고 제일 중요한 속성을 출력

In [42]:	coef = estimator_results[1].best_estimator_.feature_importances_ coef.argsort()[::-1]
Out [42]:	array([15, 5, 0, 1, 13, 2, 6, 21, 20, 24, 3, 4, 23, 7, 8, 9, 10, 11, 12, 25, 14, 22, 16, 17, 18, 19, 26])
In [43]:	X_df.columns[coef.argsort()[::-1]][:5]
Out [43]:	Index(['social_status_Mr', 'Fare', 'Pclass', 'Sex', 'social_status_Master'], dtype='object')

05 의사결정트리의 구현

- 의사결정트리를 시각화

```
In [44]: !pip install pydotplus
```

pydotplus의 원활한 수행을 위한 필수 설치 : graphviz

윈도우의 경우 pydotplus 설치 완료한 다음 가상환경에서 다음과 같이 입력하여 가상환경을 다시 설치하고 수행하는 것을 추천한다. 만약 아래 설치 중 pydotplus까지 설치를 이미 완료한 상태라면 graphviz만 설치한다.

```
conda create -n tree python=3.8
conda install jupyter
conda install pandas
conda install scikit-learn
conda install seaborn
conda install pydotplus
conda install graphviz
```

05 의사결정트리의 구현

그림 12-17 가상환경에서 graphviz 설치하기

추가적으로 윈도우 환경에서서 graphviz를 설치해야 한다. 이를 위해서 먼저 웹사이트 https://graphviz.org/download/ 에 접속하여 'graphviz-2.49.3'의 64비트 버전을 다운로드하여 설치한다.

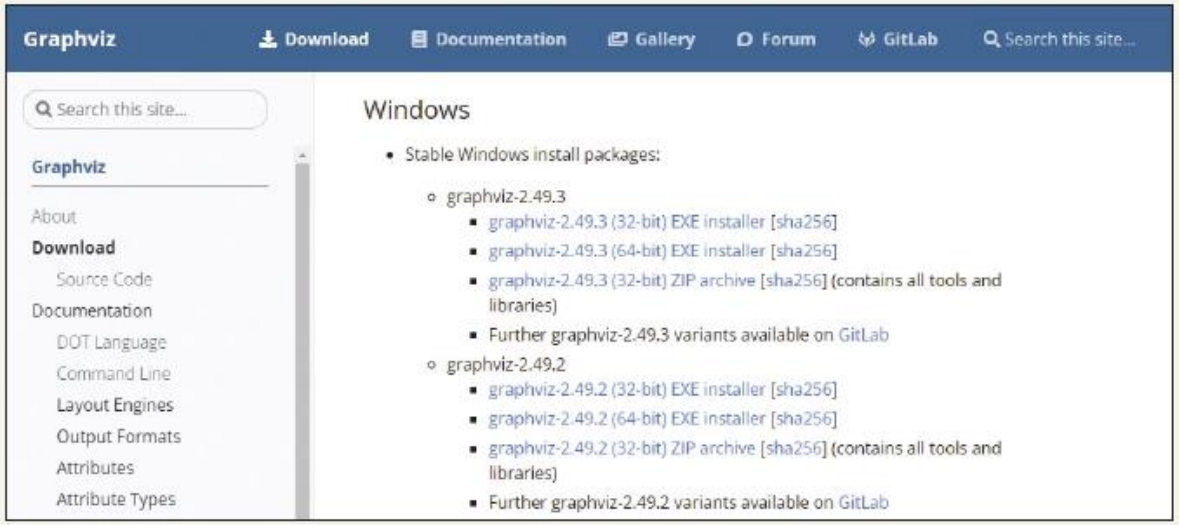


그림 12-18 Graphviz 웹사이트

마지막으로 주피터 노트북에서 다음 코드를 입력하면, PATH를 설정하여 해당 프로그램이 주피터 노트북에 설치된다. 이 과정을 모두 마쳤다면 pydotplus가 수행된다.

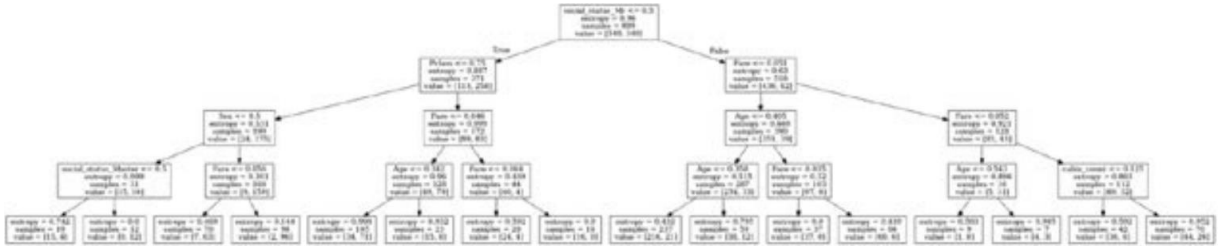
In [45]:	import os os.environ["PATH"] += os.pathsep + 'C:\Program Files\Graphviz/bin/'

05 의사결정트리의 구현

In [46]:

```
import pydotplus
from six import StringIO
from sklearn import tree

best_tree = estimator_results[1].best_estimator_
column_names = X_df.columns

dot_data = StringIO()
tree.export_graphviz(best_tree, out_file=dot_data,
                     feature_names=column_names)

graph =
pydotplus.pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png("titanic.png")
from IPython.core.display import Image
Image(filename='titanic.png')
```

Out [46]:



05 의사결정트리의 구현

- 하나의 노드를 기준으로 어떤 속성에 대해서 어떤 기준으로 가지가 발생했는지, 각 가지마다 데이터의 개수가 어떻게 나누어지는지 확인 가능

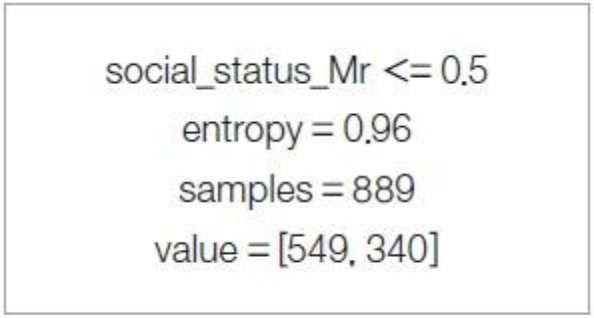


그림 12-19 titanic.png 파일에 있는 의사결정트리의 가지(branch) 일부