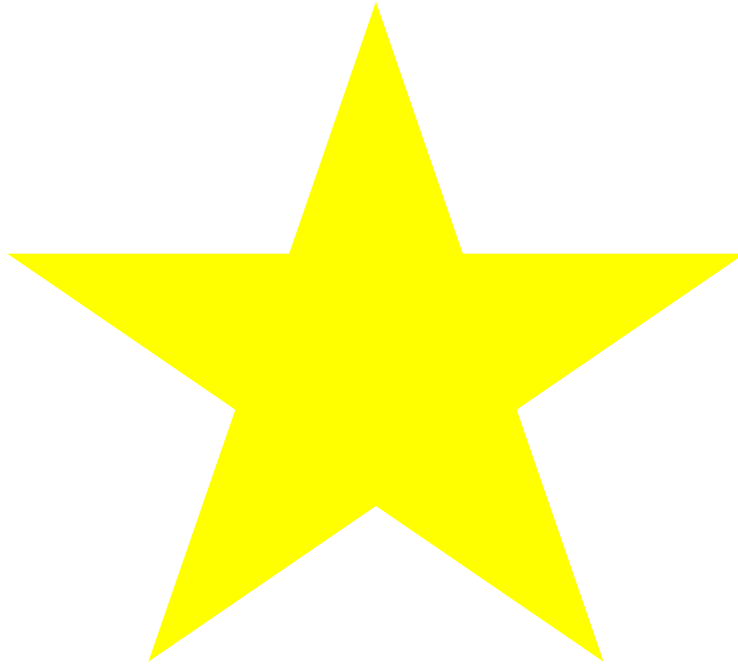


# Decision Tree & Rule Learner



3조 ( 김성산 / 이승연 / 이승한 )

# 목차

## < Decision Tree >

- 1) 개념 & 예시
- 2) 관련 용어 설명
- 3) 분류 기준 설정
- 4) 장.단점
- 5) 모델 개선하기

## < Rule Learner >

- 1) 개념 & 예시
- 2) 모델 개선하기
- 3) 장.단점

# 1) Decision Tree의 개념 & 예시

## 1) 정의

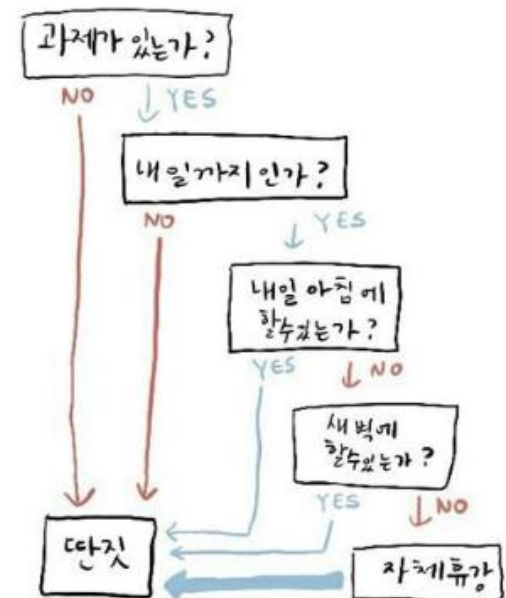
의사결정 규칙을 **나무 구조**로 나타내어,

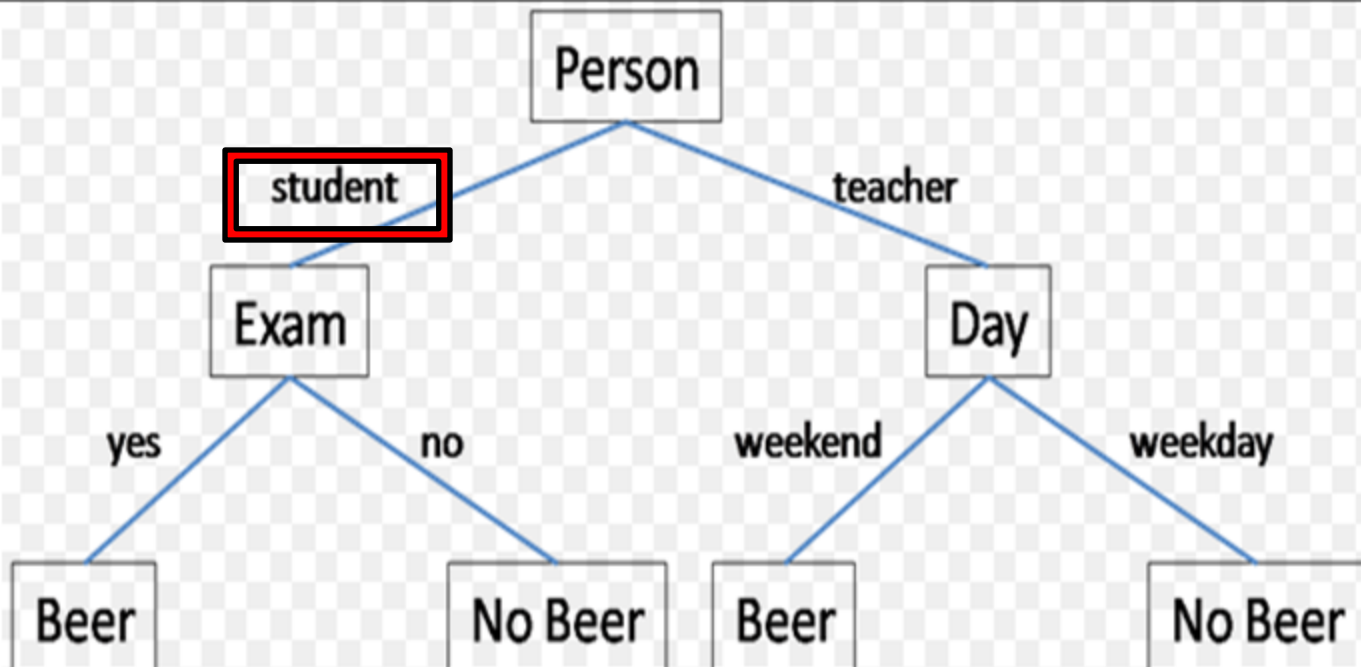
전체 자료를 **몇 개의 소집단으로 분류하거나 예측**하는 분석 방법

## 2) 종류

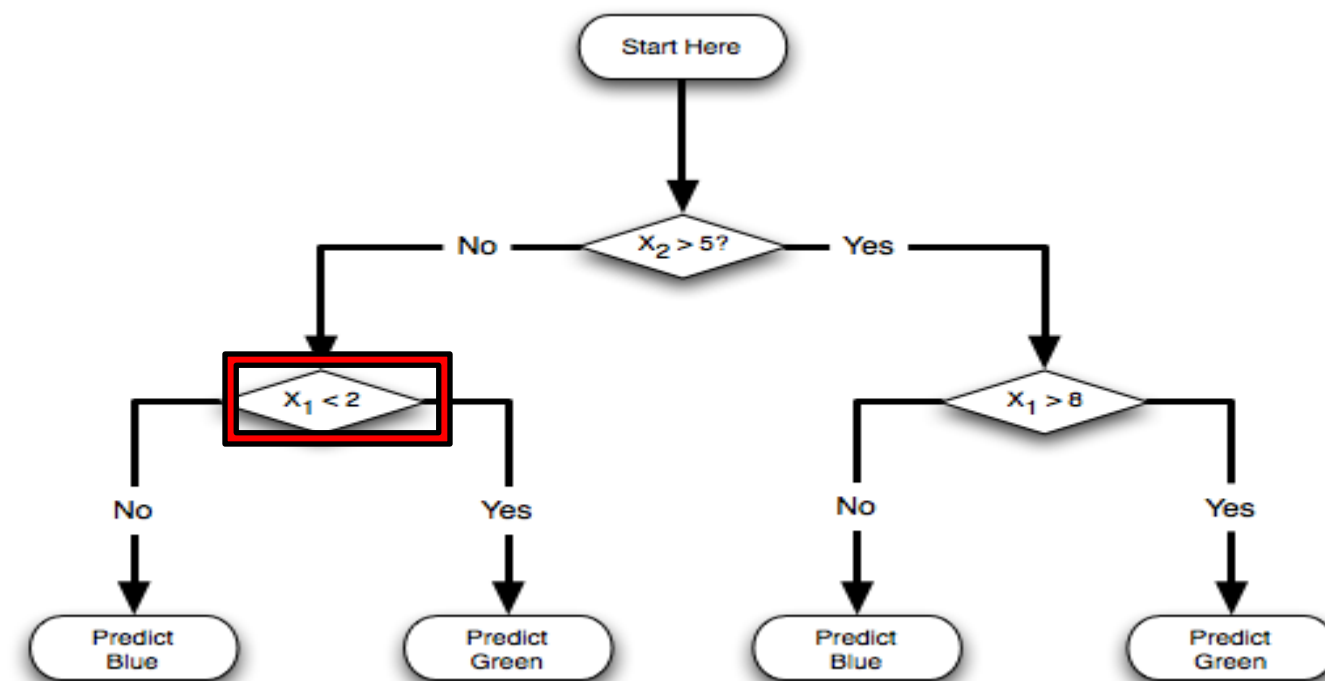
- **분류 나무** : 목표변수가 "이산형"인 경우
- **회귀 나무** : 목표변수가 "연속형"인 경우

과제 알고리즘



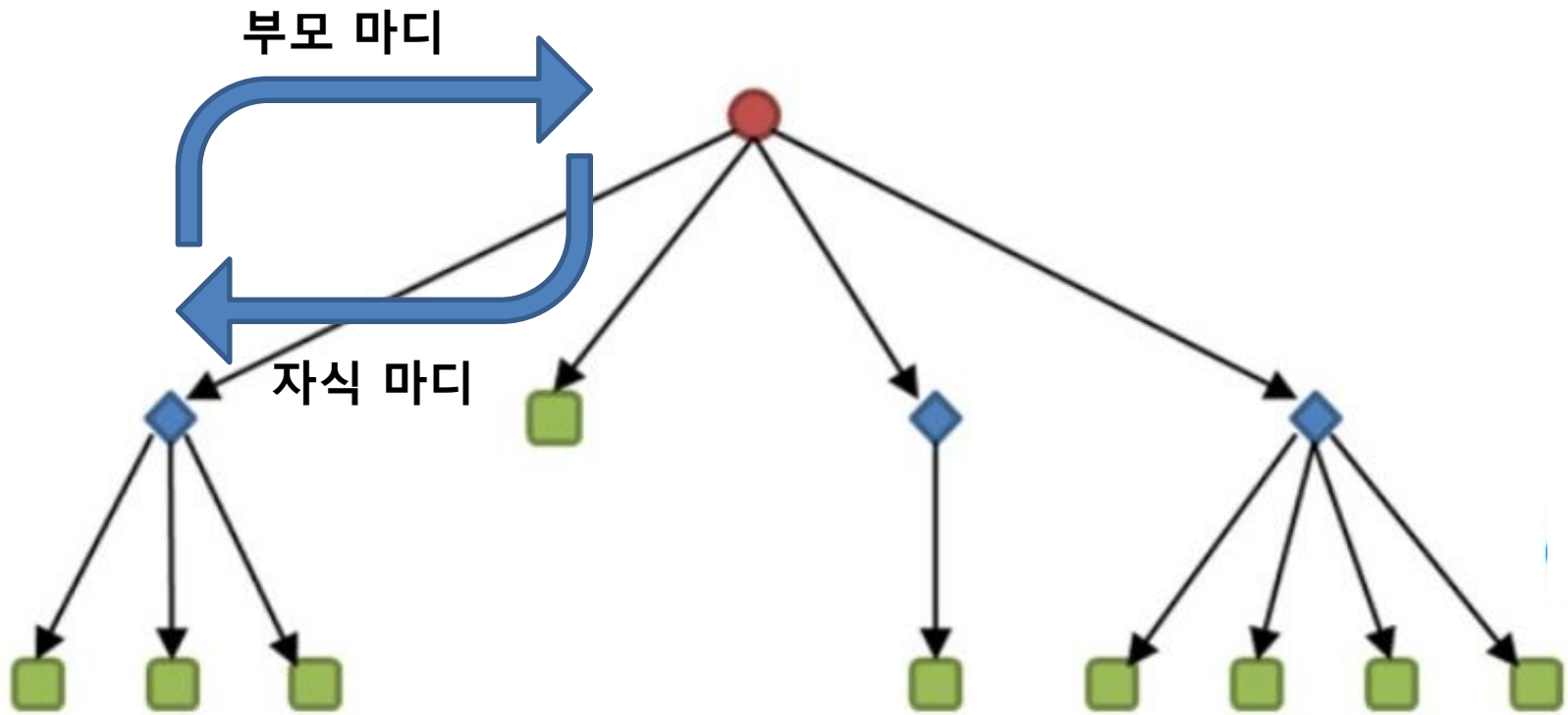


분류 기준 :  
범주형



분류 기준 :  
수치형

## 2) Decision Tree의 용어 설명



- Root Node
- ◆ Internal Node
- Leaf Node

- 뿌리 마디** : 시작되는 마디 ( 전체 자료를 포함 )
- 중간 마디** : 부모 마디와 자식 마디가 모두 있는 마디
- 최종 마디** : 자식마디가 없는 마디

- 가지(branch) : 뿌리 ~ 최종마디까지 연결된 마디들
- 깊이(depth) : 뿌리 ~ 최종마디까지의 중간 마디들의 '수'
- 가지 분할 : 나무의 가지를 생성하는 과정
- 가지 치기 : 생성된 가지를 잘라내어 모형을 단순화하는 과정

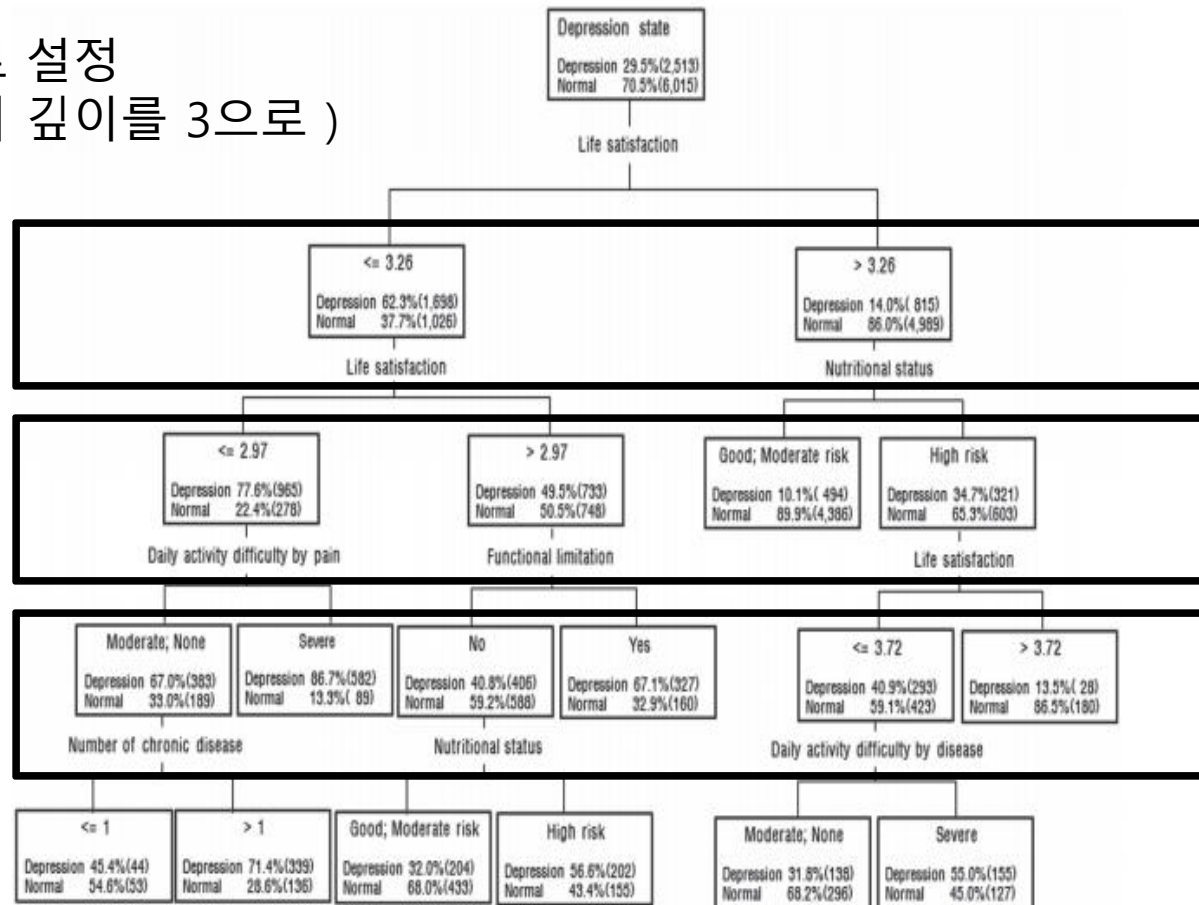
---> **for 과대적합 방지**

- 1) **사전 가지치기** ( pre-pruning ) : by 정지규칙
- 2) **사후 가지치기** ( post-pruning ) : 트리 만든 후, 데이터 포인트 적은 노드 삭제/병합

정지규칙 : 더 이상 분리가 일어나지 않고, 현재의 마디가  
최종마디가 되도록 하는 여러 규칙  
→ ex : 최대 나무의 깊이, 자식 마디의 최소 관측치 수....

## 사전 가지치기

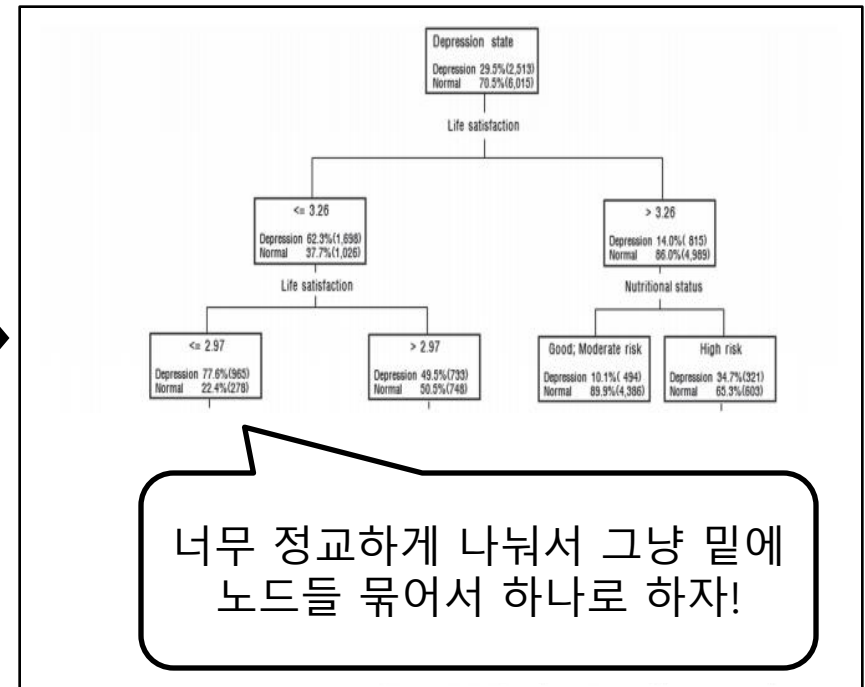
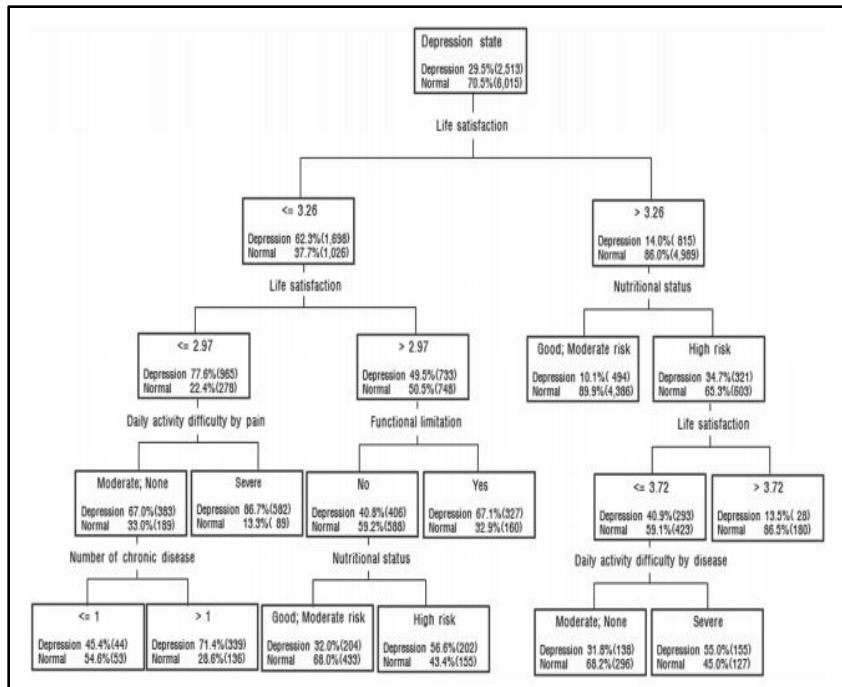
Depth=3으로 설정  
( 최대 나무의 깊이를 3으로 )



흠...완벽하게 분류되지는  
않았지만 여기서 STOP!

## 사후 가지치기

이미 트리를 다 만들고 난 후에, 몇 개의 노드를 삭제하거나 병합해서 좀 더 일반적인 model로 만들기



너무 정교하게 나뉘어서 그냥 밑에  
노드들 묶어서 하나로 하자!



### 3) Decision Tree의 분류 기준 설정

1. 엔트로피 지수

2. 지니 지수

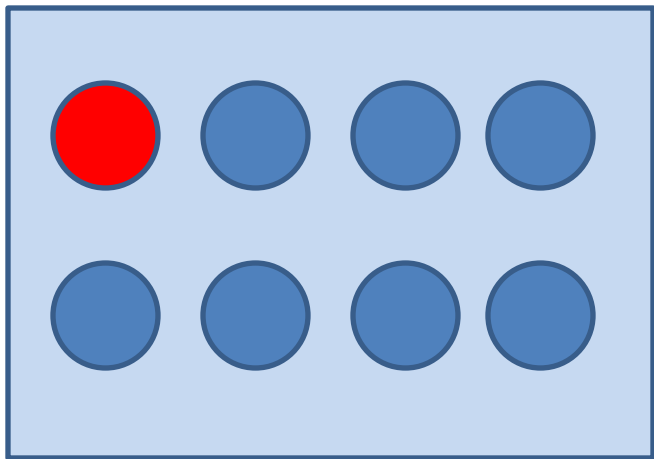
무엇을 분류 기준으로??



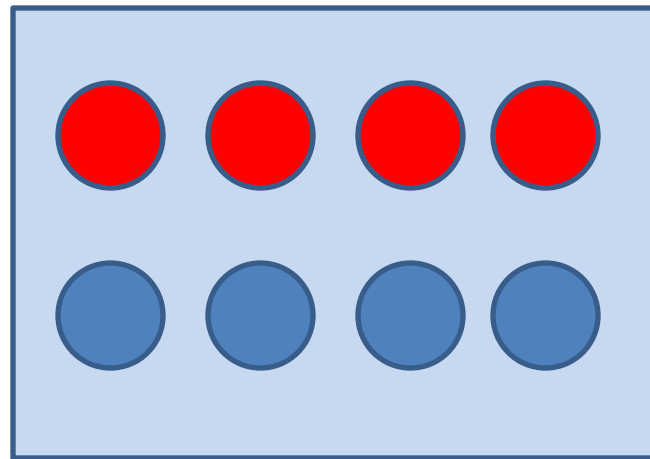
# 1. 엔트로피 지수

<한 줄 요약>

분류기준으로 의해 나누어진 "노드의 구성 클래스가 얼마나 **이질적**"인가?



노드 A : Good



노드 B : Bad

하나의 노드의 Entropy

$$\text{Entropy}(S) = \sum_{i=1}^c -p_i \log_2(p_i)$$

여러 Entropy들의 가중평균

$$\text{Entropy}(S) = \sum_{i=1}^n w_i \text{Entropy}(P_i)$$

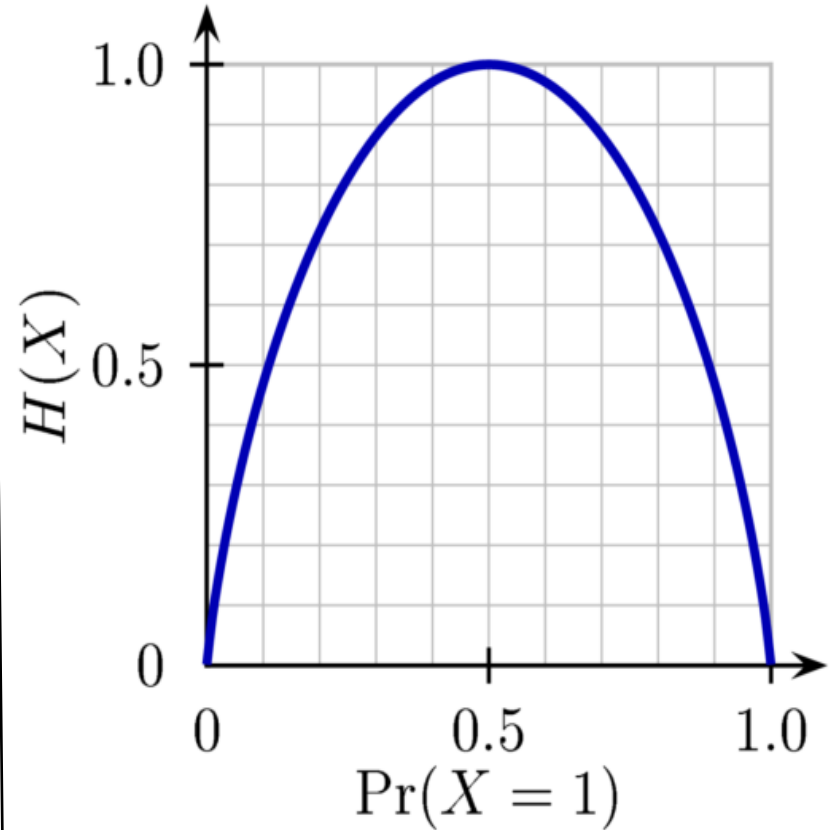
( p : 해당 클래스가 노드 내에서 차지하는 비율 )

노드 A

```
> # 노드 A는 클래스1 (0.875) 클래스2 (0.125)로 구성  
> -0.125*log2(0.125) - 0.875*log2(0.875)  
[1] 0.5435644
```

노드 B

```
> # 노드 B는 클래스1 (0.5) 클래스2 (0.5)로 구성  
> -0.50*log2(0.50) - 0.50*log2(0.50)  
[1] 1
```



# 정보 이익 (Information Gain)

$$\text{InfoGain}(F) = \text{Entropy}(S1) - \text{Entropy}(S2)$$

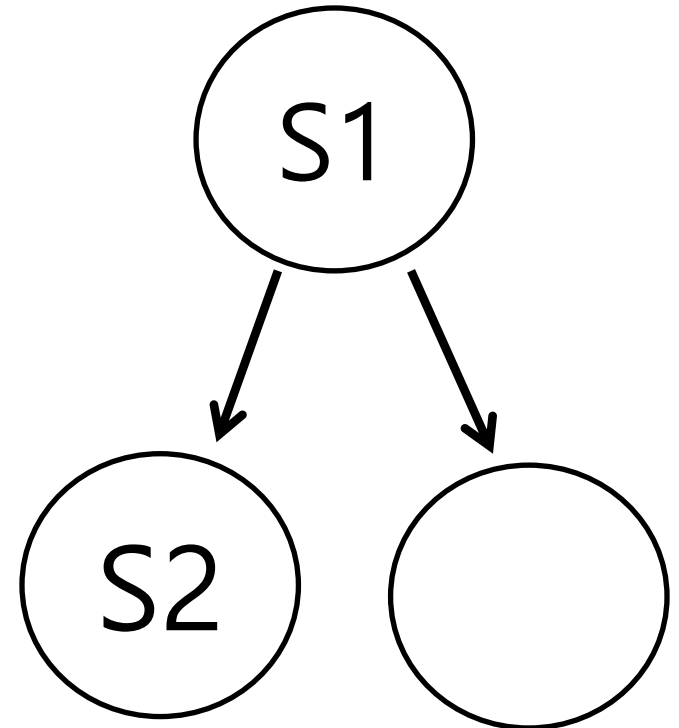
( 이 값이 "커야" 좋음 )

1) Entropy(S1)이 커야

-> 얼마나 다양하게 섞인 데이터에서

2) Entropy(S2)이 작아야

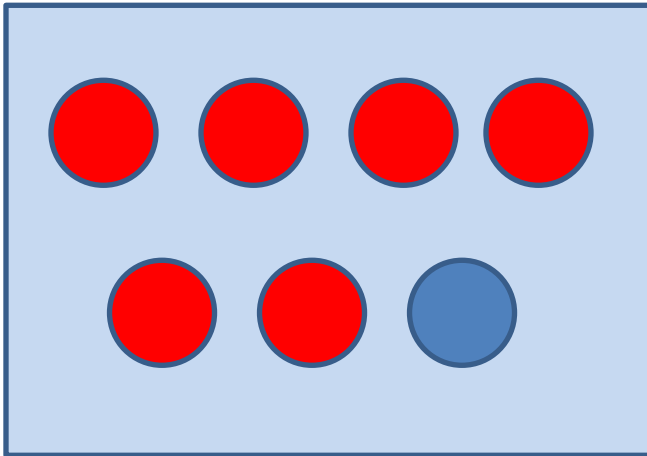
-> 얼마나 노드 내의 클래스들을 동질적으로



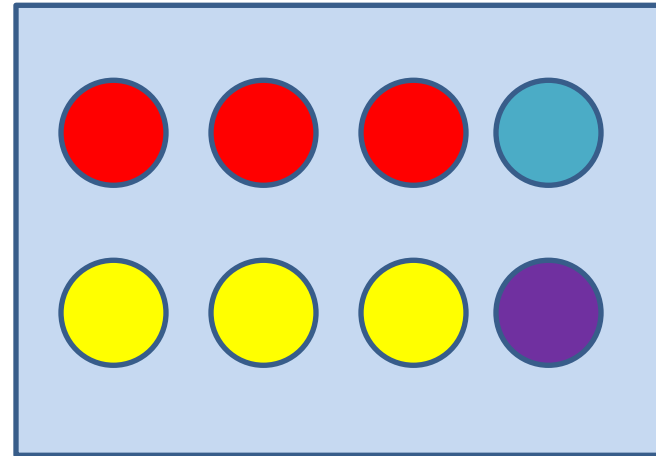
## 2. 지니 지수 (지니 불순도)

$$Gini = 1 - \sum_{l=1}^k p_l^2$$

$p_l (l = 1, \dots, k)$



노드 A : Good



노드 B : Bad

$$1 - (6/7)^2 - (1/7)^2 = \mathbf{0.24}$$

$$1 - (3/8)^2 - (3/8)^2 - (1/8)^2 - (1/8)^2 = \mathbf{0.69}$$

즉, 엔트로피 지수, 지니 지수가

둘 다 “**작도록**” 분류 기준을 설정해야!!!

# 4) Decision Tree의 장.단점

## < 장점 >

- 1) 구조가 단순하여 해석이 용이
- 2) 선형성, 정규성, 등분산성 등의 수학적 가정이 불필요한 비모수적 모형
- 3) 계산 비용이 낮아 대규모의 데이터셋에서도 빠르게 연산 가능
- 4) 수치형/범주형 변수 모두 사용가능

## < 단점 >

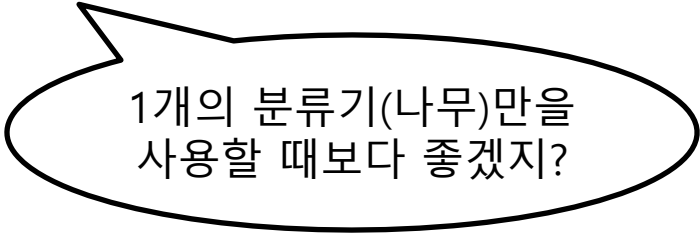
- 1) 분류 기준값의 경계선 부근의 자료값에 대해서는 오차가 크다 (비연속성)
  - 2) 로지스틱 회귀와 같이 각 예측변수의 효과 파악 어려움
  - 3) 새로운 자료에 대한 예측이 불안정
- ( -> 곧 과대적합함을 의미 )
- ( -> 가지치기 등을 통해 일반화 노력을 하지만, 그래도 다른 모델에 비해선 과대적합한 성향 O )

# 5) 모델 개선하기

## “앙상블 모형”

여러 개의 분류모형의 결과를 종합하여 분류의 정확도를 높이는 방법!

1) **배깅** : 여러 개의 분류모형에 의한 결과를 종합



1개의 분류기(나무)만을 사용할 때보다 좋겠지?

2) **부스팅** : 배깅과 유사

( 차이점 : 분류하기 더 애매한 데이터에 '가중치'를 둠

-> "경계 부근에 있는 data"를 더 신경써서 트리 생성 )

3) **랜덤 포레스트** : 배깅에 "랜덤" 과정을 추가

( 쉽게 말하면, 데이터를 자식 노드로 나누는 기준을 정할 때, '전체 변수'가 아니라

'일부 변수'만을 랜덤으로 뽑아서 하여 가지를 나눔 )



# “랜덤 포레스트”

(간단히...자세한건 11단원)

의사결정 나무만을 사용한다면 “과적합”이 일어날 확률이 높기 때문에, 이를 해결하기 위해 랜덤하게 여러 개의 트리를 만드는 방법

포레스트 : 숲 = 나무의 모음

랜덤 : 특성을 전부(X) 일부(O)만 랜덤하게 선택

ex) 기존의 의사결정나무 : 키,몸무게,국적,인종, 성별 등을 특성으로 이용해 tree 1개 만들기

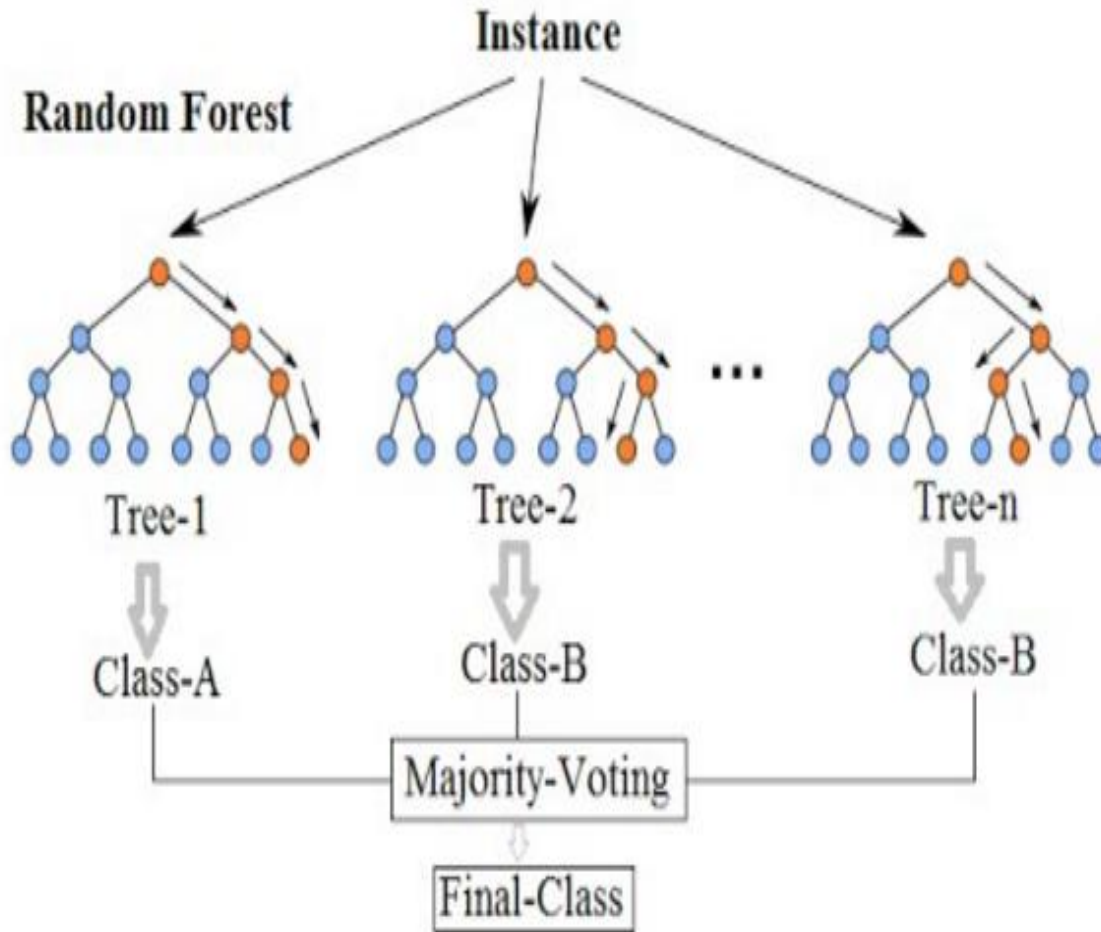
랜덤 포레스트 : 키,몸무게, 국적 이용해 tree 1개

인종,성별,국적 이용해 tree 1개

국적,몸무게,성별 이용해 tree 1개.....

-> 이런 거로 여러 개의 tree를 만듦!

# Random Forest Simplified



1개의 의사결정 나무보다...

- 1) 높은 정확성
- 2) 높은 일반화  
(과적합 해결)

# 6) Rule Learner의 개념 & 예시

Rule Learner? 최적의 분류 규칙을 찾는 알고리즘

## < 1R 알고리즘 >

전체의 데이터를 제일 잘 나눌 수 있는 "하나"의 규칙찾기!

```
> mushroom_1R
```

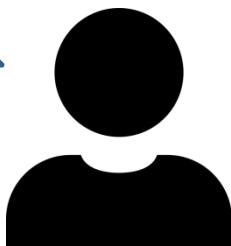
```
odor:
```

a	-> e
c	-> p
f	-> p
l	-> e
m	-> p
n	-> e
p	-> p
s	-> p
y	-> p

```
(8004/8124 instances correct)
```

8,124 송이의 버섯을 식용/독성으로 구분할 수 있는 제일 좋은 규칙(기준)은 "(odor)냄새" 이다!

( 그 많은 버섯들을 '단 하나'의 기준으로만 나누다 보면 에러가 좀 많겠지....? )



# 7) 모델 개선하기

## < RIPPER 알고리즘 >

( Repeated Incremental Pruning to Produce Error Reduction )

전체의 데이터를 제일 잘 나눌 수 있는 "여러 개"의 규칙찾기!

JRIP rules:

=====

```
(odor = f) => type=p (2160.0/0.0)
(gill_size = n) and (gill_color = b) => type=p (1152.0/0.0)
(gill_size = n) and (odor = p) => type=p (256.0/0.0)
(odor = c) => type=p (192.0/0.0)
(spore_print_color = r) => type=p (72.0/0.0)
(stalk surface below ring = y) and (stalk surface above ring = k) => type=p (68.0/0.0)
(habitat = l) and (cap_color = w) => type=p (8.0/0.0)
(stalk_color above ring = y) => type=p (8.0/0.0)
=> type=e (4208.0/0.0)
```

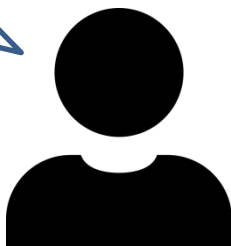
Number of Rules : 9

사용한 규칙들?

(1R) odor → 1개

(RIPPER) odor, gill\_size, gill\_color → 9개

이건 '여러 개의' 기준으로 나누다 보니,  
1R 알고리즘보다 에러가 훨씬 줄지!



# 8) Rule Learner의 장.단점

## < 1R 알고리즘 >

### < 장점 >

- 1) 이해하기 쉬움
- 2) 더 복잡한 알고리즘을 위한 기초가 될 수 있음

### < 단점 >

- 1) 하나의 특징만을 사용한다 -> 너무 단순

## < RIPPER 알고리즘 >

### < 장점 >

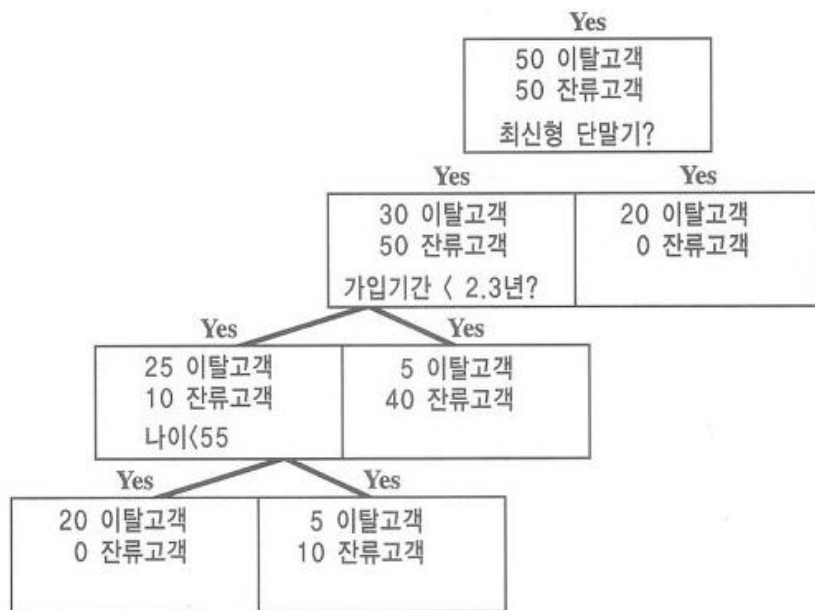
- 1) 이해하기 쉬움
- 2) 노이즈 값에 크게 영향 받지 않음
- 3) 의사결정나무 모형에 비해 간단한 모델을 만듦

### < 단점 >

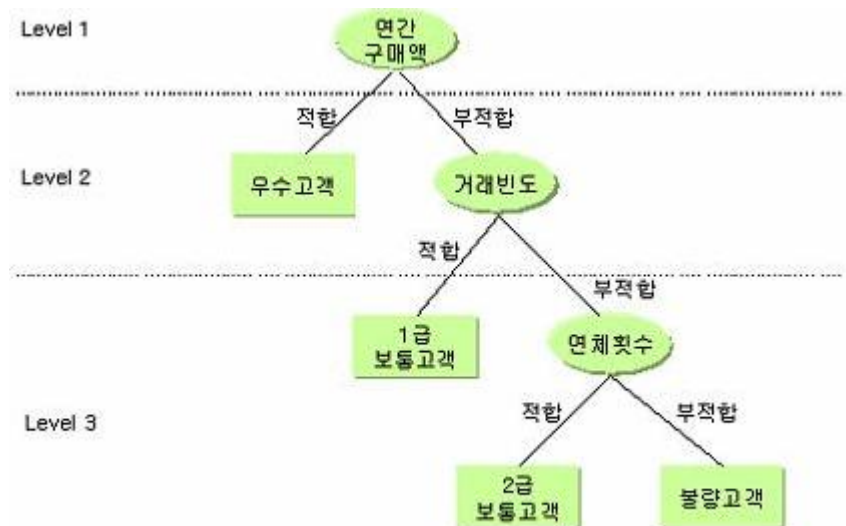
- 1) 수치형 데이터에는 적용하기 좋지 않음
- 2) 다른 복잡한 모델 만큼 좋은 성능을 내지는 못함

# 지금까지 배운 Decision Tree & Rule Learning의 활용 분야는?

고객 타겟팅, 고객들의 신용 점수화, 고객행동 예측,  
고객 세분화, 캠페인 반응 분석 등...



Ex) 기업의 고객 분석



Ex) 고객들의 신용 등급

감사합니다