분류 (Classification)

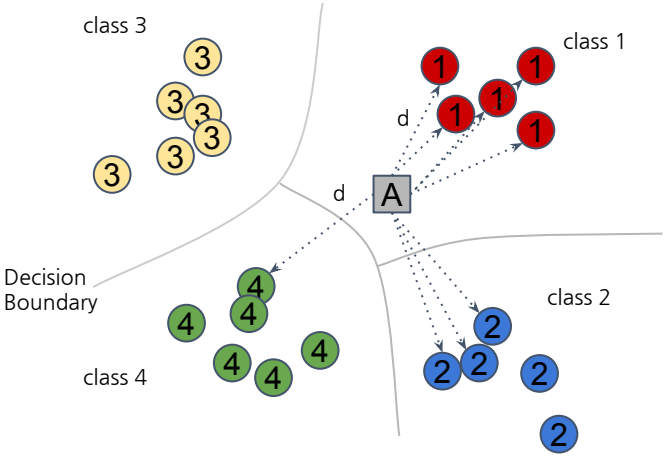
분류 (Classification)

- 분류(classification)는 학습 데이터로 주어진 데이터의 피처와 레이블값(결정 값, 클래스 값)을 머신러닝 알고리즘으로 학습해 모델을 생성하고, 이렇게 생성된 모델에 새로운 데이터 값이 주어졌을 때 미지의 레이블 값을 예측

- 대표적인 분류 알고리즘
  - 베이즈(Bayes) 통계와 생성 모델에 기반한 나이브 베이즈(Naive Bayes)
  - 독립변수와 종속변수의 선형 관계성에 기반한 로지스틱 회귀 (Logistic Regression)
  - 데이터 균일도에 따른 규칙 기반의 의사 결정 나무(Decision Tree)
  - 개별 클래스 간의 최대 분류 마진을 효과적으로 찾는 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)

KNN (K Nearest Neighbor)

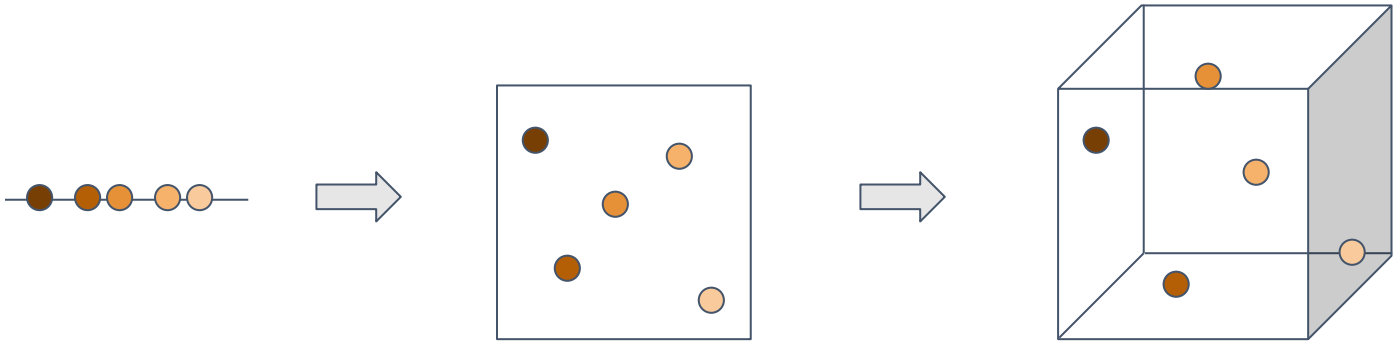
- 점 A로 부터 가까운 거리에 있는 K개의 점을 선택한 후, K 개의 점들이 가장 많이 속한 클래스를 찾아 점 A가 그 클래스에 속한다고 평가하는 알고리즘



1. 점 A와 모든 훈련 데이터 사이의 거리를 측정한다.
   -거리는 평면상의 2점 사이의 거리로 측정 (Euclidean distance)
2. 측정한 거리가 작은 순으로 K 개의 점을 찾는다 (ex : K = 9)
3. K개의 점들이 속한 클래스를 찾는다
4. K개의 점들이 속한 클래스가 가장 많은 것을 찾는다. (다수결 원칙)
   클래스 집합 = {1, 1, 1, 1, 2, 2, 2, 4} 라면 클래스 1이 가장 많다.
5. 점 A를 클래스 1로 분류한다.
6. 시험 데이터를 이용하여 분류가 잘 되었는지 평가한다
7. K값과 훈련 데이터, 시험 데이터를 바꾸어 가면서 가장 정확하게 분류하는 K값을 찾는다.
   - Cross Validation

KNN (K Nearest Neighbor)

- 차원의 저주 (Curse of Dimensionality)
  - KNN 분류기는 훈련 데이터가 충분히 많으면 좋은 성능을 나타내지만, 차원이 증가하면 (Feature 개수가 많아지면) 성능이 저하된다. (차원의 저주)
  - 차원이 증가할수록 고차원 공간을 채울 데이터가 많이 필요하고, 멀리 떨어진 훈련 데이터를 참조해야하기 때문에 '근접이웃'에 한정하기 어렵다.



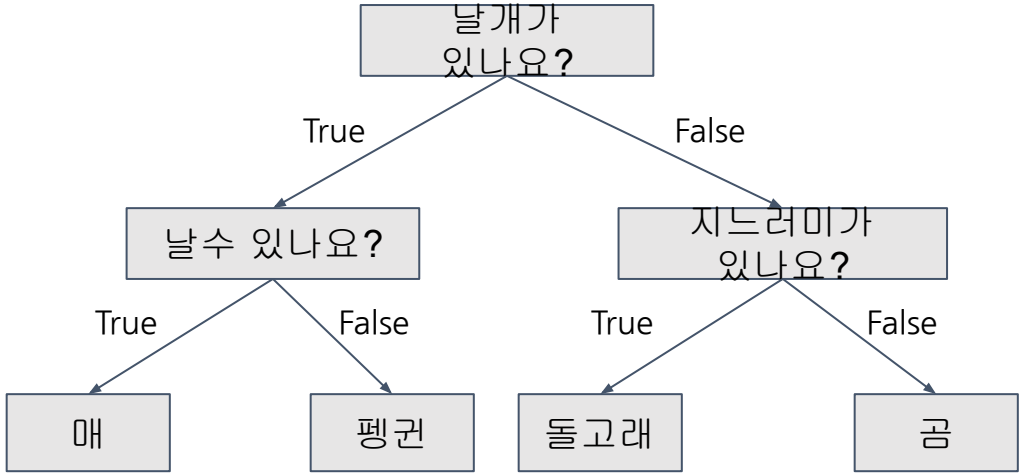
KNN

실습

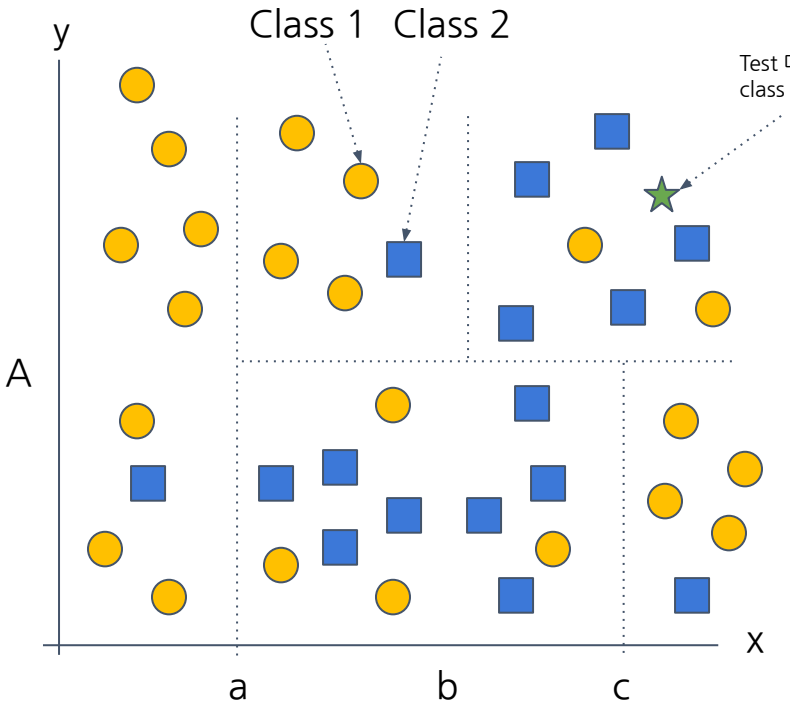
KNN

의사 결정 나무(Decision Tree)

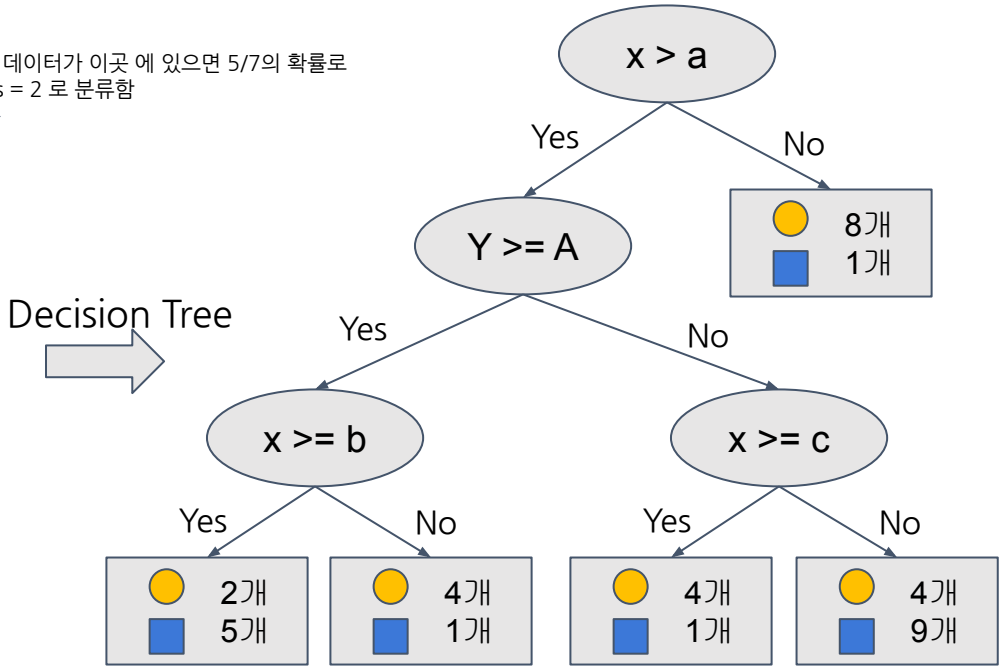
- 데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 트리 기반의 분류 규칙을 만든다
- 데이터를 어떤 기준을 바탕으로 규칙을 만들어야 가장 효율적인 분류가 될 것인가가 중요하다



의사 결정 나무 (Decision Tree)



Decision Tree



불순척도(impurity) 와 최적의 분할 선택 기준

- 불순척도 :
  - 데이터 분할 시, 분할 전보다 후에 불순한 정도가 줄어드는 것이 좋다고 가정한다. 여기서의 '불순'이란, 데이터가 섞인 정도이다.
  - 이질적인 데이터가 많을수록, 해당 노드에서 불순도가 높다고 본다.
  - 즉, 부모 노드에서 자식 노드로 갈 때 불순도가 덜해지도록, 혹은 최대한 많이 감소하도록 해야 한다.

- 엔트로피 : $H(t) = -\sum_{i=1}^{c} p(i|t) \log_2 p(i|t)$

- 지니 지수 : $Gini(t) = 1 - \sum_{i=1}^{c} p(i|t)^2$

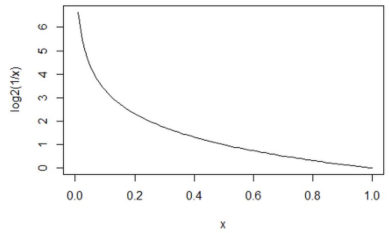
$p(i|t)^2$ 은 특정 노드 t에서 클래스 i에 속한 샘플 비율

정보 획득(information gain)

정보이론의 핵심 아이디어는 잘 일어나지 않는 사건(unlikely event)은 자주 발생하는 사건보다 정보량이 많다(informative)는 것. 예컨대 '아침에 해가 뜬다'는 메시지로 보낼 필요가 없을 정도로 정보 가치가 없다. 그러나 '오늘 아침에 일식이 있었다'는 메시지는 정보량 측면에서 매우 중요한 사건. 이 아이디어를 공식화해서 표현하면

- 자주 발생하는 사건은 낮은 정보량을 가진다. 발생이 보장된 사건은 그 내용에 상관없이 전혀 정보가 없다는 걸 뜻한다.
- 덜 자주 발생하는 사건은 더 높은 정보량을 가진다.
- 독립사건(independent event)은 추가적인 정보량(additive information)을 가진다. 예컨대 동전을 던져 앞면이 두번 나오는 사건에 대한 정보량은 동전을 던져 앞면이 한번 나오는 정보량의 두 배이다.

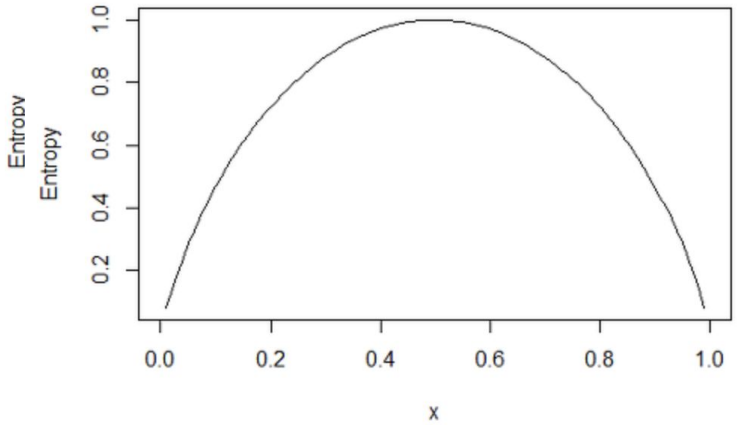
정보획득량 이라는 것은 어떤 사건이 얼마만큼의 정보를 줄 수 있는지를 수치화한 값.
정보 함수는 정보의 가치를 반환하는데 발생할 확률이 작은 사건일수록 정보의 가치가 크고 반대로 발생할 확률이 큰 사건일수록 정보의 가치가 작다. 정보 함수는 $I(x) = \log_2 \frac{1}{p(x)}$



정보 획득(information gain)

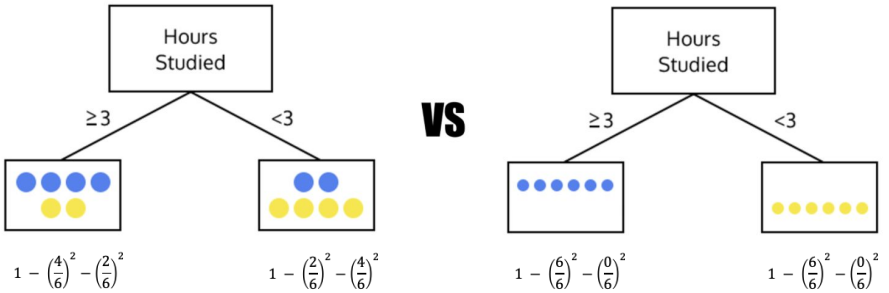
- 엔트로피란 무질서도를 정량화해서 표현한 값. 어떤 집합의 엔트로피가 높을 수록 그 집단의 특징을 찾는 것이 어렵다는 것을 의미. 따라서 의사결정트리의 잎 노드들의 엔트로피가 최소가 되는 방향으로 분류해 나가는 것이 최적의 방법으로 분류한 것이라고 할 수 있습니다.

$$E(S) = \sum p_i \, I(x_i) = \sum p_i \, \log_2 \frac{1}{p_i}$$



정보 획득(information gain)

- 지니 불순도

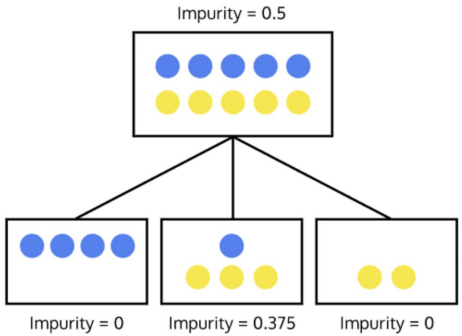


$$Gini\,(t) = 1 - \sum_{i=1}^{c} p\,(i|t)^2$$

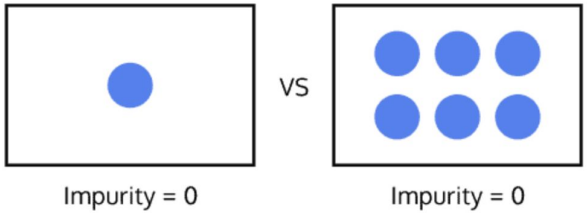
$1 - \left(\frac{4}{6}\right)^2 - \left(\frac{2}{6}\right)^2$ $1 - \left(\frac{2}{6}\right)^2 - \left(\frac{4}{6}\right)^2$ $1 - \left(\frac{6}{6}\right)^2 - \left(\frac{0}{6}\right)^2$ $1 - \left(\frac{6}{6}\right)^2 - \left(\frac{0}{6}\right)^2$

- Information Gain = 0.5 – (0 + 0.375 + 0) = 0.125

➡️ 분할된 데이터 세트들의 불순도가 작을수록 정보 획득량은 증



정보 획득(information gain)

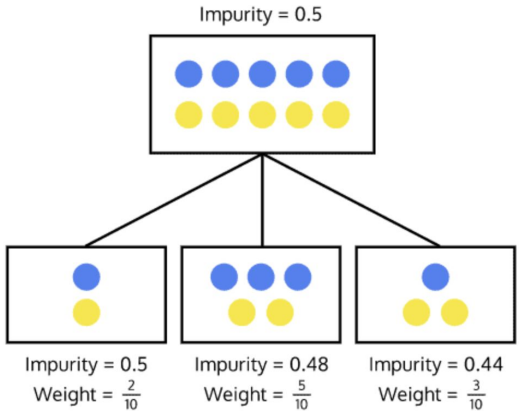


➡️ 두 개의 데이터 세트 모두 불순도는 0이지만, 오른쪽 데이터 세트가 더 의미있는 것처럼 보인다. 그 이유는 데이터 개수가 충분히 많고, 따라서 이 분류가 우연이 아니라고 확신할 수 있기 때문.

- **Weighted Information Gain**

➡️ 분할하기 전 데이터에 비해 분할 후 생성된 데이터의 크기(비율)에 I

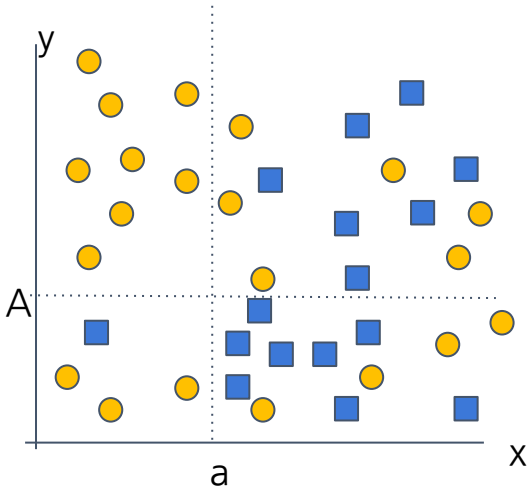
가중치를 구해놓고 이를 불순도에 곱해서 정보 획득량을 구하면 된다.



0.5 – ((2/10)*0.5 + (5/10)*0.48 + (3/10)*0.44) = 0.026

불순척도(impurity) 와 최적의 분할 선택 기준

$$Gini\,(t) = 1 - \sum_{i=1}^{c} p\,(i|t)^2$$



1. 초기 분할 : x=a인 경우

Left: 11개, 1 개

$$G_L = 1 - \left(\frac{11}{12}\right)^2 - \left(\frac{1}{12}\right)^2 = 0.153$$

Right: 10개, 16개

$$G_R = 1 - \left(\frac{10}{26}\right)^2 - \left(\frac{16}{26}\right)^2 = 0.473$$

$$E_a = \left(\frac{12}{38}\right) * 0.153 + \left(\frac{26}{38}\right) * 0.473 = 0.372$$

2. 초기 분할 : y=A인 경우

Upper: 14개, 7개

$$G_U = 1 - \left(\frac{14}{21}\right)^2 - \left(\frac{7}{21}\right)^2 = 0.444$$

Below: 7개, 10개

$$G_B = 1 - \left(\frac{7}{17}\right)^2 - \left(\frac{10}{17}\right)^2 = 0.484$$

$$E_A = \left(\frac{21}{38}\right) * 0.444 + \left(\frac{17}{38}\right) * 0.484 = 0.462$$

의사 결정 나무(Decision Tree) 특징

- 분류나 예측의 근거를 알 수 있으므로 결정 과정을 쉽게 이해할 수 있다
- 데이터의 차원이 높아져도(feature가 많아지더라도) 분류에 중요한 feature들을 제외할 수 있으므로, feature 선정 단계에서 크게 신경쓸 필요가 없다.
- feature 마다 분류에 영향을 미치는 정도를 파악할 수 있다, 중요한 feature들을 찾아볼 수 있다

  ➡️ 중요도 분석

- Decision Tree를 과도하게 분할할 경우 Tree가 복잡해지고 과잉적합(Overfitting) 문제가 발생한다
- 트리가 복잡해지고, 과잉적합 문제를 개선하기 위해서는 트리가 더 이상 커지지 않도록 하는 정지기준(Stopping rule)과 가지치기(Pruning) 방법이 있다

의사 결정 나무(Decision Tree)

실습

Decision Tree

Graphviz

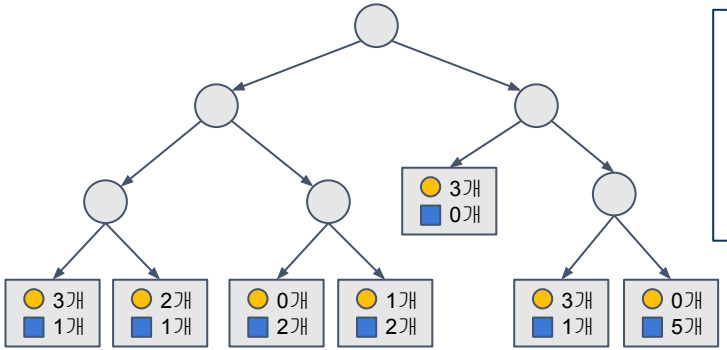
- Decision Tree 모델의 시각화
  - https://graphviz.org/ (2.38 버전 - https://www.npackd.org/p/org.graphviz.Graphviz/2.38)
  - Graphviz 실행 파일 설치
  - Graphviz 파이썬 패키지 설치 - pip install graphviz
  - OS 환경 변수 구성

가지치기(Pruning)

- 트리가 너무 복잡해지지 않도록 단순화 시킨다 → 일반화 특성을 향상 시킴
- 평가용 데이터(Validation Data)를 이용하여 일반화 특성이 좋아지는 지점에서 트리 성장을 멈추거나, 데이터 전문가에 의해 타당성이 없는 규칙은 제거한다.
- 사전 가지치기(Pre-pruning)
  - ➡️ 조기 정지 규칙에 의해 트리 성장을 멈춤. 정지 규칙으로는 트리의 깊이, 마지막 노드의 최소 데이터 수, 불순 척도 등의 임계치를 이용하는 방법 등이 있음
- 사후 가지치기(Post-pruning)
  - ➡️ 초기에는 트리를 최대 크기로 만듦(full tree). 마지막 노드의 불순 척도가 최소가 되도록 분할한다. 복잡도(complexity)가 최대가 된다.
  - ➡️ 완전히 성장한 트리를 위쪽 방향으로 다듬어 가는 절차를 수행한다. -> Trimming
  - ➡️ cross validation(cv) 시험 오차가 최소가 되는 분할 수준으로 트리를 줄인다

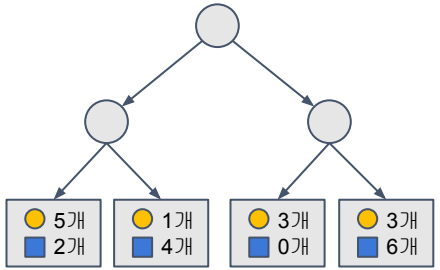
사후 가지치기(Post Pruning)

- 최대한 복잡하게 tree를 구성한 후 error가 작아지는 방향으로 사후 가지치기를 수행한다.
- Error 측정은 오분류율(e/D)에 penalty 항 (N/D)을 추가하고 leaf-node가 많아질수록 penalty를 크게 부여한다. penalty 항의 크기는 alpha로 조절한다.
- 아래 왼쪽 tree는 leaf-node 가 7개 총 error는 11/24, 오른쪽 tree는 leaf-node가 4개 총 error는 10/24
- alpha = 1.0 일 때 오른쪽 tree가 더 단순하며 (pruned) 총 error가 작다.



$$e_\alpha(T) = \frac{e + N \times \alpha}{D}$$

e: 잘못 분류된 개수
N: leaf-node 개수
D: data 개수



* 숫자가 작은 것이 잘 못 분류된 것임

$$e_\alpha(T) = \frac{4 + 7 \times 1.0}{24} = \frac{11}{24}$$

$$e_\alpha(T) = \frac{6 + 4 \times 1.0}{24} = \frac{10}{24}$$

의사 결정 나무(Decision Tree)

실습

Pruning

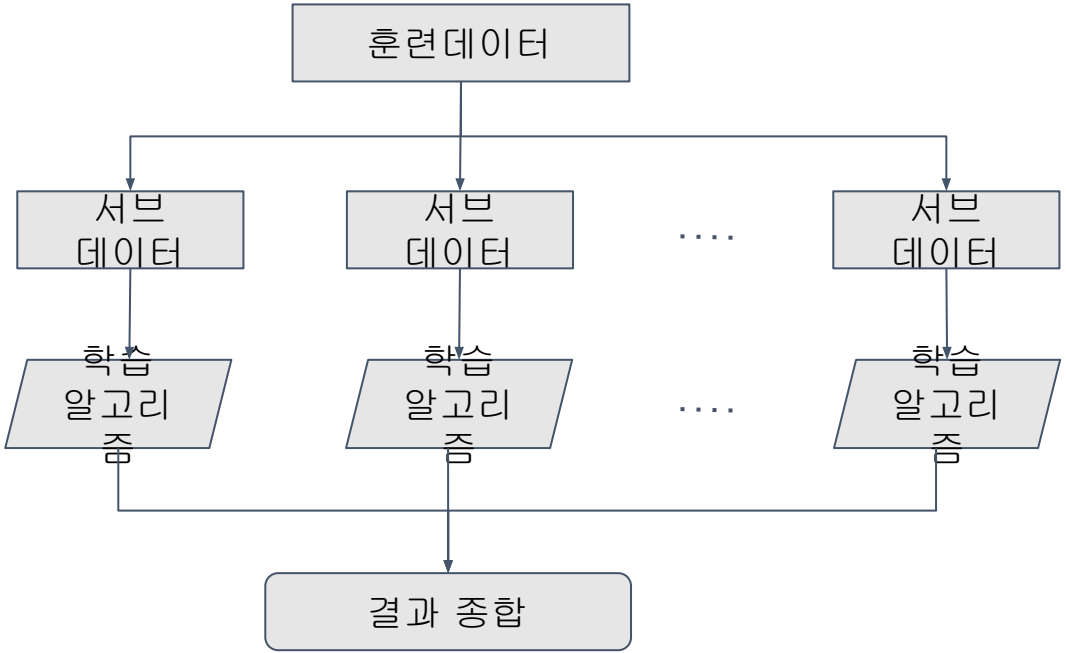
앙상블(Ensemble)

- 여러개의 분류기(classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법

  ➡️ 정확도 증가, 일반화 특성 증가

- 어려운 문제의 결론을 내기 위해 여러 명의 전문가로 위원회를 구성해 다양한 의견을 수렴하고 결정하듯이 앙상블 학습의 목표는 다양한 분류기의 예측 결과를 결합함으로써 단일 분류기보다 신뢰성이 높은 예측값을 얻기 위함
- 뛰어난 성능을 가진 모델들로만 구성하는 것보다 성능이 떨어지더라도 서로 다른 유형의 모델을 섞는 것이 오히려 전체 성능에 도움이 될 수 있음

앙상블(Ensemble)

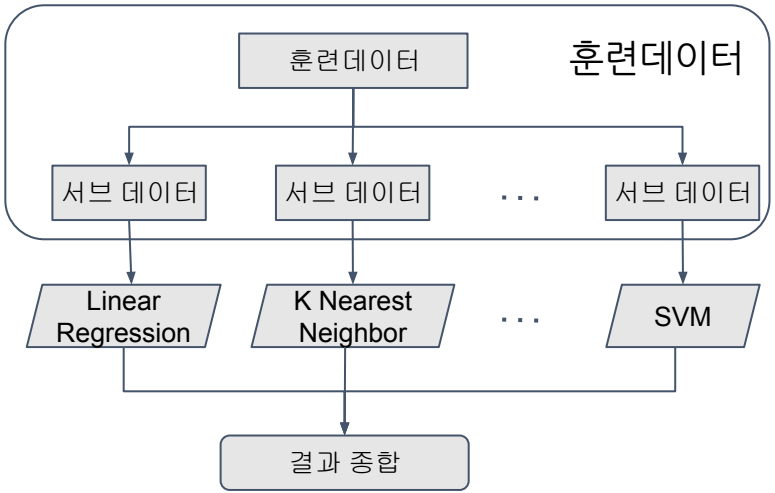


훈련데이터를 여러 개로 나눔

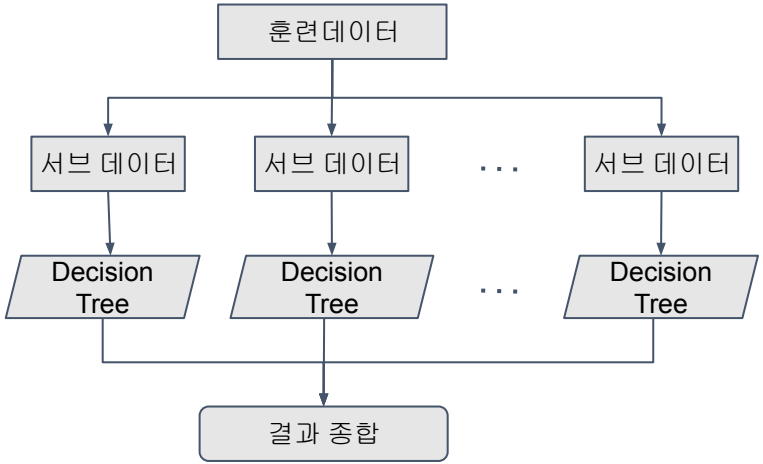
분류기로 각각 학습함

학습 결과를 종합함

앙상블(Ensemble)



Voting

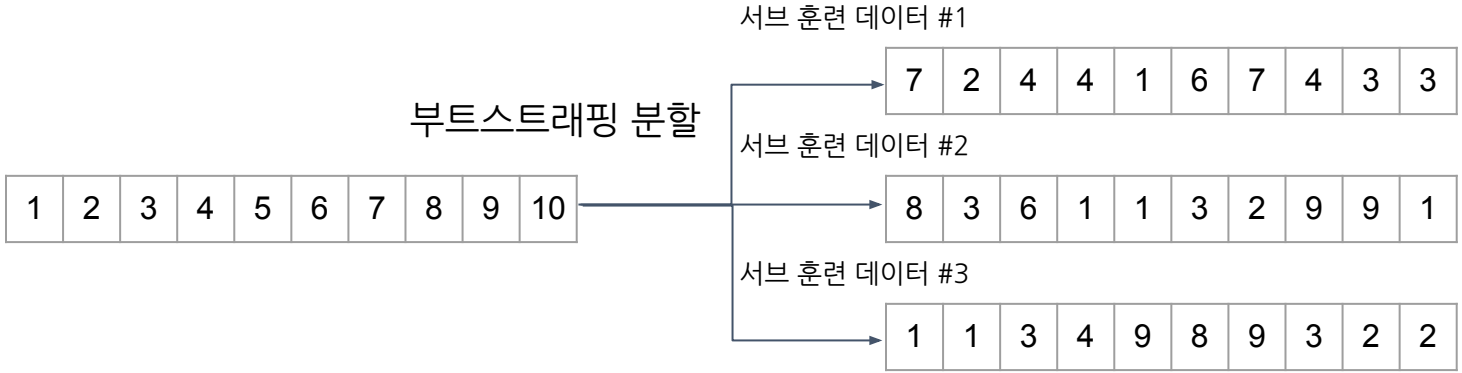


Bagging

voting은 일반적으로 서로 다른 알고리즘 분류기를 결합, bagging은 같은 알고리즘 분류기 결합

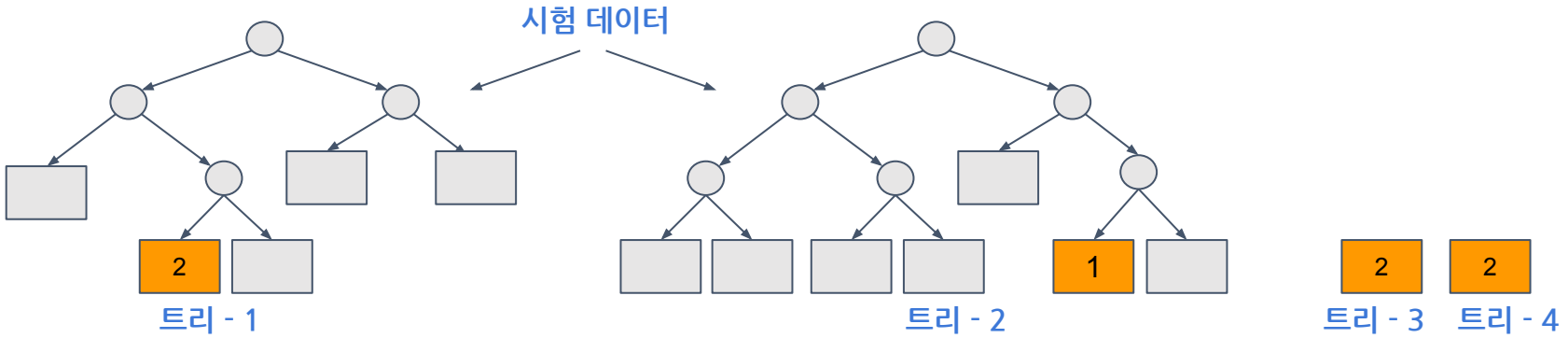
배깅(Bagging)

- 배깅(Bagging or **B**ootstrap **Agg**ression)
  - (균일한 확률분포로) 훈련 데이터를 반복적으로 샘플링하여 서브 훈련 데이터를 만듦(Bootstrap : 단순복원 임의 추출)
  - 각각의 서브 훈련 데이터는 원본 훈련 데이터와 같은 크기를 갖음 ➡️ 데이터 중복 허용
  - 결과의 분산 (변동)이 감소하고 과잉적합(overfitting)이 방지된다
  - 랜덤 포레스트 (Random Forest)



랜덤 포레스트(Random Forest)

- Decision Tree을 사용하는 앙상블 기법
- 다수의 Decision Tree의 예측을 종합하여 분류
- 배깅과 동일하게 훈련 데이터에서 n개의 서브 데이터를 샘플링하고(중복 허용, 균등 분포), 각 서브 데이터마다 Decision Tree를 구축
- 서브 데이터는 훈련 데이터로 부터 랜덤하게 추출되므로, 서브 트리들은 약간씩 다르게 구성
- 약간씩 다른 트리의 결론을 종합하므로 분산이 감소하고 일반화 특성이 향상



- 서브 데이터로 여러 개의 트리를 생성(서브 트리)
- 시험 데이터를 각 서브 트리에 적용하여 분류하고 다수의 트리가 분류하는 결과를 따름.
- 예) 트리-1 은2, 트리-2는1 이라고 분류하고, 트리-3,4 는2라고 분류하면 최종으로 2라고 분류(다수결)

의사 결정 나무(Decision Tree)

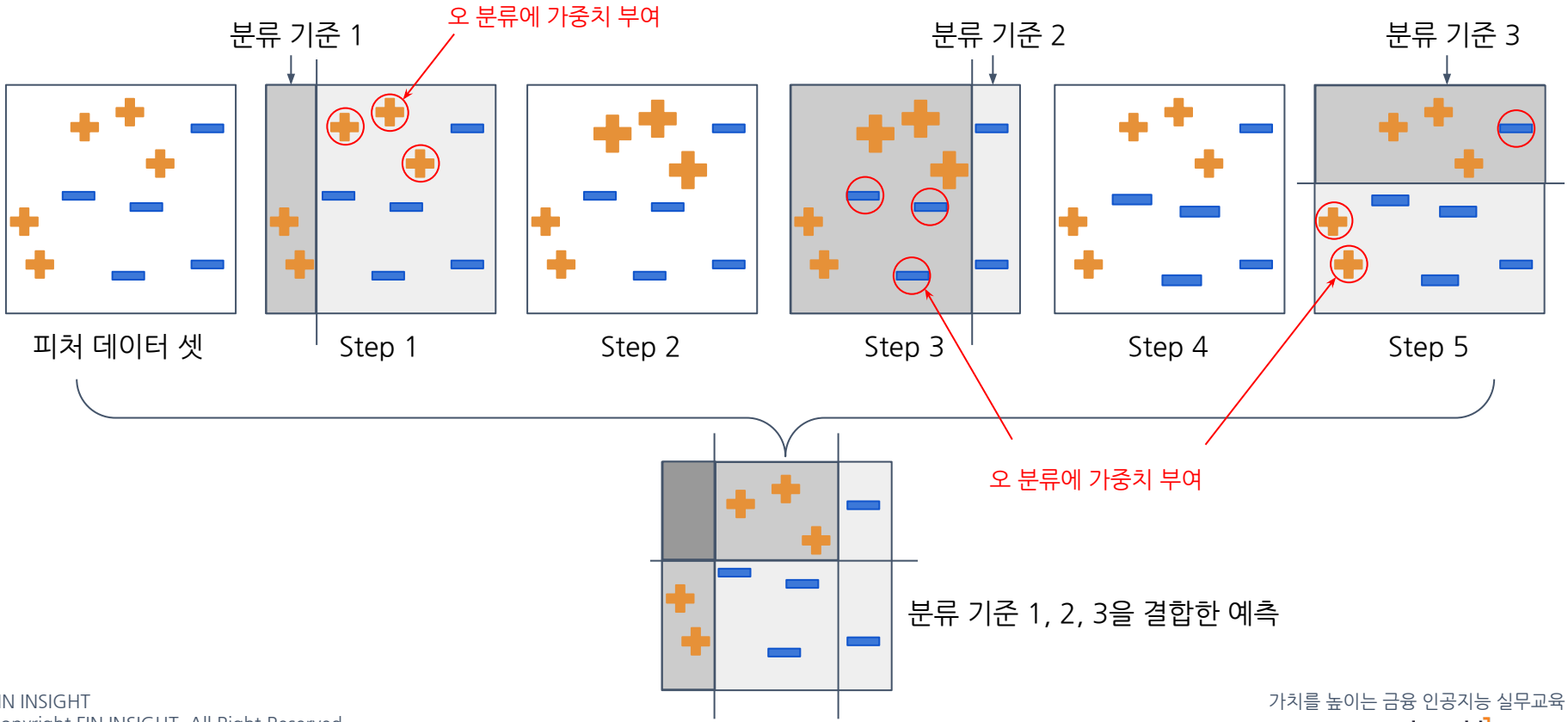
실습

random forest

부스팅(Boosting)

- 부스팅(Boosting)
  - 배깅과 마찬가지로 서브 훈련 데이터 샘플을 만듦
  - 처음 샘플링은 모든 데이터에 동일 가중치를 두어 샘플링함
  - 샘플링된 서브 훈련 데이터로 학습함
  - 분류가 잘못된 데이터의 가중치를 높이고 다시 샘플링함 ➡️ 잘못 분류된 패턴은 선택될 확률이 높아짐
  - 새로운 샘플링으로 다시 학습. 반복하여 점차 분류하기 어려운 패턴들을 많이 선택
  - 분류가 어려운 패턴에 더욱 집중하여 정확도를 높임
  - AdaBoost(Adaptive Boosting)
  - GBM (Gradient Boost Machine)
  - XGBoost (eXtreme Gradient Boost)

Ada Boost



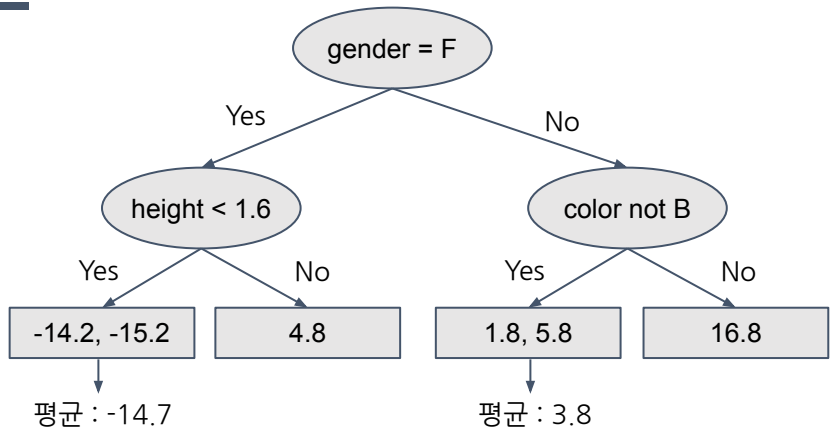
Gradient Boosting

- GBM(Gradient Boosting Machine)
  - Ada Boost와 유사하나 가중치 업데이트를 경사 하강법(Gradient Descent)을 이용
  - 회귀, 분류 모두 적용 가능
  - 레이블 데이터의 잔차를 학습하는 방식

feature (X)			target (Y)
height	color	gender	weight
1.6	B	M	88
1.6	G	F	76
1.5	B	F	56
1.8	R	M	73
1.5	G	M	77
1.4	B	F	57

Gradient Boosting

height	color	gender	weight	residual(0)	residual(1)
1.6	B	M	88	16.8	15.1
1.6	G	F	76	4.8	4.3
1.5	B	F	56	-15.2	-13.7
1.8	R	M	73	1.8	1.4
1.5	G	M	77	5.8	5.4
1.4	B	F	57	-14.2	-12.7



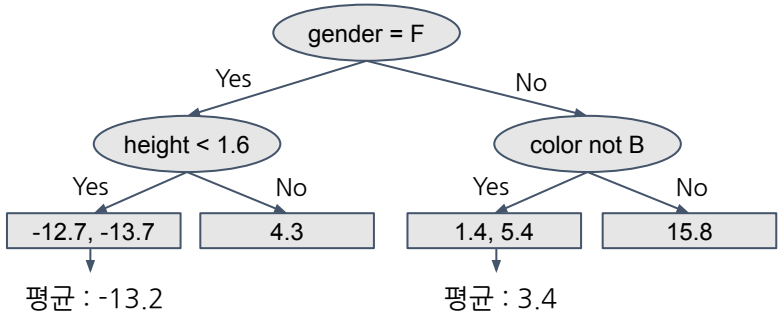
1) weight의 평균을 계산 : 평균 = 71.2
2) residual(0) 계산 : weight - 평균
   (88 - 71.2 = 16.8)
3) residual(0)에 대해 Decision Tree(1) 생성
4) Tree를 이용해서 새로운 residual(1)을 계산:
   weight - (평균 + 학습률*Tree의 leaf 평균)

88 - (71.2 + 0.1 * 16.8) = 15.1
76 - (71.2 + 0.1 * 4.8) = 4.3
56 - (71.2 + 0.1 * -14.7) = -13.7
73 - (71.2 + 0.1 * 3.8) = 15.1
77 - (71.2 + 0.1 * 3.8) = 5.4
57 - (71.2 + 0.1 * -14.7) = -12.7

Gradient Boosting

height	color	gender	weight	residual(0)	residual(1)	residual(2)
1.6	B	M	88	16.8	15.1	13.6
1.6	G	F	76	4.8	4.3	3.9
1.5	B	F	56	-15.2	-13.7	-12.4
1.8	R	M	73	1.8	1.4	1.1
1.5	G	M	77	5.8	5.4	5.1
1.4	B	F	57	-14.2	-12.7	-11.4

residual이 점점 작아지고 있다



Tree(1)과 tree(2)를 이용해서 새로운 residual(2)을 계산:

weight - (평균 + 학습률*Tree(1)의 leaf 평균 + 학습률*Tree(2)의 leaf 평균)

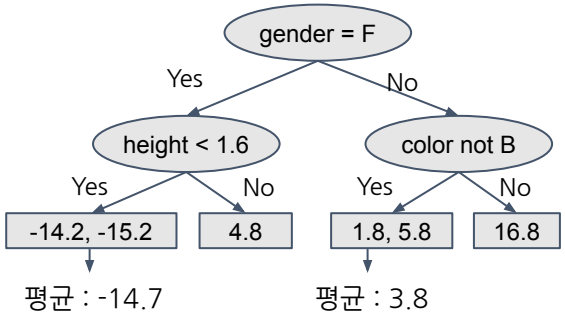
88 - (71.2 + 0.1 * 16.8 + 0.1 * 15.1) = 13.6
76 - (71.2 + 0.1 * 4.8 + 0.1 * 4.3) = 3.9
56 - (71.2 + 0.1 * -14.7 + 0.1 * -13.2) = -12.4
73 - (71.2 + 0.1 * 3.8 + 0.1 * 3.4) = 1.1
77 - (71.2 + 0.1 * 3.8 + 0.1 * 3.4) = 5.1
57 - (71.2 + 0.1 * -14.7 + 0.1 * -13.2) = -11.4

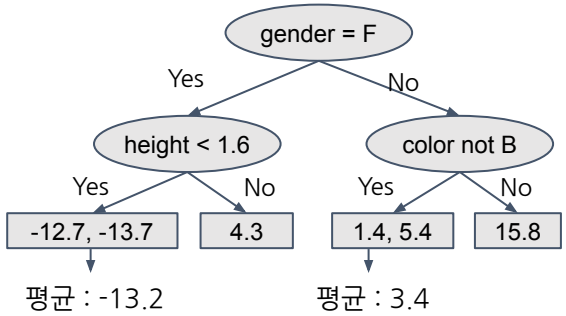
➡️ Residual이 더 이상 줄어들지 않을 때까지 이 과정을 반복한다

Gradient Boosting 추정과정

target 데이터

height	color	gender	weight
1.6	B	M	?





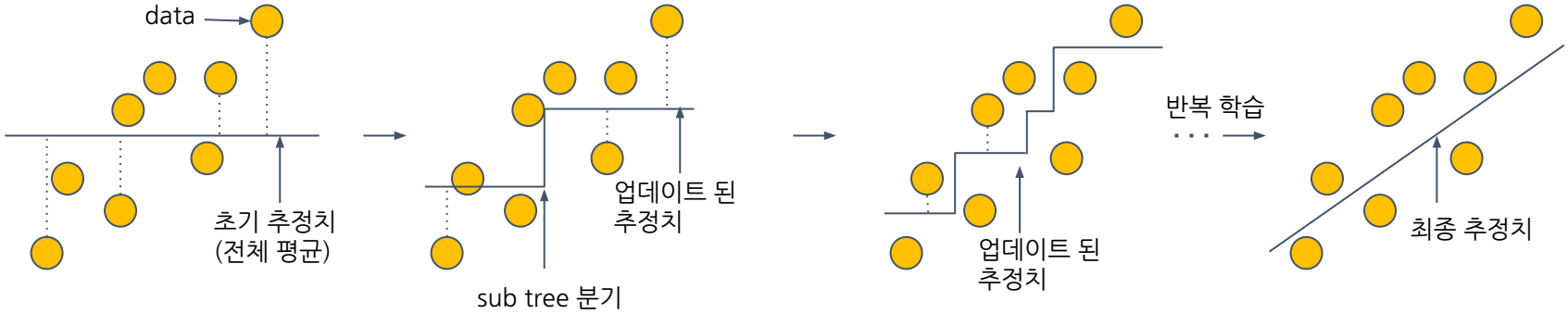
Tree (1)과 tree (2)를 이용해서 시험 데이터의 weight를 추정한다

71.2 + 0.1 * 4.8 + 0.1 * 4.3 = 72.1

tree (1) tree (2)

Gradient Boosting 학습 과정의 직관적 이해

- 모든 데이터의 평균으로 초기 추정치를 대충 설정한 다음, 잔차를 계산한다
- Decision tree를 생성해서 잔차를 줄이는 방향으로 학습하고,추정치를 업데이트하고, 새로운 잔차를 계산한다
- 새로운 decision tree를 생성해서 새로운 잔차를 학습해서 잔차를 계속 줄여 나가고, 추정치도 계속 업데이트 한다
- 잔차가 더 이상 줄어들지 않는 수준까지 decision tree를 생성하면서, 추정치를 업데이트 한다
- 최종 추정치는 아래 그림처럼 점차 데이터를 잘 설명할 수 있는 상태가 된다.



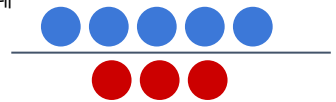
Gradient Boosting

- Gradient Boosting (for classification)
  - 전체 개념은 regression 과 동일하나 추정치를 위해 odds, log(odds), probability 개념을 사용하고 loss 함수로는 binary cross entropy 를 사용한다.
  - odds : 성공 확률이 실패 확률보다 얼마나 더 크냐?
    - class가 c1, c2인 이진 분류의 예
    - 어떤 x가 c1 으로 분류될 확률 : p

      ➡️ c2 로 분류될 확률 : 1 - p
    - p(c1|x) = y, p(c2|x) = 1-y

      ➡️ odds : y/1-y
  - logit : log(odds)

A라는 사람이 8번의 경기에서 5번을 이기고 3번을 졌을 때



odds ➡️ 5/3 = 1.7

의사 결정 나무(Decision Tree)

실습

GBM

XGBoost (Extreme Gradient Boosting)

- Gradient Boosting은 서브-트리의 leaf node 값의 평균 값을 이용하여 residual을 줄여 나갔으나, XGBoost는 loss 함수가 최소가 되는 최적값(optimal output value)을 찾아 residual을 줄여 나간다.
- XGBoost는 GBM보다 메모리 효율이 좋고 속도도 빠르다. 대용량 데이터 처리에 성능이 우수하다. (CPU 병렬 처리, GPU 지원)
- 규제(regularization)와 가지치기(pruning)을 통해 overfitting을 줄이고 일반화 특성을 좋게 만든다.

XGBoost (Extreme Gradient Boosting)

Drug dosage	Drug effect	residual (0)
13	-10	-10.5
21	7	6.5
28	8	7.5
32	-7	-7.5

$$L = \sum_{i=1}^{n} L(y_i, p_i) + \gamma T + \frac{1}{2}\lambda O^2$$

- T : 터미널 노드 개수
- O: output value
- λ regularization constant
- γ pruning 판단 상수

mse

minimize w.r.t output value (γ 생략)

$$output\ value(O) = \frac{\sum_{i=1}^{n} residual}{|residual| + \lambda}$$

$$similarity\ score = \frac{(\sum_{i=1}^{n} residual_i)^2}{|residual| + \lambda}$$

Loss 가 최소가되는 Tree 조건

$|residual|$: residual 개수

1) 초기 추정치를 임의로 설정하고 (ex : 0.5)
residual(0) 계산

residual(0) = -10 - 0.5 = -10.5
7 - 0.5 = 6.5
8 - 0.5 = 7.5
-7 - 0.5 = -7.5

2) residual(0) 에 대해 similarity와 output value를 계산한다

-10.5, 6.5, 7.5, -7.5 λ = 1 을 적용

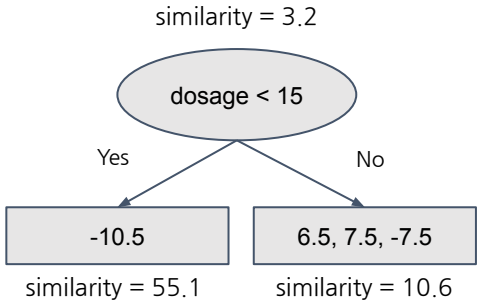
$$output = \frac{-10.5 + 6.5 + 7.5 - 7.5}{4 + 1} = -0.8$$

$$similarity = \frac{(-10.5 + 6.5 + 7.5 - 7.5)^2}{4 + 1} = 3.2$$

XGBoost (Extreme Gradient Boosting)

3) residual(0)에 대해 tree를 생성한다

Drug dosage	Drug effect	residual (0)
13	-10	-10.5
21	7	6.5
28	8	7.5
32	-7	-7.5



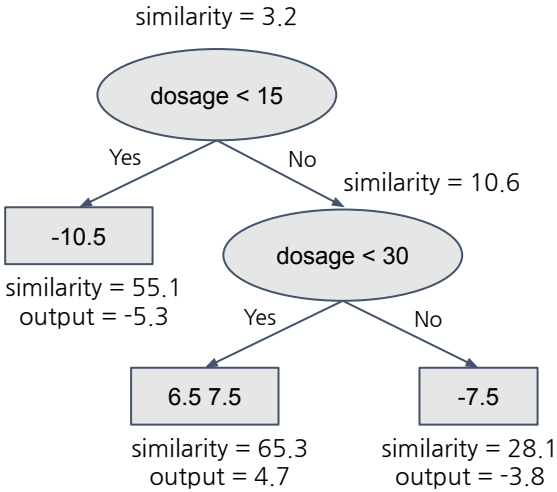
gain = 5.1 + 10.6 - 3.2 = 62.5
분할로 인해 62.5 만큼 이득이 생겼음

- 각 노드의 similarity score를 계산한다
- Gain이 가장 큰 분기를 찾아 tree를 생성한다
- 이 경우 dosage < 15 조건일 때의 gain이 가장 크다. Loss 도 작아진다
- leaf 노드에 residual이 유사한 것끼리 모여있으면 similarity score가 높다. 상쇄되는 부분이 작기 때문이다.

XGBoost (Extreme Gradient Boosting)

4) tree를 완성하고 leaf 노드의 output value를 계산한다

Drug dosage	Drug effect	residual (1)
13	-10	-10.5
21	7	6.5
28	8	7.5
32	-7	-7.5



gain = 65.3 + 28.1 - 10.6 = 82.8

- (6.5, 7.5, -7.5) 노드를 gain이 가장 큰 조건으로 분기한다.
- 이 경우는 dosage < 30 조건일 때의 gain이 가장 크다
- (6.5, 7.5) 노드는 분기해도 더 이상 이득이 없다 (gain <0)
- tree를 분기한 후 gain < γ 이라면 분기하지 않는다 (pruning)
- 예를 들어 γ = 100이라면 우측 서브 트리는 82.8 < 100 이므로 (6.5, 7.5, -7.5) 노드는 분리하지 않는다
- γ 는 분석자가 지정할 하이퍼 파라미터이다

XGBoost (Extreme Gradient Boosting)

5) 좌측 tree의 output value를 이용하여 residual(1)을 계산한다

- 첫 번째 데이터의 추정치 = 0.5 + 0.3 * (-5.3) = -1.09 ⬅️ 학습률 (ε) = 0.3 , 초기 추정치 = 0.5
  - ➡️ new residual = -10 - (-1.09) = -8.91
- 두 번째 데이터의 추정치 = 0.5 + 0.3 * (4.7) = 1.91
  - ➡️ new residual = 7 - (1.91) = 5.09
- 세 번째 데이터의 추정치 = 0.5 + 0.3 * (4.7) = 1.91
  - ➡️ new residual = 8 - (1.91) = 6.09
- 네 번째 데이터의 추정치 = 0.5 + 0.3 * (-3.8) = -0.64
  - ➡️ new residual = -7 - (-0.64) = -6.36

Drug dosage	Drug effect	residual (0)	residual (1)
13	-10	-10.5	-8.91
21	7	6.5	5.09
28	8	7.5	6.09
32	-7	-7.5	-6.36

← residual이 점점 작아지고 있다.
= 추정치가 점점 정확해지고 있다

6) residual(1)으로 또 tree를 생성하고 residual이 줄어들지 않을 때까지 반복한다

- 첫 번째 데이터의 추정치 = 초기 추정치 + ε * 이전 tree의 output + ε * 이전 tree의 output + ...

XGBoost

- 설치 conda install -c anaconda py-xgboost

실습

GBM