

[8장] 나무-기반의 방법

개요

8.1 의사결정 나무의 기초

8.1.1 회귀나무

야구선수 연봉 예제

특징 공간 층화로 예측 - 트리 만드는 방법

나무 가지치기 - 단일 트리 모델 성능 향상

8.1.2 분류나무

8.1.3 나무와 선형모형 비교

8.1.4 나무의 장점과 단점

장점

단점

8.2 배깅, 랜덤 포레스트, 부스팅 및 베이지 가법회귀나무

8.2.1 배깅

OOB(Out-of-bag) 오류 추정

변수 중요성 측도

8.2.2 랜덤 포레스트

8.2.3 부스팅

8.2.4 베이지 가법회귀나무(BART)

8.2.5 나무 앙상블 방법 요약

번외

개요



나무 기반 방법은 **예측 변수의 공간을 수 영역**으로 나눠서, 각 영역의 평균 반응값 또는 최빈 반응값을 사용하여 예측하는 방식

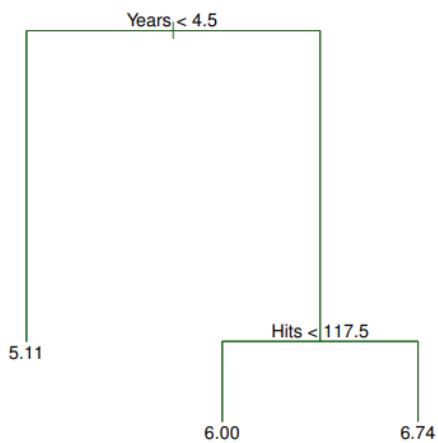
- **예측 정확도 면에** 있어선 6장과 7장에서 나왔던 고성능 지도 학습법보다는 **떨어짐**
- 나무 기반 기법의 특징
 - 종류: 배깅(Bagging), 랜덤 포레스트(random forest), 부스팅(boosting), 베이지 가법회귀나무(Bayesian additive regression trees)
 - 다중의 나무를 생성하여 결합하여 예측
 - 예측도를 극적으로 향상 ↔ 해석에 있어서 일부의 손실 발생

8.1 의사결정 나무의 기초

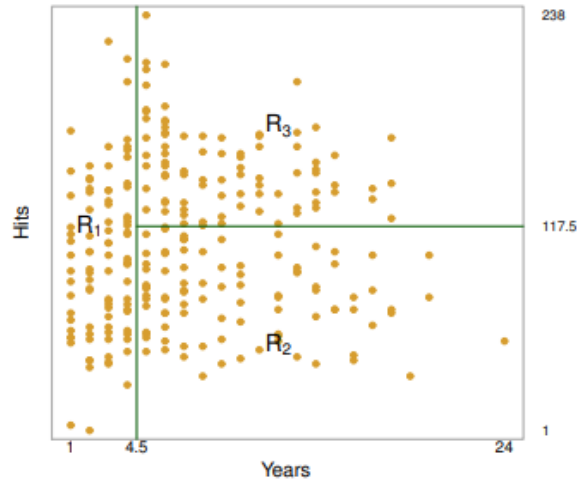
| 의사결정 나무는 회귀 문제와 분류 문제에 모두 적용 가능

8.1.1 회귀나무

야구선수 연봉 예제



[그림 1]



[그림 2]

- [그림 1]
 - 말단 마디(terminal node) : leaf라고도 하며, 각 조건의 가장 끝을 의미
 - 해당 예제에서는 평균 로그 연봉 예측 값이 해당 조건에서의 terminal node의 값이 됨
 - 내부 마디(internal node) : 예측 변수 공간이 분할되는 지점, 위에 쓰여진 조건
 - 마디에 해당하는 값은 왼쪽 가지에 해당하는 영역, 그 반대가 오른쪽 가지에 해당하는 영역
 - 가지(branch) : 각 마디들을 연결하는 선분
 - 해석
 - 가장 상위에 있는 internal node인 **years**가 영향력이 가장 큼
 - hits는 years가 4.5년 이상일 때만 의미 있음
- [그림2]
 - 그림 1을 2차 평면 상에 시각화

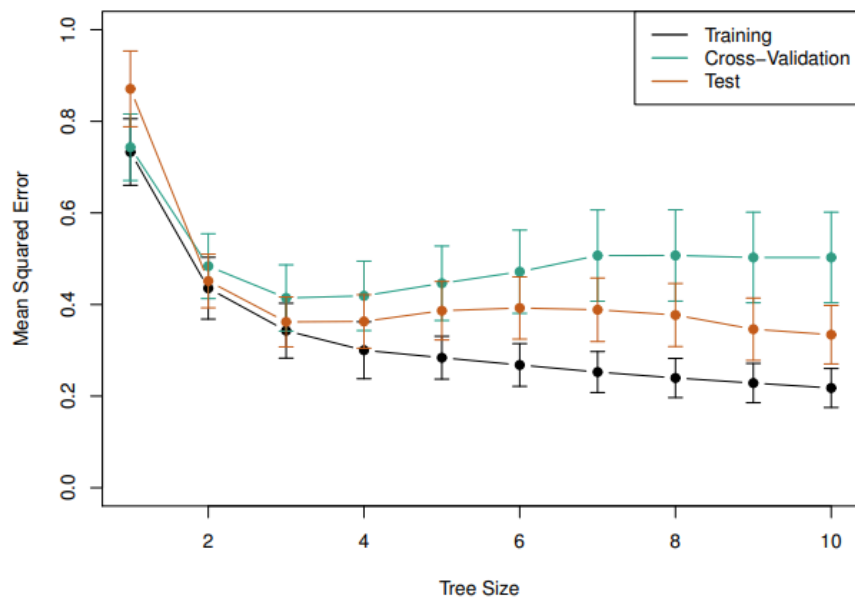
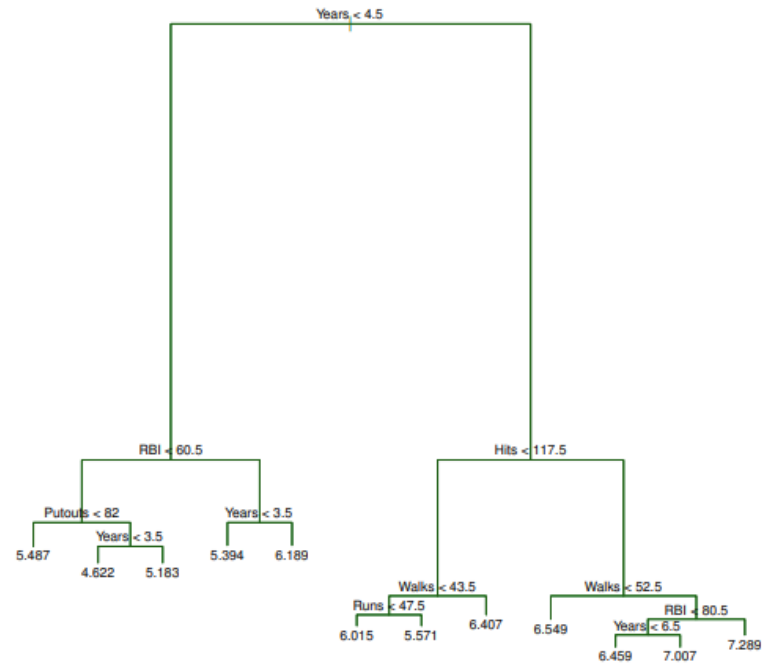
특징 공간 총화로 예측 - 트리 만드는 방법

- 회귀나무를 구축하는 절차
 - 예측변수 공간을 J개의 명확하고 겹치지 않는 영역으로 나눈다. ($R_1, R_2, R_3, \dots, R_J$)
 - [그림 2]에서는 J가 3인 영역으로 나눔
 - R_j 영역에 속하는 모든 관측을 동일한 값(예. R_j 영역의 평균 반응값)으로 예측
- R_j 영역을 만드는 방법은..?
 - 잔차 제곱합(RSS)을 최소화하는 방향으로 영역을 분할
 - 다만 J개의 영역으로 완벽하게 특징 공간을 “겹치지 않게” 분할하는 것은 잔차 제곱합을 최소화하는 식으로는 계산 상 불가능
- 재귀적 이진 분할(Recursive Binary Splitting)
 - 탑다운 방식으로 데이터를 나눠가며
 - 탐욕적(greedy)으로 각 단계에서 RSS를 가장 크게 줄이는 분할을 선택
 - 예측변수(X_j)와 cutpoint(s)를 적절히 선택해 (예. $\text{years} < 4.5$) 두 공간으로 나누며 RSS가 가장 낮아지게
 - 식으로 따지자면
 - $R_1(j, s) = X \mid X_j < s$ 와 $R_2(j, s) = X \mid X_j \geq s$ 의 공간으로 나눠
 - 잔차 제곱합인 $\sum_{i \in R_1(j, s)} (y_i - y_{R1})^2 + \sum_{i \in R_2(j, s)} (y_i - y_{R2})^2$ 가 가장 작아지도록 하는
 - j와 s를 찾음
 - 이 방식은 특정한 정지 조건이 있을 때까지 반복할 수 있다.
 - 예. 어떤 영역도 5개 이상의 관측이 포함되지 않는 경우
 - 위에서 말한 internal node가 정해지는 방식

나무 가지치기 - 단일 트리 모델 성능 향상

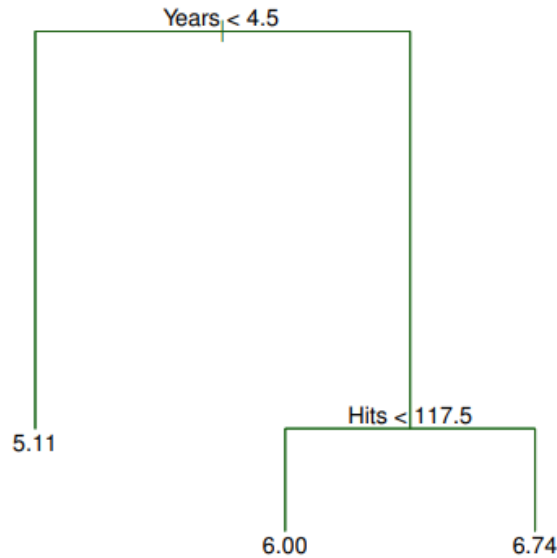
- 나무 모델은 overfitting이 일어나기 쉬워 성능 저하가 일어날 수 있음
 - 애초에 탐욕적으로 RSS가 작은 기준을 최우선을 정하는 상황에
 - 분할이 너무 많이 일어나는 경우 → 과적합이 일어날 수 있음
 - 해결1: 편향이 좀 있더라도 트리를 작게 만들면 예측 성능이 좋아질 수 있음(편향 / 분산 간 트레이드오프)

- 해결2(?): RSS 감소가 어떤 임계값을 초과할 때만 나무 구축
 - 쓸모 없어 보이는 분할이 개선점이 될 수 있을 정도로 중요할 수 있음
- 가지치기(pruning)
 - 매우매우 큰 트리를 만든다(RSS가 충분히 줄어들 때까지 계속 분할)
 - 가지치기를 통해 하위 트리(subtree)를 만든다.
 - 목표: 테스트 에러가 가장 낮은 하위 트리를 선택
 - 다만, 모든 하위 트리의 교차 검증 오류를 다 계산한다는 것은 불가능에 가까움
- 비용복잡도 가지치기(cost complexity pruning)
 - 음수가 아닌 조율모수(tuning parameter, α)를 도입해서 모델의 복잡도와 정확도 사이에서 균형을 조절
 - tuning parameter가 0에 가까워질수록 가장 큰 트리에 가까워짐
 - 이 땀 훈련 오차만 측정
 - tuning parameter가 커지면 커질수록 단순한 트리가 됨
 - terminal node가 많은 나무를 유지하는 비용이 생겨, 더 작은 subtree에서 최소화 되는 경향이 있다.
 - 가지가 예측 가능해지고, 점진적으로 나무에서 제거될 수 있음 → 부분 나무 전체의 시퀀스를 얻을 수 있음
 - α 는 검증 세트를 사용하거나 k-겹 교차검증으로 선택할 수 있음
 - 복잡도를 조절하기 위해 패널티 항을 사용한다는 점에서 라쏘와 비슷
- 예시



- 훈련 세트 132, 테스트 세트 131개로 관측 결과를 나눠 제일 큰 회귀 나무를 생성
 - 6-겹 교차 검증을 통해 MSE를 추정
 - 아래의 표에서,
 - CV가 가장 낮으면서(초록선)
 - Test 오차가 적절히 낮은(주황선)
 - 나무가 3마디일 때가 가장 적절한 나무가 되며, 3-마디 나무로 가지치기 했다고 할 수 있다.

- 이 결과는 맨 처음에 나온 나무 예제와 동일



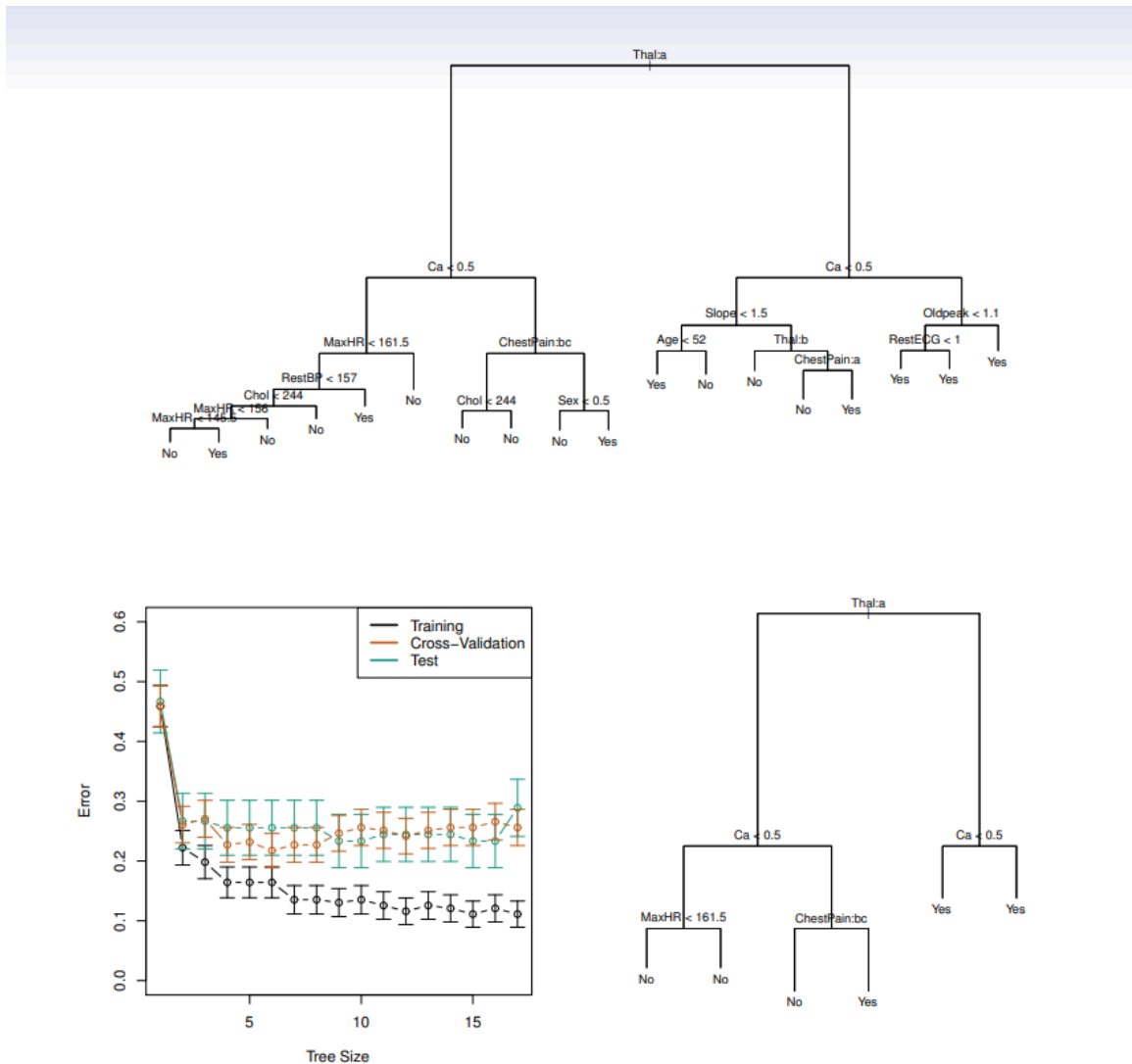
[그림 1]

8.1.2 분류나무

	회귀 트리	분류 트리
반응 형태	양적 반응(숫자형 응답)	질적 반응(범주형 응답)
말단 노드 값	훈련 샘플의 평균 값	가장 흔히 발생하는 클래스
추가 특징	-	노드 안에 있는 클래스의 비율도 중요
생성 방식	재귀적 이진 분할	재귀적 이진 분할
분할 결정 기준	RSS 최소화	분류 오류율 or 지니 지수 or 엔트로피

- **분류 오류율**: 말단 노드에서 가장 흔한 클래스가 아닌 비율(상대적으로 민감하지 않음)
- **지니 지수**: K 부류에 걸친 총 분산의 측도 → 작을수록 좋음
 - 지니 지수가 작다는 것은, 한 노드 내에 최대한 **하나의 클래스로 응집**될 수 있다는 것
 - 노드 내에 A, B 클래스가 있다고 했을 때 데이터가 A 분류로 들어가는 경우가 많을수록 해당 노드가 A라고 설명할 수 있기 때문(노드가 A로써 pure하다고 할 수 있다! 해당 노드는 A라고 예측하기 쉬워짐)
 - 지니 지수가 높다는 건 여러 클래스가 속해 있어서 해당 노드가 대표적으로 무슨 값이다 라고 설명하기 모호해짐
- **엔트로피**: 불확실성을 수치로 나타냄

- “이 노드 안에 클래스가 얼마나 섞여 있는가”
- 덜 섞여 있을수록 엔트로피가 작고, 많이 섞여 있을수록 엔트로피가 커짐
 - 이 또한 지니 지수랑 매우 흡사한 의미를 갖게 됨
 - 차이가 있다면 엔트로피는 ‘예측 값이 클래스에 개수에 따라 얼마나 헛갈리냐’에 초점
 - 지니 지수는 ‘예측이 해당 노드 내에서 틀릴 확률’에 초점을 맞춤
- 가지치기 할 때 셋 다 쓸 수 있으나,
 - 예측 정확도를 목표로 한다면 분류 오류율을 선택하는 게 유리
 - 특정 분할의 품질 평가에는 지니 지수 or 엔트로피를 사용하는 게 유리
- 예시: 심장병 유무 예측



- 예측 변수 13개를 이용하여 가장 큰 트리(상단)를 만든 후, 교차 검증(하단 좌측)을 통해 6개의 노드를 가진 나무(하단 우측)으로 가지치기 함
- 이 때, 예측 값이 같은 leaf가 생성된 사례가 존재
 - 마디 순도(node purity)를 증가 시키면서 수행됨
 - 똑같은 Yes여도 한 쪽은 9/9 순도일 수 있고, 다른 한 쪽은 7/11 순도일 수도 있다.
 - 지니 지수나 엔트로피는 노드가 '더 순수한지'를 더 잘 감지함

8.1.3 나무와 선형모형 비교

- 선형 회귀 모델

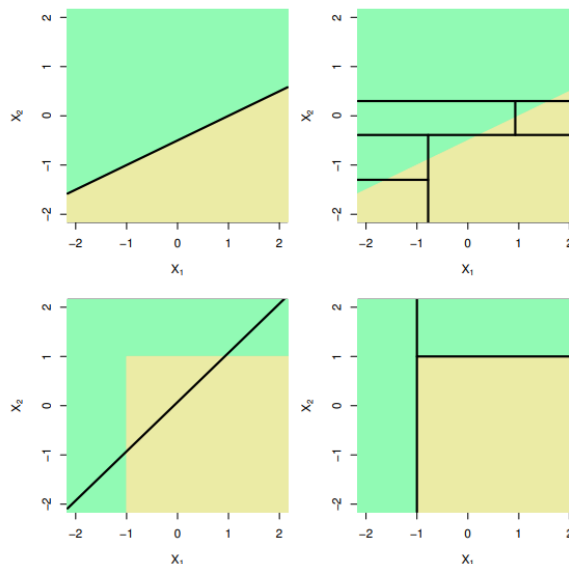
$$f(X) = \beta_0 + \sum_{j=1}^p X_j \beta_j$$

- 회귀나무 모델

$$f(X) = \sum_{m=1}^M c_m \cdot (X \in R_m)$$

- R_m 은 공간 분할 영역, c_m 은 각 영역의 예측값(평균값)

- 모델 선택 기준



- 데이터가 선형인 경우: **선형 회귀**를 이용하는 것이 성능이 훨씬 좋음
- 데이터가 비선형이며 복잡한 경우: **회귀 트리**가 훨씬 유연하며 성능이 좋음

▼ GPT한테 물어본 선형회귀 vs 회귀나무에 대한 좀 더 디테일한 표 정리

항목	선형 모델 (예: 선형 회귀)	트리 기반 모델 (예: 회귀/분류 트리)
모델 형태	연속적인 선형식 $f(X) = \beta_0 + \sum X_j \beta_j$	영역별 상수 함수 $f(X) = \sum c_m \cdot 1(X \in R_m)$
관계 가정	변수와 응답 사이에 선형적 관계를 가정	변수와 응답 사이의 비선형적, 복잡한 구조도 허용
해석 용이성	계수 β 를 통해 변수의 영향 해석이 직관적 임	트리 구조로 의사결정 경로가 명확하므로 시각적 해석이 쉬움
예측 정확도 (단순 구조)	선형 구조가 잘 맞을 경우 우수함	단순 선형 구조일 경우 과소적합 위험 있음
예측 정확도 (복잡 구조)	고차항이나 상호작용 없이는 잘 대응 못함	복잡하거나 비선형적인 관계에 대해 유리함
과적합 경향	적절한 정규화 없으면 과적합 가능성 있음	과도하게 깊은 트리 는 과적합 우려 → 가지치기(pruning) 필요
모델 선택 기준	MSE, AIC, BIC, cross-validation 등	주로 cross-validation , pruning 시엔 복잡도-비용 균형 고려
성능 평가 방법	교차 검증, 테스트셋 오차 등	교차 검증, 테스트셋 오차 등 (같음)
범주형 변수 처리	일반적으로 더 복잡한 인코딩 필요 (e.g. one-hot)	직접 분할 가능 (e.g. "Sex: female vs. male")
적용 예시	선형적으로 예측 가능한 문제 (e.g. 집값, 키와 몸무게)	구조적 경계가 있는 문제 (e.g. 질병 유무, 고객 분류 등)

8.1.4 나무의 장점과 단점

장점

- 설명이 매우 쉽다
 - 트리는 사람들에게 설명하기 쉬우며, 경우에 따라 **선형 회귀보다 더 쉽게 설명**될 수 있다.
- 사람의 사고 방식과 유사하다
 - 어떤 사람들은 트리 방식이 인간의 의사결정 구조와 더 가깝다고 느낀다.
- 시각화가 쉽고 해석이 간단하다
 - 트리는 **그래픽으로 표현이 가능**하고, 특히 작을 경우 **비전문가도 이해하기 쉬움**
- 범주형 변수도 자연스럽게 처리 가능하다
 - 더미 변수(dummy variable)를 만들 필요 없이 **그대로 분할 가능**함

단점

- 예측 정확도가 다른 방법들보다 낮은 편

- 일반적으로 트리는 **선형 회귀나 로지스틱 회귀** 등 기존 방법들보다 **예측력이 떨어질 수 있음**
- **비견고성(non-robust)**
 - 데이터가 약간만 바뀌어도 트리의 구조가 크게 바뀔 수 있음 → 즉, 과적합이나 민감도 문제 발생 가능

8.2 배깅, 랜덤 포레스트, 부스팅 및 베이즈 가법회귀나무

단순한 의사결정나무만으로는 모델의 성능이 다소 떨어질 수 있음
이 때, 배깅, 랜덤 포레스트, 부스팅, 베이즈 가법회귀나무 기법 등의 앙상블 기법을 이용하면 예측 성능이 크게 개선 될 수 있다.

8.2.1 배깅

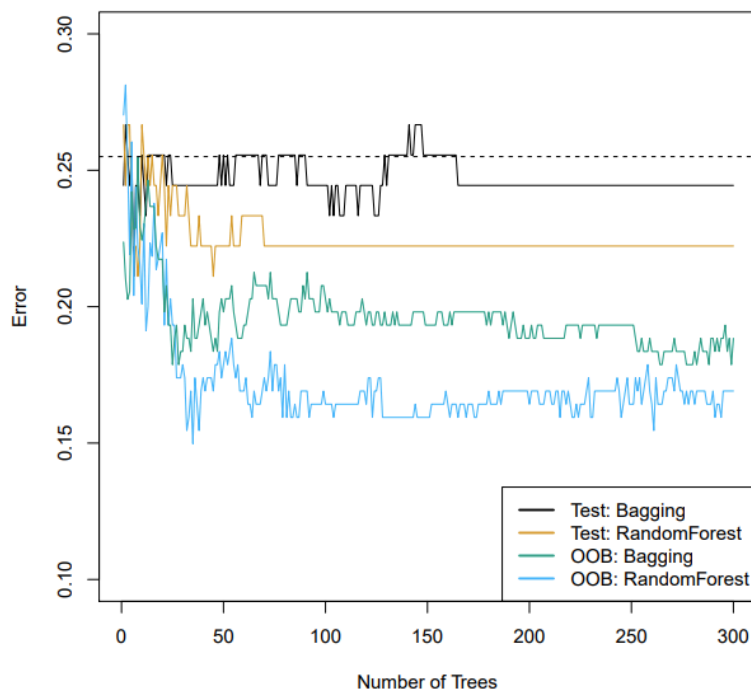
- 5장에서 나왔던 부트스트랩을 머신러닝 성능 향상에서 사용할 때, 배깅이라고 함
 - **부트스트랩** : 어떤 통계량(ex. 표준편차)등을 직접 계산하기 어렵거나 불가능할 때 사용
- 기본적으로 의사결정나무는 분산이 크다
 - 훈련 데이터를 무작위로 나누고 각각 트리를 적용했을 땐 전혀 다른 트리가 만들어 질 수 있다는 의미
 - 여러 모델의 평균을 구하여 분산을 줄임 → 배깅의 기본 아이디어
 - 관측 값이 Z_1, \dots, Z_n 일 때, 평균의 분산은 $Var(\bar{Z}) = \sigma^2/n$ 으로 평균을 내면 분산이 줄어든다는 것이 직관적으로 보임
- 머신 러닝에 적용한다면?

$$\hat{f}_{avg}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x)$$

- \hat{f}_b 는 서로 다른 학습 세트로 훈련한 예측 함수들이며, B로 나누면서(=평균으로 만들어) 저분산 통계 학습 모형으로 만들
- 현실적으로는 여러 훈련 세트를 사용하는 건 불가능하므로, 단일 훈련 세트를 이용해서 부트스트랩 기법으로 훈련 데이터세트를 이용함

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(x)$$

- 회귀나무와 배깅
 - 회귀나무에 배깅을 쓰려면 B개의 회귀나무를 만들어 예측 평균을 내면 됨 → 가지치기는 안되어 있음
 - 트리는 분산이 크고 편향이 낮은 모델
 - 예측 평균을 낸다는 것 → 분산도 같이 낮아진다는 것 → 정확도 향상
- 분류에서는?
 - 정량적 Y대신 범주형 Y로 치환해서 사용
 - 각 트리의 예측 클래스를 모아서 **다수결**로 최종 예측
- 테스트 에러 관련 그래프 예시



- 트리 개수가 많아지면서 오차율은 줄어듦
- 어느정도 되었을 때(약 100개의 트리일 때 쯤부터) 오차율이 안정화 되므로, 최소 100개 정도의 트리를 만들어 평균 분산값을 내면 충분히 좋은 모델이 됨

OOB(Out-of-bag) 오류 추정

- 배깅에서 테스트 오차 추정하는 방법
- 원리

- 배경: 부트스트랩 샘플링으로 각 트리를 학습
 - 훈련 데이터 중 일부만 뽑아서 트리를 학습 시킨다는 의미
- 통계적으로, 각 트리는 평균적으로 전체 데이터의 2/3 정도만 사용
 - 나머지 1/3의 데이터는 트리 학습에 사용하지 않음 → 이 때의 데이터가 **OOB(out of bag) 샘플**이라 함
- 각 관측값 i 에 대해 이 샘플이 포함되지 않았던 트리가 있음
 - 해당 트리에서의 OOB 샘플이므로 이 트리들에서만 예측 수행(통계적으로 약 B/3 개의 트리)
 - 회귀 → 예측값들의 평균으로 OOB 예측, 분류 → 예측값들의 다수결로 OOB 예측
 - 위에서 얻은 OOB 예측값과 실제 값을 비교
 - 전체 관측값에 대해
 - 회귀인 경우는 MSE, 분류인 경우에는 분류 오류율을 계산
- OOB 에러는 외부 테스트 데이터 없이도 테스트 에러를 추정하기가 매우 쉬움
 - 부트스트랩 기법의 특징을 통해 가능했던 것!!
- 예시(상단 그래프)
 - OOB 에러는 트리 개수가 충분히 클 때 LOO 교차 검증 결과와 거의 같아짐
 - 그렇기 때문에 데이터가 크고 cross-validation이 너무 느리다면 OOB를 쓰는 게 매우 빠르다 → 대규모 데이터 셋에서 유용!

변수 중요성 측도

- 배경은 많은 트리를 조합하기 해서 예측이 정확하지만, 해석이 어려움
 - 어떤 변수가 중요한지 직관적으로 파악하기 힘들
 - 그럼에도 **변수별 중요도 요약**이 가능함
- 측정 방법
 - 회귀 트리
 - 각 트리에서 특정 변수로 나눈 분할이 RSS를 얼마나 줄였는지 기록
 - B개의 트리에서 얻어낸, 각 특정 변수마다 RSS 감소 기록값의 평균을 구함
 - RSS 감소량이 클수록 중요한 변수
 - 트리에서 변수를 정하는 방법이 RSS 최소화 하는 건데, 최소화 하는 양 자체가 큰 건 주요 변수라는 얘기
 - 분류 트리

- 각 트리에서 변수별 분할이 지니 지수를 얼마나 줄였는지 기록
- B개의 트리에서 얻어낸, 각 특정 변수마다의 지니 지수 감소량을 찾아 평균을 구함
- 가장 감소량이 큰 변수일 수록 중요한 변수
- 개인적으로 궁금해진 것!! → 엔트로피나 분류 오류율을 써도 되는가?
 - 분류 오류율은 덜 민감해서 이론상 가능하지만 변수 중요성 측정에는 쓰지 않음
 - 엔트로피는 사용할 수 있지만 다음의 기준에서 선택해서 쓸 수 있음

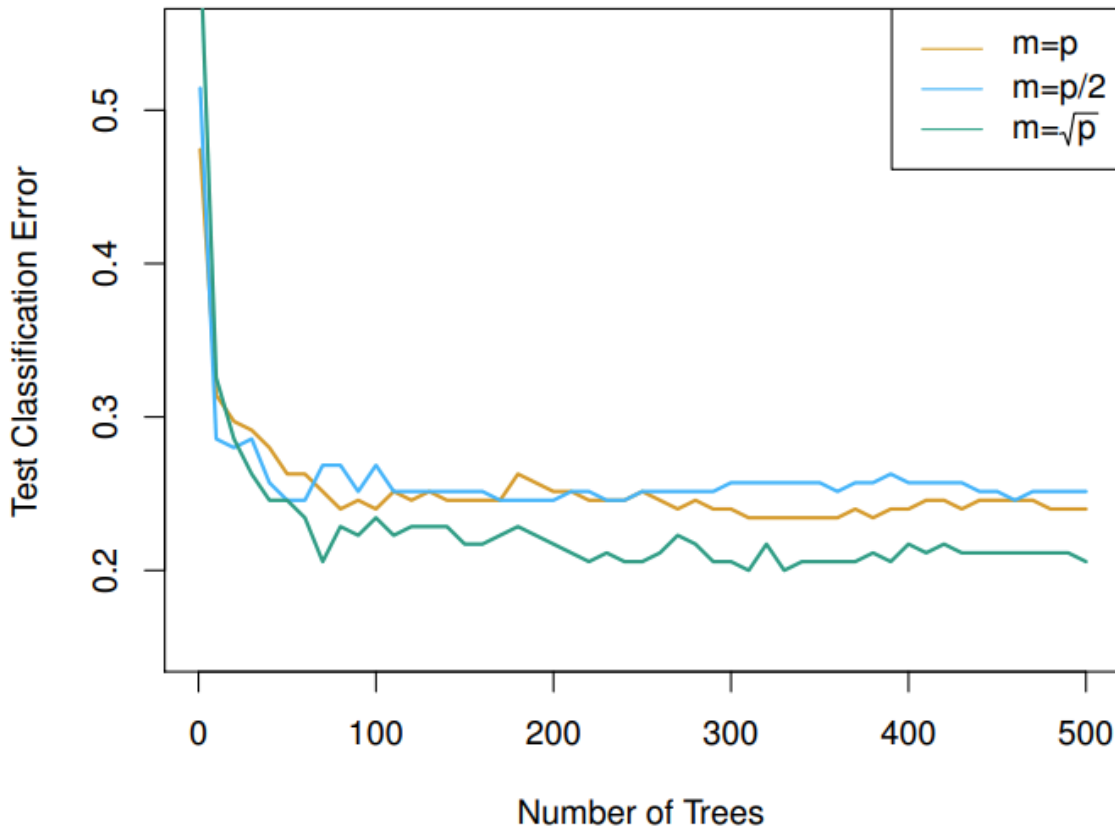
상황	추천 기준
계산 비용 신경 쓴다	✅ 지니 지수
해석을 명확히 하고 싶다 (정보 이득 기반 설명)	✅ 엔트로피
다수 클래스 vs 소수 클래스 구분을 예민하게 하고 싶다	엔트로피가 약간 유리할 수 있음
실험적으로 성능 비교 중이다	둘 다 돌려보고 확인

- 대신 성능 상으로는 차이가 미미하고 속도는 지니 지수를 사용하는 게 더 빠르므로 지니 지수 사용을 권장!

8.2.2 랜덤 포레스트

- 랜덤 포레스트는 배깅의 확장 버전
 - 기본적으로 부트스트랩 샘플링된 여러 데이터셋에 대해 트리를 만든 후 평균을 내는 방식이라는 점은 배깅과 동일
- 핵심 차이점: 무작위 변수 선택
 - 트리를 만들 때, 매 분할마다 전체 변수 p 중 무작위로 고른 m 개의 변수만 후보로 사용
 - 일반적으로 $m \approx \sqrt{p}$ 로 구해 고른다
 - 각 트리의 구조가 더 다양해짐 → 상호 상관이 줄어들음
- 유용한 이유
 - 변수 하나가 너무 강력하다면, 모든 트리가 첫 분류에 그 변수를 기준으로 사용해서 트리 구조들이 비슷해짐
 - 배깅을 해도 예측들 간의 상관이 높아져, 분산 감소 효과가 적어짐
 - 랜덤하게 변수를 뽑기 때문에, 트리들이 덜 유사해져 예측 분산이 많이 줄어들음
- m 의 효과
 - $m = p$ 인 경우: 랜덤 포레스트가 곧 배깅이 됨

- $m = \sqrt{p}$ 인 경우: 일반적인 랜덤 포레스트 설정
- m 이 작을수록 변수 다양성이 커지고 상관성은 줄어들지만, 너무 작으면 약한 분할만 생겨서 성능이 다소 떨어질 수 있음



8.2.3 부스팅

- 앞선 배깅이나 랜덤 포레스트와 달리 순차적으로 학습 시킴
 - 이전 트리들의 잘 학습 안된 부분을 다음 트리가 보완해줌
 - 즉, 앞선 결과를 토대로 다음 결과가 정해질 수 있음(좋은 쪽으로)
 - 부트스트랩 사용x
- 부스팅의 주요 하이퍼 파라미터
 - **B (트리 개수)**
 - 너무 크면 과적합 가능 → 교차 검증으로 조정
 - **λ (learning rate / shrinkage rate)**
 - 학습 속도 조절
 - 작을수록 느리게 학습되며 일반화 잘 됨

- 보통 0.01 또는 0.001
- λ 가 작으면 더 많은 트리(B) 필요해짐
- d (트리의 분할 수 = interaction depth)
 - 트리 한 개가 얼마나 복잡한지를 나타내는 척도
 - d = 1이면 stump (단일 분할) → 가산 모델과 유사해짐
 - 일반적으로 낮게 설정하는 것이 좋음
- 부스팅의 기본 아이디어 (회귀 기준) : 분류나무 부스팅도 비슷하지만 디테일이 다름(복잡함)
 1. 처음엔 단순한 모델로 시작함.
 2. 현재 모델의 잔차를 계산함: 실제 값 - 예측 값
 3. 이 잔차에 대해 새로운 결정 트리를 학습시킴
 4. 학습된 트리를 기존 모델에 조금씩(λ 만큼) 추가함
 5. 잔차를 업데이트
 6. 1~4의 과정을 B번 반복함
 - a. 각 트리는 작고 단순하게 만들고, 전체적으로 조화롭게 오류를 줄임

▼ 부스팅 vs. 배깅

항목	부스팅(Boosting)	배깅(Bagging)
트리 생성 방식	순차적 (앞선 결과 반영)	병렬 (독립적으로)
데이터 샘플링	원본 데이터, 단 잔차로 학습	부트스트랩 사용
목적	잔차(error)를 점진적으로 줄이기	분산 감소
각 트리의 역할	이전 트리의 오류 보정에 집중	동등하게 기여

8.2.4 베이즈 가법회귀나무(BART)

- 기존 앙상블과의 차이

방식	설명
배깅/랜덤 포레스트	트리를 각각 독립적으로 만들고 예측값을 평균함
부스팅	트리를 순차적으로 만들되, 이전 트리의 오류(잔차)를 보정함
BART	트리를 순차적으로 학습하되, 매 단계에서 기존 트리를 살짝씩 수정 (perturbation)해가며 개선해나감 + 베이지안 관점 도입

- 작동 방식

- **K개의 트리**를 준비 (보통 작고 단순한 트리)
- 각 반복(iteration)에서:
 - 트리 하나씩 차례대로 업데이트함
 - 나머지 트리의 예측값을 제외하고 해당 트리에서만 보정해야 할 잔차(residual)를 계산
 - 기존 트리를 기반으로 작은 변화(perturbation)를 시도:
 - 트리 구조 변경 (가지 추가/제거)
 - 단말 노드 값 변경
 - 이 변화를 통해 **잔차에 더 잘 맞는 방향으로 조정**
- **B번 반복**하고, 초반 L번은 **burn-in**이라 버림
→ 최종 예측값은 이후의 평균

$$f(x) = \frac{1}{B-L} \sum_{b=L+1}^B f^b(x)$$

- **신뢰 구간**이나 **percentile**도 함께 계산 가능 → 불확실성 추정 가능
- perturbation 방식의 장점
 - 매번 새 트리를 훈련하는 게 아니라 **기존 트리를 조금씩 바꿔가며 적합**
 - 데이터를 “**심하게 적합하지 않도록 하여(overfitting 억제)**” 안정적임
 - 개별 트리는 작고, 전체 모델은 점진적으로 정확도 상승
- 베이지안 해석
 - 매번 트리를 조금씩 수정하는 과정은, 사실상 **베이지안 사후 분포로부터 트리를 샘플링** 하는 과정임
 - BART 알고리즘 전체는 **MCMC (Markov Chain Monte Carlo)** 알고리즘의 일종으로 해석 가능

8.2.5 나무 앙상블 방법 요약

방법	특징 요약
Bagging	<ul style="list-style-type: none"> - 각 트리를 독립적으로, 부트스트랩 샘플에서 학습 - 트리들이 서로 비슷해져서 모델 공간 탐색이 제한적일 수 있음 - 과적합 억제에 효과적이거나, 다양성 부족으로 성능 한계

Random Forests	<ul style="list-style-type: none"> - Bagging과 거의 같지만, 각 분할마다 무작위로 일부 변수만 사용 - 트리 간 **상관성(decorrelation)**을 줄여서 더 다양한 트리 생성 → Bagging보다 더 넓은 모델 공간 탐색 가능
Boosting	<ul style="list-style-type: none"> - 원본 데이터만 사용, 부트스트랩 없음 - 트리를 순차적으로 학습, 잔차(residual)를 점진적으로 보정 - 학습률(λ)을 작게 설정해 천천히 학습함 (slow learning) → 과적합을 조심스럽게 피하면서 정교한 모델 구성
BART	<ul style="list-style-type: none"> - Boosting과 유사하게 순차적 학습, 원본 데이터만 사용 - 단, 이전 트리를 “조금씩 수정” (perturbation)하는 방식으로 개선 - 로컬 최적(local minima)에 갇히지 않도록 모델 공간을 더 잘 탐색함 - 베이지안 방식과 연결됨 (사후 분포에서 트리 샘플링처럼 작동)

▼ 추가 표(by. GPT)

비교 기준	Bagging	Random Forest	Boosting	BART
트리 학습 방식	병렬 (독립적으로 생성)	병렬 (독립적으로 생성) + 변수 무작위 선택	순차적 (앞선 트리 잔차에 맞춰 학습)	순차적 (기존 트리를 작게 수정하며 학습)
데이터 샘플링	부트스트랩 샘플 사용	부트스트랩 샘플 사용 + 변수 하위 집합 사용	부트스트랩 사용 안함 , 원본 데이터 반복 사용	부트스트랩 사용 안함 , 원본 데이터 반복 사용
트리 간 의존성	없음 (완전 독립)	없음 (변수 무작위화로 약한 다양성 부여)	있음 (이전 트리 기반으로 다음 트리 구성)	있음 (이전 트리를 수정하며 점진적 개선)
트리 다양성 유도 방식	데이터 샘플링만	데이터 샘플링 + 변수 무작위 선택	잔차 기반 학습 (error-corrective)	트리 구조 및 값의 무작위 수정 (perturbation) 기반 다양화
모델 업데이트 단위	전체 트리 예측값 평균	전체 트리 예측값 평균	각 트리 예측값에 가중치 λ 곱해 누적	매 반복마다 트리 수정 후 전체 예측 누적
과적합 위험	낮음 (분산 감소 효과)	낮음 (분산 감소 + 다양성)	있음 (B , λ 조절 필요)	낮음 (작은 트리, 느린 학습, 베이지안 구조로 제어)

비교 기준	Bagging	Random Forest	Boosting	BART
트리 크기	깊은 트리 (prune 안함)	깊은 트리 (prune 안함)	보통 작은 트리 (depth 1~2, stump 추천)	보통 아주 작은 트리 (크기 제한 필수)
성능 향상 원리	분산 감소	분산 감소 + 트리 상관성 제거	편향 감소 (잔차 보정), 정교한 학습	편향 감소 + 베이저안적 모델 공간 탐색
베이저안 해석 가능성 여부	✗	✗	✗	✓ (MCMC와 사후 분포 샘플링 관점)
대표 하이퍼파라미터	B (트리 수)	B, m (분할 변수 개수)	B (트리 수), λ (학습률), d (트리 깊이)	B (반복 수), K (트리 수), L (burn-in 길이)

번외

▼ 솔직히 부스팅이랑 BART는 뭔 소리인지 이해가 안돼서 GPT한테 다른 방식으로 알려 달라고 했습니다...



아 너무 어렵다..ㅠ

혹시 베이저안의 사후 분포에 대해서 진짜 겁나 쉽게 설명해줄 수 있어?
비유를 할 수 있으면 비유를 해줘도 되고..



비유: 라면 취향 찾기 실험

너랑 친구가 있어.

너는 "내가 진짜 좋아하는 라면은 어떤 맛일까?" 궁금해서 실험을 하기로 해.

1. 🎱 사전 확률(prior): 처음 갖고 있는 예상

처음에 너는 생각해.

"나는 매운 라면을 좋아할 확률이 높을 것 같아.

치즈라면은 좀 애매하고,

된장라면은 별로일 거 같아."

이게 바로 **사전 확률**이야.

→ 데이터를 보기 전, 내 추측

2. 🍜 데이터 관찰(likelihood): 실제로 라면을 먹어본 결과

며칠간 실험을 해봤어.

- 매운 라면 먹고: 너무 맵기만 해서 별로
- 치즈라면 먹고: 생각보다 괜찮음
- 된장라면 먹고: 오, 이거 은근 맛있는데?

이게 바로 데이터 = 관찰 결과 = likelihood

3. 🧠 사후 확률(posterior): 관찰 결과를 반영한 새로운 생각

그럼 이제 너는 이렇게 생각해:

“음... 처음엔 매운 라면 좋아할 줄 알았는데,
먹어보니까 된장라면이 내 입맛이야!
이제는 된장라면일 가능성이 가장 높다고 생각해.”

이게 바로 **사후 확률(posterior distribution)**이야.

→ “내 예상 + 실제 경험”을 합친 업데이트된 믿음

🎯 정리하면:

용어	의미	라면 예시
사전 확률 (Prior)	경험 전 나의 믿음	“매운 게 좋을 듯”
우도 (Likelihood)	실제 관측된 데이터	“매운 거 별로, 된장 맛있음”
사후 확률 (Posterior)	업데이트된 믿음	“된장이 내 스타일이네”



그럼 BART의 매번 트리를 조금씩 수정하는 과정은, 사실상 베이지안 사후 분포로부터 트리를 샘플링하는 과정임 이 설명은 너의 라면 예시?에서 확장?해서 설명 가능해?

🍜 BART = 라면 취향 찾기 + 매번 살짝 바꿔보는 과정

아까 넌 된장라면을 좋아하는 걸 데이터 기반으로 깨달았다고 했잖아?

근데 너는 진짜 호기심이 많아서 이렇게 생각함:

"그래도 혹시...

된장 70%, 치즈 30% 비율이면 더 좋을 수도 있지 않을까? 그리고 조리 시간을 좀 늘린다면?? 거기에 두부토핑을 살짝 넣으면?" 등..

이제 트리로 생각하면?

BART에서는:

- 너의 **라면 취향** = 하나의 결정 트리
- 트리는 특정 입력(라면 특성)에 대해 "맛있음 or 별로"를 알려주는 룰

BART의 핵심은?

단 하나의 정답 트리를 고정하지 않아!

대신 **여러 개의 그럴듯한 트리들을 사후 분포에서 뽑아가며**,
전체 평균으로 예측을 만들어.

그러니까 매번 트리를 조금씩 바꾸는 과정은?

내가 생각하는 최적의 라면 레시피를, 살짝씩 조정해보면서
"이게 더 맛있을 수도 있잖아?" 하고 **탐색하는 과정**이야.

- 된장 80% → 된장 60%
- 치즈 없는 버전 → 치즈 살짝 넣은 버전
- 조리시간 3분 → 4분

이런 식으로 트리를 살짝 바꿔가며 성능(데이터와의 일치도)을 평가하고,
그게 좋다면 **사후 분포에서 더 높은 확률로 채택**되는 거지.

BART는 이렇게 동작해

1. 초기엔 라면 레시피(트리)를 막 지어봐 → prior
2. 데이터를 보고: "이 조합은 생각보다 별로였네, 수정해야지"
3. 조금씩 바꿔가며:

- 트리를 grow/split/merge/change 해서 다양하게 만들어봄
- MCMC 방식으로 사후 분포에서 **트리 샘플**들을 계속 뽑는 거임

4. 예측은?

- 여러 트리들의 예측을 ****평균(앙상블)****해서 쓰는 거야

✅ 요약: 너의 라면 비유에서 BART는...

요소	의미
라면 레시피	하나의 트리 모델 구조
실험 결과	데이터
레시피 수정	트리 구조 변경 (MCMC step)
여러 레시피 평균	여러 트리 평균 → 최종 예측
이 모든 걸 감싸는 틀	베이지안 사후 분포 기반 탐색과 추론

즉, BART는 ****"라면 취향을 탐색하는 베이지안 미식가"****라고 보면 돼 ㅋㅋ
 매번 조금씩 레시피(트리)를 바꿔보면서, 가장 맛있는 조합을 찾아가는 거지.

... 그럼 위 예시로 부스팅도 설명해줘

🍜 부스팅 = "라면 레시피 계속 보완해나가기"

앞에서 너 라면 실험을 하다가

"된장라면이 꽤 괜찮네"

라는 결론을 얻었지?

그럼 부스팅은 뭐냐면...

"된장라면이 전체적으로 괜찮긴 한데, 뭔가 2% 부족한데?"

"그 부족한 부분을 메꾸는 보완책을 하나 더 만들어보자"

→ 이게 바로 부스팅이야.

🎯 부스팅이 하는 일 (라면 버전)

1. 1차 시도

- 된장라면 기반 레시피 만들었어 → 약간 짜다

2. 2차 시도 (잔여 오류 보완)

- "간을 잡아주는 치즈를 살짝 넣자" → 치즈 10% 추가

3. 3차 시도 (또 남은 아쉬움 보완)

- "면발이 심심한데 매운 기름을 살짝?" → 매운 오일 약간

4. ...

5. 최종 예측(맛) = 이 모든 시도들의 가중 평균

기술적으로 다시 말하면:

- 첫 번째 트리가 rough하게 예측함
- 그 예측이 **틀린 부분(잔차)**을 보고
- 두 번째 트리가 "이 틀린 부분만 잘 고치자"
- 계속 반복해서, 각 단계마다 **에러 보완**
- 최종 예측은: $\text{Final Model} = T_1 + \text{small step} \times T_2 + \text{small step} \times T_3 + \dots$

$$\text{Final Model} = T_1 + \text{small step} \times T_2 + \text{small step} \times T_3 + \dots$$

→ 한 번에 끝내는 게 아니라, 매우 신중하고 점진적으로 완성해 나감

베이지안과 차이점 (BART vs 부스팅)

항목	BART	부스팅
철학	여러 후보를 뽑아서 평균 (탐색 기반)	한 방향으로 보완해감 (누적 개선)
방식	트리들을 사후 분포에서 샘플링	트리를 순차적으로 학습하고 잔차를 보완
예측	샘플된 모든 트리 평균	앞 단계까지 누적한 결과 + 새 트리
라면 비유	다양한 레시피 뽑아서 평균 맛 찾기	계속 맛의 부족한 부분만 집중해서 보완하기

요약: 부스팅은 이런 라면 미식가

“맛있지만 완벽하진 않아.

그러니까 지금 이 조합에서 **뭔가 아쉬운 부분만** 딱 잡아서
조금씩 **더 완벽한 맛**으로 다듬어보자."

→ 이렇게 **점진적으로 개선**해 나가는 방식이 바로 부스팅.