December 10, 2025

▣ **강의명:** CS109A: 데이터 과학 입문

▣ **주차:** Lecture 19

▣ **교수명:** Pavlos Protopapas, Kevin Rader, Chris Gumb

▣ **목적:** Lecture 19의 핵심 개념 학습

Contents

개요: 결정 트리 학습의 두 가지 핵심

이 문서는 결정 트리(Decision Tree) 모델의 두 가지 핵심적인 확장 개념인 **회귀(Regression)**와 **가지치기(Pruning)**에 대해 다룹니다.

데이터 사이언스 초심자도 쉽게 이해할 수 있도록 각 개념의 필요성부터 작동 원리, 그리고 실제 적용 시 주의사항까지 단계별로 설명합니다.

1. 회귀 트리 (Regression Trees): '예/아니오' 같은 분류(Classification) 문제뿐만 아니라, '집값', '매출액' 등 연속적인 숫자(Quantitative outcome)를 예측하는 회귀 문제에 결정 트리를 사용하는 방법입니다. 핵심은 **'불순도(Impurity)' 대신 'MSE(평균 제곱 오차)'를 기준**으로 트리를 분할하는 것입니다.

2. 가지치기 (Pruning): 트리가 훈련 데이터에만 과도하게 최적화되는 **과적합(Overfitting)**을 방지하기 위한 핵심 기술입니다. 트리를 일단 최대로 성장시킨 후, 불필요한 가지들을 체계적으로 잘라내어 모델의 일반화 성능을 높입니다. 핵심은 **'비용 복잡도(Cost Complexity)'**라는 정규화 항을 도입하는 것입니다.

1 핵심 용어 정리

본격적인 학습에 앞서, 자주 등장하는 핵심 용어들을 정리합니다.

용어	원어	쉬운 설명	비고
회귀 트리	Regression Tree	연속적인 숫자(예: 가격)를 예측하는 결정 트리	분류 트리의 반대
분류 트리	Classification Tree	범주형 값(예: 생존/사망)을 예측하는 결정 트리	
분할 기준	Splitting Criterion	트리의 가지를 나눌 때 사용하는 기준	분류: 지니 불순도, 엔트로피 회귀: **MSE (평균 제곱 오차)**
단말 노드	Leaf Node (Terminal Node)	트리의 가장 마지막에 위치한 노드 (예측값 결정)	
과적합	Overfitting	모델이 훈련 데이터에만 너무 잘 맞아, 새로운 데이터에 대한 예측 성능이 떨어지는 현상	
가지치기	Pruning	과적합을 막기 위해 트리의 복잡도를 줄이는 과정	(사후 가지치기)
비용 복잡도	Cost Complexity Pruning (CCP)	**모델의 오류(Error)**와 **복잡도(Complexity)**를 동시에 고려하는 가지치기 공식	
복잡도 매개변수	Complexity Parameter (α)	트리의 복잡도에 얼마나 큰 페널티를 줄지 정하는 값	하이퍼파라미터

Table 1: 회귀 트리 및 가지치기 관련 핵심 용어

2 회귀 트리 (Regression Trees)

우리는 결정 트리를 '스무고개'와 비슷하다고 배웠습니다. "A인가?", "B인가?" 질문을 통해 대상을 구분하는 것은 **분류(Classification)** 문제입니다.
그렇다면 "이 사람의 나이는 몇 살인가?" 또는 "이 집의 가격은 얼마인가?"처럼 **연속적인 숫자(Continuous Variable)**를 예측해야 하는 **회귀(Regression)** 문제에는 결정 트리를 어떻게 적용할 수 있을까요?
회귀 트리 (Regression Tree)란? 회귀 트리는 예측 결과값이 범주(Category)가 아닌 **연속적인 숫자**인 결정 트리 모델입니다.

- **분류 트리:** 각 단말 노드(Leaf Node)는 **다수결(Majority Vote)**을 통해 가장 흔한 클래스(예: '생존')를 예측합니다.
- **회귀 트리:** 각 단말 노드(Leaf Node)는 해당 노드에 속한 훈련 데이터 샘플들의 **평균(Mean)** 값을 예측합니다.

타이타닉 생존 예측 (분류 트리)

- 질문: "성별이 여성인가?" → 예
- 질문: "객실 등급이 1등급인가?" → 예
- 예측: 해당 노드에 '생존' 80명, '사망' 5명이 있다면, 예측값은 **'생존'** (다수결)

보스턴 집값 예측 (회귀 트리)

- 질문: "방 개수가 6개 이상인가?" → 예
- 질문: "범죄율이 1% 미만인가?" → 예
- 예측: 해당 노드에 속한 집들의 가격이 {5억, 6억, 5.5억} 이라면, 예측값은 **5.5억** (평균)

2.1 핵심 질문: 어떻게 분할 기준을 정할까? (Splitting Criteria)

분류 트리에서는 '지니 불순도(Gini)'나 '엔트로피(Entropy)'를 사용하여, 분할 후 두 그룹이 얼마나 더 '순수해졌는지(Pure)'를 측정했습니다. "생존/사망이 명확히 갈리는가?"가 중요했습니다.
회귀 트리에서도 비슷한 목표를 가집니다. "분할된 두 그룹의 값들이 얼마나 **서로 비슷비슷하게 모여있는가?**"

- **나쁜 분할:** 한 그룹에 {1억, 5억, 10억}이 섞여있음 → 평균 5.3억은 누구도 대표하지 못함 (분산이 큼)
- **좋은 분할:** 한 그룹에 {1억, 1.1억, 1.2억}이 모여있음 → 평균 1.1억은 그룹을 잘 대표함 (분산이 작음)

여기서 "값이 얼마나 흩어져 있는가(분산)"를 측정하는 가장 좋은 지표가 바로 **MSE(Mean Squared Error, 평균 제곱 오차)**입니다.
회귀 트리의 분할 기준: MSE 최소화 회귀 트리는 분할 후 생성될 두 자식 노드(R_1, R_2)의 **MSE의 가중평균**을 최소화하는 지점(변수 p와 임계값 t_p)을 찾습니다.
각 노드 R의 MSE는 해당 노드의 **분산(Variance)**과 같습니다.

$$MSE(R) = \frac{1}{n_R} \sum_{i \in R} (y_i - \bar{y}_R)^2$$

(n_R: 노드 R의 샘플 수, y_i: 실제 값, $\bar{y}_R$: 노드 R의 평균 예측값)

따라서, 트리가 찾는 최적의 분할 (p, t_p)는 다음 공식을 최소화하는 것입니다.

$$\min_{p, t_p} \left[\frac{N_1}{N} MSE(R_1) + \frac{N_2}{N} MSE(R_2) \right]$$

(N: 부모 노드의 총 샘플 수, N_1, N_2: 각 자식 노드의 샘플 수)

2.2 회귀 트리의 성장 및 예측 과정

1. **분할 (Greedy Algorithm):** 모든 변수(Predictors)와 모든 가능한 분할 지점(Unique values)을 하나씩 다 시도해봅니다. 그 중 MSE 가중 평균을 가장 많이 낮춰주는 '최적의 질문' 하나를 선택하여 노드를 분할합니다.
2. **성장:** 위 1번 과정을 재귀적으로 반복하며 트리를 키워나갑니다.
3. **중단 (Stopping Conditions):** 미리 정해둔 중단 조건에 도달하면 성장을 멈춥니다.
   - '$\max_{d}epth$' :
   - '$\min_{s}amples_{l}eaf$' :
   - **Accuracy Gain (MSE Reduction):** 분할로 인해 MSE가 줄어드는 양('Gain(R)')이 정해진 임계값(Threshold)보다 작으면 더 이상 분할하지 않습니다.

$$Gain(R) = MSE(R_{\text{parent}}) - \left(\frac{N_1}{N} MSE(R_1) + \frac{N_2}{N} MSE(R_2) \right)$$

4. **예측 (Prediction):** 새로운 데이터(x_i)가 들어오면, 트리의 질문을 따라가며 특정 단말 노드에 도달합니다. 해당 단말 노드에 저장된 **훈련 데이터의 평균값($\bar{y}_{\text{leaf}}$)**을 최종 예측값($\hat{y}_i$)으로 반환합니다.

3 범주형 변수 처리 (Categorical Attributes)

결정 트리는 "변수 < 임계값" 형태의 비교를 기반으로 작동합니다.

- **수치형(Numerical):** "Sepal width > 4.0?" (가능)
- **범주형(Categorical):** "Color < Red?" (불가능)

'Red'보다 작다는 것은 수학적으로 의미가 없습니다.

3.1 잘못된 접근: 순서형 인코딩 (Ordinal Encoding)

가장 간단하게 숫자를 부여하는 방식입니다. (예: Yellow=0, Red=1, Purple=2)
순서형 인코딩은 절대 사용하면 안 됩니다 (데이터가 실제 순서를 갖지 않는 한).
이 방식은 모델에게 **인공적인 순서(Artificial Order)**를 강제로 학습시킵니다. (예: "Purple(2) > Red(1)")
만약 인코딩 순서를 "Yellow=2, Red=0, Purple=1"로 바꾸면, 트리의 분할 결과 자체가 완전히 달라집니다. 이는 모델의 성능이 데이터의 본질이 아닌, 프로그래머의 임의적인 인코딩 순서에 의존하게 됨을 의미합니다.

3.2 올바른 접근: 원-핫 인코딩 (One-Hot Encoding, OHE)

범주의 각 값(Category)을 자신만의 새로운 **이진(Binary) 변수**로 만드는 것입니다.
원-핫 인코딩 (OHE) 'Color'라는 하나의 변수를 'is_Yellow', 'is_Red', 'is_Purple' 이라는 여러 개의 0 또는 1 값을 갖는 변수로 변환합니다.

변환 전 (Original)

Sepal width	Color
3.0 mm	Yellow
3.5 mm	Red
3.7 mm	Purple

변환 후 (One-Hot Encoded)

Sepal width	Color_Yellow	Color_Red	Color_Purple
3.0 mm	1	0	0
3.5 mm	0	1	0
3.7 mm	0	0	1

Table 2: 원-핫 인코딩 예시

이렇게 변환하면, 트리는 "Color_Red >= 1?" (즉, 색상이 빨간색인가?) 과 같은 논리적인 질문을 수치형으로 수행할 수 있게 됩니다.
Scikit-Learn 사용 시 주의사항
'sklearn.tree.DecisionTreeClassifier' 나 'DecisionTreeRegressor'는 **범주형 변수를 자동으로 처리해주지 않습니다.**
모델에 데이터를 넣기 전에, 사용자가 직접 Pandas의 'get$_d ummies$' '$sklearn.preprocessing.OneHotEncoder$' OH
$XGBoost, LightGBM, CatBoost$.)

4 가지치기 (Pruning): 과적합과의 전쟁

결정 트리를 아무런 제한 없이 끝까지 성장시키면(Full Tree), 훈련 데이터의 모든 샘플을 완벽하게 구분(분류)하거나 완벽하게 예측(회귀, R^2=1)하려 합니다.
이는 훈련 데이터의 사소한 노이즈(Noise)까지 모두 학습하는 **과적합(Overfitting)** 상태로 이어집니다.
이런 모델은 새로운(Unseen) 데이터가 들어왔을 때 형편없는 성능을 보입니다.

과적합의 비유 (R^2=1 밈)

"오늘 R^2=1인 모델을 만들었어요!" "당장 내 집에서 나가!"
R^2=1은 모델이 훈련 데이터를 100% 완벽하게 설명한다는 뜻입니다. 이는 현실에서는 불가능하며, 훈련 데이터에만 과적합된 '암기 기계'를 만들었다는 의미입니다. 이런 모델은 실제 문제 해결에 아무런 도움이 되지 않기 때문에 데이터 과학자들이 가장 경계하는 상황입니다.

4.1 과적합 제어: 사전 중단 vs 사후 가지치기

1. **사전 중단 (Pre-stopping):** 트리가 성장하는 중에 멈추는 방식입니다. ('$\max_depth$', '$min_samples_leaf$')
   2. **문제점:** 최적의 중단 시점을 미리 알기 어렵습니다. 너무 일찍 멈추면 **과소적합(Underfitting)**이, 너무 늦게 멈추면 **과적합(Overfitting)**이 발생합니다.

사후 가지치기 (Post-pruning / Pruning): "일단 끝까지 키우고, 나중에 잘라낸다."

- **장점:** 트리의 전체 구조를 본 후에 불필요한 부분을 제거하므로, 더 유연하고 정교한 모델 복잡도 제어가 가능합니다.
- **방법:** 가장 표준적인 방법이 **비용 복잡도 가지치기(Cost Complexity Pruning, CCP)**입니다.

4.2 비용 복잡도 가지치기 (Cost Complexity Pruning, CCP)

CCP는 모델의 **오류(Error)**와 **복잡도(Complexity)** 사이에 균형점을 찾는 정규화(Regularization) 기법입니다.
비용 복잡도 (Cost Complexity) 공식 CCP는 트리의 '비용' $C(T)$를 최소화하는 것을 목표로 합니다.

$$C(T) = \text{Error}(T) + \alpha \cdot |T|$$

- $C(T)$: 트리 T의 총 비용 복잡도
- $\text{Error}(T)$: 트리 T가 훈련 데이터에서 발생시키는 **총 오류** (예: 분류 오류 수, 또는 총 MSE)
- $|T|$: 트리 T의 **단말 노드(Leaf) 개수**. (트리의 복잡도를 나타내는 척도)
- α (알파): **복잡도 매개변수 (Complexity Parameter)**. 사용자가 정하는 하이퍼파라미터입니다.

α는 '복잡도에 대한 페널티'입니다.

- $\alpha = 0$: 복잡도 페널티 없음. $C(T) = \text{Error}(T)$가 되므로, 훈련 오류가 가장 낮은 **가장 큰 트리(Full Tree)**가 선택됩니다.
- $\alpha \to \infty$: 복잡도 페널티가 매우 큼. 단말 노드가 2개인 것보다 1개인 것을 선호하게 되므로, 결국 **가장 단순한 트리(Root Node만 있는 트리)**가 선택됩니다.
- $0 < \alpha < \infty$: α 값이 커질수록, 오류(Error)가 조금 늘어나더라도 더 단순한(단말 노드가 적은) 트리를 선호하게 됩니다.

$\alpha = 0.2$ 일 때, 두 트리를 비교해봅시다.

트리 T (Full Tree):

- Error(T) = 0.32

- $|T|$ (단말 노드 수) $= 8$
- $C(T) = 0.32 + (0.2 \times 8) =$ **1.92**

트리 T_{small} (가지치기 후):

- $\text{Error}(T_{small}) = 0.33$ (훈련 오류는 약간 증가)
- $|T_{small}|$ (단말 노드 수) $= 7$
- $C(T_{small}) = 0.33 + (0.2 \times 7) =$ **1.73**

결론: 비록 T_{small}이 훈련 오류는 0.01 더 높지만, 전체 비용 복잡도 $C(T)$는 더 낮습니다 (1.73 < 1.92). 따라서 $\alpha = 0.2$ 일 때는 T_{small}이 T보다 '더 좋은 트리'로 간주됩니다.

4.3 최적의 α와 트리 찾는 과정 (Nested Cross-Validation)

CCP의 알고리즘은 복잡해 보이지만, 본질은 **2단계 교차 검증**입니다.

1. **[Inner Loop] 특정 α에 대한 최적의 트리 찾기**
   먼저 α 값을 하나 고정합니다 (예: $\alpha = 0.1$). 알고리즘은 Full Tree(T_0)에서 시작하여, '가장 약한 고리 (Weakest Link)' (즉, 잘라냈을 때 $C(T)$가 가장 적게 증가하거나 오히려 감소하는 노드)부터 차례대로 제거합니다.
   이 과정을 통해 $\alpha = 0.1$일 때 가능한 후보 트리들의 목록을 만듭니다: $\{T_0(\text{Full}), T_1, T_2, ..., T_L(\text{Root})\}$
   이 후보 트리들을 **검증 데이터(Validation Set)**에 적용하여, $\alpha = 0.1$일 때 검증 성능(예: Validation MSE)이 가장 좋은 트리 $T_{\text{best_for_0.1}}$를 하나 선택합니다.
2. **[Outer Loop] 모든 α 중 최적의 α 찾기**
   이제 다른 α 값 (예: $\alpha = 0.2, \alpha = 0.5, ...$)에 대해 1번 과정을 **반복**합니다.
   - $\alpha = 0.1$ 일 때 최적 트리: $T_{\text{best\_for\_0.1}}$ (검증 점수: 85점)
   - $\alpha = 0.2$ 일 때 최적 트리: $T_{\text{best\_for\_0.2}}$ (검증 점수: 90점)
   - $\alpha = 0.5$ 일 때 최적 트리: $T_{\text{best\_for\_0.5}}$ (검증 점수: 88점)
3. **최종 선택**
   모든 α 후보들 중에서 가장 높은 검증 점수(90점)를 기록한 $\alpha = 0.2$와, 그 때의 트리 $T_{\text{best_for_0.2}}$를 **최종 모델**로 선택합니다.

5 FAQ 및 자가 점검

강의 중 나온 퀴즈와 자주 묻는 질문들을 통해 이해도를 점검합니다.

Q1: 결정 트리 모델은 '탐욕적 알고리즘(Greedy Algorithm)'을 사용하는데, 왜 이것이 정당화되나요?

A: '탐욕적'이라는 것은 매 분할 시점마다 당장 MSE(또는 Gini)를 가장 많이 줄여주는 최적의 분할을 찾는다는 의미입니다. 이렇게 찾은 분할이 나중에 전체 트리의 최적(Global Optimum)을 보장하지는 않습니다. 하지만, 모든 가능한 트리 조합을 탐색하는 것은 계산적으로 거의 불가능(NP-hard)합니다. 탐욕적 접근 방식은 계산 비용이 훨씬 저렴하면서도 **현실적으로 매우 준수한 성능의 모델**을 찾아주기 때문에 널리 사용됩니다.

Q2: 하나의 결정 트리 안에서 동일한 변수(Feature)가 여러 번 사용될 수 있나요?

A: 예, 가능합니다. 예를 들어, 상위 노드에서 "Work experience > 2"로 분할한 후, 그 자식 노드 중 하나에서 "Work experience > 4"로 **더 세분화된 분할**을 할 수 있습니다. 이는 모델이 데이터의 복잡한 관계를 더 정교하게 학습할 수 있게 해줍니다.

Q3: 훈련된 회귀 트리(Regression Tree)가 새로운 데이터 포인트를 예측할 때, 마지막 단계는 무엇인가요?

A: 새로운 데이터가 트리를 따라 내려가 도달한 **단말 노드(Leaf Node)에 저장된 값**을 예측값으로 반환합니다. 이 저장된 값은 바로 해당 단말 노드에 속해있던 **훈련(Training) 데이터들의 평균(Average)** 값입니다.

Q4: 어떤 회귀 트리가 4개의 단말 노드(Region)를 생성했습니다. 이 트리의 전체 MSE는 어떻게 계산하나요?

- R1: 샘플 90개, MSE = 0.2
- R2: 샘플 5개, MSE = 1.2
- R3: 샘플 3개, MSE = 1.5
- R4: 샘플 2개, MSE = 1.8

A: 트리의 전체 MSE는 각 노드 MSE의 **가중 평균(Weighted Average)**입니다.

- 총 샘플 수(N) = $90 + 5 + 3 + 2 = 100$
- 전체 MSE = $(\frac{90}{100} \times 0.2) + (\frac{5}{100} \times 1.2) + (\frac{3}{100} \times 1.5) + (\frac{2}{100} \times 1.8)$
- 전체 MSE = $(0.18) + (0.06) + (0.045) + (0.036) = \mathbf{0.321}$

대부분의 샘플(90%)이 R1에 속해있으므로, 전체 MSE는 R1의 MSE(0.2)에 가장 가깝게 나옵니다.

Q5: '$\mathbf{max}_{d}epth = 1$' $(DecisionBoundary)$ $(Linear)$, $(Non-linear)$?

A: 선형 (Linear) 입니다. **'$\mathbf{max}_{d}epth = 1$' 단 하나의 분할** .(: "$x >$ 6.5") ( ) , . 2 .