DATA MINING 101

Somewhere in between Statistics and AI

Contents

1. Tree-based Models

A. 의사 결정 나무 (Decision Trees)

지난 시간 다뤘던, 특정 주택의 부동산 가격을 예측하는 회귀 문제를 다시 불러와보자. Bedrooms, Sq.feet, Neighborhood 와 같은 요소들을 독립 변수로 두고, 이를 토대로 Sale price 를 산출하는 문제 상황이었다. 컴퓨터에게 '네가 알아서 회귀식 적합해줘!' 라고 할 수도 있지만 비교적 적은 양의 데이터가 주어졌고 충분히 수동적으로 이들 데이터의 패턴을 발견할 수 있다면, 아래 그림에서 보는 것처럼 if 조건문을 활용해 알고리즘을 설계할 수 있을 것이다.

Bedrooms	Sq. feet	Neighborhood	Sale price
3	2000	Normaltown	$250,000
2	800	Hipsterton	$300,000
2	850	Normaltown	$150,000
1	550	Normaltown	$78,000
4	2000	Skid Row	$150,000

[training data]

Bedrooms	Sq. feet	Neighborhood	Sale price
3	2000	Hipsterton	???

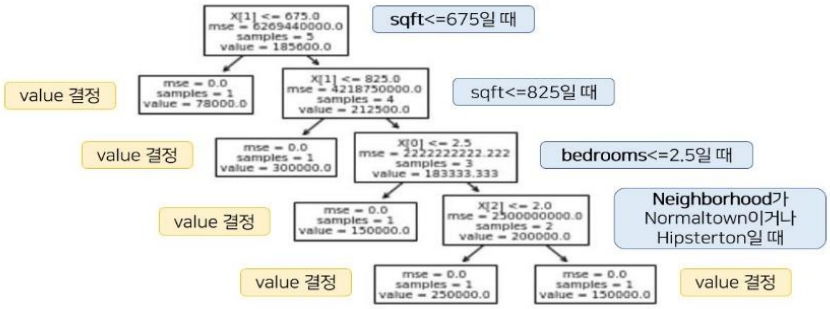
[test data]

```
def estimate_house_sales_price(num_of_bedrooms, sqft, neighborhood):
  price = 0
  price_per_sqft = 200
  if neighborhood == "hipsterton":
      price_per_sqft = 400
  elif neighborhood == "skid row":
      price_per_sqft = 100

  price = price_per_sqft * sqft
  if num_of_bedrooms == 0:
      price = price — 20000
  else:
      price = price + (num_of_bedrooms * 1000)
return price
```

두번째 그림에서 neighborhood 가 'hipsterton'인 경우 더 높은 sqft 당 가격이 더 높게 생성된 것처럼, 주어진 데이터셋들을 설명하는 요소들 중 이들 요소가 어떠한 조건이 갖춰졌을 때 이에 대한 산출값이 결정되는지, 다시 말해 **어떤 조건이 결정적으로 충족되었을 때** 그 최종값이 형성(또는 범주가 결정)되는지에 대해 궁금해할 수 있다. 이러한 논리를 기반으로 하여 설계된 일련의 모델들을 오늘 함께 살펴보려 한다.

위의 예시를 오늘 다룰 트리 모형으로 표현하면 다음과 같다.

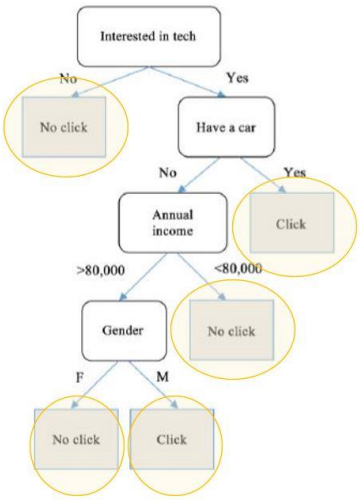


*Neighborhood에서 Normaltown, Hipsterton, Skid Row를 임의로 1,2,3 값 가지도록 인코딩

위 그림을 보고 스무고개 놀이가 생각난다면 당신은 직관이 좋은 사람! 스무고개 놀이에서 답을 찾아내기 위해 일련의 질문들을 던져 정답에 근사하도록 조건들을 제약하듯, 의사 결정 트리(decision tree)로 대표되는 트리 기반 모델들은 각 단계에서 조건들을 효과적으로 제약할 수 있도록 순차적으로 질문을 던진다. 가장 포괄적인 질문부터 시작하여 일련의 질문들을 통해, 다시 말해 관측치들을 나눌 때 결정적으로 정의되는 조건들을 하나하나 제시하여 이들 조건을 만족하는 관측치들을 배치하는 과정이라고 이해할 수 있겠다.

큰 그림에서 이해하자면, 이는 독립변수들이 차지하는 공간을 일정한 영역들로 분할(partition)하는 것과 같은 개념이다. 다시 말해, x변수들이 차지하는 공간, 즉 input space를 직사각형의 형태로 나누는 것이다. 다른 형태를 취해도 되지만, 해석과 이해에 용이하도록 직사각형 형태를 취한다고 보면 되겠다. 트리 모델은 회귀 문제와 분류 문제 모두에 적용 가능하며 뒤에 나올 앙상블 기법들을 더하여 모델을 설계할 때 그 성능이 매우 우수하게 나타나곤 하기에 자주 사용된다.

우선, 용어의 통일을 꾀하자. 의사 결정 나무의 요소는 다음과 같이 정리될 수 있다.



위 분류 문제에서, 노랑색 원으로 표시된 항목들은 이미 그 범주가 결정되어 더 이상 분기하지 않음을 확인할 수 있다. 이와 같이 하부 구조 상의 노드(node)들을 추가적으로 가지지 않는 항목들을 leaf node 또는 terminal node 라고 일컫는다. 'Interested in tech?', 'Have a car?' 와 같은 질문들은 라벨(label)이라는 이름으로 통용된다.

주어진 데이터셋에 대해 모델이 경향성을 파악할 수 있도록 하는 과정을 '적합한다'고 했는데, 의사 결정 나무의 경우 그 과정 마다 terminal node 에 특정한 상수 값이 적합된다. 회귀 문제의 경우에는 terminal node 에 배치된 y 값들의 평균으로 기술되며, 분류 문제라면 terminal node 에서 가장 많이 등장하는 범주, 즉 최빈값으로 제시된다.

이제 간단히 알고리즘을 이해해보자. 그 전에 몇 가지 질문이 제기될 수 있을 것이다. **1) 어떤 질문을 던졌을 때 의사결정 나무가 효과적으로 분기하는가?** 그리고, **2) terminal node 에 적합하는 상수 c 예측값을 어떻게 도출해낼 수 있는가?**

B. CART (Classification And Regression Trees)

트리 생성 알고리즘으로는 ID3, C4.5, CHAID 등 다양한 종류가 있으나, 본 교안에서는 CART(Classification And Regression Trees)에 집중하여 그 원리를 이해해보고자 한다. 우리의 모델을 다음의 수식으로 표현할 수 있다. c_m을 terminal node 에 적합하는 평균값(회귀 문제) 혹은 최빈값(분류 문제)이라고 이해하고 아래 수식을 살펴보자.

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

$$when \bigcup_{m=1}^{M} R_m = \mathbb{R}^p \; and \; R_m \cap R_p = \emptyset \; (non-overlapping, distinct)$$

$$and \; \mathbb{R}^p \; is \; a \; p-dimensional \; input \; space$$

먼저, 회귀 문제에서 위 알고리즘이 어떻게 적용되는지 확인해보자.

1) Decision Tree Regressor

모델 적합 시, 실제 f 에 근접한 $\hat{f}$를 찾는 것이 우리의 목표라고 했다. 또, 회귀 문제에서는 f 와 $\hat{f}$가 서로 얼마나 떨어져 있는지에 대한 지표로 MSE(Mean Squared Error)를 사용한다고 했다. 사실, MSE 는 잔차제곱합(Residual Sum of Squares, RSS)을 자유도로 나눠준 것이라 할 수 있는데, 트리 모델은 RSS 값을 줄여 나가는 방향으로 작동한다. 오차를 줄여 나간다는 원리는 동일하다. 이를 다음과 같은 수식으로 표현 가능하다.

$$min_{c_m} \sum_{i=1}^{N} \{y_i - f(x_i)\}^2 = min_{c_m} \sum_{i=1}^{N} [y_i - \sum_{m=1}^{M} c_m I\ (X\ \in\ R_m)]^2$$

c_m이 왜 모든 관측치들의 평균값으로 정의되는지 증명하는 수학 전개식은 부록으로 빼놓았으니 참고 바란다.

회귀 트리는 **'어떤 독립변수를 기준으로 분기를 할지'**와 같은 결정의 순간순간마다 최적의 선택을 취하는, 탐욕적 알고리즘(greedy algorithm)의 flow 를 따른다. 랜덤하게 feature 를 선택해서 RSS 계산을 해보고 RSS 값이 줄어드는 방향으로 모델링 과정을 이어 나가기 때문이다. 다음의 예시를 통해 그 원리를 파악해보자.

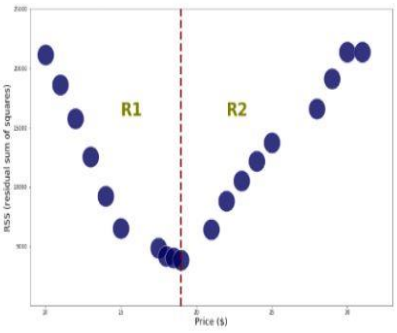
랜덤 기준 1
RSS=15762

랜덤 기준 n
RSS=3874
최종 선택

	Price ($)	Review Scores Rating
0	10.0	10
1	11.0	10
2	12.0	10
3	13.0	10
4	14.0	13
5	15.0	20
6	17.5	35
7	18.0	44
8	18.5	52
9	19.0	55
10	21.0	80
11	22.0	83
12	23.0	80
13	24.0	83
14	25.0	85
15	28.0	100
16	29.0	100
17	30.0	100
18	31.0	100
19	31.0	100

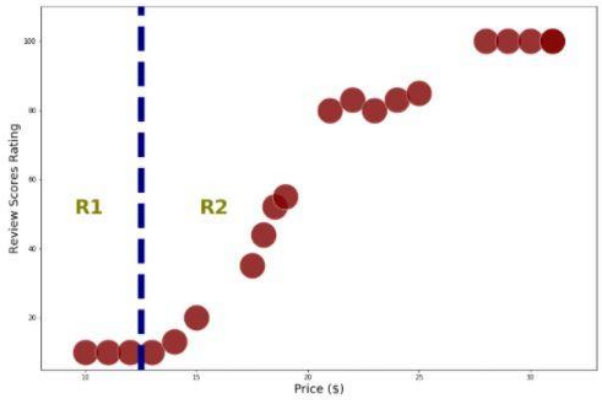
독립변수가 다음과 같이 한 개(드라이클리닝 가격)로 주어졌을 때, 리뷰 점수를 예측하는 모델링을 실시한다 해보자. 일반적인 예시는 아니나, 개념 이해를 위해 먼저 독립변수를 한 개만 두고 시작해보자. 이와 같이 독립변수가 하나만 존재하는 경우라면 Price 기준을 무엇으로 삼을지 랜덤하게 선택하여 RSS 계산을 해보면 된다. RSS 값이 줄어드는 방향으로 랜덤하게 기준을 세워주면 되고, 멈춤 조건은 RSS 값이 더 이상 줄어들지 않을 때이다.

여기서 'Price<=19'를 기준으로 삼았을 때 RSS 가 가장 작은데, 이를 바탕으로 input space 를 R1 과 R2 로 나눌 수 있다.



terminal node 에 평균값이 위치하고, 이를 바탕으로 RSS 를 계산한다고 했는데 아직 잘 와 닿지 않을 수 있다. 위의 두 가지 경우들에 대해 RSS 를 직접 계산해보자.

i. price 가 **13** 일 때를 기준으로, 이보다 작은 값/큰 값으로 나누기



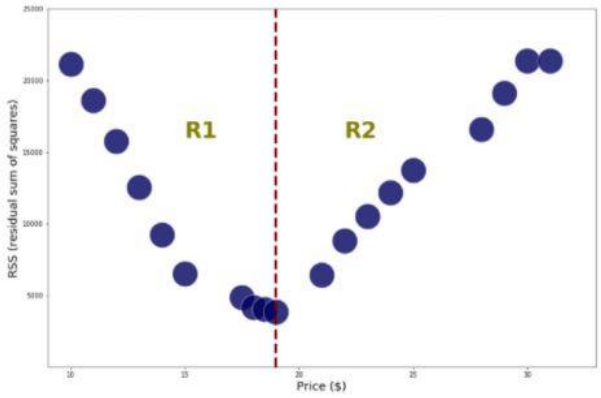
-> terminal node 에 평균값이 위치한다는 게 무슨 느낌인지..!

왼: Price 가 13 인 경우를 제외, 이보다 작은 값을 가지는 관측치들의 종속변수 평균

오: Price 가 13 인 경우를 제외, 이보다 큰 값을 가지는 관측치들의 종속변수 평균

$$RSS = (10-10)^2 + (10-10)^2 + \cdots + (100-65)^2 + (100-65)^2 = 15762.0$$

ii. price 가 **19** 일 때를 기준으로, 이보다 작은 값/큰 값으로 나누기



$$RSS = (10-25.9)^2 + (10-25.9)^2 + \cdots + (100-91.1)^2 + (100-65)^2 = 3873.79$$

독립변수 2개 추가됨!

	price ($)	cleaning_fee ($)	license	review scores rating
0	10.0	0.0	0	10
1	11.0	1.0	0	10
2	12.0	2.0	1	10
3	13.0	3.0	0	10
4	14.0	4.0	1	13
5	15.0	5.0	0	20
6	17.5	7.5	1	35
7	18.0	8.0	1	44
8	18.5	8.5	1	52
9	19.0	9.0	0	55
10	21.0	11.0	1	80
11	22.0	12.0	0	83
12	23.0	13.0	1	80
13	24.0	14.0	1	83
14	25.0	15.0	0	85
15	28.0	18.0	1	100
16	29.0	19.0	0	100
17	30.0	20.0	0	100
18	31.0	21.0	1	100
19	31.0	21.0	1	100

*각 변수 안에서 최소 RSS 산출 조건

여기서 독립변수가 하나씩 더 추가된다고 생각하고, 문제 상황을 확장해보자. 어떤 독립변수를 먼저 큰 분기점의 기준으로 잡아줘야 RSS 값을 꾸준히 감소시킬 수 있을까? 먼저, 각 독립변수들 중 가장 작은 RSS 값을 내는 기준 선택을 하고 이들 기준을 같은 선상에 놓아, 이들 중 가장 작은 RSS 값을 내는 기준을 선택하는 것이다. 다시 말해, 최적의 기준들 중에서 다시금 가장 최적의 기준을 선별해내는 방식을 취하는 것이다. (왕들 중 왕, 연예인들의 연예인 이런 느낌이랄까)

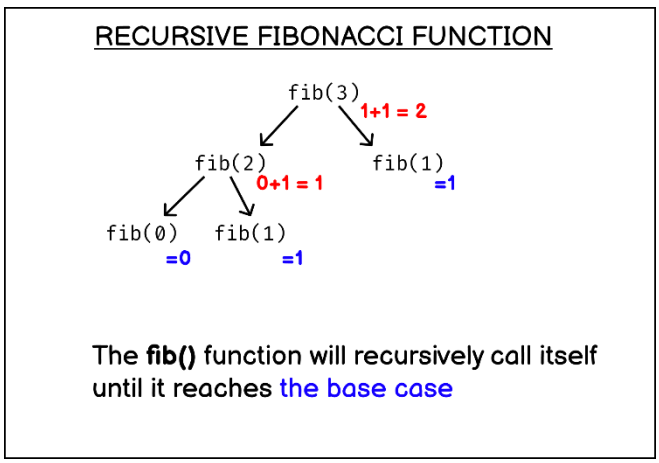
i. Price에 대한 RSS 최적 기준과 RSS: 19 / **3873.9**

ii. Cleaning_fee에 대한 RSS 최적 기준과 RSS: 9 / 64214.8

iii. Review_scores_rating에 대한 RSS 최적 기준과 RSS:

0 / 11685.5

그러므로, 각 독립변수 안에서 최소 RSS를 산출하는 조건들 중 가장 작은 RSS를 산출하는 Price 독립변수와 이의 세부 기준 Price<=19를 첫번째 분기점의 라벨로 삼는다. 그럼, 그 다음 라벨은? R1, R2로 나눠진 input space 각각에서 최소 RSS 산출 조건을 찾는다. R1이 R11, R12로 나눠졌고 R2가 R21, R22로 나눠졌다고 해보자. 위에서 이 과정의 멈춤 조건은 RSS 값이 더 이상 작아지지 않을 때라고 했다. 이제 R11, R12, R21, R22 각각에서 공간 분할을 시도해보면 되는 것이다 (멈춤 조건 도달하기 전까지). 이렇게, 분할된 공간에 대해 다시금 분할을 취해주는 방식을 '재귀(recursion)'라고 하며, 동일한 작업을 수행하는 함수를 해당 함수 안에서 계속적으로 호출하는 방식으로 작성된다. 피보나치 수열을 만드는 함수를 대표적인 재귀 함수라고 볼 수 있겠다. (중요한 건 아닌데 궁금할까봐)

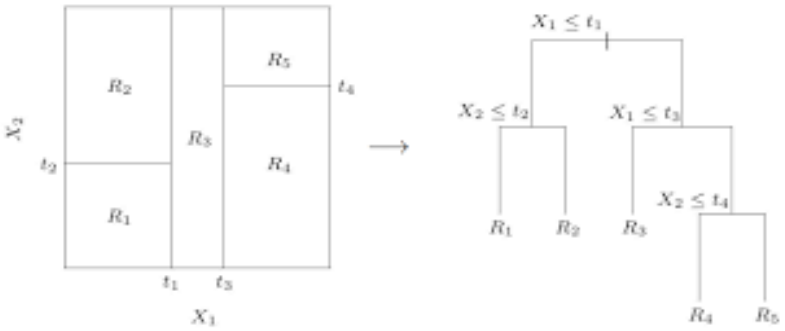


지금까지의 내용을 다음의 수식으로 요약할 수 있다. 여기서 j는 특정 독립변수이고 s는 분기 지점, 다시 말해 기준이 되는 관측치의 종속변수 값이다. c_1과 c_2는 분기된 terminal node들의 좌/우 종속변수 평균값(m번째 분할에 들어가는 관측치들의 종속변수 평균값)을 의미한다.

$$Find\ j\ and\ s\ satisfying$$

$$min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2$$

독립변수가 두 개일 때, 최종 분할된 input space는 다음과 같은 형태로 표현될 수 있다.

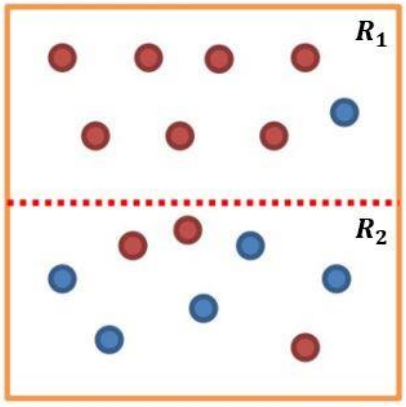


2) Decision Tree Classifier

회귀 문제를 확인해봤으니, 동일한 구조를 통해 분류 문제를 해결해보자. 분류 문제에서 terminal node 의 적합값은, 해당 분할에 포함된 범주들의 최빈값이라 밝힌 바 있다. 다수결의 원칙을 생각하면 쉽겠다. 비슷한 종류의 관측치들끼리 '몰려 있는 형태'를 지닌다면 가장 이상적이라고 판단한다. 앞선 회귀 문제의 예시에서 RSS 가 줄어드는 방향으로 트리가 분기하도록 했듯, 분류 과정에서도 RSS 와 같이 실제 값과 우리가 모델로서 예측한 값 사이의 괴리를 설명할 수 있는 지표가 필요하다.

이 때 등장하는 개념이 **불순도(impurity)**인데, 분류 나무에서는 이 불순도가 감소하는 방향으로 학습이 진행된다. 다시 말해 학습이 진행됨에 따라, 다음 분기 이후 terminal node 에 등장하는 관측치들이 어떤 카테고리에 속할 지에 대해서 불확실성(uncertainty)이 줄어드는 방향으로 작동한다고 할 수 있다. 불순도와 불확실성이 줄어든다는 것은 곧 순도가 증가/불확실성이 감소한다는 것인데, 이를 정보 이론 (information theory)의 관점에서 풀어내면 정보 획득(information gain)이라고 부를 수 있는 것이다.

도합 16 개의 빨간색 구슬과 파란색 구슬이 있다고 가정해보자. 이 중 빨간색 구슬이 10 개, 파란색 구슬이 총 6 개라면, 어떻게 구슬들을 배치했을 때 가장 균일한 분류 상태에 다다를 수 있을까?



위와 같이 임의의 구슬들이 두 개의 부분공간 R_1, R_2에 대하여 8 개씩 배치되어 있다고 해보자. 위 그림에서, R_2 구역이 R_1 구역에 비해 불순도가 더 높다. R_2 구역에는 빨간 구슬과 파란 구슬이 3:5 의 비율으로 섞여 있기 때문에 적절히 분류되었다고 보기 힘들다. 각 부분공간에 다수결을 차지하는 범주를 선택한다 했으므로 R_1 부분공간의 예측값은 빨강, R_2 부분공간의 예측값은 파랑이 되지만 추가적인 분류 작업이 필요해 보인다.

분류가 잘 되었는지, 그러니까 분할된 부분공간의 '순도'는 어떻게 정량적으로 계산할 수 있을까? 오분류오차(misclassification error)와 지니계수(gini index), 엔트로피(entropy)와 같은 종류가 있으나 본 교안에서는 엔트로피의 개념에 초점을 맞춰 설명을 전개하고자 한다.

물리학 분야에서도 등장하는 '엔트로피'는 익히 '혼란'이라는 개념으로서 설명된다. 이를 우리의 상황에 적용해본다면 '불순도' 또는 '불확실성' 이라는 개념으로 번역될 수 있을 것이다. 이러한 불확실성을 줄이는 것이 목표이기에 트리의 분할 후 얼마나 해당 부분공간에 대하여 순도가 좋아졌는지를 평가하는 것이 우리의 flow 일 것이다. Before 와 After 를 비교해서 트리 분할을 통해 불확실성이 얼마나 줄어드는지를 측정하는 것이다. 분할 전 엔트로피와 분할 이후 엔트로피를 직접 계산해보자.

엔트로피 공식은 다음과 같다.

$$Entropy(X) = -\sum_{k=1}^{m} p_k \log_2(p_k)$$

전체 input space 를 X라고 두었을 때 엔트로피를 계산하는 방법이다. p_k는 X 영역에 속한 관측치 들 중 k 범주에 속하는 관측치들의 비율을 의미한다. 분할 전 엔트로피는 다음과 같이 계산된다.

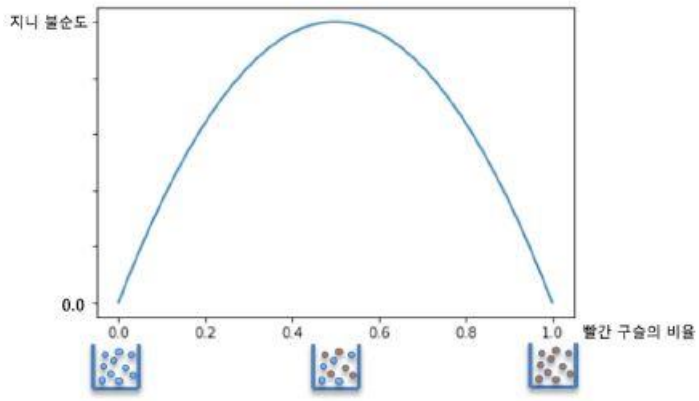
$$Entropy(X) = -\frac{10}{16}\log_2\left(\frac{10}{16}\right) - \frac{6}{16}\log_2\left(\frac{6}{16}\right) \approx 0.95$$

빨강색 점선을 그어 두 개의 부분공간 (R_1, R_2)을 생성, 이때의 엔트로피를 계산해보자면

$$0.5 \times \left(-\frac{7}{8}\log_2\left(\frac{7}{8}\right) - \frac{1}{8}\log_2\left(\frac{1}{8}\right)\right) + 0.5 \times \left(-\frac{3}{8}\log_2\left(\frac{3}{8}\right) - \frac{5}{8}\log_2\left(\frac{5}{8}\right)\right) \approx 0.75$$

트리를 분할한 후 약 엔트로피가 0.2 정도 감소했음을 확인할 수 있다. 분할 전의 엔트로피와 분할 후의 엔트로피의 차이가 곧 정보 획득이므로 이 값이 0.2 임을 짚어볼 수도 있겠다.

불순도 개념을 복습하는 겸, 다음의 그래프를 보고 그 뒤에 따라 나오는 일련의 문장들을 곱씹어보며 내용을 정리해보자.



*불확실성이 높다는 것은 클래스 별로 관측치들이 고르게 분포되어 있다는 것을 의미한다.

*모든 클래스에 똑 같은 비율로 데이터가 존재할 때, 여기서 어떤 데이터를 임의로 뽑는다면 이 데이터가 어떤 클래스에 속할지 불확실하다.

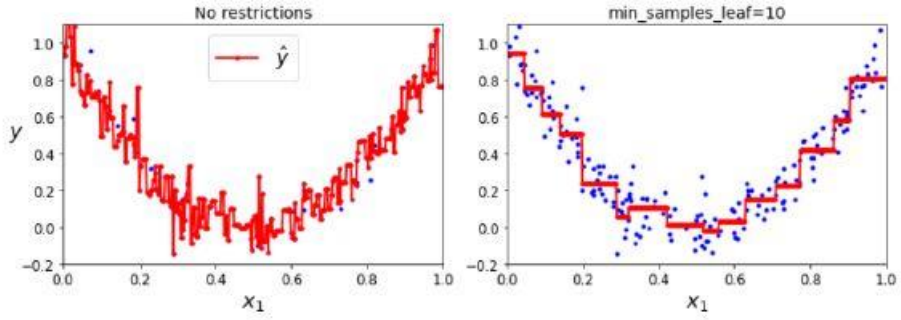
*반대로, 관측치들이 특정 클래스에 편중되어 있다면 데이터의 클래스 분류에 대한 불확실성이 낮다.

*클래스 두 개일 때, 모든 관측치가 하나의 클래스에만 속한다면 엔트로피는 $-(1 \times \log_2 1 + 0) = \mathbf{0}$

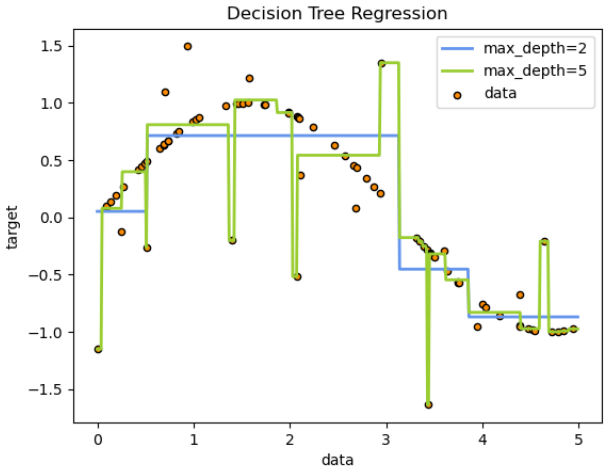
*반면 모든 관측치가 두 개의 클래스에 절반씩 속한다면 엔트로피는 $-(0.5 \times \log_2 0.5 + 0.5 \times \log_2 0.5) = \mathbf{1}$

3) How to Avoid Overfitting in Tree-based Models

어떤 모델이 절대적으로 그 복잡도(model complexity)가 높고 어떤 것이 무조건 그 복잡도가 낮다고 말을 할 순 없겠지만, 1 주차 때 언급한 선형 회귀 모델과 비교하자면 트리 모델은 확실히 그 복잡도가 크다고 할 수 있겠다. 하이퍼 파라미터를 어떻게 튜닝해주냐에 따라 동일한 데이터셋에 대해서 그 모양이 확연히 달라질 수 있기 때문이다. 아무런 제약 조건을 주지 않았을 때와 min_samples_leaf 파라미터의 값을 10 으로 제약했을 때의 모습을 비교해보자.



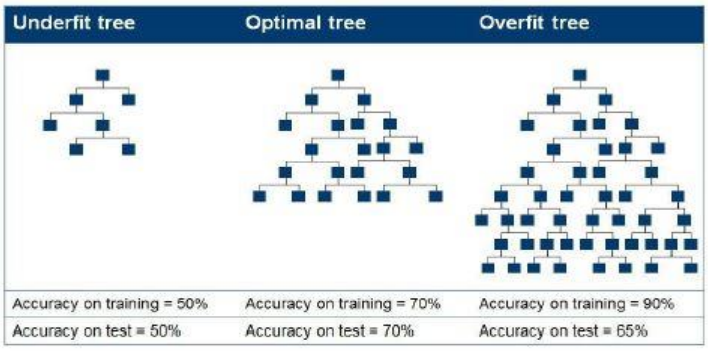
트리의 깊이(레벨)를 의미하는 'max_depth' 파라미터에 대해서 다른 값을 주었을 때도 역시 모델의 형태가 크게 달라짐을 확인할 수 있다.



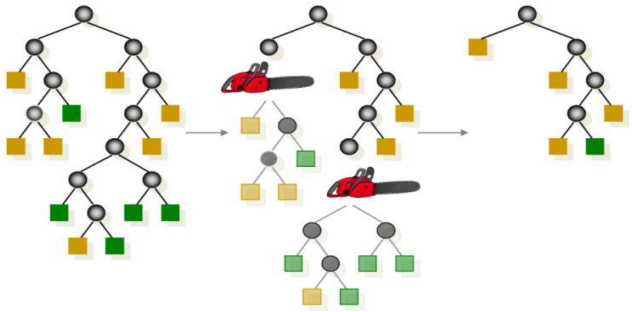
이처럼 모델 간 분산(variance)이 큰 경우, 단 하나의 모델만을 취해 '자, 끝~! 이걸로 하자.' 라고 말하는 것은 위험하다. **'과연 한 개의 나무로 충분히 설명/적용 가능할까?'**라는 질문에서 출발, 여러 개의 트리 모델을 적합시켜 이들을 종합적으로 평가하고자 하는 방법론이 이후 나올 랜덤 포레스트 모델링 기법(Random Forest)이다.

더불어 학습 데이터를 이용해 적합 시킨 모델에, 검증 데이터/테스트 데이터를 놓고 그 성능을 확인해보고자 한다면 (재사용성의 측면에서도 그렇고), 오버피팅이 되는 상황은 필히 경계해야 할 것이다. '너무 많은 변수를 사용했어!' 라는 말을 트리 모델에 적용한다면, '너무 분기를 많이 했어!' 정도가 적당할 듯하다. 수차례 분기를 통해 세부적인 기준 각각에 대한 terminal node 를 생성한다면 관측치들을 매우 자세하게 분류한 셈이 되겠고,

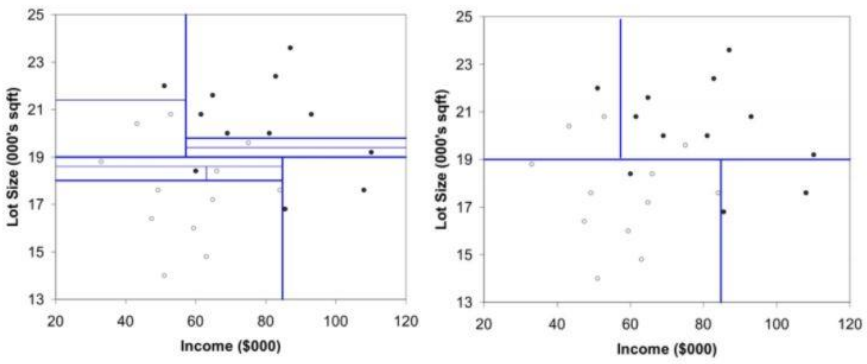
몇 번 분기하지 않는 작은 트리의 경우에는 분기 라벨들이 비교적 러프하게 정의가 되기에 언더피팅의 상황에 도달할 확률이 높다. 바로 아래 그림과 같이 말이다.



분기가 너무 많아서 overfit 한 tree 가 생성이 되었다면, 필요하지 않은 가지는 쳐 내주고 유사한 특성을 보이는 부분 공간들은 합쳐주면 되는 법! Pruning 이라고 불리는 가지치기 방법은 오버피팅 상황에서의 솔루션으로 제공되기도 한다. 앞에서 가지를 '쳐 내준다' 라고 했으나, 그렇다고 하여 관측치들을 아예 삭제하는 것이 아니라 분기를 합치는(merge) 개념으로 보는 것이 맞다. 모든 terminal node 의 불순도가 0 인 경우, 적절히 이를 다시 불순하게 만들어줘야(?) 새로운 데이터가 등장했을 때에도 탄력적으로 반응할 수 있다.



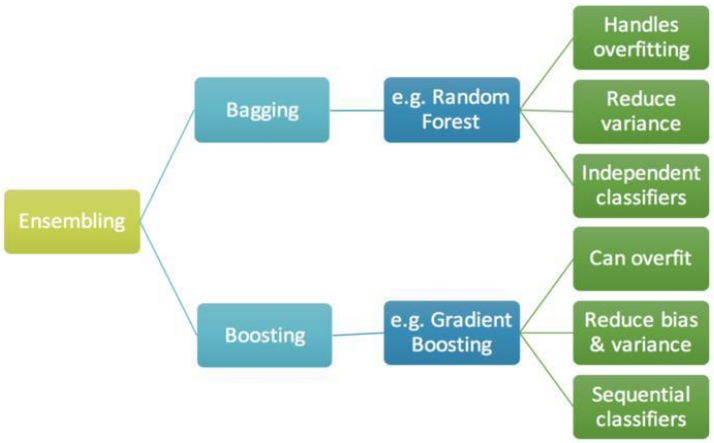
이를 통해 적당히 불순하게 만들어준 input space 의 부분공간으로의 분할 모습은 오른쪽과 같이 표현될 수 있을 것이다.



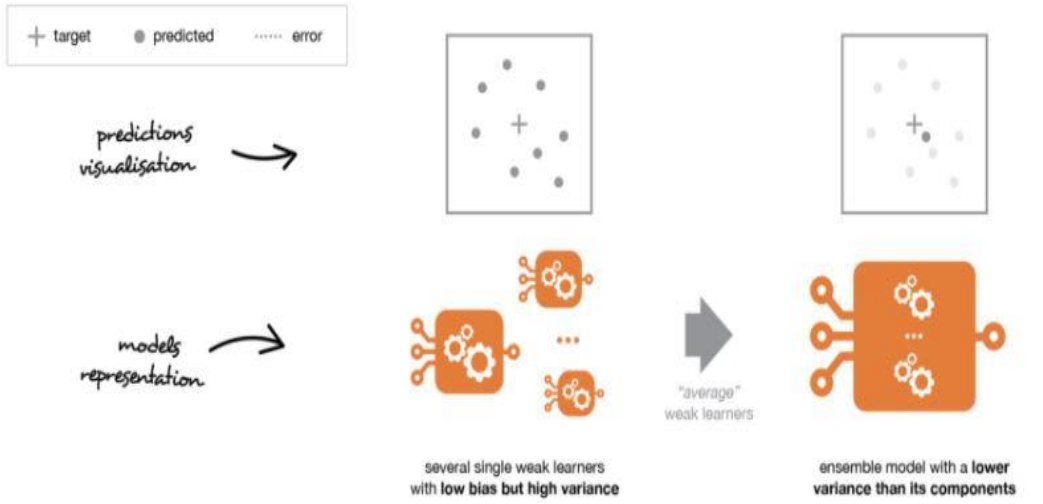
단일한 모델의 오버피팅 확률을 최소화하고자 하는 노력으로 가지치기를 제시했다면, 이제 이들 모델들(복수형!)을 조합하는 방법론인 앙상블 기법(Ensemble Method)이 무엇인지 배워보자.

2. Ensemble Methods

앙상블 기법은 여러 모델을 독립적으로 학습시킨 다음, 각 모델이 만들어 낸 결과를 조합하여 최종 결과를 생성하는 방법이라 할 수 있겠다. 핵심은 여러 개의 약한 학습기(weak learner, 우수한 성능을 내지 않는 학습기)를 결합하여 더욱 성능을 잘 내도록 하는 강한 학습기(strong learner)를 만들어 내는 것이다. 각 방법론들에 대해 자세히 논하기 전 큰 갈래부터 그려보자.

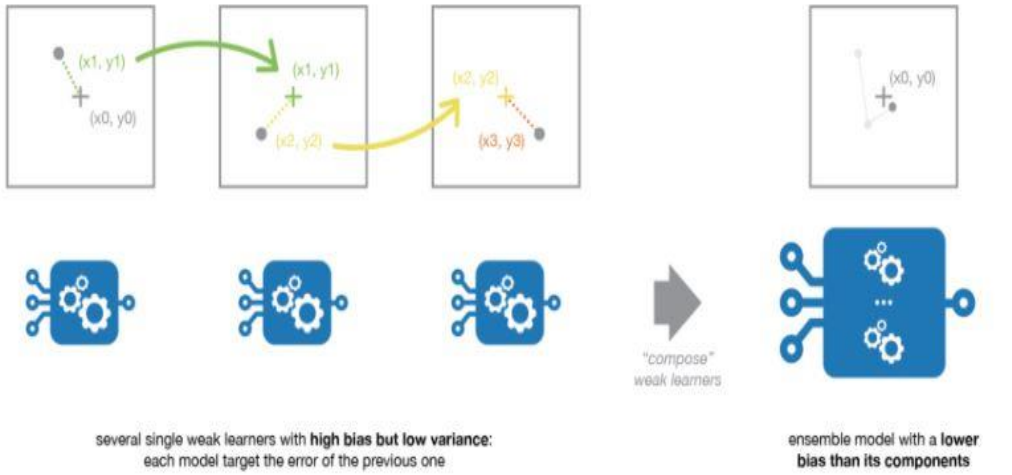


앙상블 기법을 구현하는데 있어 가장 먼저 이루어져야 할 작업은 당연하게도 어떤 회귀/분류기(estimator)를 사용할지 선택하는 것이다. 대부분의 경우에서는 한 종류의 estimator 를 여러 개 결합하는 homogeneous ensemble 의 전략을 취하지만, 동일한 데이터셋에 대해 서로 다른 종류의 estimator 를 결합하는 heterogeneous ensemble 의 전략도 제시되곤 한다. 앙상블하기 위한 estimator 를 선정하는 데 있어 주의해야 할 점이 있다면, low bias ~ high variance 의 모델은 동일하게 low bias ~ high variance 의 특성을 지닌 모델들과 결합되어야 한다는 것이고 low variance ~ high bias 의 모델은 마찬가지로 low variance ~ high bias 의 특성을 보이는 모델들과 결합이 되어야 한다는 것이다. 전자의 경우에는 모델의 분산을 크게 줄여 더욱 강건한 모델을 설계함으로써 성능 개선을 꾀하고자 하는 것이고, 높은 bias 를 보이는 후자의 경우에는 이 bias 를 줄이는 방향으로 작업이 이루어져 성능을 올리고자 하는 것이기 때문이다.



LOW BIAS HIGH VARIANCE WEAK LEARNERS

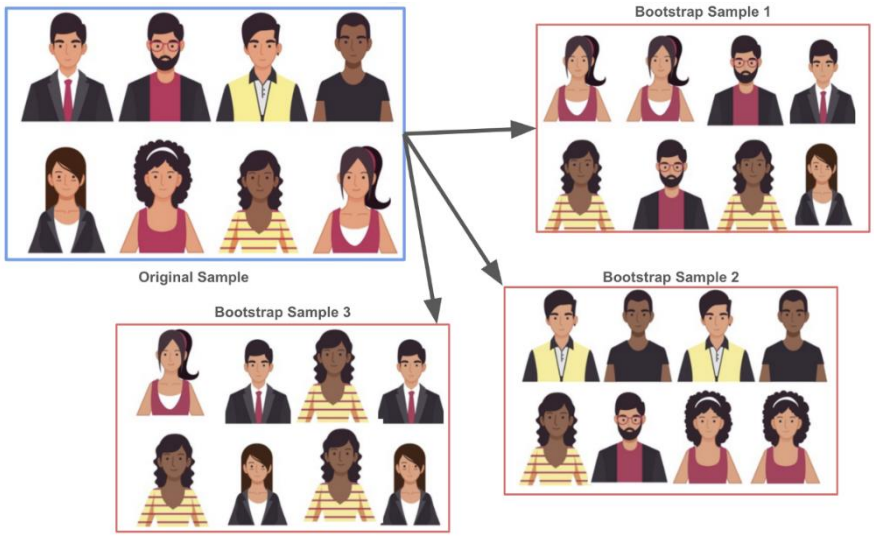
LOW VARIANCE HIGH BIAS WEAK LEARNERS



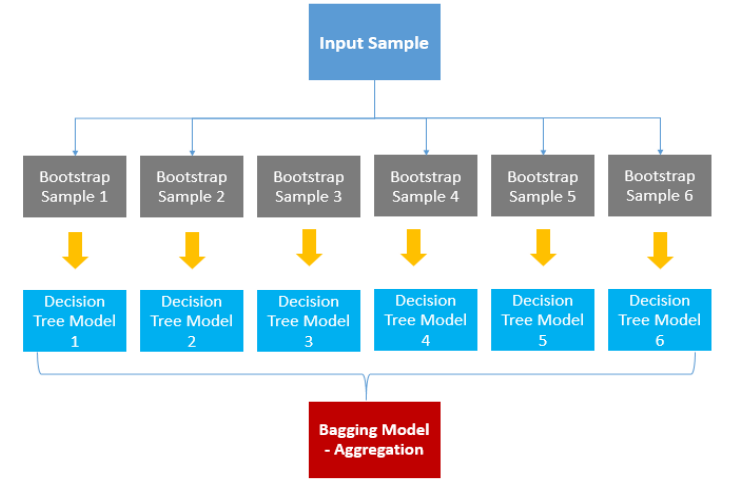
A. Bagging

먼저, 배깅의 개념부터 살펴보자(가방이랑 아무 상관없다). **Bagging** 은 Bootstrap+Aggregating 의 의미로, 부트스트랩 방법을 통해 샘플링한 표본들을 바탕으로 모델링을 실시하는 기법이다.

부트스트랩 방법은 무엇인가? 아래 그림을 통해 그 원리를 이해해보자.

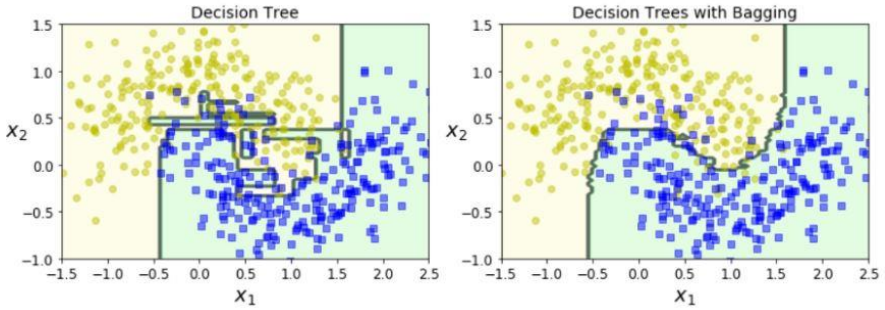


위와 같이 총 8 명의 사람들에 대한 정보가 담긴 데이터셋으로부터, 동일한 표본 개수를 포함하는 데이터셋을 여러 개 만들 수 있다. 확인할 수 있듯, 하나의 샘플 데이터셋에 같은 사람이 두 번 들어가는 복원 추출(sampling with replacement)이 취해졌다. 이렇게 하는 이유는 데이터셋이 다르게 구성됨에 따라 이들 데이터를 모델에 적합 시켰을 때 발생할 수 있는 모델 간 variance 를 최소화하고자 함이다. 우리의 맥락에서 보았을 때 생성된 세 개의 부트스트랩 샘플 각각으로 트리 모델을 적합, 이들의 평균(또는 최빈값)을 최종 예측값으로 생각하는 것이 트리 모델에서의 배깅이다.



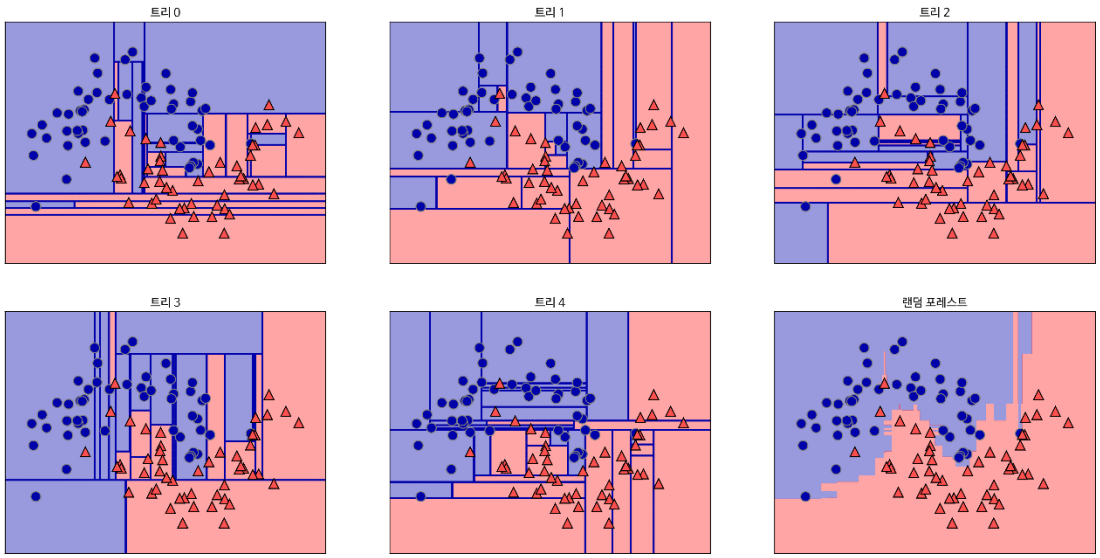
배깅을 통해 의사결정나무를 적합한 경우 아래 그림에서와 같이 더 작은 variance 를 지니게 된다. 평균 또는 최빈값을 구하는 과정을 'average out', 또는 'voting'의 개념으로 이해할 수 있는데, 여기에 iid 확률분포의

개념을 차용하여 설명해볼 수 있다. 모집단의 관측치 N 개에서 단일한 표본만을 추출한다면 이때의 variance 는 σ^2이지만, 여러 개의 표본을 추출할 경우에는 사실상 표본평균의 variance 인 $\frac{\sigma^2}{N}$ 를 가지는 것과 동일하기에 배깅을 취한 경우 모델의 variance 가 줄어든다고 말할 수 있는 것이다.



1) Random Forest

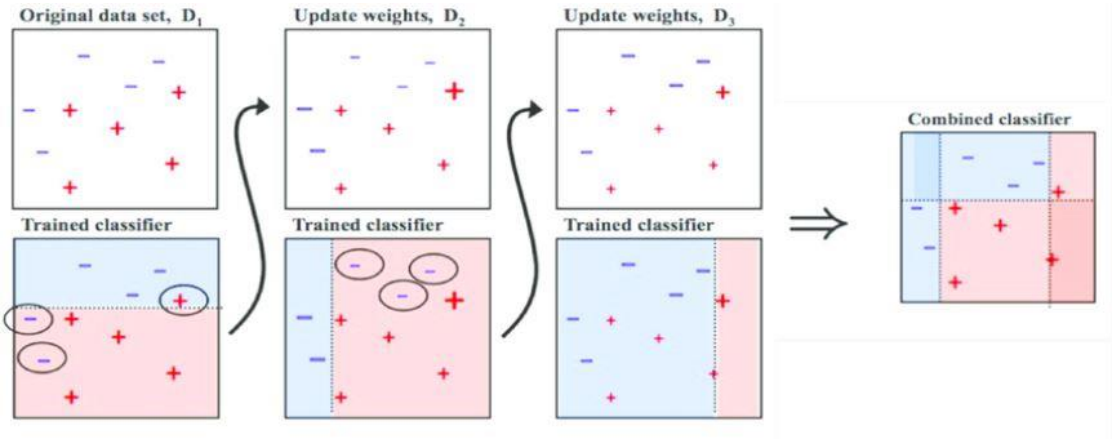
기존의 데이터셋을 pseudo population 으로 두고 다시금 샘플링을 시도하는 이 방법론은 기발하지만, 부트스트랩 샘플들이 서로 너무 비슷하여 그 상관관계가 매우 높게 형성된다는 점을 치명적인 단점으로 지닌다. 이를 극복하는 방법으로, 매 모델링마다 사용할 독립변수를 임의로 선택하여 트리를 적합시키는 **랜덤 포레스트(random forest)**가 제시된다.



여러 개의 트리 모델들을 종합적으로 고려하여 하나의 숲(forest)을 이루는 방식인 random forest 는 앞서 제시된 트리 간 상관관계 문제를 효과적으로 해결한다. 단순히 배깅만을 사용할 경우 대부분 혹은 모든 트리들이 아주 결정적인 설명변수를 상단의 분할에서 사용하기 때문에, 생성되는 트리들 서로 유사하다. 하지만, 모델 적합에 사용되는 변수의 종류를 다양하게 구성한다면 이와 같은 문제점이 발생할 확률이 낮아져 더욱 다양한 트리들을 많이 만들어낼 수 있는 것이다. 정리하자면, 부트스트랩 샘플들에 대해 배깅을 실시한 후 트리를 적합시키는 단계에서 일부 X변수를 사용하는 과정을 통해 decorrelation을 달성하는 flow라고 이해하면 될 것 같다.

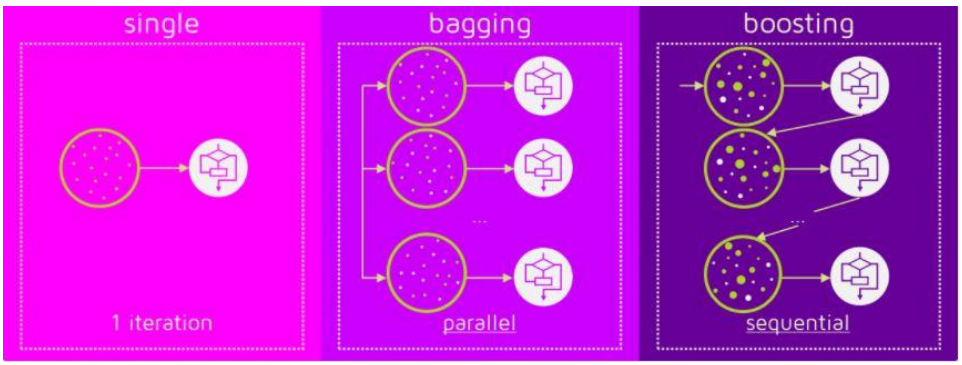
B. Boosting

배깅과 마찬가지로 **부스팅** 기법 역시 여러 개의 약한 estimator 를 강한 estimator 로 만들어주는 목표를 지니고 있으나 그 방법에 있어서는 조금 차이를 보인다. 배깅에서는 여러 개의 estimator 적합이 독립적으로 전개된 방면, 부스팅은 이전 단계에서의 estimator 적합에서 예측한 결과에 따라 다음 단계에서 가중치를 부여, 모델을 적합시키는 방식을 취한다. 한마디로 모델 간 팀워크가 이루어진다는 것인데 그림을 통해 직관적으로 파악해보자.



파란색의 -와 빨간색의 +로 구성된 데이터셋을 분류하는 문제에 부스팅 기법을 적용하면 위와 같은 흐름으로 전개된다. D1 에서 잘못 분류된 +와 두 개의 -는 다음 단계인 D2 에서 더욱 큰 가중치를 지니게 된 것을 볼 수 있다. 마찬가지로 D2 에서 잘못 분류된 세 개의 - 역시 다음 단계에서 큰 값으로 가중치를 지니게 된다. 최종 분류기는 D1, D2, D3 의 분류기를 모두 합친 형태로 생성되며 명확하게 +와 -를 분류해준다.

더 나아가기 전에 배깅과 부스팅의 차이에 대해 정리해보자. 배깅은 병렬적인 학습이 이루어지는 반면, 부스팅은 순차적인 과정을 밟으며 가중치의 개념을 도입하여 모델을 적합시킨다. 부스팅은 배깅에 비해 성능이 좋게 나타나곤 하지만, 속도가 느리고 오버 피팅이 될 가능성이 있다. 그렇기 때문에 상황에 맞추어 배깅과 부스팅 중 더 적절한 방법을 취하는 것이 바람직하다.



약분류기들로부터 강분류기를 만들 때 필요한 가중치들을 한 번에 빡! 찾는 최적화 문제(optimization problem)를 해결하기란 어려운 일이기에, 각 iteration 마다 최적의 가중치를 찾아내는 방법을 고안해낸 것이다. 1996 년 제시된, 부스팅 계열의 시초인 **Adaboost** 는 Adaptive Boosting 의 줄임말로 아래와 같은 표현으로 요약될 수 있다.

$$Difficult\ to\ optimize\ problem\ when\ its\ algorithm\ written\ as:$$

$$s_L = \sum_{l=1}^{L} c_l \times w_l \quad where\ c_l s\ \ are\ coefficients\ and\ \ w_l s\ are\ weak\ learners$$

$$Instead\ take\ approach:$$

$$s_l = s_{l-1} + c_l \times w_l$$

$$that\ satisfies\ the\ following:$$

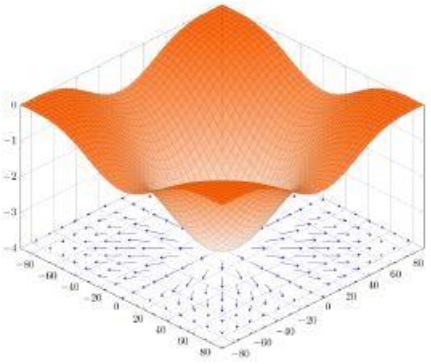
$$(c_l, w_l) = argminE(s_{l-1} + c_l \times w_l)$$

한편, Adaboost 방법은 또 그 나름대로 오래 되었던 지라 이후 다양한 부스팅 계열 알고리즘들이 등장해 더 좋은 성능을 보이고 있다. 이중 GBM 계열 모델들을 통해 기본적인 원리를 파악해보자.

1) LGBM

LGBM 은 Light GBM 의 약자로서, 2017 년 Microsoft 에서 소개한 알고리즘이다. 여기서 **GBM** 은 Gradient Boosting Machine 인데, 단어 그 자체에서 볼 수 있듯 경사하강법을 이용한 최적화문제(Gradient Descent Optimization) 해결 방식을 취한다. 이게 무슨 말인지 먼저 그림을 통해 확인해보자.

경사하강법 개념을 설명할 때 정말 많이 등장하는 비유다. 등산을 마친 후 집으로 돌아가기 위해 아래 방향으로 차츰차츰 내려오듯, 우리가 모델로서 산출해낸 예측값과 실제 값의 차이(잔차)를 손실함수(cost function 또는 loss function)로서 표현했을 때 오차가 가장 낮은 지점에 다다르는 것을 목표로 삼는다는 의미이다.

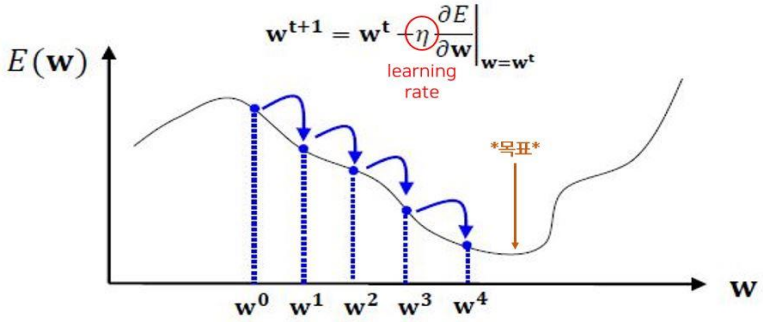


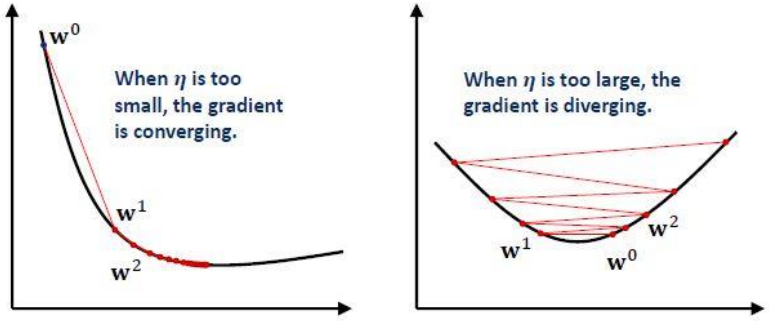
이를 직관적으로 이해할 수 있도록 먼저 기하학적인 접근을 취해보자. 손실함수는, 우리가 흔히 아는 2 차 곡선의 형태와 같이 취해질 수 있는데 이 경우 convex 하다고 말한다. 아래로 볼록한 형태의 함수를 공간에 위치시켜보면 다음과 같이 표현된다.

이를 수학적으로도 설명해볼 수 있겠다. Gradient 란 어떤 점에 대한 편미분으로 정의된다. 수식으로는 오른쪽과 같이 표현된다.

$$\nabla f(\mathbf{w}) = \nabla f(w_1, w_2, \dots, w_m) = \begin{bmatrix} \frac{\partial f(\mathbf{w})}{\partial w_1} \\ \frac{\partial f(\mathbf{w})}{\partial w_2} \\ \vdots \\ \frac{\partial f(\mathbf{w})}{\partial w_m} \end{bmatrix}$$

결론부터 말하자면 우리는 negative gradient 를 구해야 하는데, 앞서 말했듯 우리는 손실함수의 값이 점점 작아지다가 마지막에는 가장 작은 값에 도달하는 목표를 갖고 있기 때문이다. 여기서 learning rate, 학습률이라는 개념이 제시가 되는데 이 부분은 딥러닝 팀에서 자세하게 설명해줄 내용으로 가볍게만 짚고 넘어간다. 경사하강법 관점을 취하자면 이는 step size 와 동일한 개념인데, learning rate 를 너무 작게 설정하면 지역 최솟값(local minima)에 빠질 우려가 있으며 너무 크게 설정하면 수렴하지 못하고 overshoot 할 수 있다.





잔차를 줄여가는 방향으로 모델을 적합시킨다는 아이디어를 유념하여 다음의 예시를 살펴보자. 사람들의 키와 그들이 좋아하는 색깔, 그리고 성별을 설명변수로 두었을 때 몸무게를 어떻게 예측할 수 있을까?

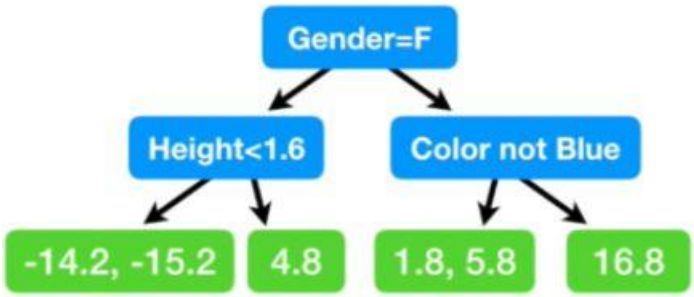
1. 모든 관측치의 종속변수(weight)에 대해 평균을 구한다.

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

2. 1 에서 구한 평균값과 종속변수 값과의 차이를 pseudo-residual 로써 표현한다. Pseudo-residual 이라고 부르는 이유는, 최종적으로 결정된 residual 값이 아니라 각 단계마다 새롭게 계산이 되는 것이기 때문이다. 모델을 적합시켜 감에 따라 이 pseudo-residual 값이 어떻게 변하는지 확인해보자. 우리는 종속변수 값 그 자체를 예측하는 것이 아니라, 사실상 residual 을 예측하는 것과 같은 과정을 따른다.

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

3. 구한 pseudo-residual을 트리의 terminal node에 위치하도록 모델을 적합 시키고, 마지막 node에 두 개의 residual 값이 있는 경우 이들 값을 평균 내어 넣어주자.

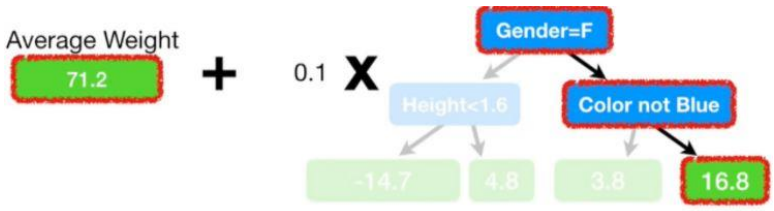


Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

이 과정까지 실행 후, 첫번째 관측치의 종속변수 값을 예측해본다면 88(71.2+16.8)로 완벽히 일치하는 것을 확인할 수 있다. 우리가 만들어낸 모델의 성능이 매우 우수하다고 생각할 수 있지만, 사실 여기에는 함정이 있다. 훈련 데이터에만 너무 fit한 모델이기 때문에 오버피팅 되었다고 보는 것이 더욱 적절하다.

바로 이 지점에서 앞서 가볍게 살펴본 learning rate라는 개념이 등장한다. Learning rate은 0부터 1 사이의 값을 가지는데, 마치 가중치와 같은 역할을 수행하며 오버피팅을 방지해준다.

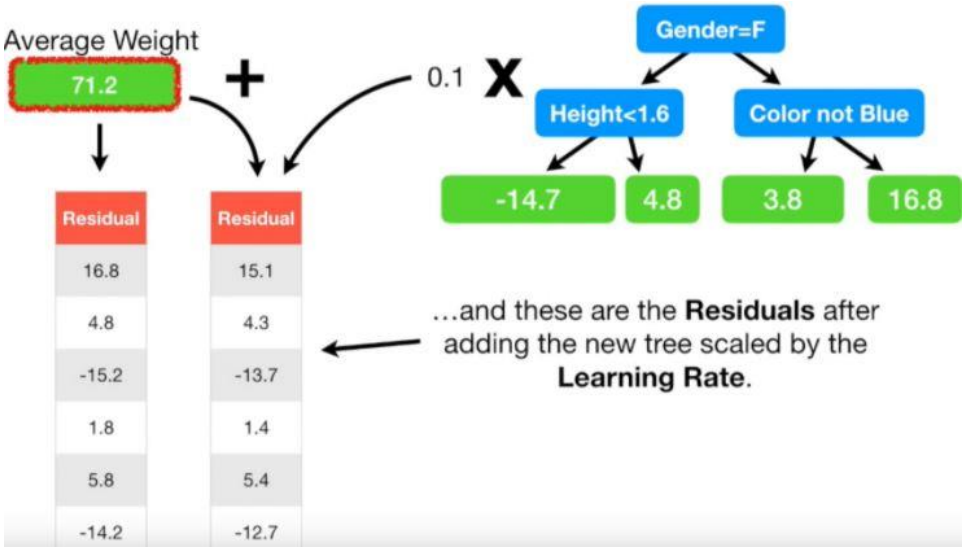
Learning rate을 0.1로 두었을 때의 모습은 다음과 같다.



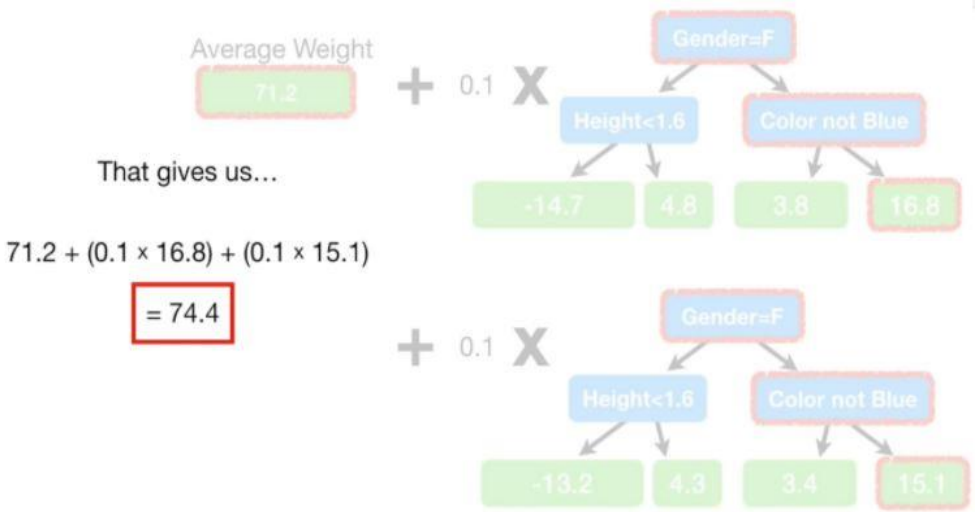
Now the **Predicted Weight** = 71.2 + (0.1 × 16.8) = 72.9

학습률을 고려하여 예측한 값은 72.9로, 처음 예측해낸 평균값보다 실제 종속변수 값 88에 더 가까워졌음을 알 수 있다.

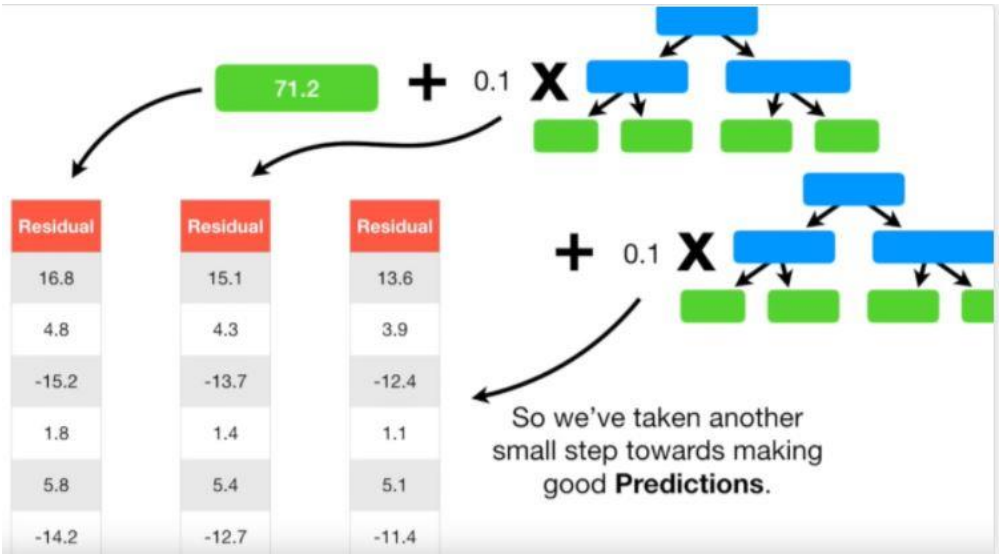
4. 3에서 예측한 값을 토대로 pseudo-residual을 다시 계산해준다. 1단계에서 구한 평균값 71.2에 대한 pseudo-residual값보다 더욱 줄었음을 확인할 수 있다. 이제 GBM이 어떻게 점차 residual을 줄여 나가는지 파악할 수 있겠지?



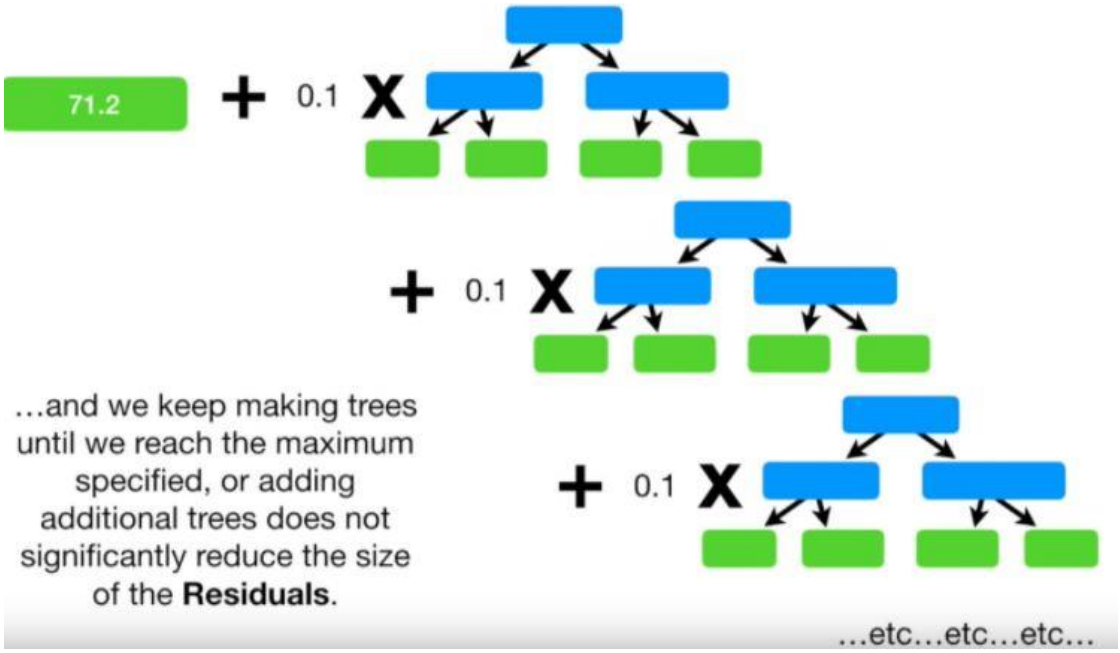
5. 4에서의 값을 바탕으로 새로운 트리 모델을 적합시킨다. 3에서와 마찬가지로 terminal node에 단일한 값이 위치할 수 있도록 평균을 구해주고, 학습률 0.1을 적용한다. 초기 예측값 71.2에 그 다음 스텝의 모델의 값, 그리고 이 단계에서 구한 모델의 값을 전부 고려하여 예측값을 계산하면 74.4가 나온다. 실제 값인 88과 점점 더 가까워지고 있다.



마찬가지로 residual 값도 줄어들었다.

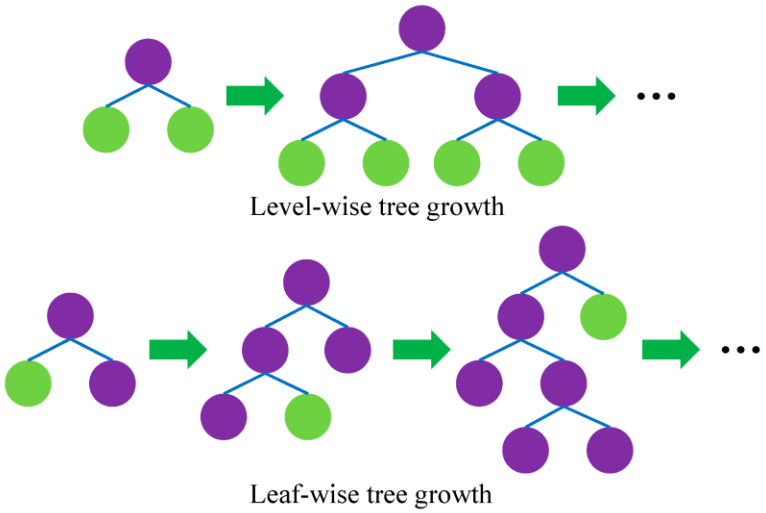


6. 잔차가 더 이상 작아지지 않을 때까지, 또는 사용자가 하이퍼 파라미터로 지정해 놓은 iteration 횟수에 도달할 때까지 위 과정을 반복한다. 이를 통해 잔차를 점차 줄이는 방향으로 이동하며, 앞선 모델의 결과로부터 더하여 순차적으로 전개됨을 확인할 수 있다.

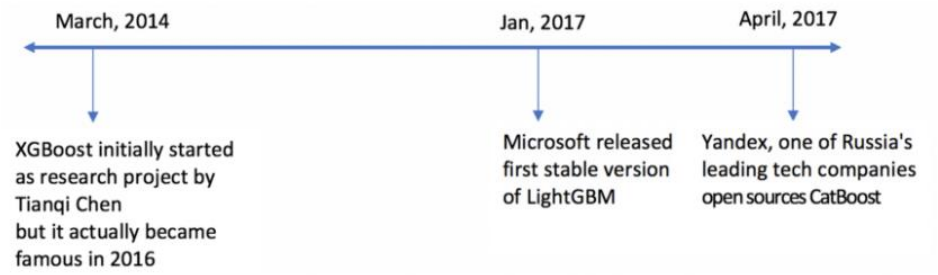


지금까지 이야기한 GBM 모델을 개선시킨 것이 Light GBM, 그러니까 LGBM이 될 텐데 'light'하다는 말 뜻 그대로 다른 모델들에 비해 더욱 빨리 작동한다는 이점이 있다. 큰 사이즈의 데이터에 대해서도 잘 작동하며, 그러면서도 메모리는 많이 차지하지 않아 많은 인기를 끌고 있는 알고리즘이다. Microsoft사에서 2016년 소개한 이 모델은 Kaggle Competition에서 가장 많이 우승한 알고리즘이기도 하다!

LGBM의 주요 특성이라 하면 우선 GOSS(Gradient Based One Side Sampling) 방법을 취한다는 것인데 큰 gradient, 그러니까 큰 error를 보이는 관측치들에 대해서만 이들 error를 줄이는 데에 집중한다고 볼 수 있다. 깊이 우선 방식인 leaf-wise 방식을 취하는 것과도 일맥상통하는데, LGBM은 균형적으로 트리를 분할(너비 우선 방식, level-wise 방식)하는 것이 아니기 때문에 비대칭적으로 트리가 아주 깊어지게 된다. 이러한 기법을 통해 예측 오류를 최소화하고자 하는 것이 LGBM의 목표라 할 수 있겠다.

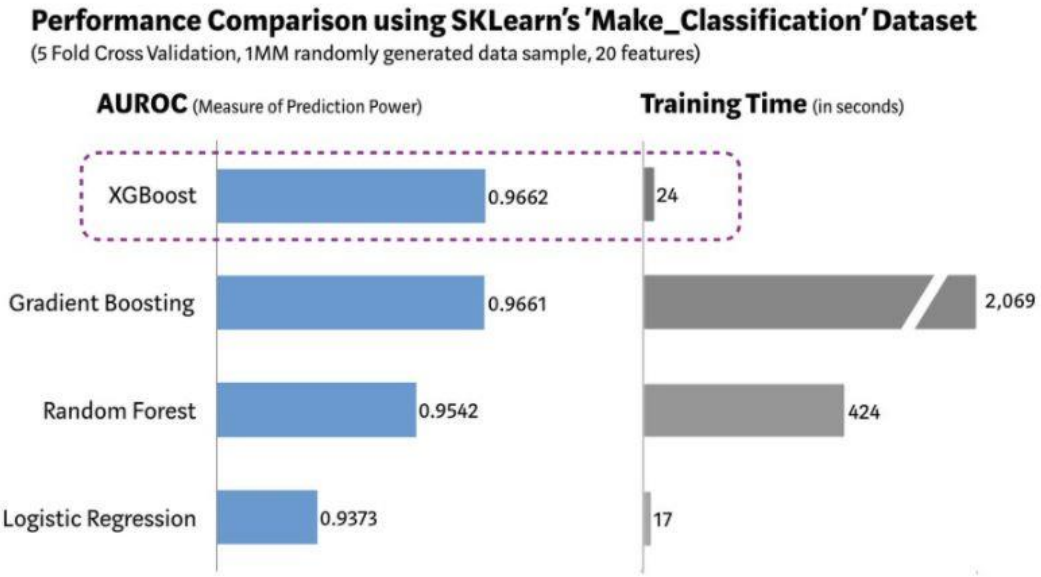


2) XGBoost



GBM Killer라고도 부르는 이 알고리즘은 LGBM과 함께 Kaggle Competition의 최강자 쌍두마차를 이루며 좋은 성능을 보이곤 한다. 간단하게 말하자면 GBM 베이스라인에 오버피팅 방지를 위한 regularization term을 추가한 것으로 회귀와 분류 문제 모두를 정확하게 수행하는 알고리즘이라 할 수 있겠다. 이외에도 다른 이점들이 여러 개 있는데, 다음 그림을 통해 파악해보자.

마지막으로, 동일한 데이터셋에 대해 여러 종류의 모델들을 적합시켰을 때의 그 성능과 속도를 비교해보고자 한다. 아래 그림을 통해 오늘 우리가 학습한 내용을 복습해보자.



3. Appendix

*Decision Tree Regressor 에서 terminal node 의 값으로 왜 종속변수들의 평균값을 취하는지 수학적으로 증명.

$$y_i = c_i + \varepsilon_i$$

$$min_{c_m} RSS = min_{c_m} \sum_{i=1}^{N} \{y_i - f(x_i)\}^2 \, , let\ L\ min_{c_m} \sum_{i=1}^{N} \{y_i - f(x_i)\}^2\ be\ the\ objective\ function$$

$$\frac{dL}{dc_m} = -2 \sum_{i=1}^{N} y_i - c_m = 0$$

$$\equiv \sum_{i=1}^{N} y_i - c_m = \sum y_i - N c_m = 0$$

$$\therefore \widehat{c_m} = \overline{y_i}$$

실습 타임

간단한 실습을 해보겠습니당! 학습 목표는 아래와 같습니당!

1. Python (데이터셋은 sklearn의 load_breast_cancer 데이터를 사용합니다)

   -sklearn에서 기본적으로 제공하는 데이터셋들을 어떻게 불러올 수 있는지 알아보겠습니다!

   -XGBclassifier를 이용하여 분류 문제를 해결해보겠습니다!

   -미니 숙제 나가요~~

   1) XGBClassifier()에서 objective 파라미터가 의미하는 바는?

   2) 기존의 XGBClassifier에서 정의된 파라미터 값 1개 변경, 2개 새롭게 추가하여 성능 비교 해보기

   3) 지난 시간 다룬 RandomizedSearchCV와 합쳐볼까요? 코드 짜오기~

(R에서 모델링하는 내용은 패키지로 나갈 테니, 해보고 어려운 점 있으면 질문!)