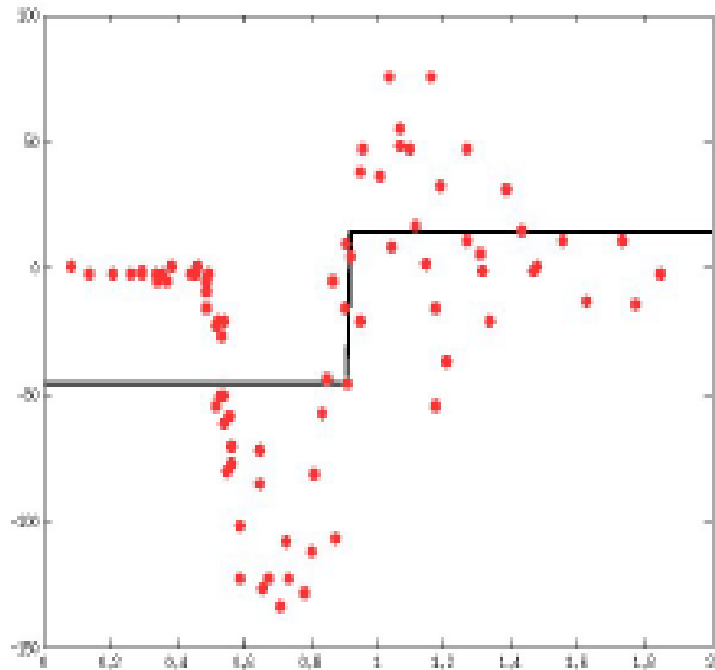


Gradient Boosting

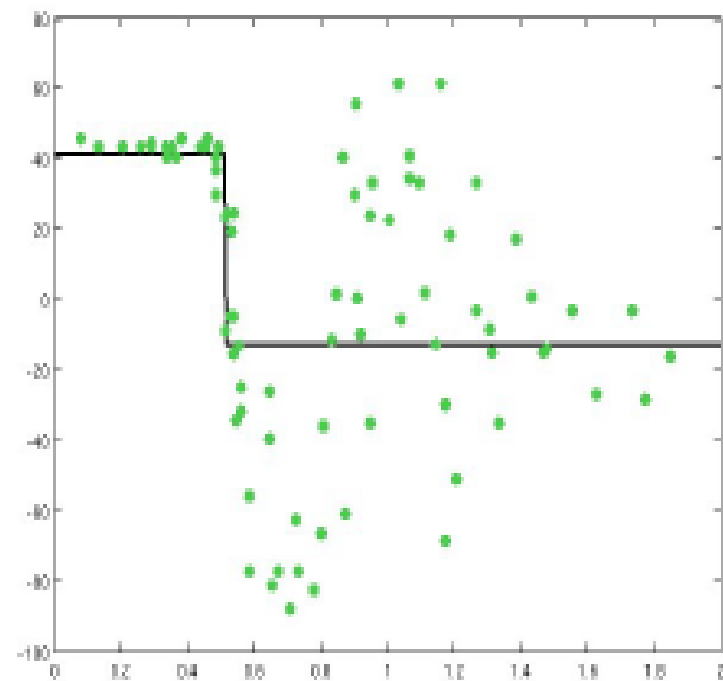
Gradient Boosting

- Gradient Boosting 리뷰

Learn a simple predictor...



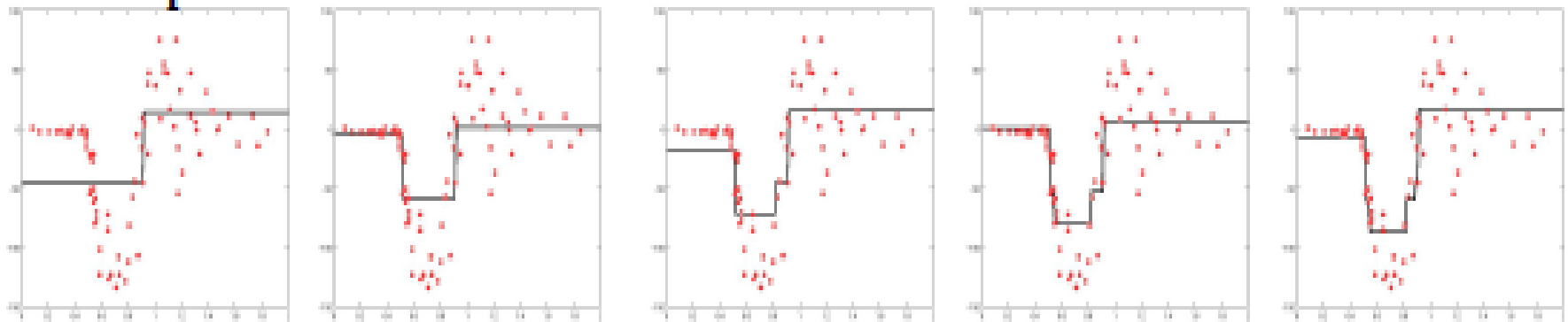
Then try to correct its errors



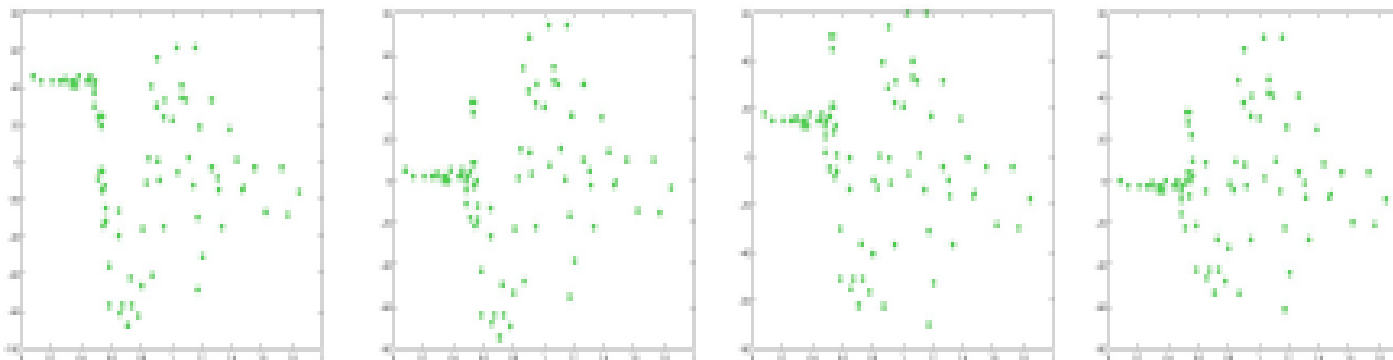
Gradient Boosting

- Gradient Boosting

Data & prediction function



Error residual



그래디언트 부스팅 해설

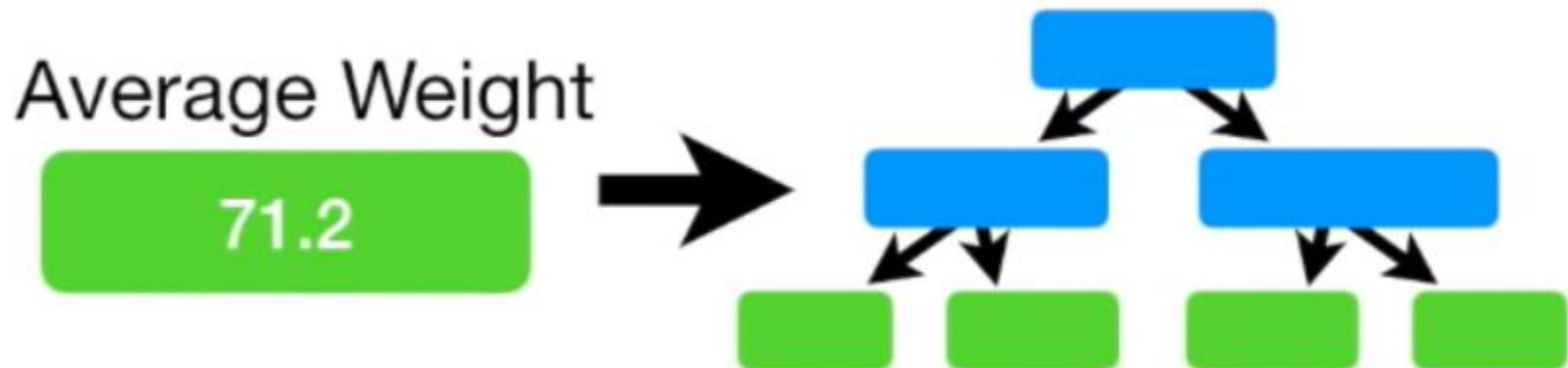
그래디언트 부스팅

- 그래디언트 부스트 모델 예제

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

그래디언트 부스팅

- 그래디언트 부스트 모델 예제: 싱글 리프로 출발
 - 몸무게를 평균으로 추정: $(88 + 76 + 56 + 73 + 77 + 57) / 6 = 71.2$



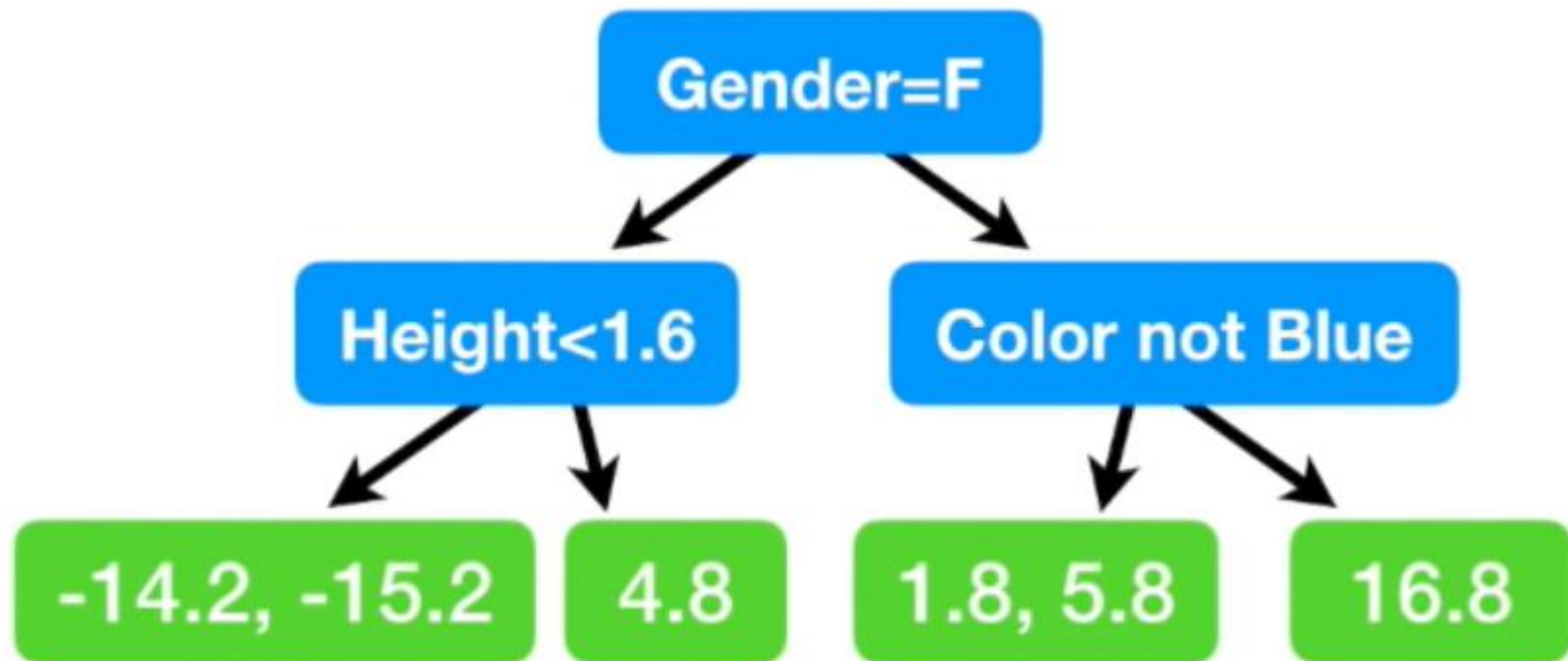
그래디언트 부스팅

- 잔여오차(Residual)

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

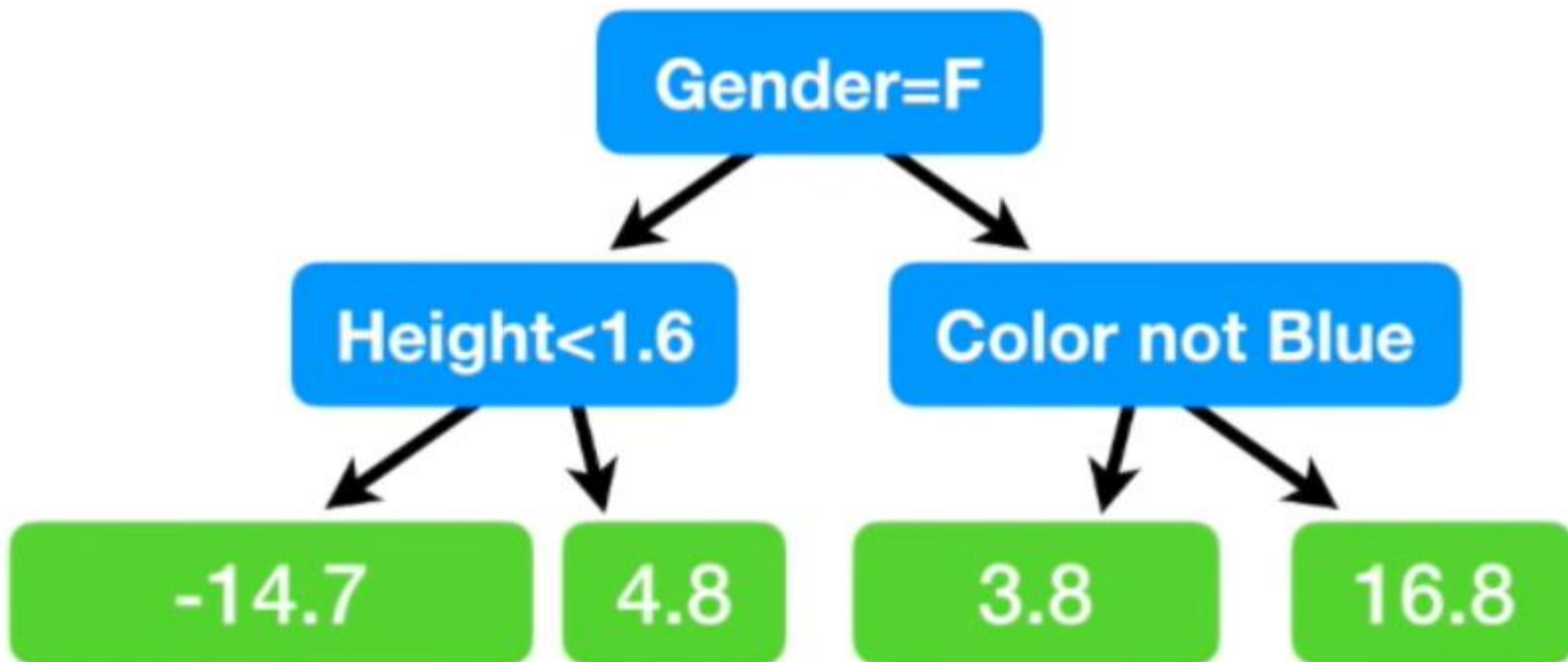
그래디언트 부스팅

- 키, 좋아하는 색깔, 성별을 통해 잔여오차(Residual)를 예측하는 트리



그래디언트 부스팅

- 여기서 마지막 leaf 노드에 두개의 residual 값이 있는 경우가 있다. 그럴 땐 두 값의 평균으로 치환해 넣어주면 된다. $\{(-14.2) + (-15.2)\} / 2 = -14.7$ 이므로 -14.7을 첫번째 leaf 노드에 넣습니다. 세번째 leaf 노드도 마찬가지로 평균 값을 넣어준다.



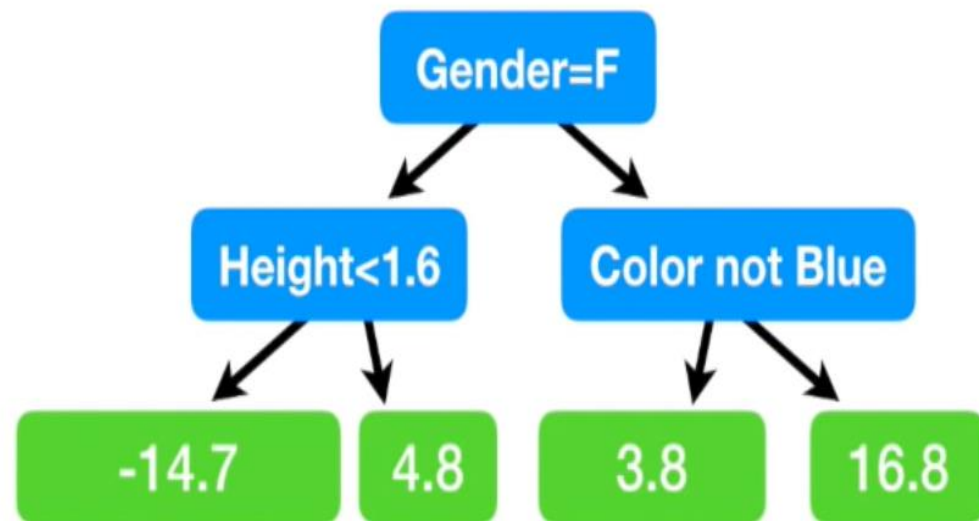
그래디언트 부스팅

- 이제, 초기에 구한 트리(여기서는 single leaf)와 두번째로 구한 트리를 조합하여 몸무게를 예측해보자.

Average Weight

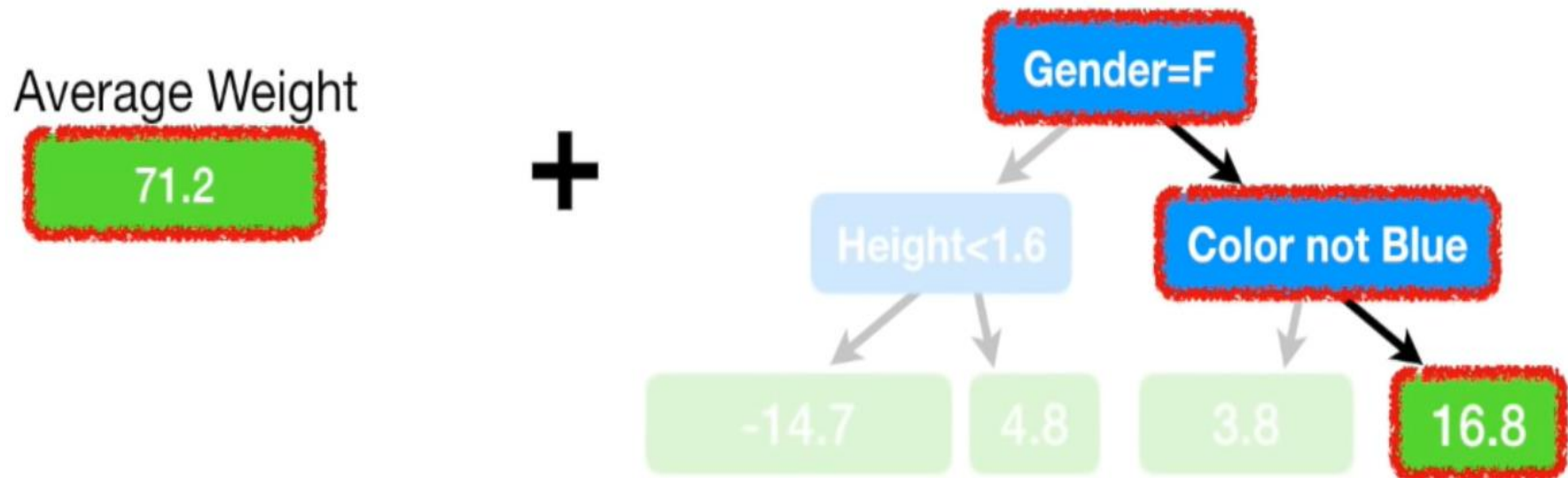
71.2

+



그래디언트 부스팅

- 두번째 트리에서 성별이 남자고, 좋아하는 색깔이 파란색이면 residual을 16.8로 예측한다.



...so the **Predicted Weight** = $71.2 + 16.8 = 88$

그래디언트 부스팅

- 첫번째 트리의 몸무게 예측 값인 71.2kg와 두번째 트리의 residual 예측 값인 16.8kg을 더하면 88kg이 된다. 즉, 성별이 남자고, 좋아하는 색깔이 파란색이면 그 사람의 몸무게는 88kg라고 예측을 한 것이다.

$$\text{Predicted Weight} = 71.2 + 16.8 = 88$$

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

...which is the same as the **Observed Weight**.

- 과적합

우리가 만들어준 모델이 예측한 몸무게와 실제 몸무게가 88kg으로 일치한다. 놀라운가? 놀라운 일이 아니다. 이는 훈련 데이터에 너무 과적합화된 모델이다.

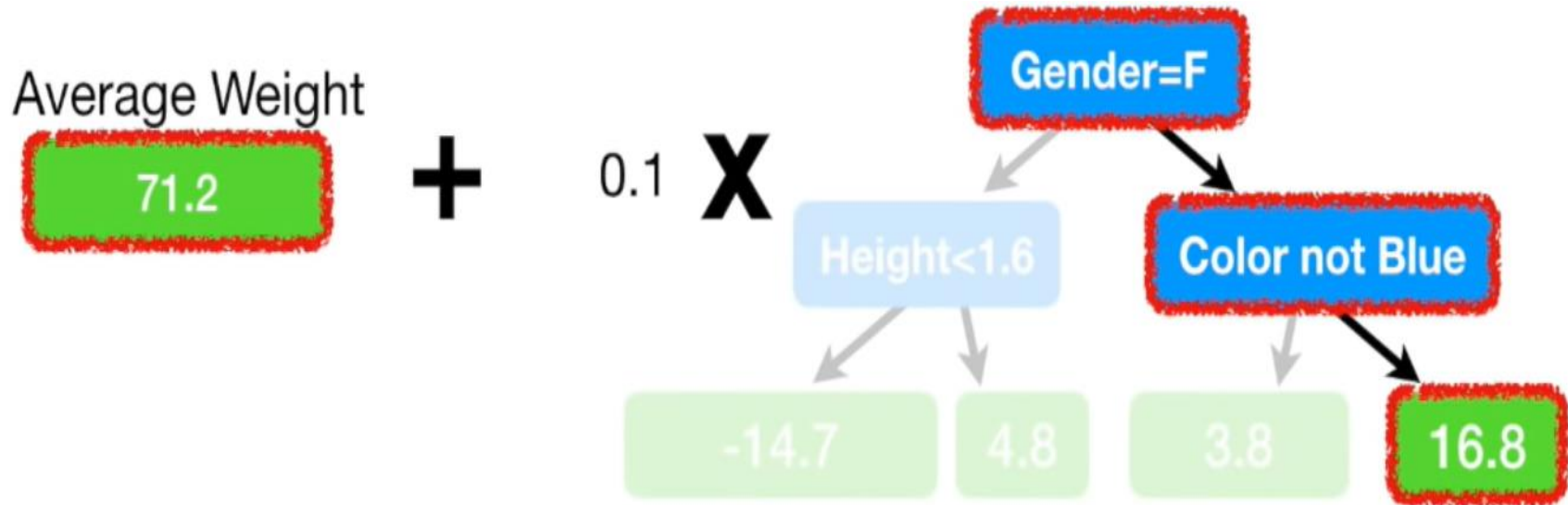
그래디언트 부스팅

- 과적합 문제를 해결하기 위해 학습률(Learning Rate)이라는 것을 활용한다. 학습률은 0부터 1사이의 값을 가진다. Residual을 예측하는 모델에 학습률을 곱해줌으로써 과적합을 해결할 수 있다.



그래디언트 부스팅

- 본 예제에선 학습률을 0.1로 설정해보자.



Now the **Predicted Weight** = $71.2 + (0.1 \times 16.8) = 72.9$

- 학습률이 0.1일 경우 모델이 예측한 몸무게는 72.9kg이다. 첫 single leaf 모델이 예측한 71.2kg보다는 실제 값(88kg)에 더 가까워졌다. Gradient Boost 모델은 이런식으로 실제 값에 조금씩 가까워지는 방향으로 학습을 한다.
- 올바른 방향으로 (즉, 실제 값과 가까워지는 방향으로) 조금씩 다가가는 식으로 학습된 모델은 테스트 데이터에서 좋은 성능을 보여준다.

그래디언트 부스팅

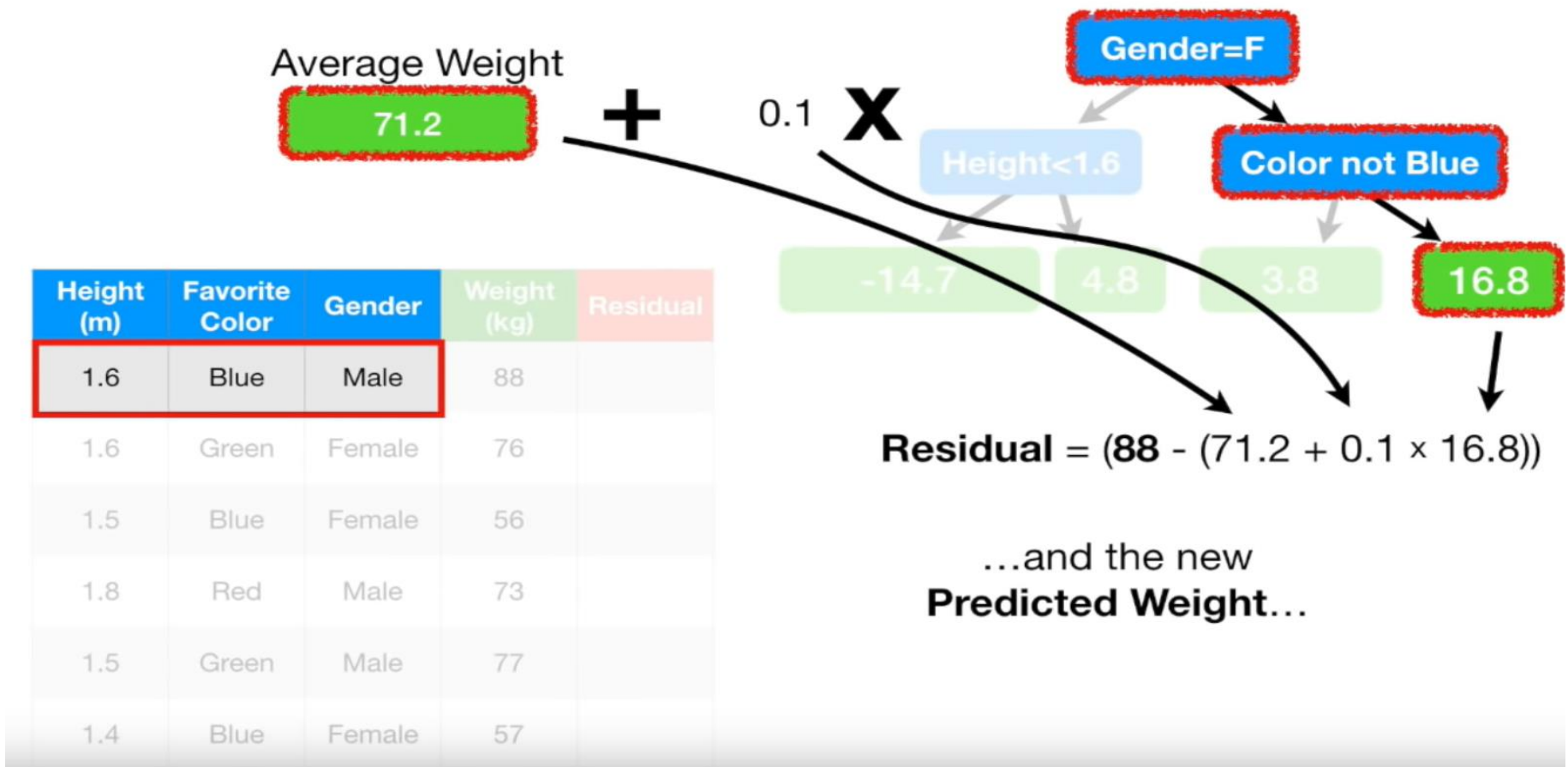
- 올바른 방향으로 한 걸음 더 가보자! 지금까지 만든 모델을 기반으로 잔여오차(Residual)를 다시 구해보자.

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
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	

← **Residual = (Observed - Predicted)**

그래디언트 부스팅

- 앞에서와 같이 잔여오차(Residual)는 (실제값 - 예측값)이다.



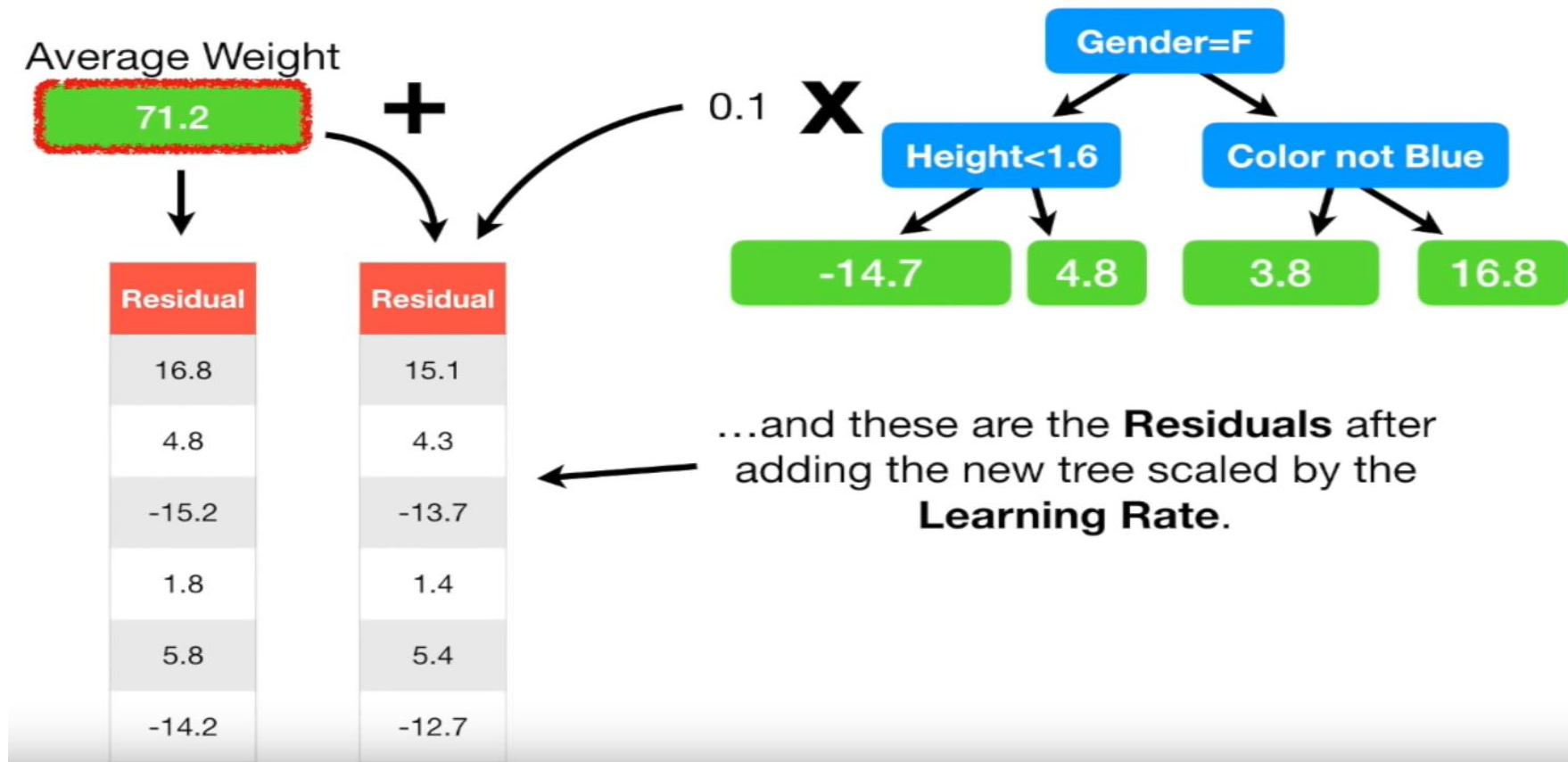
그래디언트 부스팅

- 첫번째 데이터를 예로 들면, 실제값은 88kg이고 새로운 모델의 예측값은 $71.2 + 0.1 * 16.8$ 이다.
 - 따라서 Residual은 $88 - (71.2 + 0.1 * 16.8) = 15.1\text{kg}$ 이다.
 - 새로 구한 Residual을 아래와 같다.

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	4.3
1.5	Blue	Female	56	-13.7
1.8	Red	Male	73	1.4
1.5	Green	Male	77	5.4
1.4	Blue	Female	57	-12.7

그래디언트 부스팅

- 맨 처음 구한 Residual과 결합된 모델로 구한 Residual을 비교해보자.

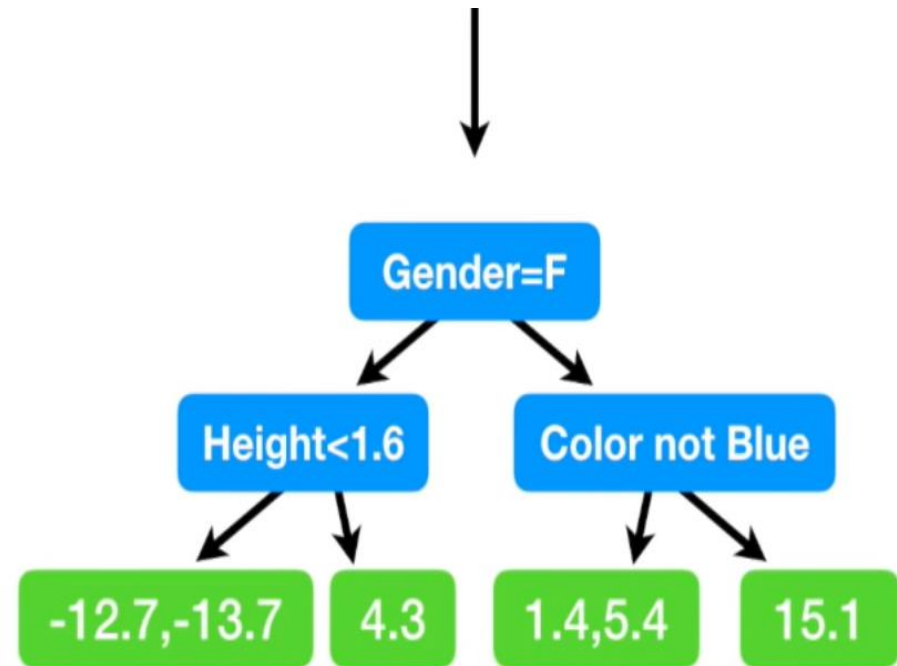


- 왼쪽이 처음 구한 Residual이다. 이는 평균값인 71.2만 활용해서 구한 값이다. 오른쪽이 결합된 모델로 구한 Residual이다. 새로운 Residual은 초기 Residual보다 모두 작은 값이다. Residual이 작아졌다는 뜻은 다른 말로 하면 실제 값과 예측 값의 차이가 작아졌다는 것이다. 즉 조금씩 실제 값으로 다가가고 있다고 할 수 있다.

그래디언트 부스팅

- 한 걸음 더 나아가 보자. 새로 구한 Residual로 트리도 새로 만들어보자.

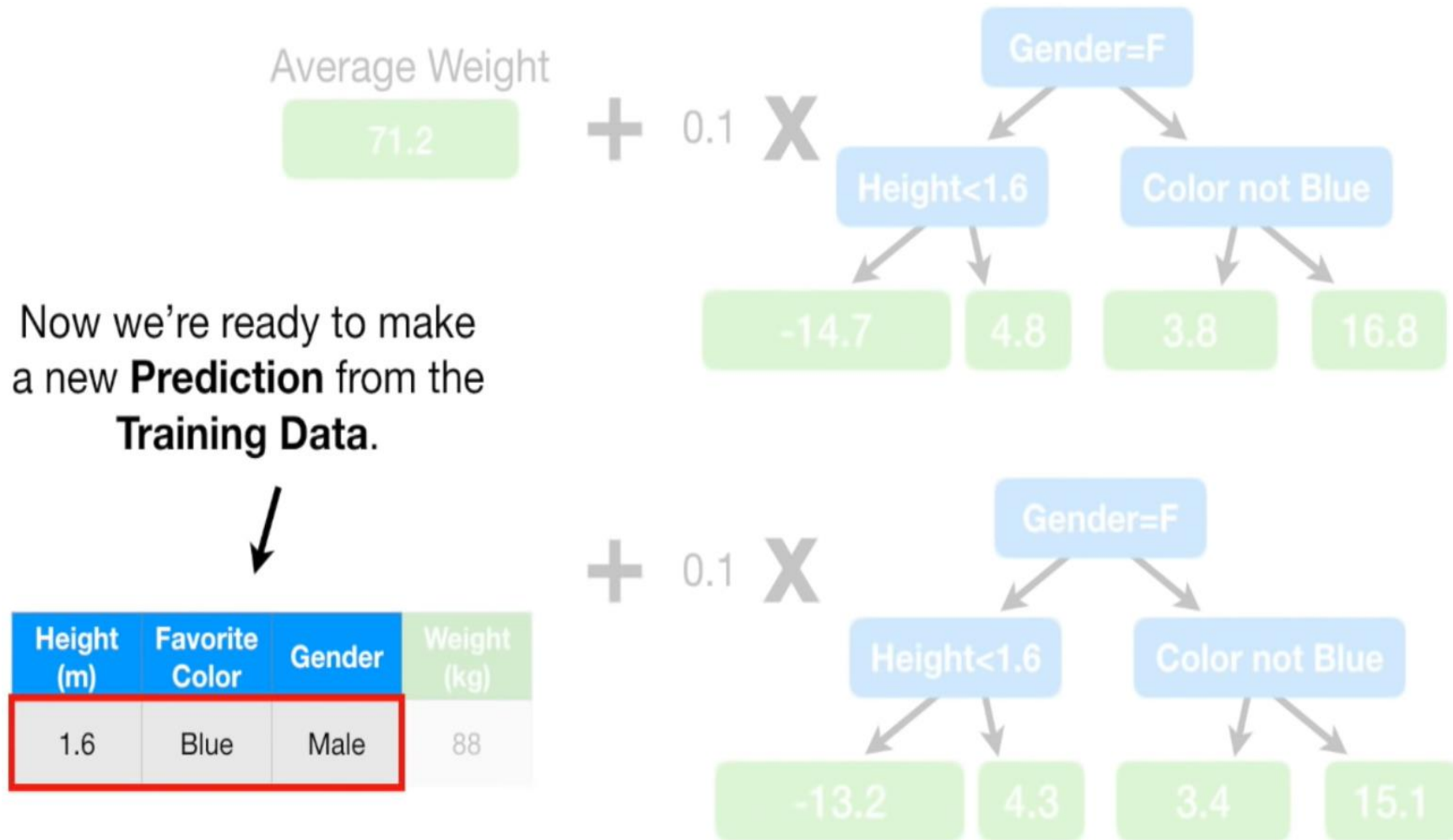
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	4.3
1.5	Blue	Female	56	-13.7
1.8	Red	Male	73	1.4
1.5	Green	Male	77	5.4
1.4	Blue	Female	57	-12.7



- 맨 처음 구한 Residual과 결합된 모델로 구한 Residual을 비교해보자..
- 본 예제에서는 leaf가 4개인 트리를 계속 사용하지만 실제로는 매 iteration마다 트리의 모양이 바뀔 수 있다. 즉, 첫 iteration에서는 leaf가 8개인 트리를 사용했다가, 두 번째 iteration에서는 leaf가 16개인 트리를 사용할 수도 있다. 마찬가지로 두 값이 들어가 있는 첫번째 leaf와 세번째 leaf를 -12.7, -13.7의 평균값인 -13.2로 변경해주고, 1.4, 5.4의 평균값인 3.4로 변경해준다.

그래디언트 부스팅

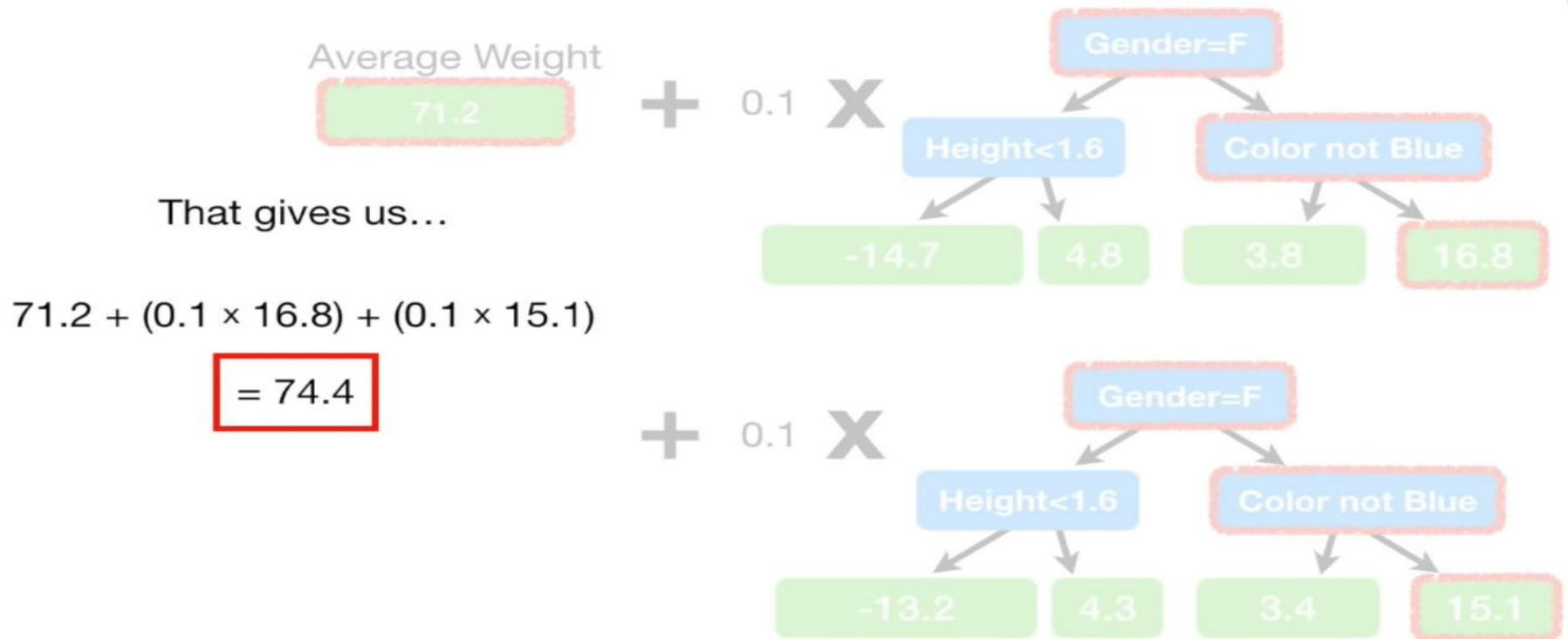
- 이제, 지금까지 구한 트리에 학습률을 곱한 뒤 합한다.



- 즉, ' 초기 leaf 트리+ (학습률 X 첫번째 residual 예측 트리) + (학습률 X 두번째 residual 예측 트리) ' 가 지금까지 구한 모델이다.

그래디언트 부스팅

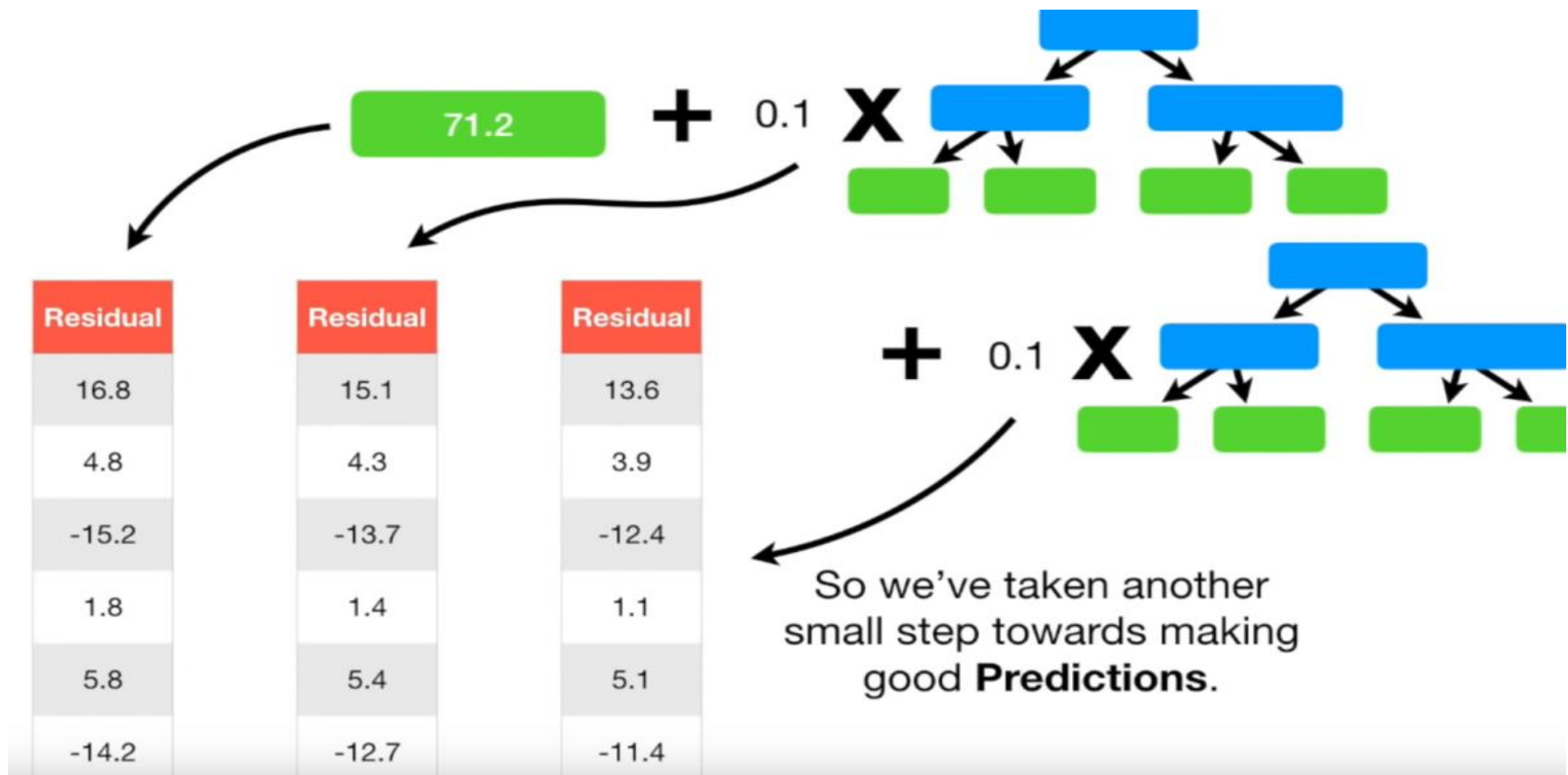
- 지금까지 구한 모델을 통해 키 1.6m에 남자이고, 좋아하는 색깔이 파란색인 사람의 몸무게를 예측해보자.



- 74.4kg로 예측한다. 초기에는 평균 값인 71.2kg로 예측했고, 그 다음 스텝의 모델에선 $71.2 + 0.1 \times 16.8 = 72.9\text{kg}$ 로 예측했고, 현재 모델에선 $71.2 + 0.1 \times 16.8 + 0.1 \times 15.1 = 74.4\text{kg}$ 로 예측한다. 실제 값은 88kg인데 실제 값과 조금씩 가까워지고 있다.

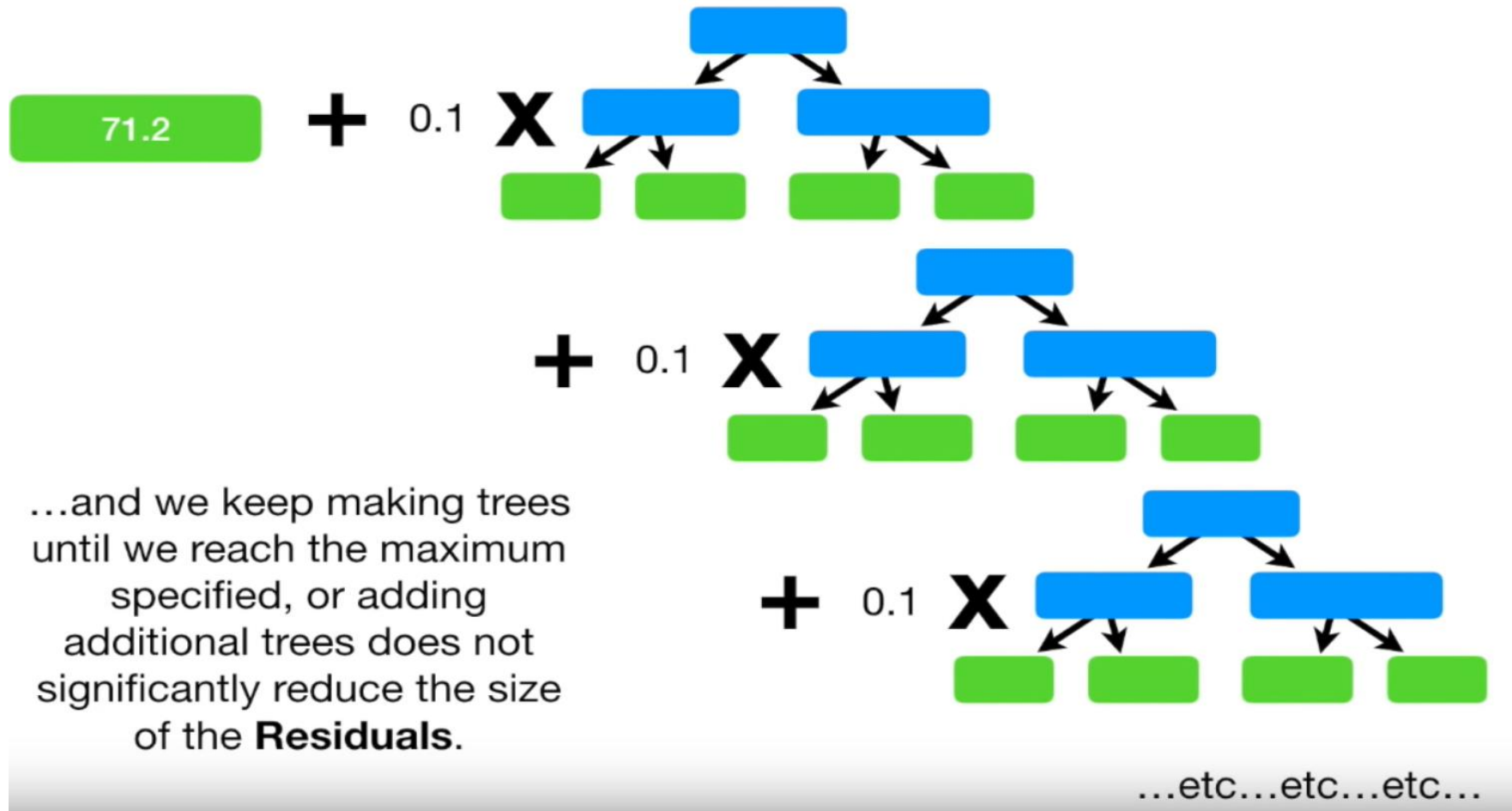
그래디언트 부스팅

- Residual도 구해보면 아래와 같다.



그래디언트 부스팅

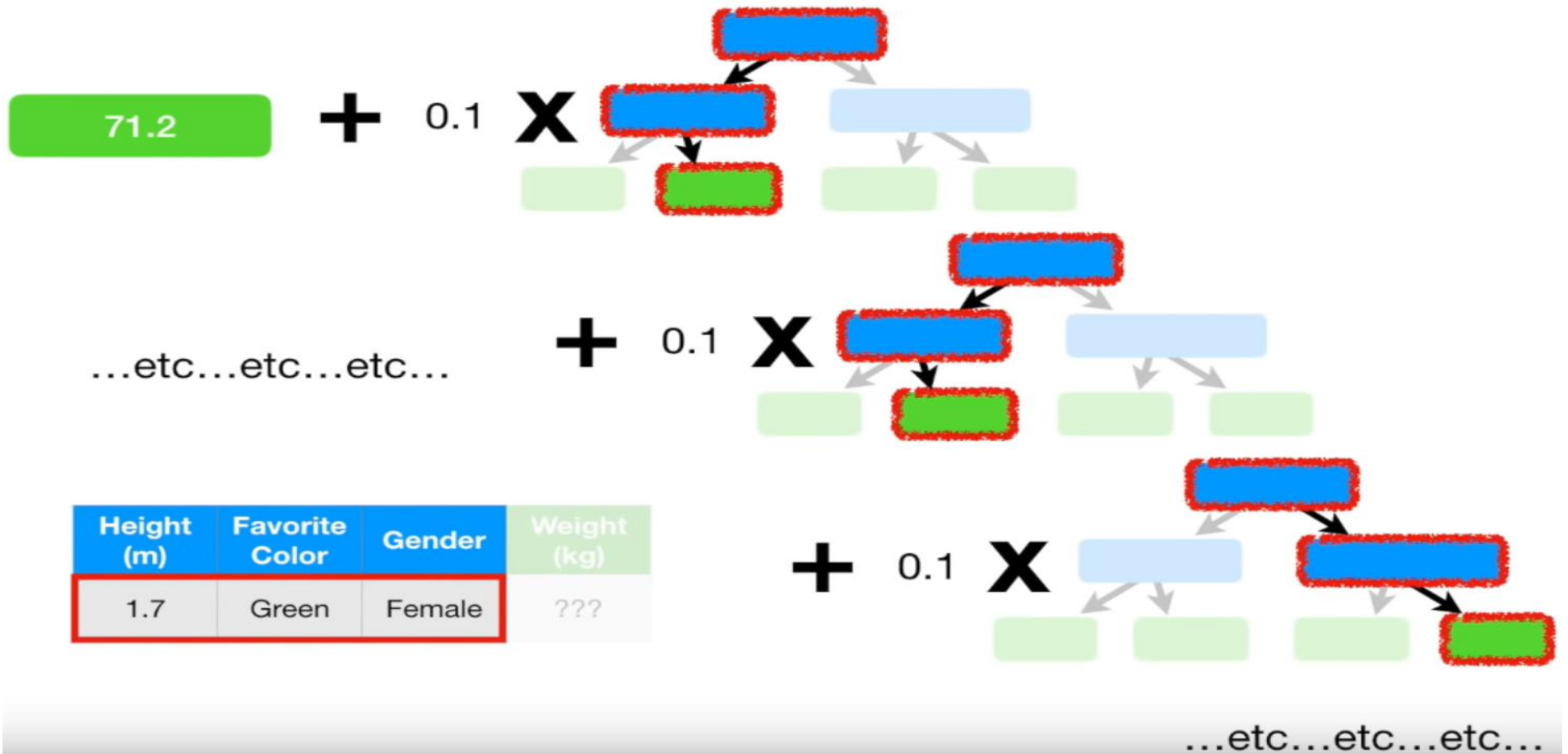
- 보시는 바와 같이 Residual을 점차 감소하고 있다. iteration을 할수록 Residual이 감소한다. 즉, 예측 정확도가 더 높아진다는 뜻이다.



- 이렇게 계속 반복한다. 언제까지? 사전에 하이퍼 파라미터로 정해 놓은 iteration 횟수에 도달하거나 더 이상 residual이 작아지지 않을 때까지 반복한다.

그래디언트 부스팅

- 모든 반복이 완료되었다면 최종적으로 Gradient Boost 모델이 구축된 것이다. 이제 새로운 데이터가 주어지면 본 모델로 몸무게를 예측할 수 있다.



- 초기 평균 값인 71.2에 모든 트리의 학습률 * residual을 더해주면 몸무게를 예측할 수 있다. 키가 1.7m이고 좋아하는 색깔이 초록색이고 성별이 여자이면 몸무게는 70kg라는 결과를 얻을 수 있다.

그래디언트 부스팅

- Gradient Boost 모델은 이런 방식으로 학습한다..
- Gradient Boost를 발전시킨 모델이 XGBoost, Light GBM, CatBoost이다. 더 빠르고 성능이 좋은 모델들이라고 보면 된다.

그래디언트 부스팅 [코드 리뷰]

그래디언트 부스팅

- 그래디언트 부스팅: 샘플의 가중치를 수정하는 대신 이전 예측기가 만든 잔여 오차에 새로운 예측기를 학습시킨다.

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)
```

- 첫번째 예측기에서 생긴 잔여 오차에 두번째 결정트리를 훈련시킨다.

```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)
```

- 두번째 예측기에 생긴 잔여 오차에 세번째 회귀모델을 훈련시킨다.

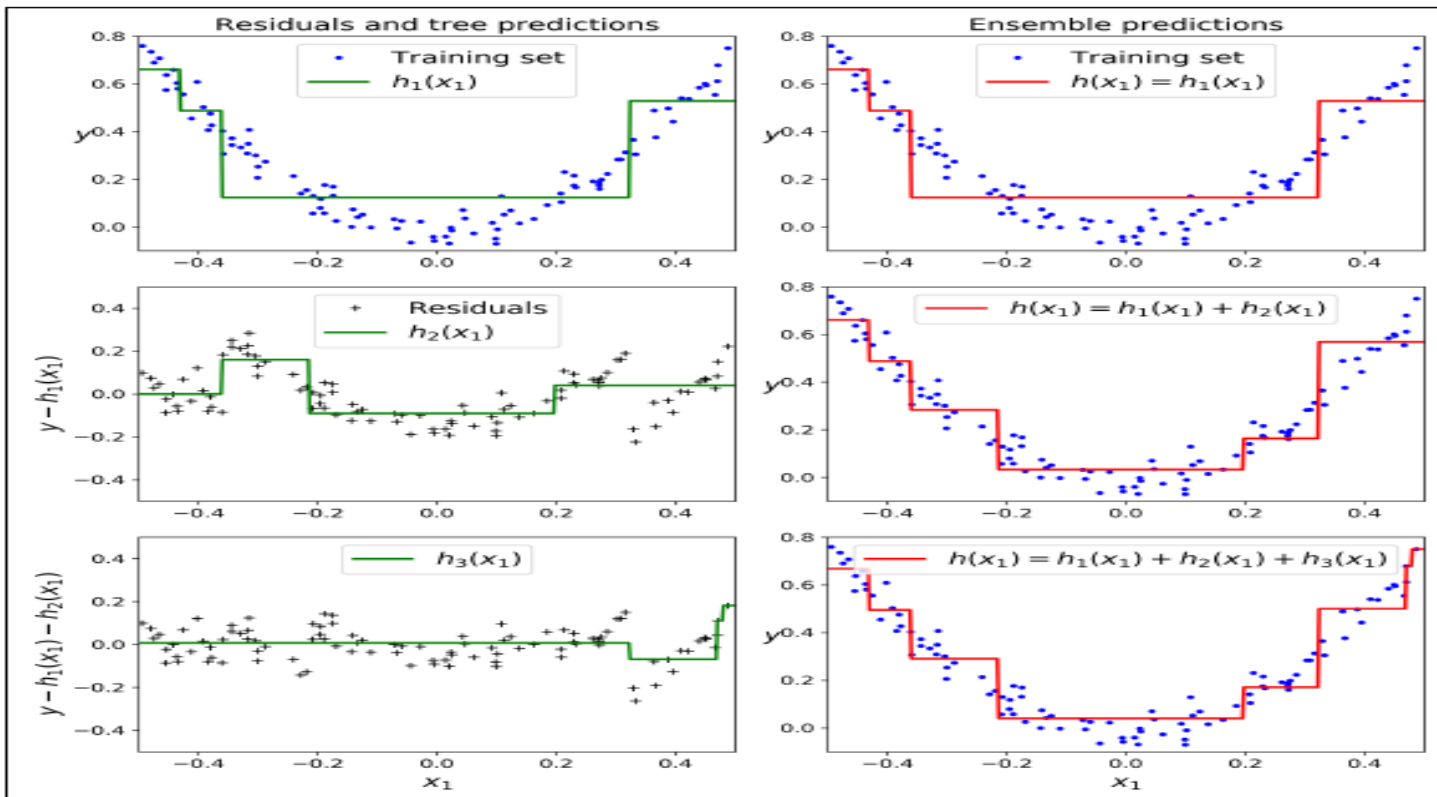
```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)
```

그라디언트 부스팅

- 이제 세 개의 트리를 포함하는 앙상블 모델이 생겼습니다. 새로운 샘플에 대한 예측을 하려면 모든 트리의 예측을 더하면 된다.

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

- 개별 예측기와 앙상블 예측기의 비교



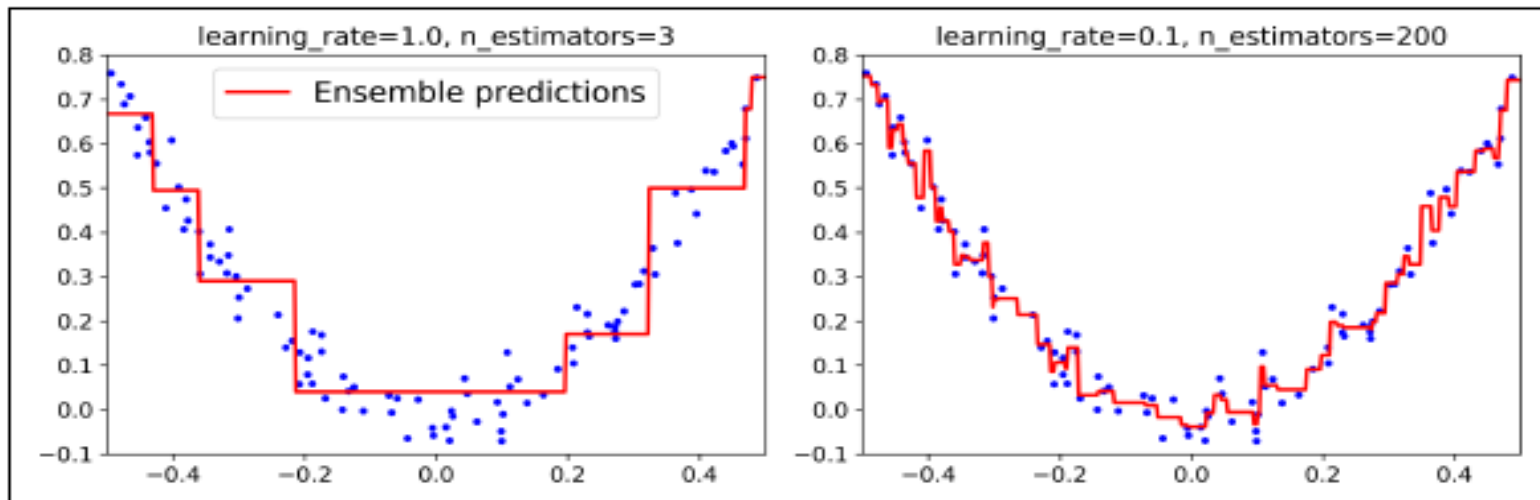
그래디언트 부스팅

- 사이킷런의 GradientBoostingRegressor를 사용하면 간단하게 훈련시킬 수 있다.

```
from sklearn.ensemble import GradientBoostingRegressor
```

```
gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0)  
gbrt.fit(X, y)
```

- Learning rate 매개변수를 0.1처럼 낮게 설정하면 훈련을 위해 많은 트리가 필요하지만 예측의 성능은 좋아진다.



그래디언트 부스팅

- GradientBoostingClassifier

```
In [73]: from sklearn.ensemble import GradientBoostingClassifier

X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)

gbrt = GradientBoostingClassifier(random_state=0)
gbrt.fit(X_train, y_train)

print("훈련 세트 정확도: {:.3f}".format(gbrt.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(gbrt.score(X_test, y_test)))
```

훈련 세트 정확도: 1.000
테스트 세트 정확도: 0.958

n_estimators=100, max_depth=3

과대적합

```
In [74]: gbrt = GradientBoostingClassifier(random_state=0, max_depth=1)
gbrt.fit(X_train, y_train)

print("훈련 세트 정확도: {:.3f}".format(gbrt.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(gbrt.score(X_test, y_test)))
```

훈련 세트 정확도: 0.991
테스트 세트 정확도: 0.972

트리 깊이 제한

```
In [75]: gbrt = GradientBoostingClassifier(random_state=0, learning_rate=0.01)
gbrt.fit(X_train, y_train)

print("훈련 세트 정확도: {:.3f}".format(gbrt.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(gbrt.score(X_test, y_test)))
```

훈련 세트 정확도: 0.988
테스트 세트 정확도: 0.965

학습률 감소

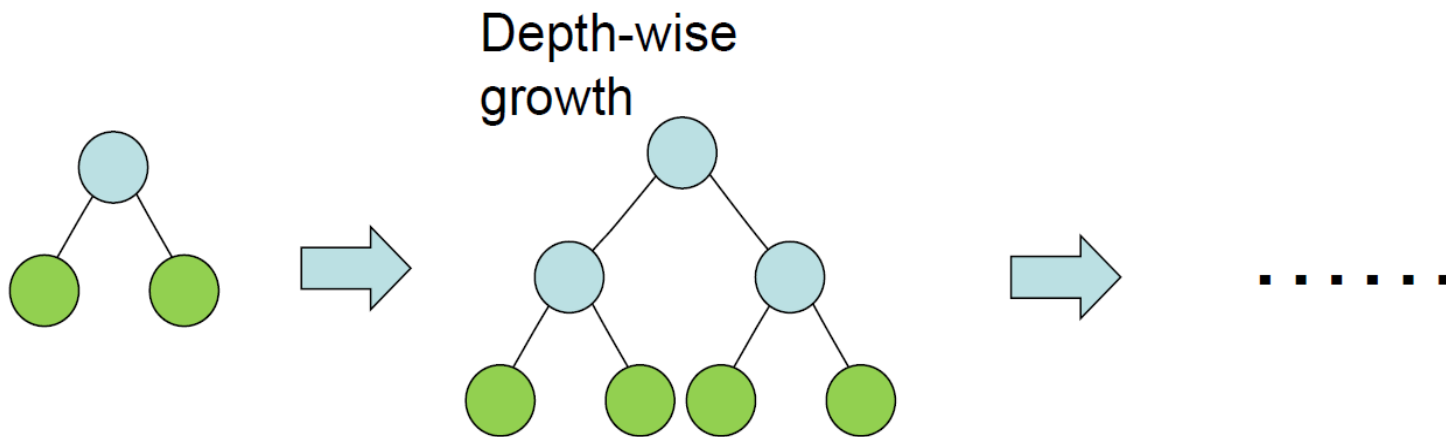
XGBoost

XGBoost의 소개

- XGBoost는 LightGBM이 나오기 전의 대세였던 강력한 GBDT 방법 (아직 많이 사용)
 - Kaggle 경연에서 많은 우승자를 배출
 - 지금은 XGBoost, LightGBM(마이크로소프트), CatBoost(러시아 엔덱스)의 블렌디드 모델이 강력한 엔진으로 등장

데이터 중심 접근

XGBosst 트리 생성 구조는 동일 깊이 구조를 갖는다. 한쪽으로만 자라지 않는다.



XGBoost

- 주요 특징

- 손실함수로부터 직접 트리의 분지점을 구하고자하며, (LightGBM과 동일)
- (1) 2차함수 근사의 손실함수 최소와 함께 규제화를 사용
- (2) Histogram-based algorithm과 (메모리에서 정렬하는) Pre-sorted algorithm 두개를 사용

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

where $\Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2$

규제화

2계최적화

$$\mathcal{L}_{split} = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma$$

XGBoost

- 기타 주요 특징

Approximate Greedy Algorithm

Parallel Learning

Weighted Quantile Sketch

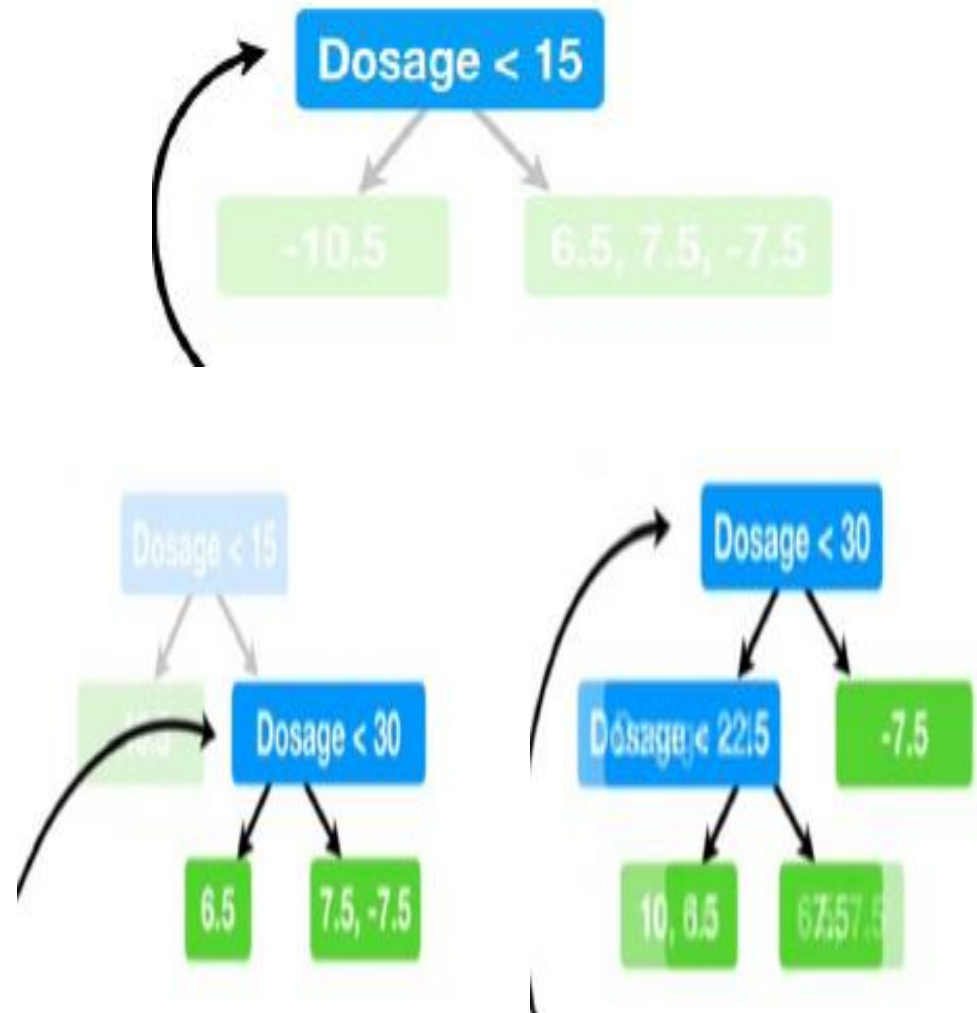
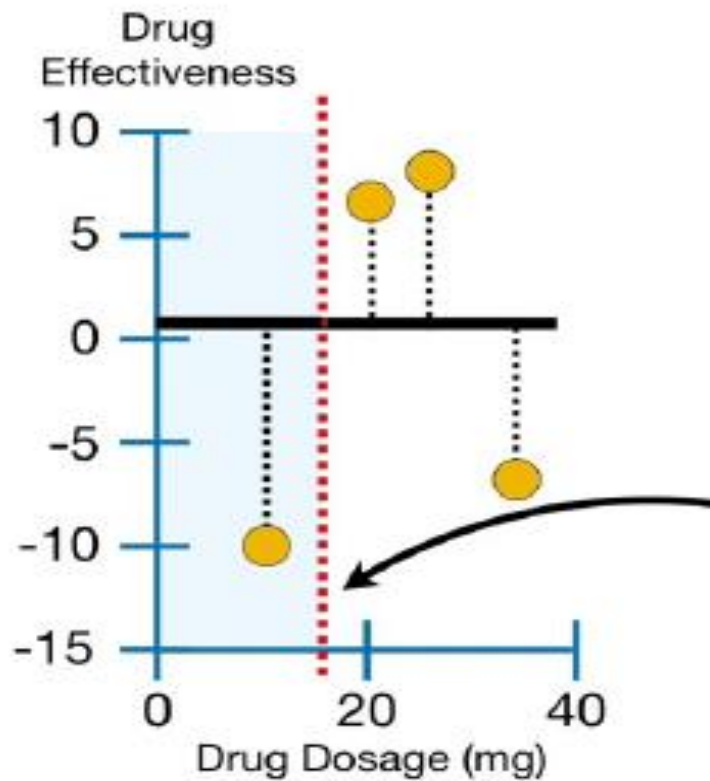
Sparsity-Aware Split Finding

Cache-Aware Access

Blocks for Out-of-Core Computation

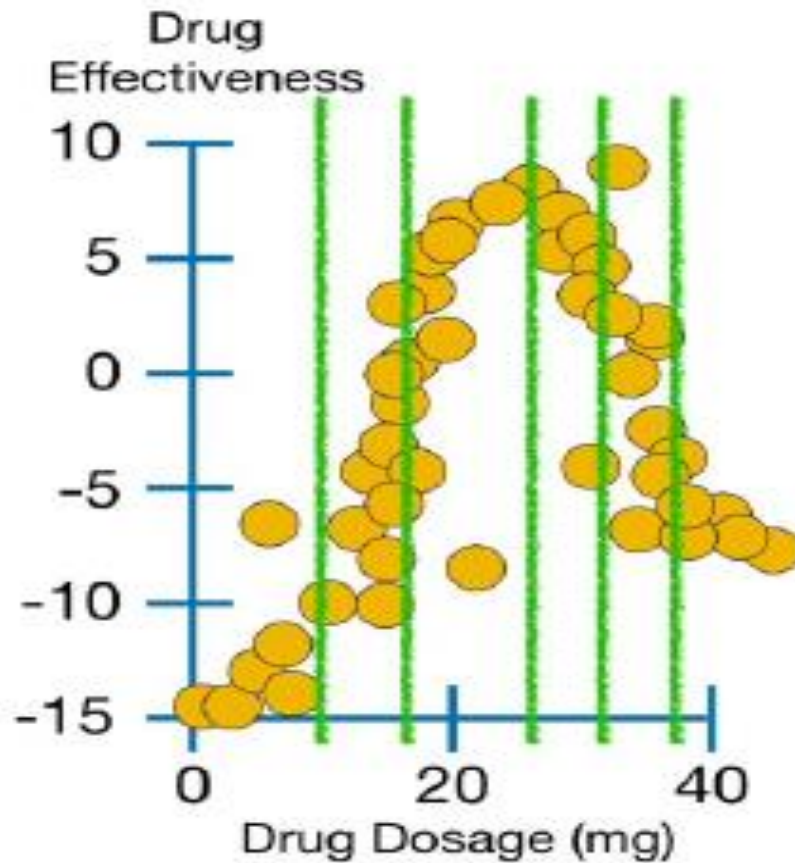
XGBoost

- Greedy Algorithm: 분할시 한단계만 고려한다.



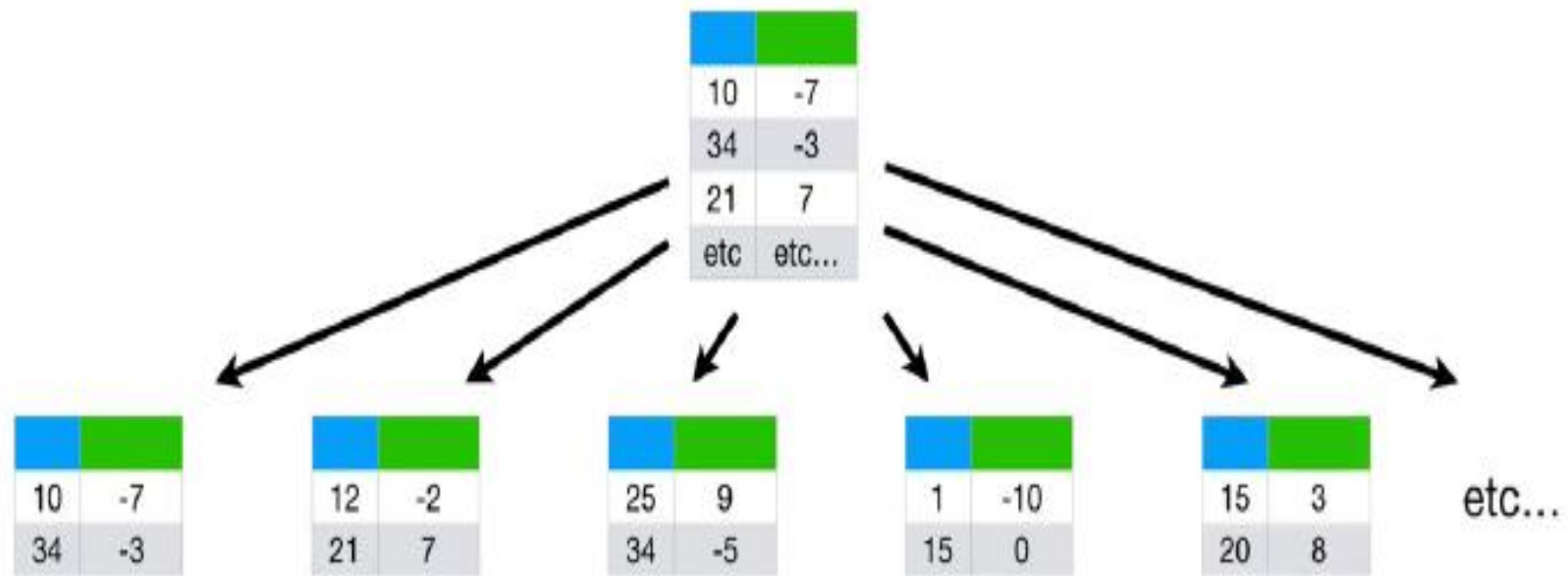
XGBoost

- Approximate Greedy Algorithm: 분할시 qunantile 포인트만 고려한다.



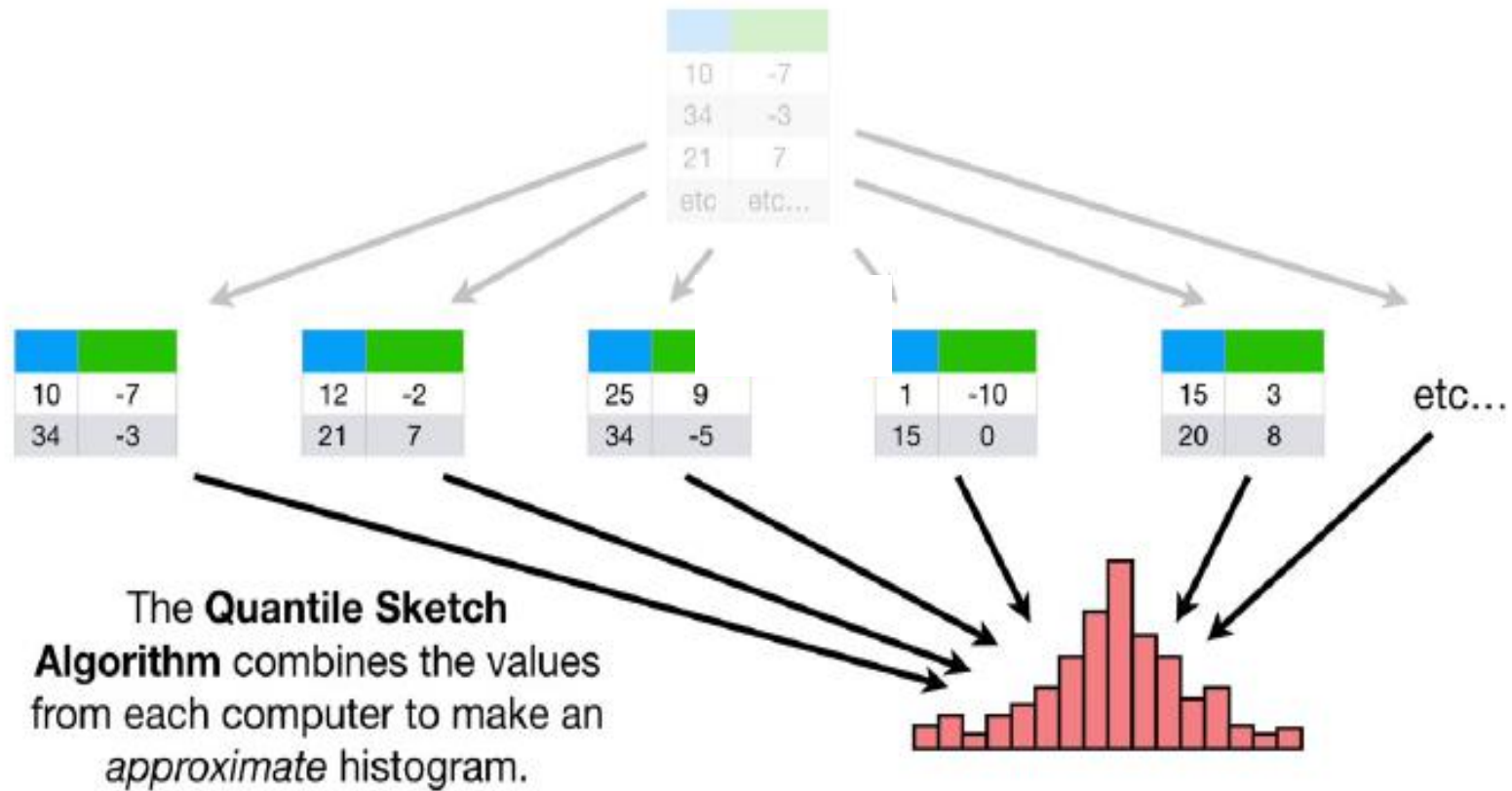
XGBoost

- Approximate Greedy Algorithm: 스케치 알고리즘



XGBoost

- Approximate Greedy Algorithm: 쿼타일 스케치 알고리즘

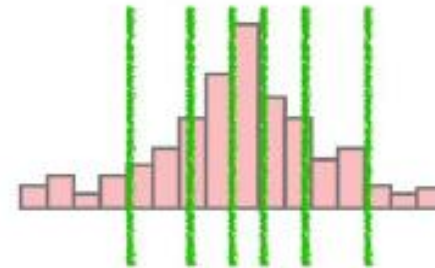


XGBoost

- 근사적 탐욕 알고리즘(Approximate Greedy Algorithm)는 다음과 같은 가중분위스케치(Weighted Quantile Sketch) 알고리즘을 사용한다.

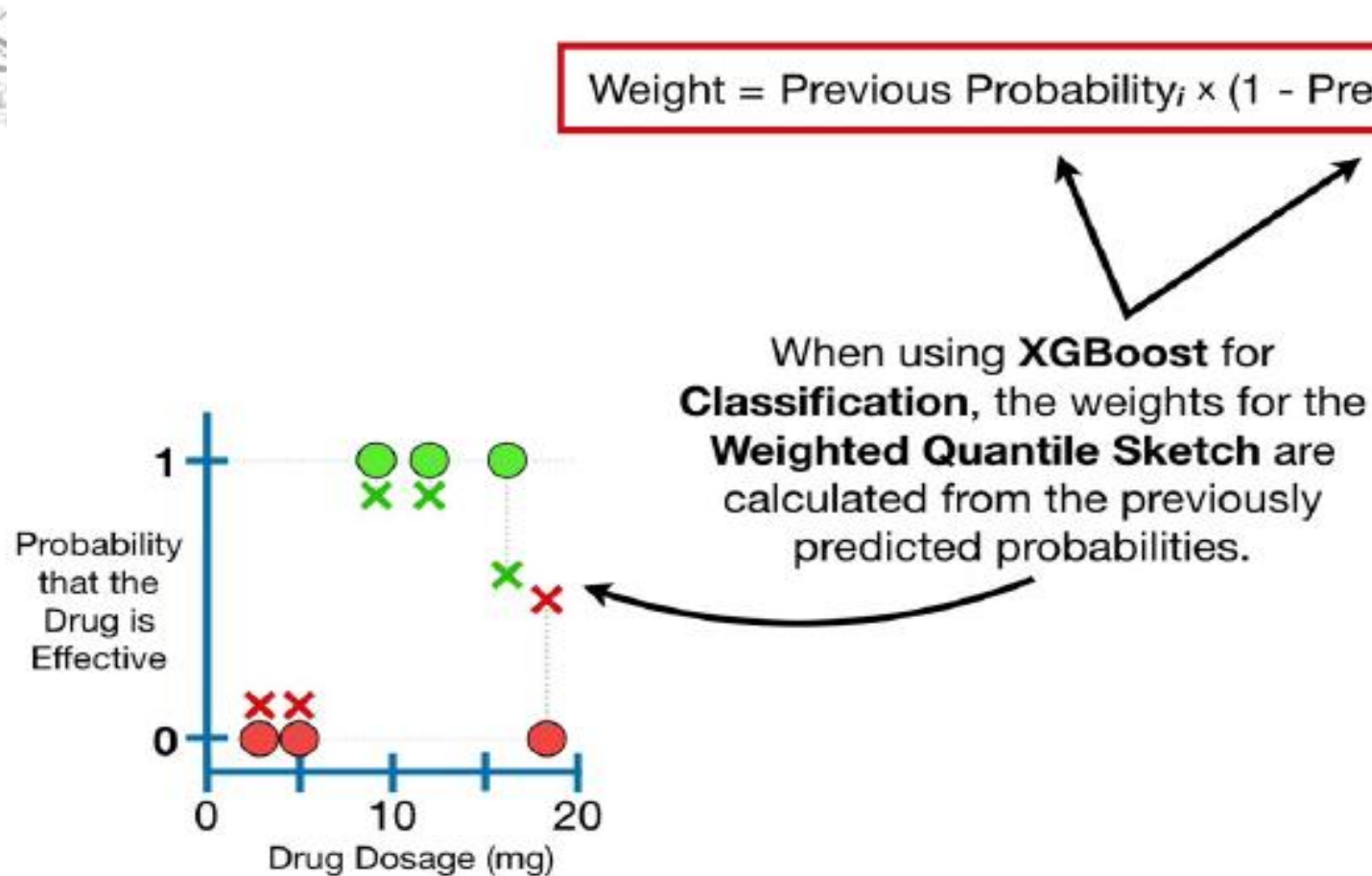


...but **XGBoost** uses a **Weighted Quantile Sketch**.



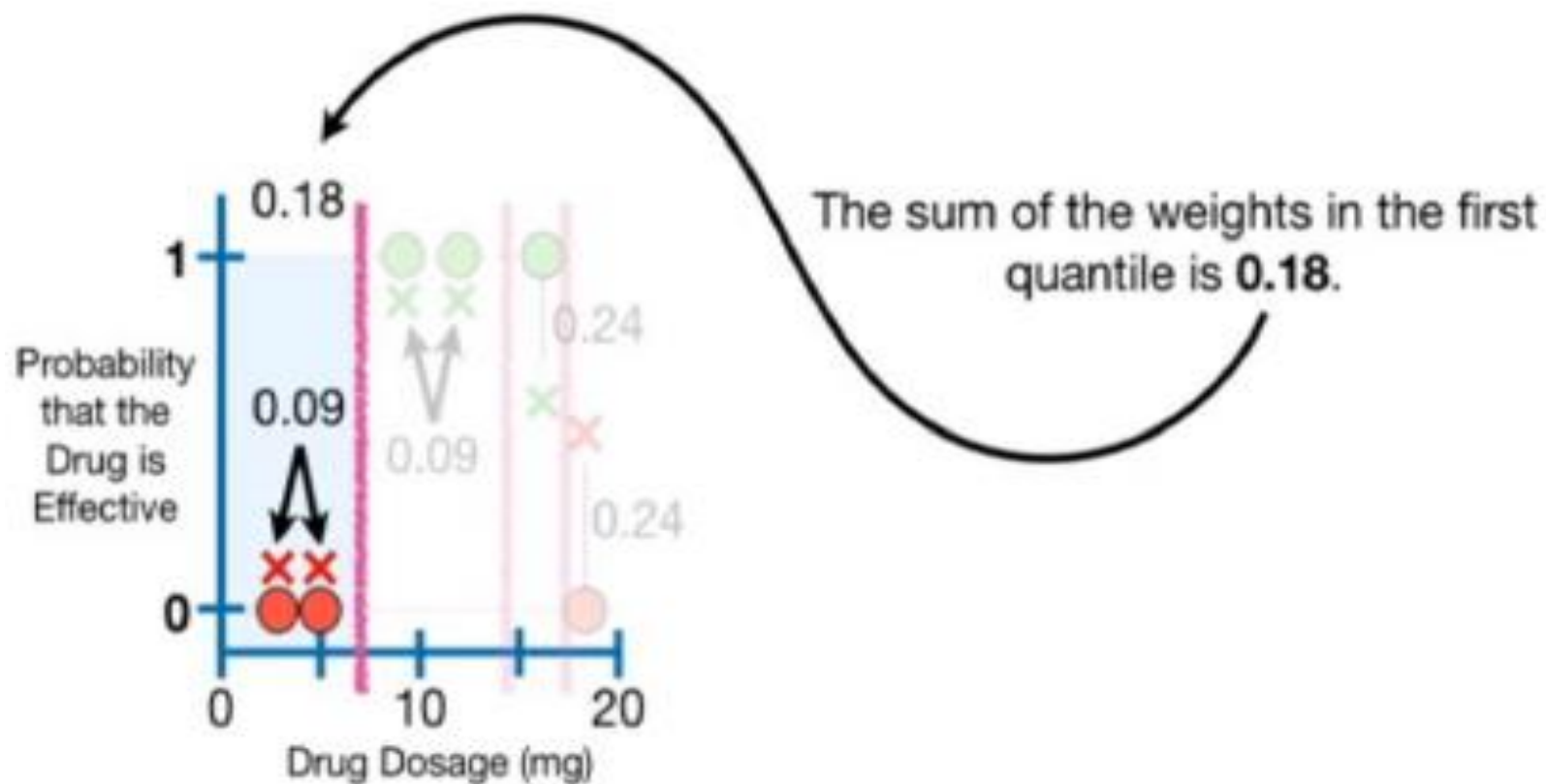
XGBoost

- 가중분위스케치(Weighted Quantile Sketch)란? 가중치를 다음과 같은 방식으로 계산을 해 Qunatile을 나눌 때 이 가중치를 고려한다.



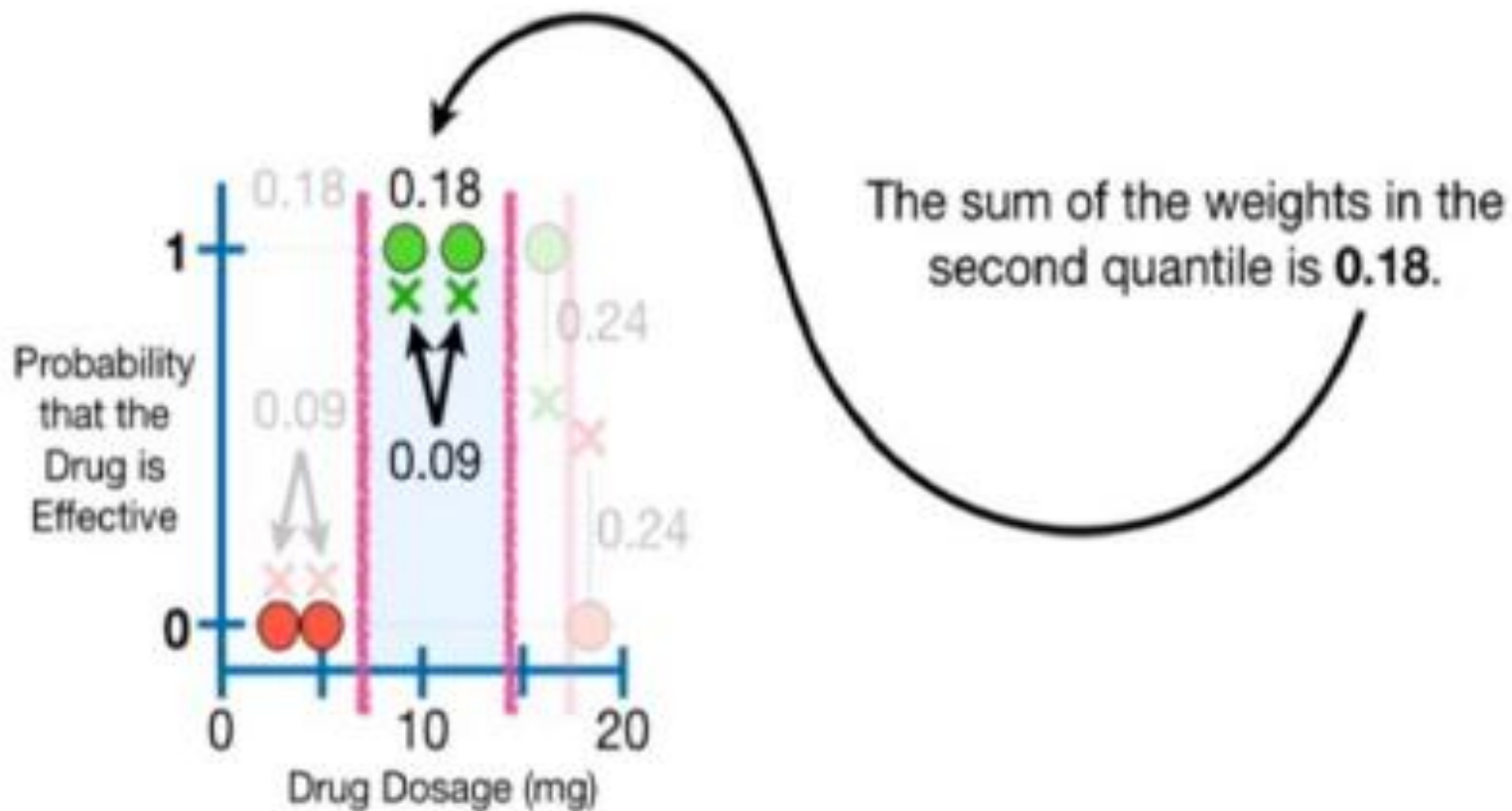
XGBoost

- 분위기를 나누는데 가중치를 사용함으로써 다음 그림에서 보듯이 분기의 불순도를 감소시킨다.)



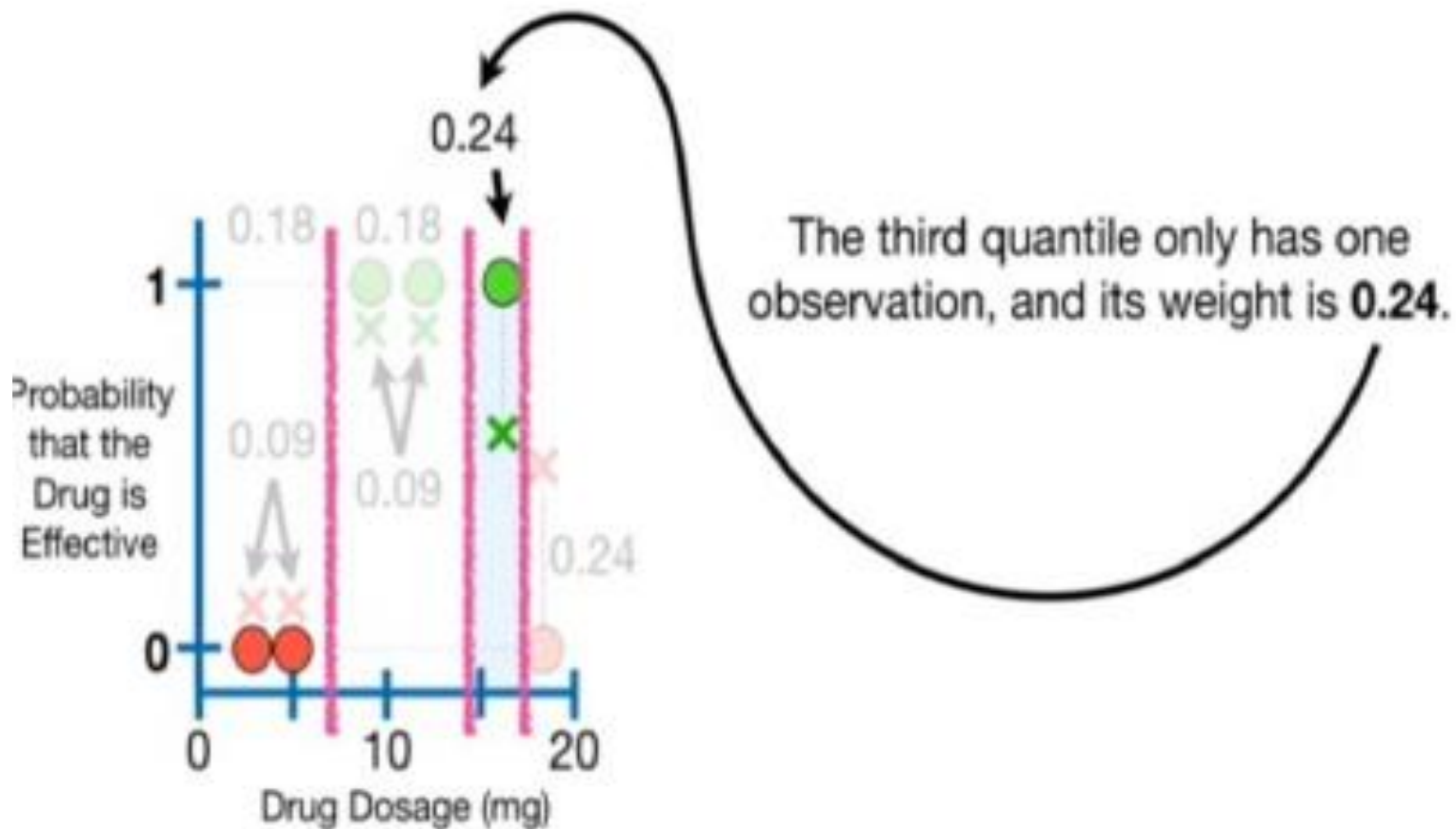
XGBoost

- 분위기를 나누는데 가중치를 사용함으로써 다음 그림에서 보듯이 분기의 불순도를 감소시킨다.)



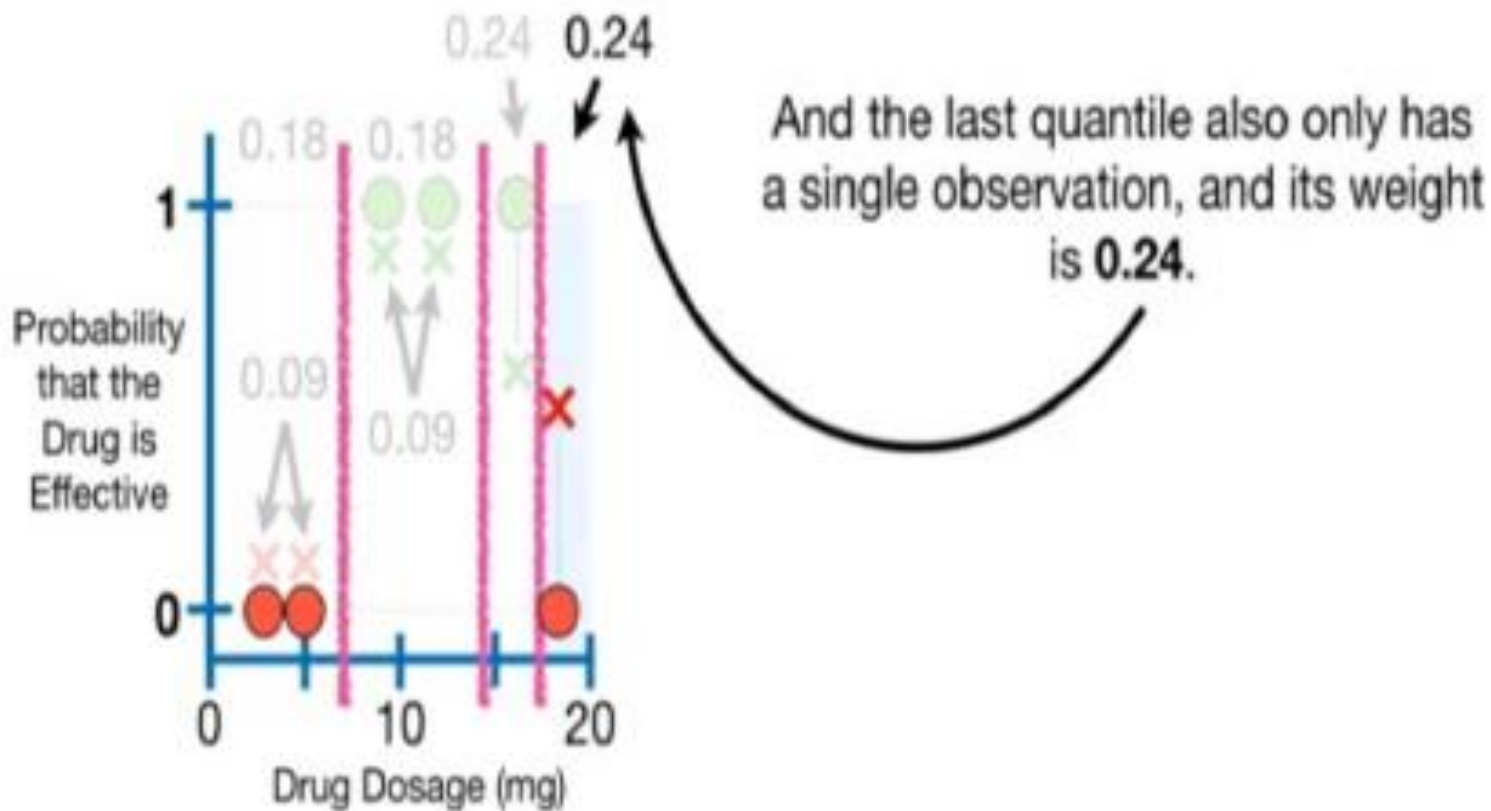
XGBoost

- 분위를 나누는데 가중치를 사용함으로써 다음 그림에서 보듯이 분기의 불순도를 감소시킨다.)



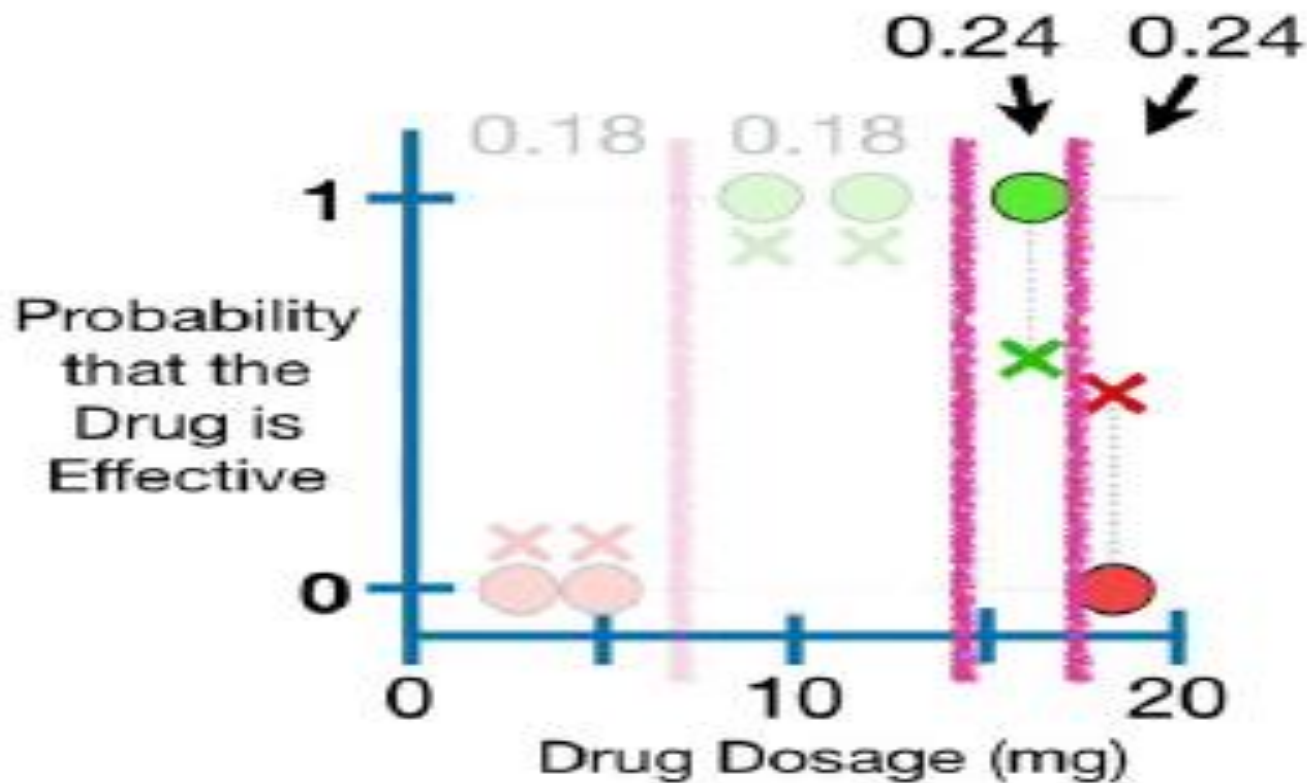
XGBoost

- 분위기를 나누는데 가중치를 사용함으로써 다음 그림에서 보듯이 분기의 불순도를 감소시킨다.)



XGBoost

- 분위기를 나누는데 가중치를 사용함으로써 다음 그림에서 보듯이 분기의 불순도를 감소시킨다.)



XGBoost

- 데이터셋을 동시에 여러 컴퓨터 작업할 수 있도록 분할하는 병렬 학습을 사용한다.



Gradient Boost-(ish)

Regularization

A Unique Regression Tree

Approximate Greedy Algorithm

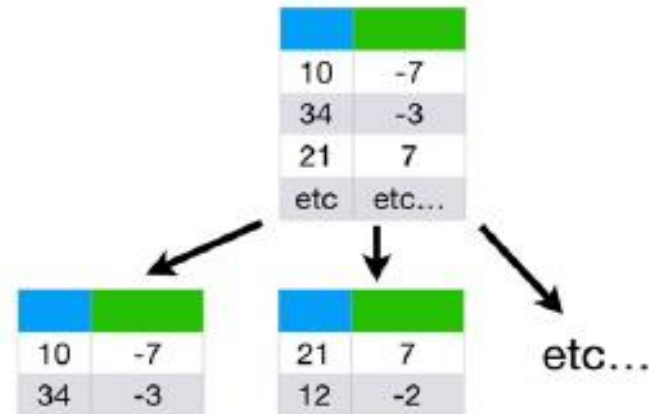
Parallel Learning

Weighted Quantile Sketch

Sparsity-Aware Split Finding

Cache-Aware Access

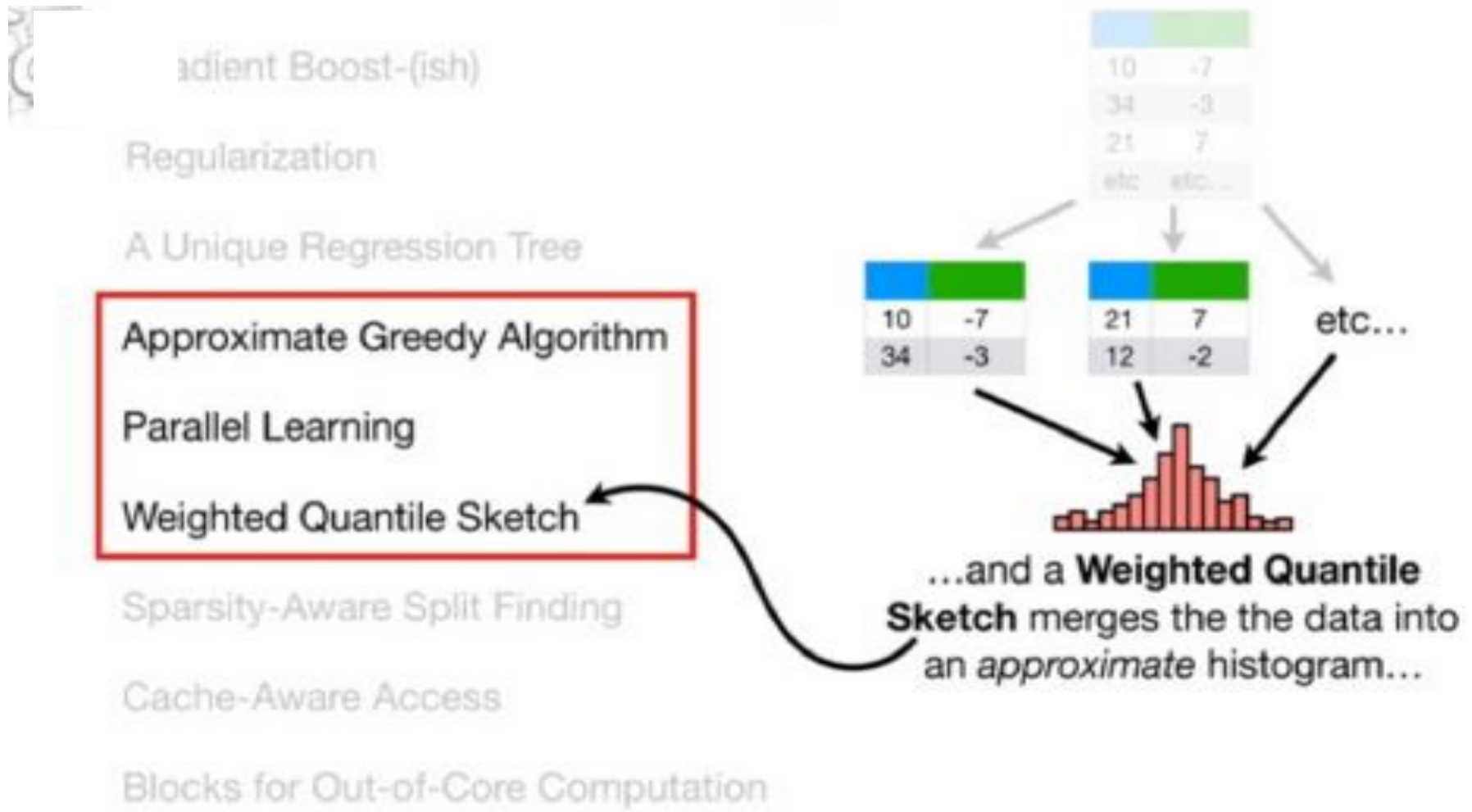
Blocks for Out-of-Core Computation



...and that means using **Parallel Learning** to split up the dataset so that multiple computers can work on it at the same time...

XGBoost

- 이후 WQS(Weighted Quantile Sketch)는 데이터를 근사적인 히스토그램으로 병합한다.



XGBoost

- 희소 인지 분할(Sparsity aware splitting): 결측치 처리



Predicted Drug Effectiveness

0.5

Dosage	Drug Effectiveness	Residuals
10	-7	-7.5
???	-3	-3.5
21	7	6.5
25	8	7.5
5	-5	-5.5
???	-2	-2.5



Predicted Drug Effectiveness

0.5

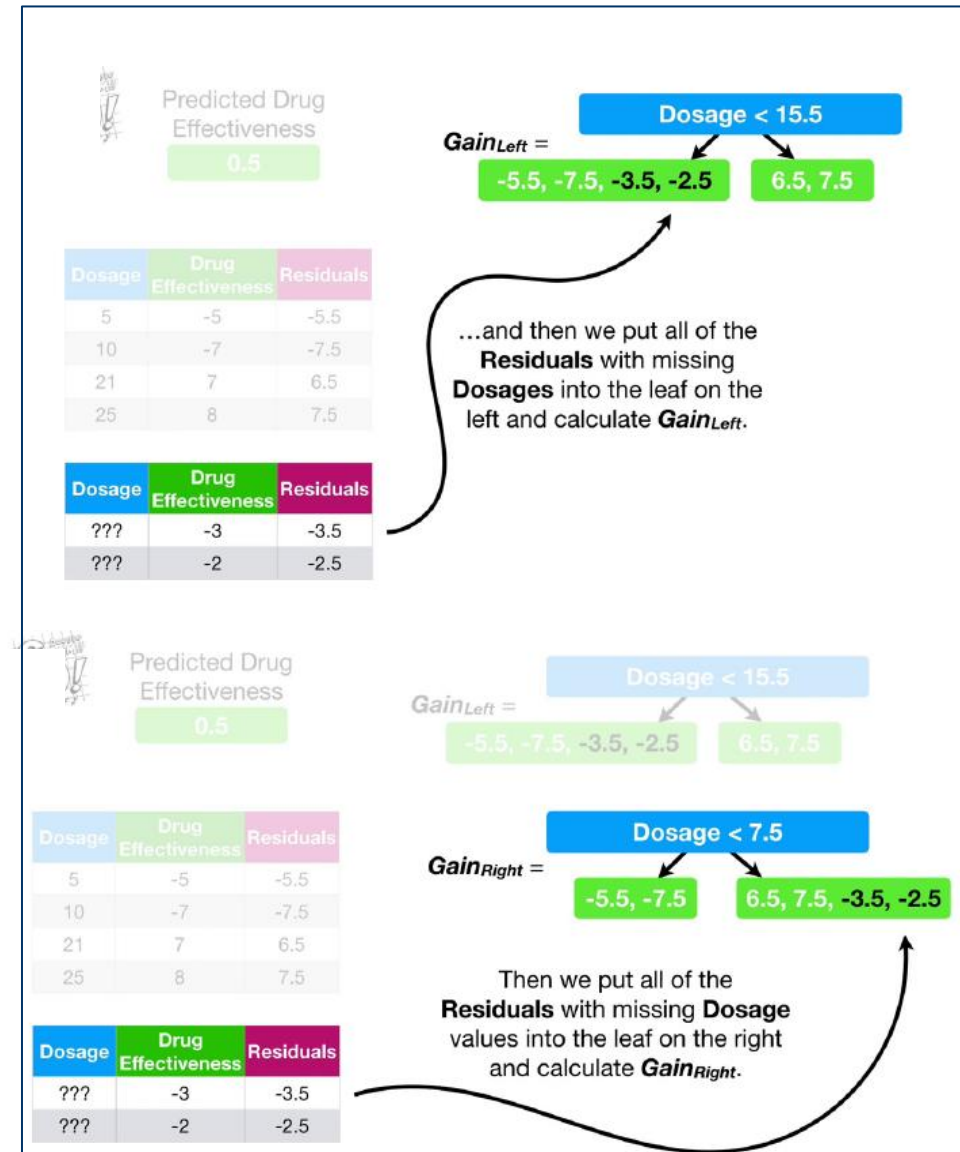
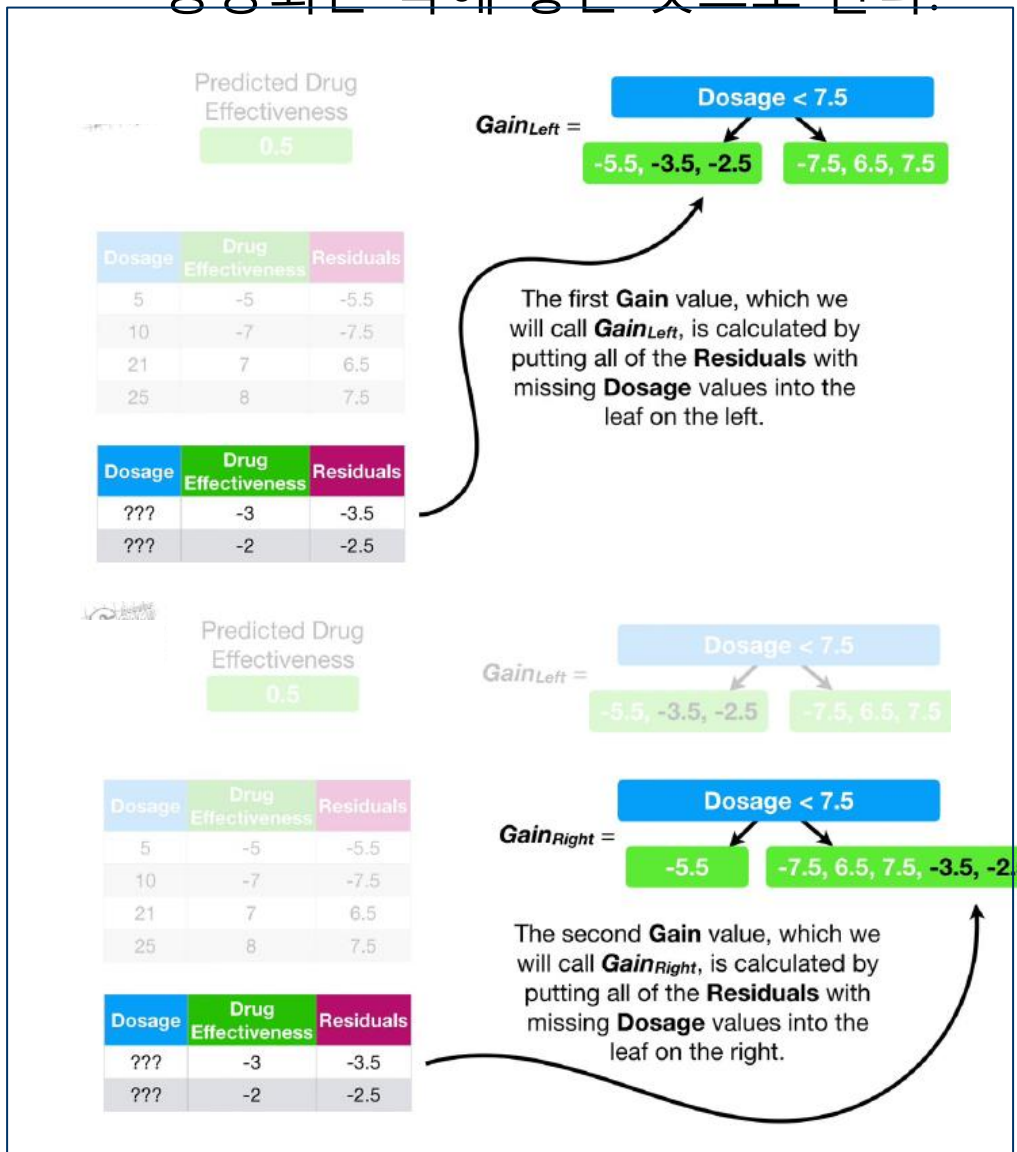
Dosage	Drug Effectiveness	Residuals
10	-7	-7.5
???	-3	-3.5
21	7	6.5
25	8	7.5
5	-5	-5.5
???	-2	-2.5

-7.5, -3.5, 6.5, 7.5, -5.5, 2.5

And just like we normally do when we build **XGBoost Trees**, we can put all of the **Residuals** into a single leaf.

XGBoost

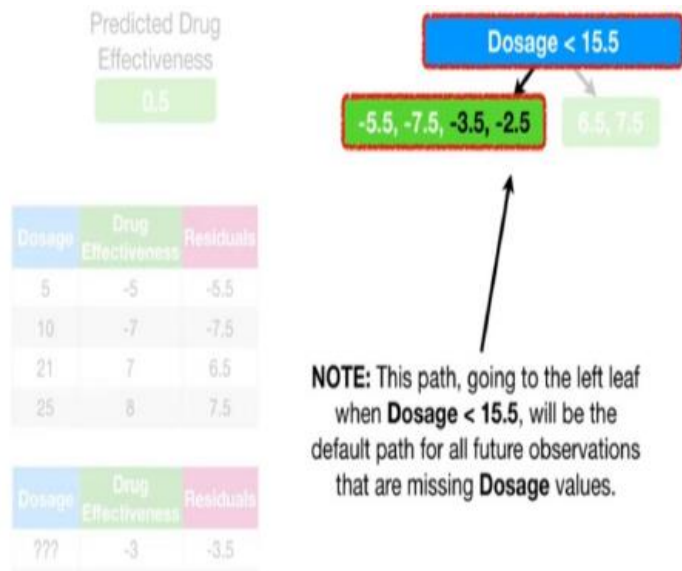
- 희소 인지 분할(Sparsity aware splitting): 결측치는 불순도 감소가 향상되는 쪽에 넣는 것으로 한다.



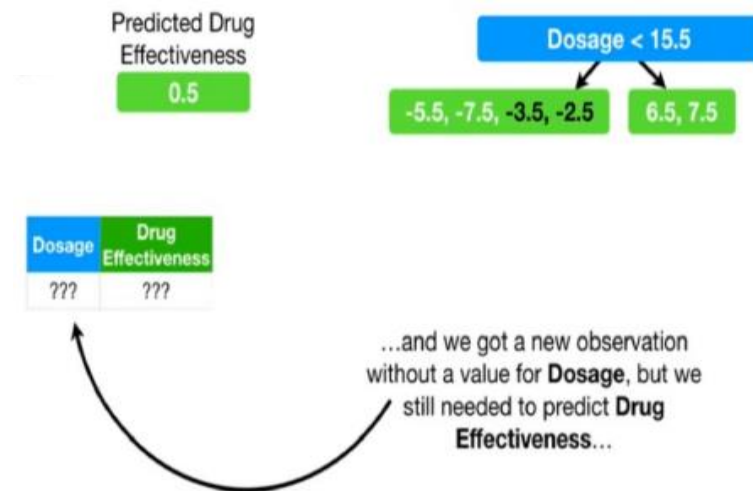
XGBoost

- 희소 인지 분할(Sparsity aware splitting): 결측치 처리

(1) 샘플의 결측치 처리

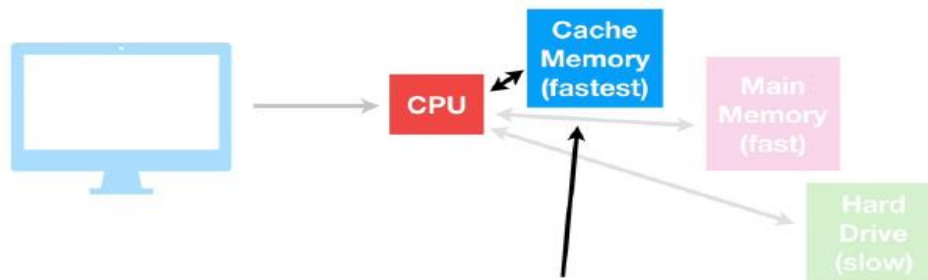


(2) 새로운 결측치 샘플 처리

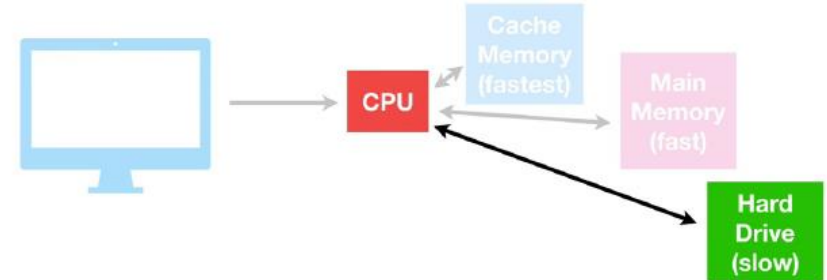


XGBoost

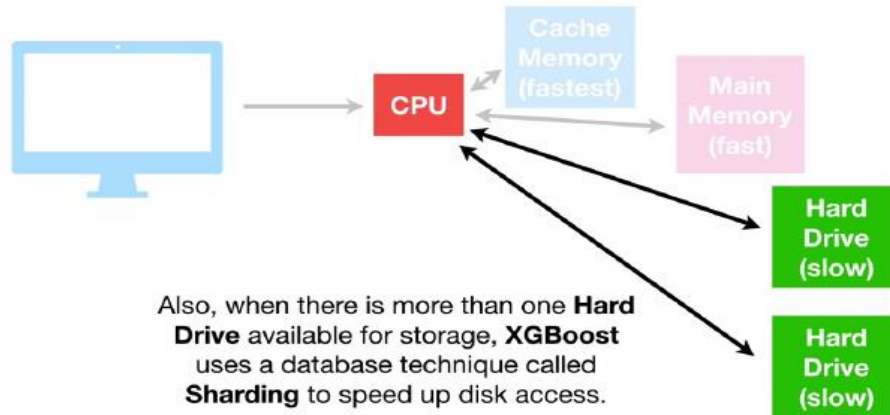
- 캐시 인지 접근 (Cache aware access): CPU 메모리를 최대한 사용해 속도를 향상시킨다. 하드드라이브를 쓸 때 CPU로 압축시키고 분산/병렬 저장을 한다.



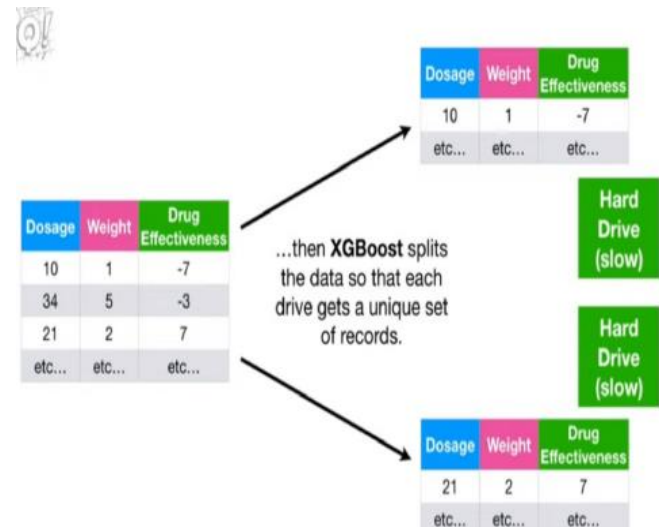
So **XGBoost** puts the **Gradients** and **Hessians** in the **Cache** so that it can rapidly calculate **Similarity Scores** and **Output Values**.



In other words, by spending a *little* bit of **CPU** time uncompressing the data, we can avoid spending a *lot* of time accessing the **Hard Drive**.



Also, when there is more than one **Hard Drive** available for storage, **XGBoost** uses a database technique called **Sharding** to speed up disk access.

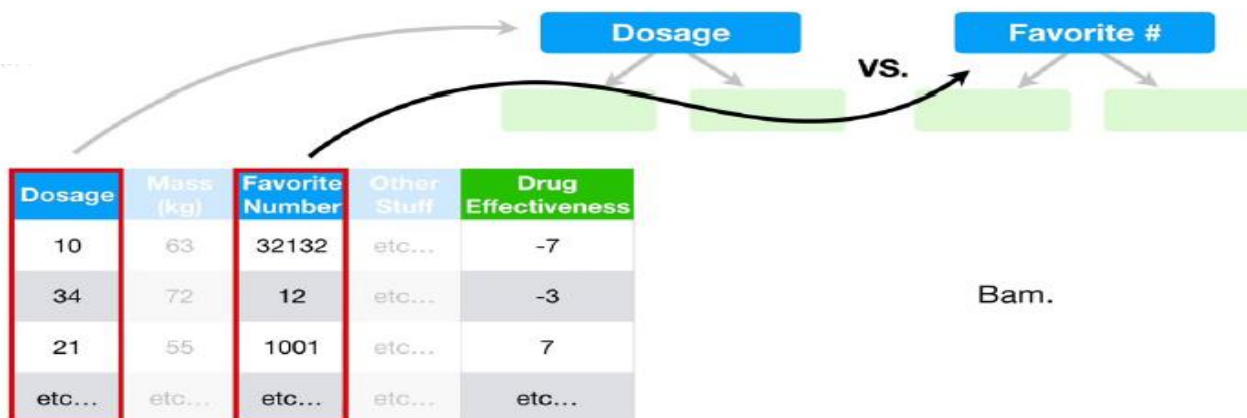


XGBoost

- 샘플과 특성의 랜덤 부분집합을 사용한다. (랜덤포레스트와 유사)

Dosage	Mass (kg)	Favorite Number	Other Stuff	Drug Effectiveness
10	63	32132	etc...	-7
34	72	12	etc...	-3
21	55	1001	etc...	7
etc...	etc...	etc...	etc...	etc...

Lastly, I need to mention that **XGBoost** can also speed things up by allowing you to build each tree with only a random subset of the data.



XGBoost

- More than 통계학!

Gradient Boost-(ish)

Regularization

A Unique Regression Tree

Approximate Greedy Algorithm

Parallel Learning

Weighted Quantile Sketch

Sparsity-Aware Split Finding

Cache-Aware Access

Blocks for Out-of-Core Computation

And that means **Machine Learning** is more than just applied **Statistics**.



LightGBM

히스토그램 기반의 분지 전략

- 분지점에서의 스코어를 분산으로 사용하는 경우

Predict **age** of a person from Titanic Dataset.

sex	survived	age
female	1	29
male	1	1
female	0	2
male	0	30
female	0	25
male	1	48
female	1	63
male	0	39
female	1	53
male	0	71

Variance: 498.29

calculate variances

survived	Var	#people
0	502.64	5
1	479.36	5

sex	Var	#people
male	524.56	5
female	466.24	5

weighted average

Variance

491.0

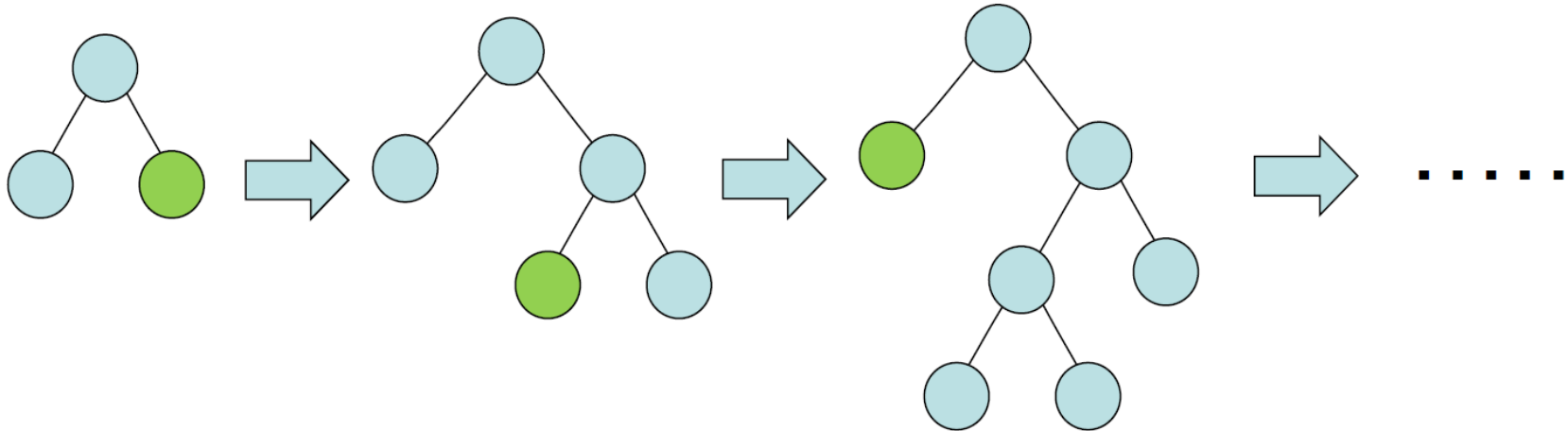
7.29 Down

495.4

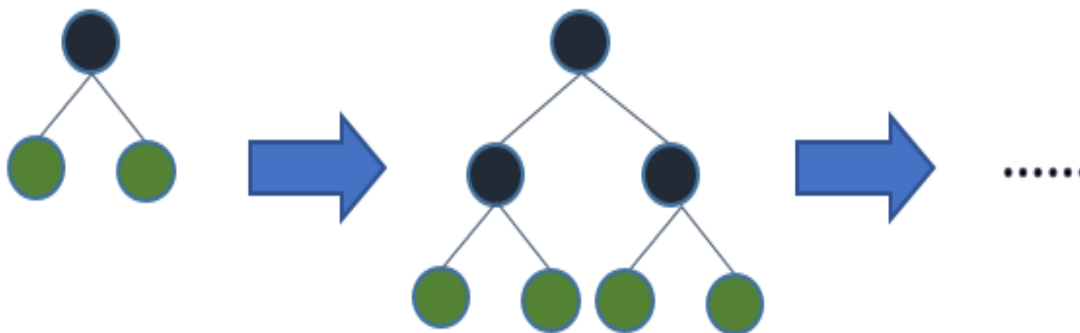
2.11 Down

데이터 중심 접근

- LightGBM 트리 생성 구조: 트리가 비대칭적으로 깊이 자랄 수 있다. 옆 노드의 정보가 그다지 중요하지 않을 때는 그쪽은 무시하고 중요한 노드 쪽을 집중적으로 판다. => 속도의 향상
- Leafwise Tree (Depthwise Tree)



- Levelwise Tree

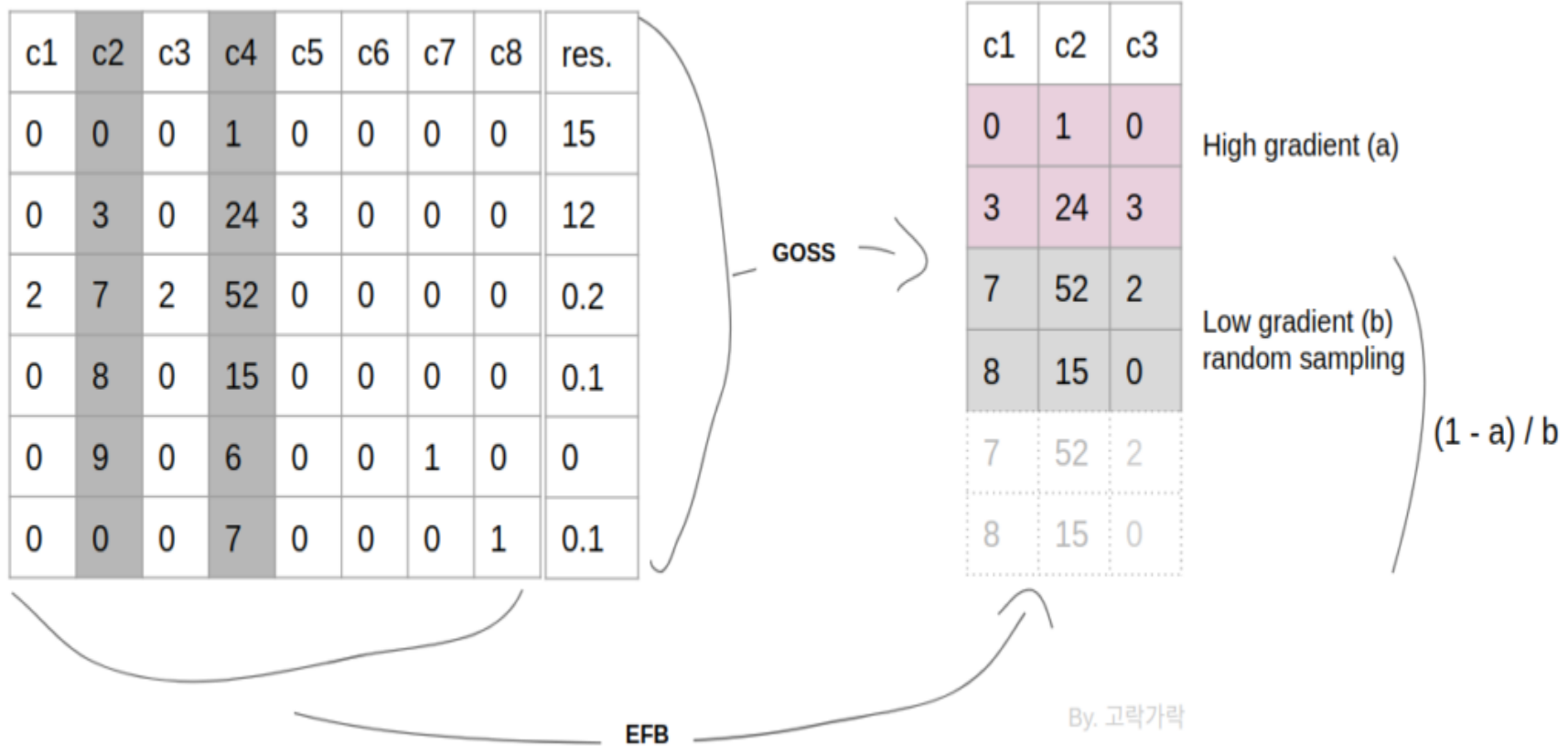


LightGBM

$$\begin{array}{c} \text{GBDT (Gradient Boosting Decision Tree)} \\ + \\ = \text{GOSS (Gradient-based One-side Sampling)} \\ + \\ \text{EFB (Exclusive Feature Bundling)} \end{array}$$

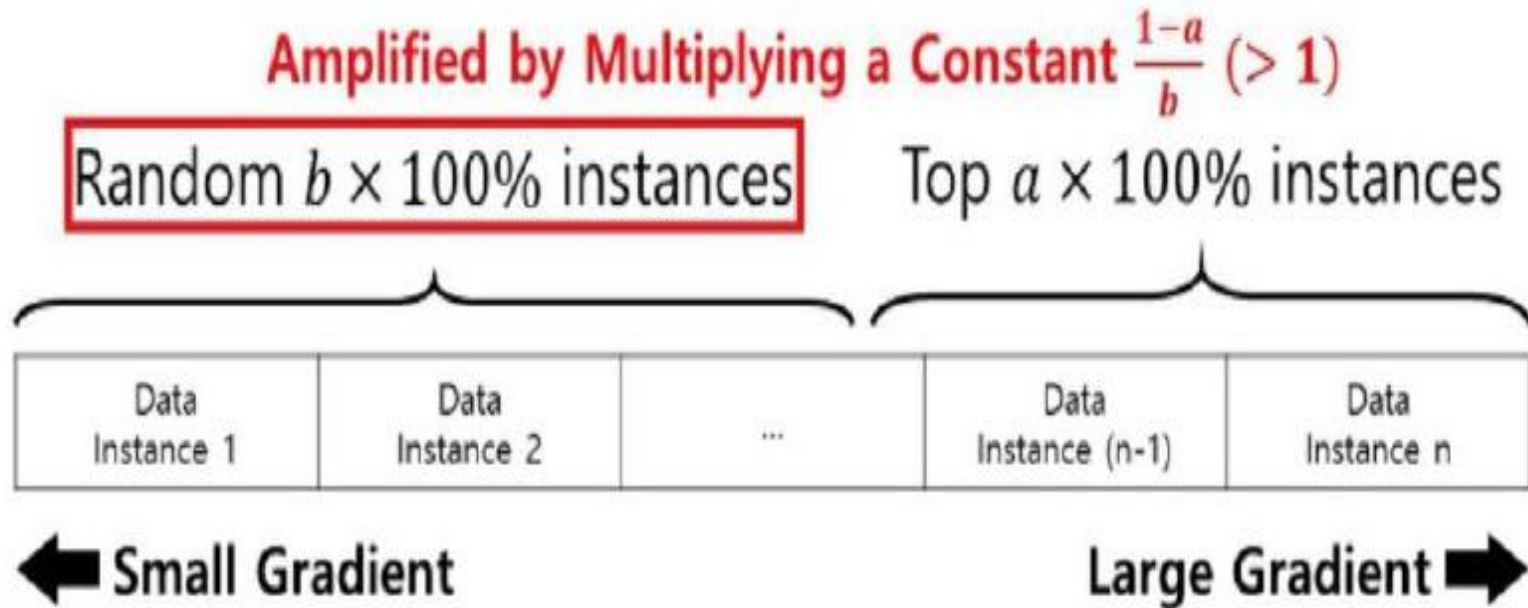
GOSS와 EFB 개요

- GOSS와 EFB



GOSS(Gradient-based one-side sampling)

- GOSS의 기본 원리



<https://cdm98.tistory.com/m/31>

EFB(Exclusive Feature Bundling)

- 원값에 변수를 구별하기 위한 offset(예: 기준변수의 최대값)을 더한다.

	x_5	x_1	x_4	x_2	x_3
l_1	1	1	0	1	0
l_2	1	0	1	0	1
l_3	2	1	0	2	0
l_4	1	0	3	0	2
l_5	3	2	0	1	0
l_6	1	3	0	3	0
l_7	2	0	0	0	3
l_8	3	1	4	2	3
l_9	0	1	0	0	1
l_{10}	2	2	0	3	0

	x_5	x_{14}	x_{23}
l_1	1		1
l_2	1	4	4
l_3	2	1	2
l_4	1	6	5
l_5	3	2	1
l_6	1	3	3
l_7	2	0	6
l_8	3	1	2
l_9	0	1	4
l_{10}	2	2	3

Add the offset 3 to the nonzero values of x_4

Add the offset 3 to the nonzero values of x_3

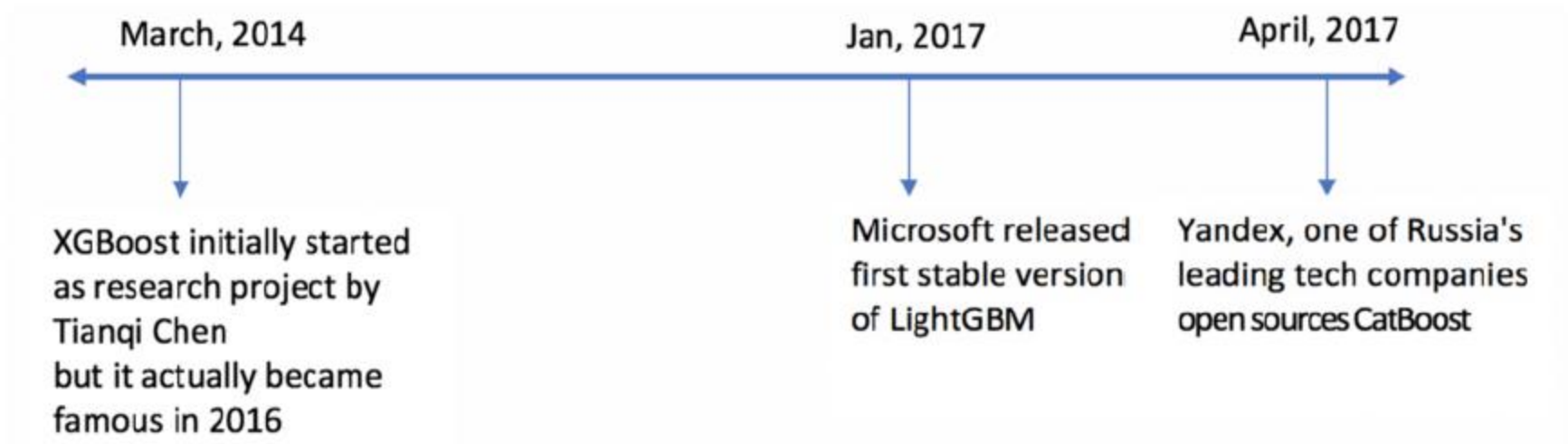
Conflict: Use the value of x_1

Conflict: Use the value of x_2

CatBoost

CatBoost

- 발전 경과



CAT Boost 일반

1. 기존 부스팅 방법

1. 실제 값들의 평균과 실제 값의 차이인 잔차(Residual)를 구한다.
2. 데이터로 이 잔차들을 학습하는 모델을 만든다.
3. 만든 모델로 예측하여, 예측 값에 Learning_rate 를 곱해 실제 예측 값(평균 + 잔차예측 값*lr) 을 업데이트 한다.
4. 1~3 반복

2. 문제점

1. 느린 속도
2. 과대적합

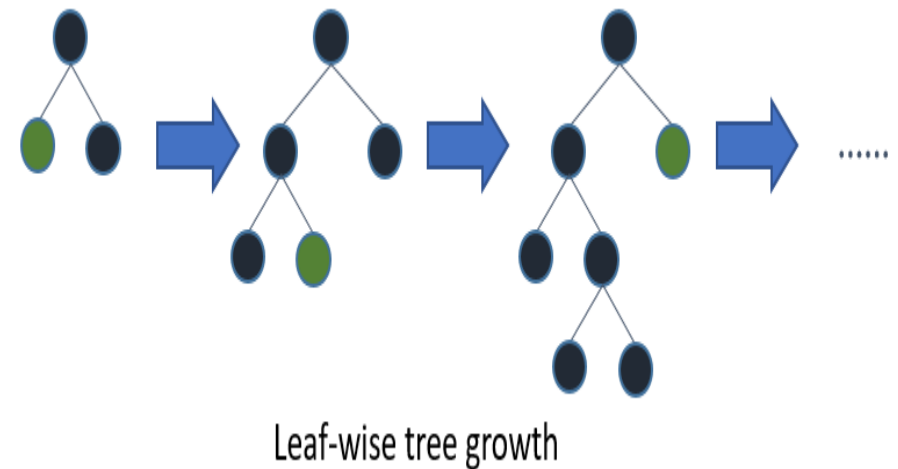
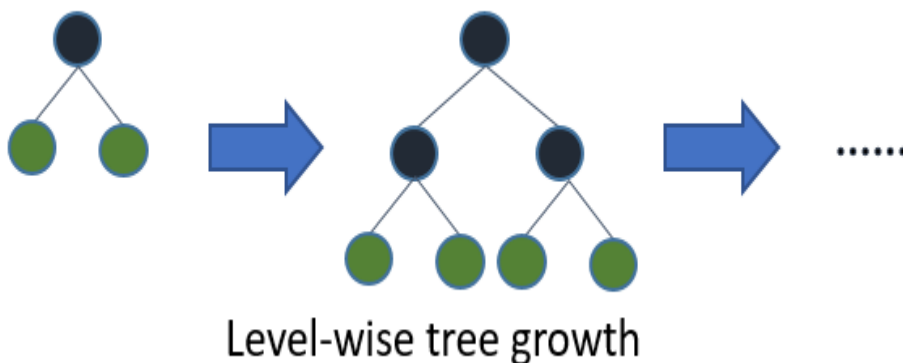
Catboost의 특징

1. Level wise 트리

- XGBoost 와 더불어 Catboost 는 Level-wise 로 트리를 만들어나간다.

(반면 Light GBM 은 Leaf-wise 다)

- Level-wise 와 Leaf-wise 의 차이는, 그냥 직관적으로 말하면 Level-wise 는 BFS(Breadth First Search) 같이 트리를 만들어나가는 형태고, Leaf-wise 는 DF(Depth First Search) 같이 트리를 만들어나가는 형태다. 물론 $\text{max_depth} = -1$ 이면 둘은 같은 형태지만, 대부분의 부스팅 모델에서의 트리는 $\text{max_depth} \neq -1$ 이기 때문에 이 둘을 구분하는 것이다.



Catboost의 특징

2. Ordered Target Encoding

- Target Encoding, Mean Encoding, Response Encoding 이라고 불리우는 기법 (3개 다 같은 말이다.)을 사용한다.
- 범주형 변수를 수로 인코딩 시키는 방법 중, 비교적 가장 최근에 나온 기법인데, 간단한 설명을 하면 다음과 같다.

time	feature 1	class_labels (max_temperature on that day)
sunday	sunny	35
monday	sunny	32
tues	cloudy	15
wed	cloudy	14
thurs	mostly_cloudy	10
fri	cloudy	20
sat	cloudy	25

위 데이터에서 time, feature1로 class_label 을 예측해야한다고 해보자.

$$\text{cloudy} = (15 + 14 + 20 + 25) / 4 = 18.5$$

cloudy 를 cloudy 를 가진 데이터들의 class_label 의 값의 평균으로 인코딩 하는 것이다.
이 때문에 Mean encoding 이라 불리기도 한다

Catboost의 특징

2. Ordered Target Encoding

time	feature 1	class_labels (max_temperature on that day)
sunday	sunny	35
monday	sunny	32
tues	cloudy	15
wed	cloudy	14
thurs	mostly_cloudy	10
fri	cloudy	20
sat	cloudy	25

- 그런데 위는 우리가 예측해야하는 값이 훈련 셋 피처에 들어가버리는 문제, 즉 Data Leakage 문제를 일으킨다. 이는 오버피팅을 일으키는 주 원인이자, Mean encoding 방법 자체의 문제이기도 하다.

- Catboost 는 이에대한 해결책으로, 현재 데이터의 인코딩하기 위해 이전 데이터들의 인코딩된 값을 사한다. 예를 들면 다음과 같다.

- Friday 에는, $\text{cloudy} = (15+14)/2 = 15.5$ 로 인코딩 된다.
- Saturday 에는, $\text{cloudy} = (15+14+20)/3 = 16.3$ 로 인코딩 된다.

즉, 현재 데이터의 타겟 값을 사용하지 않고, 이전 데이터들의 타겟 값만을 사용하니, Data Leakage 가 일어나지 않는 것이다.

TBS (Target Based Statistics)

- 범주형 변수 -> 실수형 변수

- 문제: 새로 들어온 SDE의 값은 무엇으로?

i		SDE		1
		SDE		1
		SDE		0
		PR		
		SDE		1
		PR		

$$i \longrightarrow \frac{1 + 1 + 0 + a * \text{Prior}}{3 + a}$$

Catboost의 특징

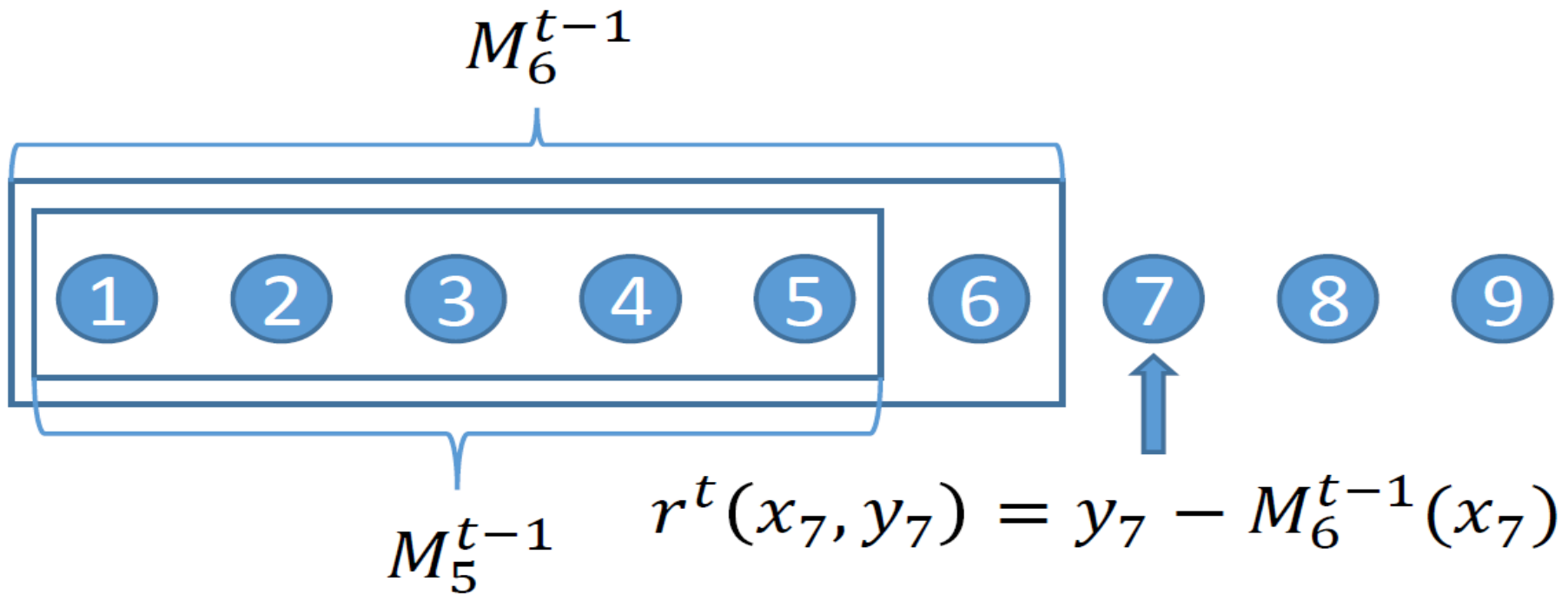
3. Order Boosting

time	datapoint	class label
12:00	x1	10
12:01	x2	12
12:02	x3	9
12:03	x4	4
12:04	x5	52
12:05	x6	22
12:06	x7	33
12:07	x8	34
12:08	x9	32
12:09	x10	12

1. 먼저 x1 의 잔차만 계산하고, 이를 기반으로 모델을 만든다. 그리고 x2 의 잔차를 이 모델로 예측한다.
2. x1, x2 의 잔차를 가지고 모델을 만든다. 이를 기반으로 x3, x4 의 잔차를 모델로 예측한다.
3. x1, x2, x3, x4 의 잔차를 가지고 모델을 만든다. 이를 기반으로 x5, x6, x7, x8 의 잔차를 모델로 예측한다.
4. ... 반복

Ordered Boosting

- Ordered boosting을 도식으로 설명하면 다음과 같다. 7번째 타겟을 6번째까지의 잔차로 만든 모델로 예측하고 7번째 잔차를 구한다. 이제까지의 잔차를 사용해 7번째 모델을 형성한다.



Catboost의 특징

4. 최적화된 파라미터 튜닝

- Catboost 는 기본 파라미터가 기본적으로 최적화가 잘 되어있어서, 파라미터 튜닝에 크게 신경쓰지 않아도 된다고 한다. (반면 xgboost 나 light gbm 은 파라미터 튜닝에 매우 민감하다.)

- 사실 대부분 부스팅 모델들이 파라미터 튜닝하는 이유는, 트리의 다형성과 오버피팅 문제를 해결하기 위함인데, Catboost 는 이를 내부적인 알고리즘으로 해결하고 있으니, 굳이.. 파라미터 튜닝할 필요가 없는 것이다.

- 굳이 한다면 learning_rate, random_strength, L2_regulariser 과 같은 파라미터 튜닝인데, 결과는 큰 차이가 없다고 한다.

Catboost의 한계

- 데이터 대부분이 수치형인 경우, Light GBM보다 학습속도가 느리다. (즉 대부분이 범주형 변수인 경우 성능이 좋다.)

XGBoost, LightGBM, Catboost 구현

구현요약

- pip install xgboost
- pip install lightgbm
- pip install catboost

```
from xgboost import XGBClassifier  
from xgboost import XGBRegressor  
from lightgbm import LGBMClassifier
```

```
from lightgbm import LGBMRegressor  
from catboost import CatBoostClassifier  
from catboost import CatBoostRegressor
```

구현요약

```
#XGBoost
```

```
params={'objective':'reg:linear', 'booster':'gbtree', 'eta':0.03, 'max_depth':10,  
'subsample':0.9, 'colsample_bytree':0.7, 'silent':1, 'seed':10}
```

```
num_boost_round=6000
```

```
dtrain=xgb.DMatrix(Xtrain,ytrain)
```

```
dvalid=xgb.DMatrix(Xtest,ytest)
```

```
watchlist=[(dtrain,'train'),(dvalid,'eval')]
```

```
# 모델 적합화
```

```
gbm=xgb.train(params,dtrain,num_boost_round,evals=watchlist,  
early_stopping_rounds=100,feval=rmspe_xg,verbose_eval=True)
```

```
yhat=gbm.predict(xgb.DMatrix(Xtest))
```

```
error=rmspe(np.expm1(ytest),np.expm1(yhat))
```

구현요약

#LightGBM

```
train_ds = lgb.Dataset(x_train2, label = y_train)
```

```
val_ds = lgb.Dataset(x_val2, label = y_val)
```

```
params = {'learning_rate': 0.01, 'max_depth': 16, 'boosting': 'gbdt', 'objective': 'regression', 'metric': 'auc', 'is_training_metric': True, 'num_leaves': 144, 'feature_fraction': 0.9, 'bagging_fraction': 0.7, 'bagging_freq': 5, 'seed': 2018}
```

#모델 적합화

```
model = lgb.train(params, train_ds, 1000, val_ds, verbose_eval=10, early_stopping_rounds=100)
```

모델 예측

```
yhat= model.predict(x_test2)
```

구현요약

#catboost 사례 (간단한 경우)

```
from catboost import CatBoostClassifier, Pool
train_data = Pool(data=[[1, 4, 5, 6], [4, 5, 6, 7], [30, 40, 50, 60]],
label=[1, 1, -1], weight=[0.1, 0.2, 0.3])
model = CatBoostClassifier(iterations=10)
model.fit(train_data)
preds_class = model.predict(train_data)
```

구현요약

#catboost 사례: 최고의 결과

```
from catboost import CatBoostClassifier, Pool
train_data = [[0, 3], [4, 1], [8, 1], [9, 1]]
train_labels = [0, 0, 1, 1]
eval_data = [[2, 1], [3, 1], [9, 0], [5, 3]]
eval_labels = [0, 1, 1, 0]
eval_dataset = Pool(eval_data, eval_labels)
model = CatBoostClassifier(learning_rate=0.03, custom_metric=['Logloss',
'AUC:hints=skip_train~false' ])
model.fit(train_data, train_labels, eval_set=eval_dataset, verbose=False)
print(model.get_best_score())
```

#catboost 사례: 최고의 반복시행

```
model = CatBoostClassifier(learning_rate=0.03, eval_metric='AUC' )
model.fit(train_data, train_labels, eval_set=eval_dataset, verbose=False)
print(model.get_best_iteration())
```

구현요약

#catboost 표준 구현 (Pool을 사용, 아니면 싸이킷런과 동일)

```
from catboost import Pool, CatBoostClassifier
train_data = [["summer", 1924, 44], ["summer", 1932, 37], ["winter", 1980, 37], ["summer", 2012, 204]]
eval_data = [["winter", 1996, 197], ["winter", 1968, 37], ["summer", 2002, 77], ["summer", 1948, 59]]
cat_features = [0]
train_label = ["France", "USA", "USA", "UK"]
eval_label = ["USA", "France", "USA", "UK"]
train_dataset = Pool(data=train_data, label=train_label, cat_features=cat_features)
eval_dataset = Pool(data=eval_data, label=eval_label, cat_features=cat_features)

# Initialize CatBoostClassifier
model = CatBoostClassifier(iterations=10, learning_rate=1, depth=2,
loss_function='MultiClass' )

# Fit model
model.fit(train_dataset)
# Get predicted classes
preds_class = model.predict(eval_dataset)
# Get predicted probabilities for each class
preds_proba = model.predict_proba(eval_dataset)
# Get predicted RawFormulaVal
preds_raw = model.predict(eval_dataset, prediction_type='RawFormulaVal')
```

구현요약: 하이퍼파라미터

- 1.XGBoost가 사이킷런 규칙에 따라 자신의 하이퍼 파라미터 변경함
- 2.LightGBM은 XGBoost와 기능이 많이 유사해서 사이킷런 래퍼의 파라미터를 XGBoost에 맞춰서 변경함

3.예:

유형	파이썬 래퍼 LightGBM	사이킷런 래퍼 LightGBM	사이킷런 래퍼 XGBoost
	num_iterations	n_estimators	n_estimators

4. 기본적으로 n_estimators, max_depth와 learning_rate이 중요함.

예제 catboost: parameters = { ' depth ' : [6,8,10], ' learning_rate ' : [0.01, 0.05, 0.1], ' iterations ' : [30, 50, 100] }

참고 catboost에서 learning_rate을 eta라고도 한다.

5. 참조 (코딩+하이퍼파라미터)

<http://incredible.egloos.com/m/7478695>

<http://incredible.egloos.com/m/7479081>