Data Analysis:
Machine Learning Base Model(1)

ESC 2024 Summer Special Session 3차
발제자: 심재윤

목차 / List

1

Decision Tree

Pruning
Split Criterion
Information gain

Decision Tree

Decision Tree란?

데이터를 분석하고 예측 모델을 만들기 위해 사용되는 통계적 방법이다.

분류와 회귀분석에 모두 적용 가능한데, 분류 분석에는 대상 변수가 범주형 변수이고(Classification Tree), 회귀분석에서는 연속적인 값일 경우(Regression Tree) 사용된다.

Node 와 Edge로 이루어진 Tree 모형은 특정 변수에 따라 하위 노드가 분리된다.

핵심 원리는 정보이론 (Information Theory)에서 유래한다. 정보이론은 데이터의 불확실성을 측정하는데 사용되고, Decision Tree는 각 분기에서 최대한 많은 정보를 얻을 수 있는 변수를 선택한다.

Decision Tree

왜 사용하는가?

분석 결과를 해석하기 쉬우며, 변수 간의 상호작용과 비선형성을 반영하여 정확한 예측이 가능하다.

모델의 직관성과 가시성이 높아, 어떤 결정에 대한 이유를 설명하기 용이하다

이러한 Decision Tree는 비즈니스 분야에서 마케팅, 고객분석, 고객이탈 예측 등에 활용되고, 의료 분야에서는 질병예측, 진단, 치료법 추천 등에 적용된다.

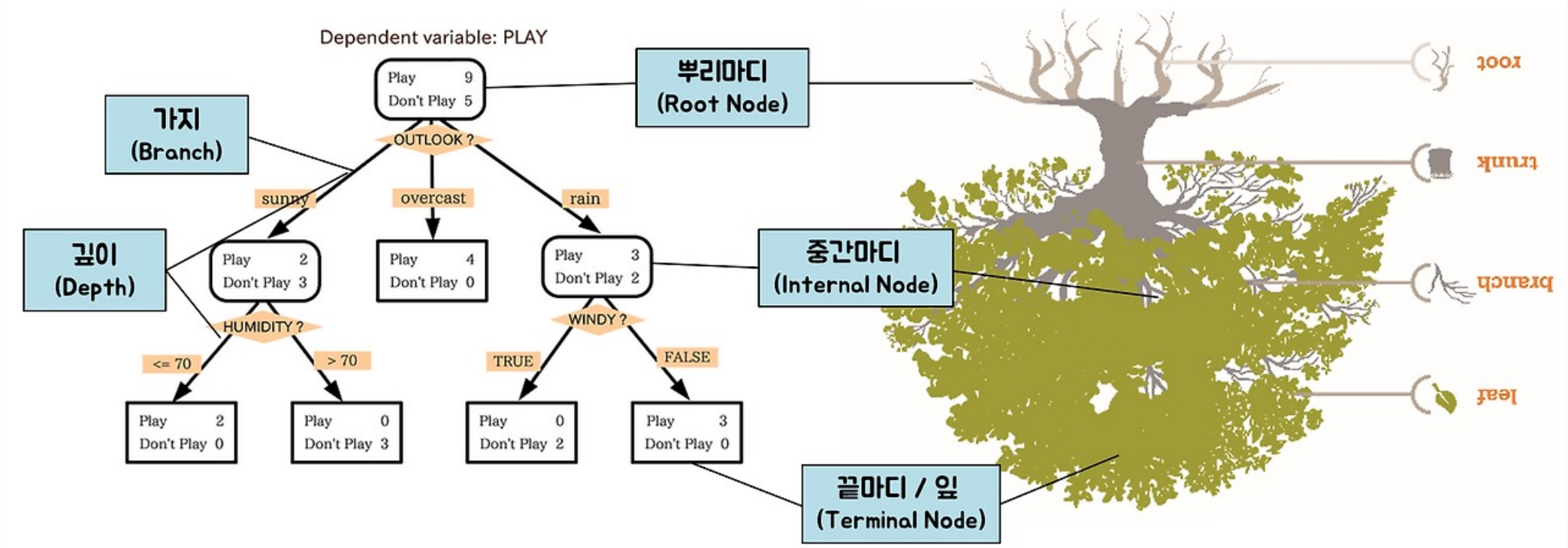
Decision Tree

결정 트리 모델

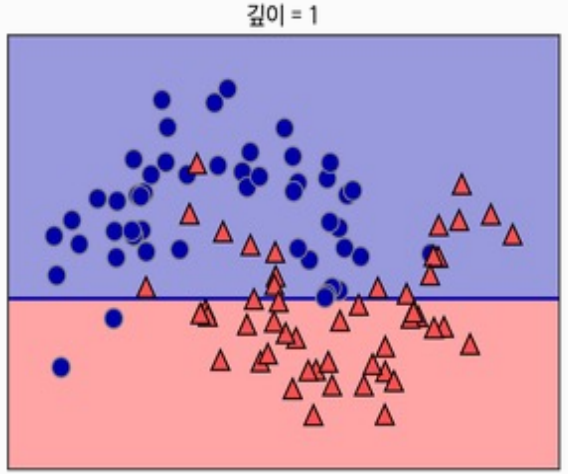
:특정 기준(질문)에 따라 데이터를 구분하는 모델

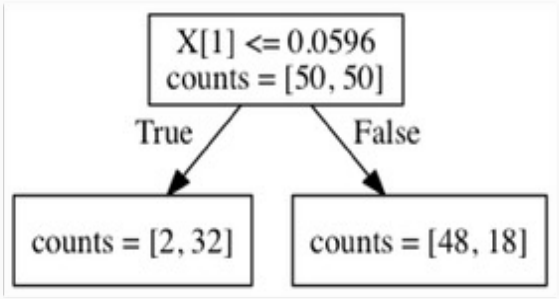
-한번의 분기 때마다 변수 영역을 두 개로 구분한다.

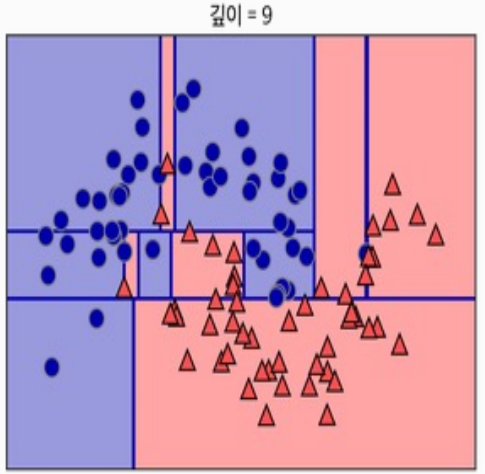
-맨 처음 분류 기준 (즉, 첫 질문)을 Root Node라고 하고, 맨 마지막 노드를 Terminal Node 혹은 Leaf Node라고 한다.

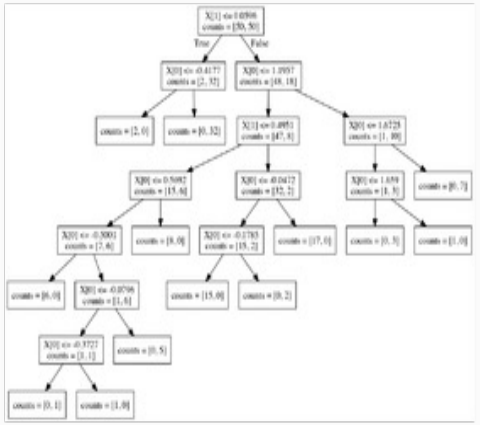


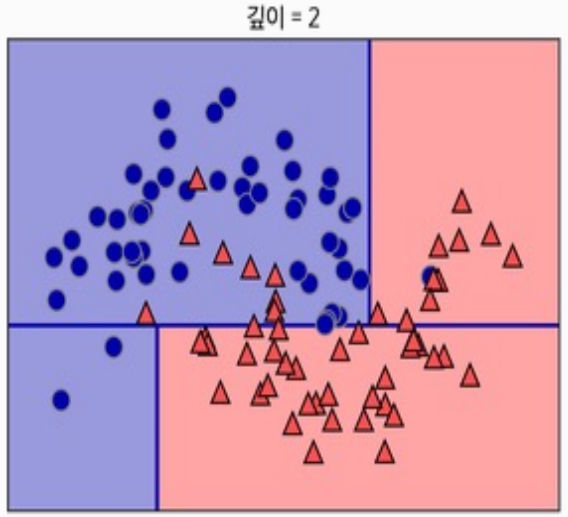
Process of Decision Tree

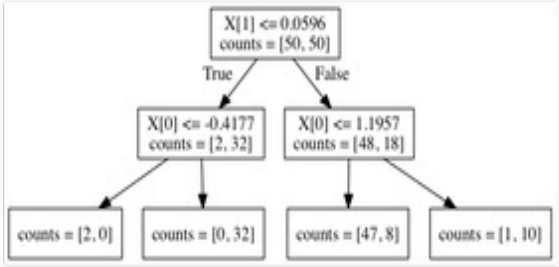












데이터 분류를 지나치게 많이 하면
즉, 결정 트리에 아무 파라미터를 주지 않고 모델링하면
오버피팅이 발생하여 일반화 성능이 떨어진다.

Pruning of Decision Tree (가지치기)

그렇다면 어떻게 오버피팅을 막을 수 있을까?

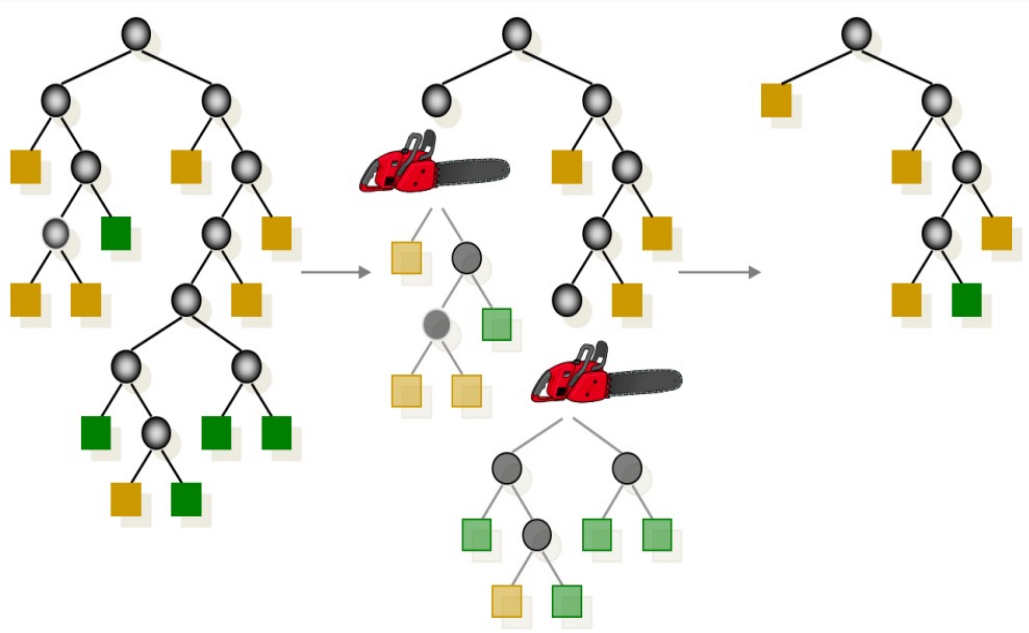
모든 terminal node의 순도가 100%인 상태를 Full tree라고 하는데, 이렇게 Full tree를 생성한 뒤 적절한 수준에서 terminal node를 제거하는 것을 Pruning이라 한다.

Pruning(가지치기)

- 최대 깊이나 터미널 노드의 최대 개수, 혹은 한 노드가 분할하기 위한 최소 데이터 수를 제한하는 것이다.

- Min_sample_split = 10 -> 한 노드에 10개의 데이터가 있다면 더이상 분기하지 않는다.

- Max_depth = 4 -> 깊이가 4보다 크게 가지를 치지 않는다.



Split Criterion: Impurity Algorithm

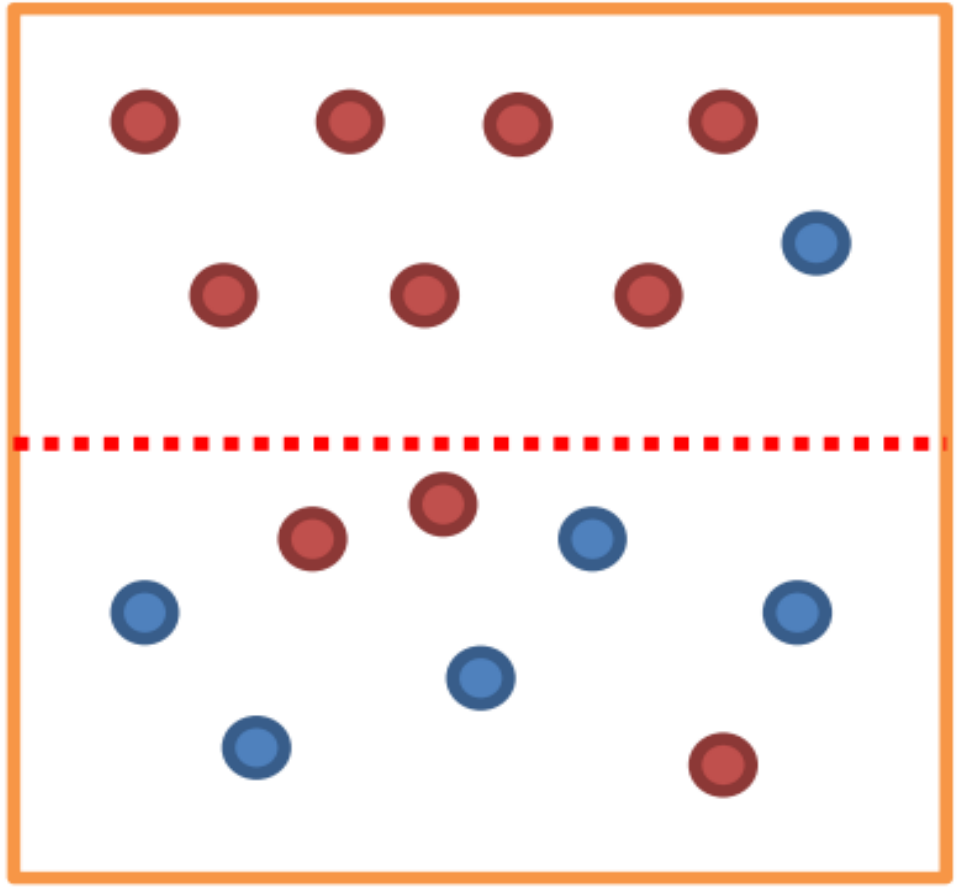
Decision Tree의 분할 기준은 그룹이 최대한 동질 하도록 반복적으로 레코드 하위를 분리한다.

즉, 불순도 알고리즘을 이용한 계산식에 의해 Tree를 구성한다.

불순도(Impurity)
:해당 범주 안에 서로 다른 데이터가 얼마나 섞여 있는지를 뜻한다.

한 범주에 하나의 데이터만 있다면 불순도가 최소이고, 한 범주 안에서 서로 다른 두 데이터가 정확히 반반 있다면 불순도가 최대가 된다.

결정 트리는 불순도를 최소화 하는 방향으로 학습을 진행하는데, 이를 Impurity Algorithm이라 한다..



Split Criterion: Impurity Algorithm

엔트로피(Entropy)는 불순도(Impurity)를 수치적으로 나타낸 척도이다.

엔트로피가 높다는 것은 불순도가 높다는 뜻이고, 엔트로피가 낮다는 것은 불순도가 낮다는 뜻이다.

엔트로피가 1이면 불순도가 최대가 되고, 그 뜻은 한 범주 안에 서로 다른 데이터가 정확히 반반 있다는 뜻이다.

$$\text{Entropy} = -\sum_i (p_i) \log_2 (p_i)$$

(Pi = 한 영역 안에 존재하는 데이터 가운데, 범주 i에 속하는 데이터의 비율)

따라서 엔트로피를 계산하였을 때 가장 낮은 엔트로피를 생성하는 분류기준을 선택하는 것이다.

경사	표면	속도 제한	속도
steep	bumpy	yes	slow
steep	smooth	yes	slow
flat	bumpy	no	fast
steep	smooth	no	fast

Information gain: Greedy Algorithm

정보 획득(Information gain)
: 특정 분할 기준에 의해 데이터가 얼마나 잘 분할되는지 측도
(분기 이전의 엔트로피에서 분기 이후의 엔트로피를 뺀 값)

Ex) 엔트로피가 1인 상태에서 0.7인 상태로 바뀌었다면 정보 획득(information gain)은 0.3이다.

Information gain = entropy(parent) – [weighted average] entropy(children)

분기 이후 엔트로피에 대해 가중 평균을 하는 이유는 분기를 하면 범주가 2개 이상으로 쪼개지기 때문이다.

결정 트리 알고리즘은 정보 획득을 최대화하는 방향으로 학습이 진행된다.

즉, 어느 feature의 어느 분기점에서 정보 획득이 최대화되는지 판단을 해서 분기가 진행된다.

이러한 알고리즘을 Greedy Algorithm이라고 한다.

Information gain: Greedy Algorithm

경사	표면	속도 제한	속도
steep	bumpy	yes	slow
steep	smooth	yes	slow
flat	bumpy	no	fast
steep	smooth	no	fast

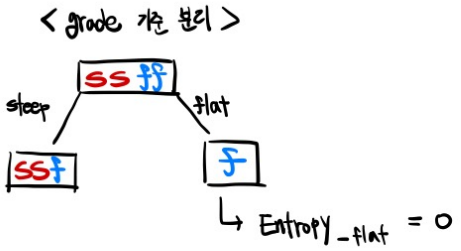
Let i : slow, fast

$\Rightarrow P_{slow} = 0.5 \;,\; P_{fast} = 0.5$

$\Rightarrow \text{Entropy} = -\sum_i (P_i) \log_2 (P_i)$

$= -0.5 \log_2(0.5) - 0.5 \log_2(0.5)$

$= 1$ (Parent)



- $\text{Entropy\_steep} = -\frac{2}{3}\log\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}$

$= 0.9184$

- [weighted average] entropy (children) = W.A of steep * entropy_steep + W.A of flat * entropy_flat

$= \frac{3}{4} * 0.9184 + \frac{1}{4} * 0$

$= 0.6888$

$\Rightarrow$ Information gain = entropy (Parent) - [weighted average] entropy (children)

$= 1 - 0.6888$

$= 0.3112$

Decision Tree 실습

Classifier를 만들고, fitting한 뒤, Test해보는 방식으로 모델을 사용할 수 있다.

결정 트리의 default는 max_depth, min_sample_split 제한이 없으므로 한 범주에 한 종류의 데이터가 남을 때까지 가지를 친다.

따라서 훈련 세트의 정확도는 100%인데 테스트 세트의 정확도는 93.7%이다.

반면, max_depth=4로 설정해주면 오버피팅을 막아 훈련 세트 정확도는 전 보다 떨어지지만 테스트 세트 정확도가 더 높아졌습니다.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("훈련 세트 정확도: {:.3f}".format(tree.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(tree.score(X_test, y_test)))

>>> 훈련 세트 정확도: 1.000
>>> 테스트 세트 정확도: 0.937
```

```
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)

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

>>> 훈련 세트 정확도: 0.988
>>> 테스트 세트 정확도: 0.951
```

2
Ensemble Learning

Bagging
Boosting

Ensemble Learning

Ensemble

: 함께, 동시에, 협력하여 등을 뜻하는 프랑스어

Decision Tree는 분석과정과 결과를 직관적으로 이해할 수 있기 때문에 설명력이 필요한 경우에 많이 쓰인다.

하지만 오버피팅 문제가 발생하여 정확도가 떨어질 위험이 크다. 때문에 ' 앙상블 ' 기법을 사용한다.

그 종류에는 투표(Majority Voting), 배깅(Bagging), 부스팅(Boosting)이 있다.

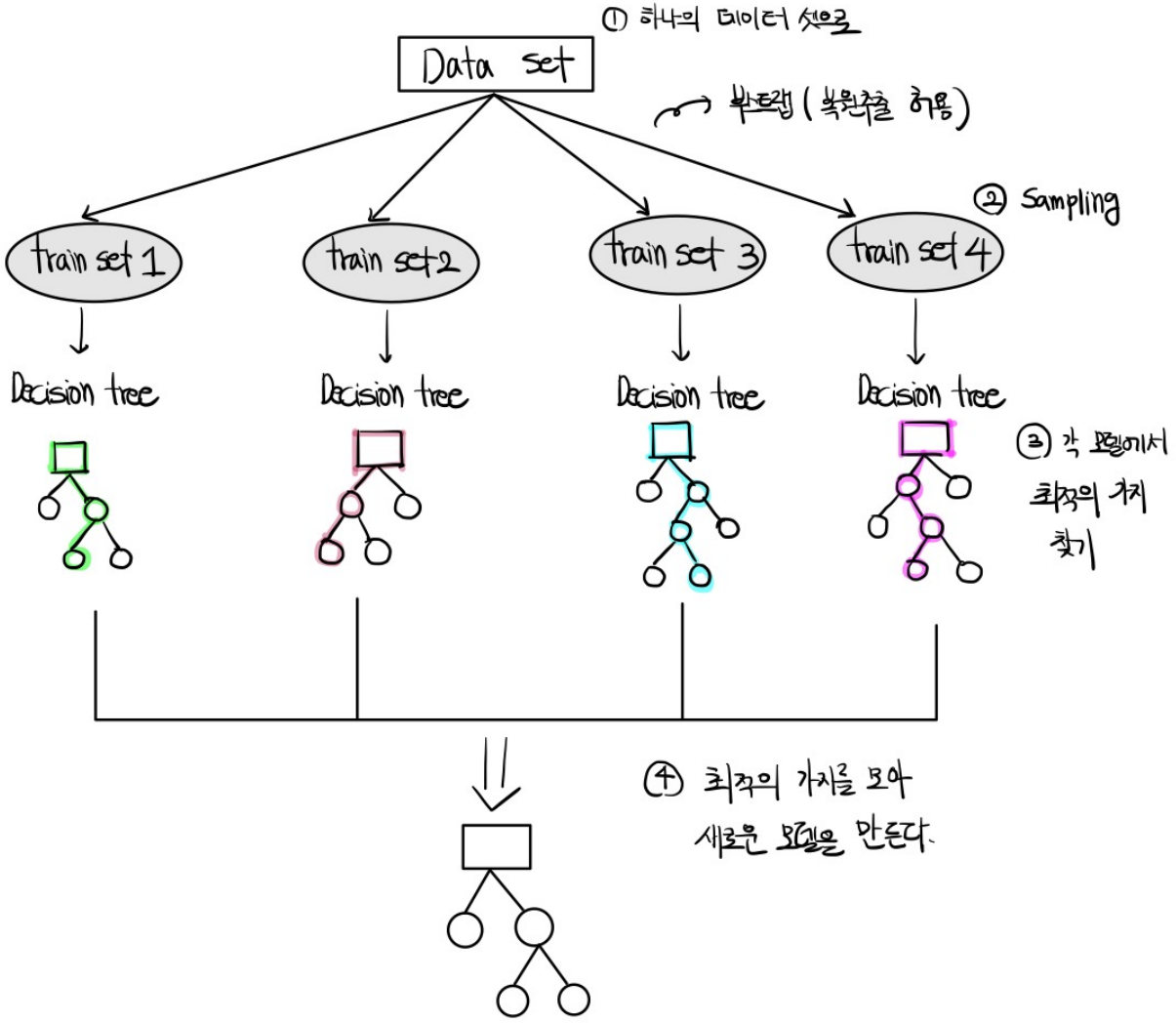
Ensemble Learning: Bagging

Bagging

1. 하나의 data set을 가지고 여러 개의 train set을 만든다.

2. 이 때 Random sampling with replacement 를 사용하여 train set을 추출한다.

3. 각 train set으로 모델을 만드는데, 이 때 decision tree를 모델로 사용하면 최적의 가지를 선택하여 결합한 결과는 random forest가 된다.



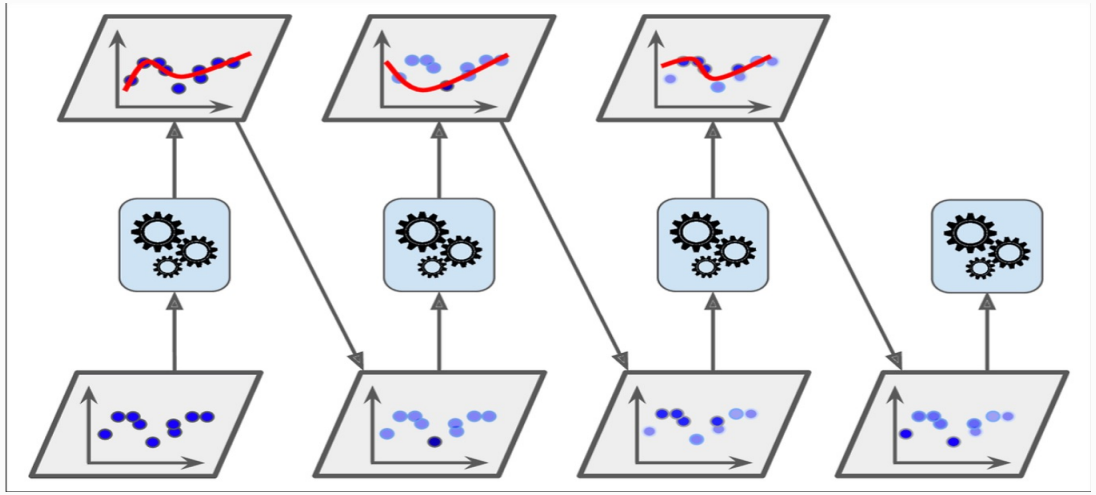
Ensemble Learning: Boosting

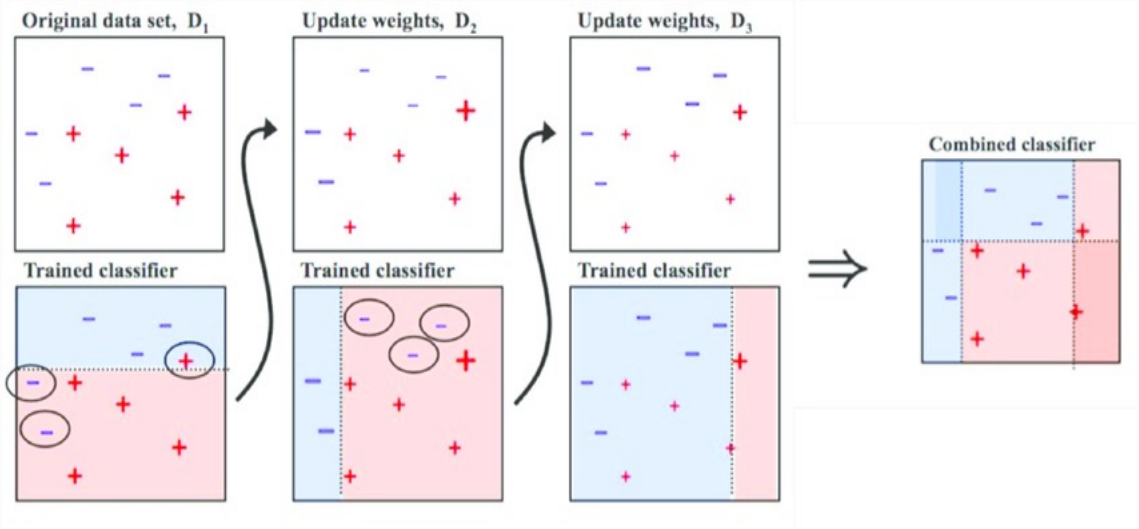
Boosting

전체 훈련 세트에서 random sampling 하여
잘못 분류된 데이터에 가중치를 적용하는 방법이다.

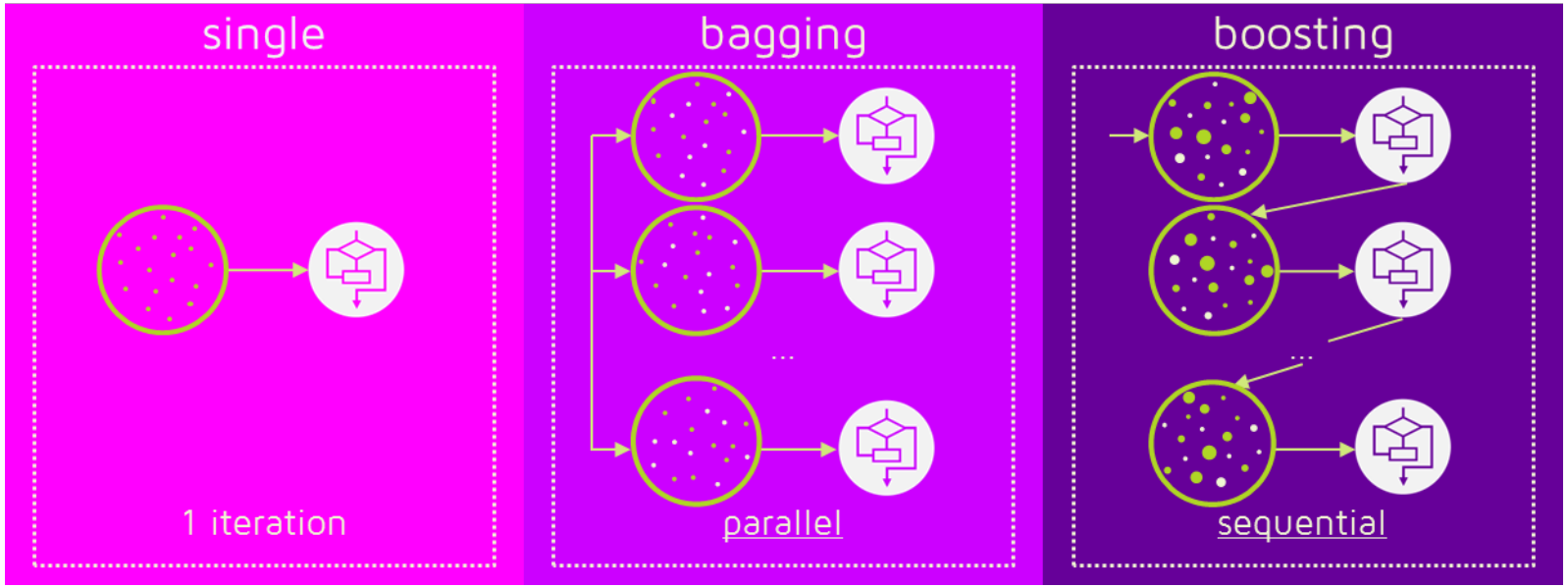
즉, 추출된 데이터로 하나의 트리를 만들고, 이 때 오차를 계산하여 그 오차가 작아지도록 가중치를 주어 다음 모델을 만들 때 그것을 반영하는 방법이다.

이렇게 이전에 만들었던 tree의 오차를 개선하여 다음 모델을 만들기 때문에 voting, bagging 보다 성능이 비교적 좋다.





Ensemble Learning



위 그림에서 나타내는 바와 같이 bagging은 병렬로 학습하는 반면, boosting은 순차적으로 학습한다.

오답에 대해서는 높은 가중치를 부여하고, 정답에 대해서는 낮은 가중치를 부여하기 때문에 오답을 정답으로 맞추기 위해 오답에 더 집중할 수 있게 되는 것이다. 따라서 Boosting이 Bagging에 비해 error가 적다. 즉, 성능이 좋다. 하지만 속도가 느리고 오버 피팅이 될 가능성이 있다.

그렇다면 실제 사용할 때는 Bagging과 Boosting 중 어떤 것을 선택해야 할까? 상황에 따라 다르다고 할 수 있는데, 개별 Decision Tree의 낮은 성능이 문제라면 Boosting이 적합하고, 오버 피팅이 문제라면 Bagging이 적합하다.

3
Random Forest

Random Forest

나무가 모여 숲을 이루듯, 결정 트리(Decision Tree)가 모여 랜덤 포레스트(Random Forest)를 구성한다.

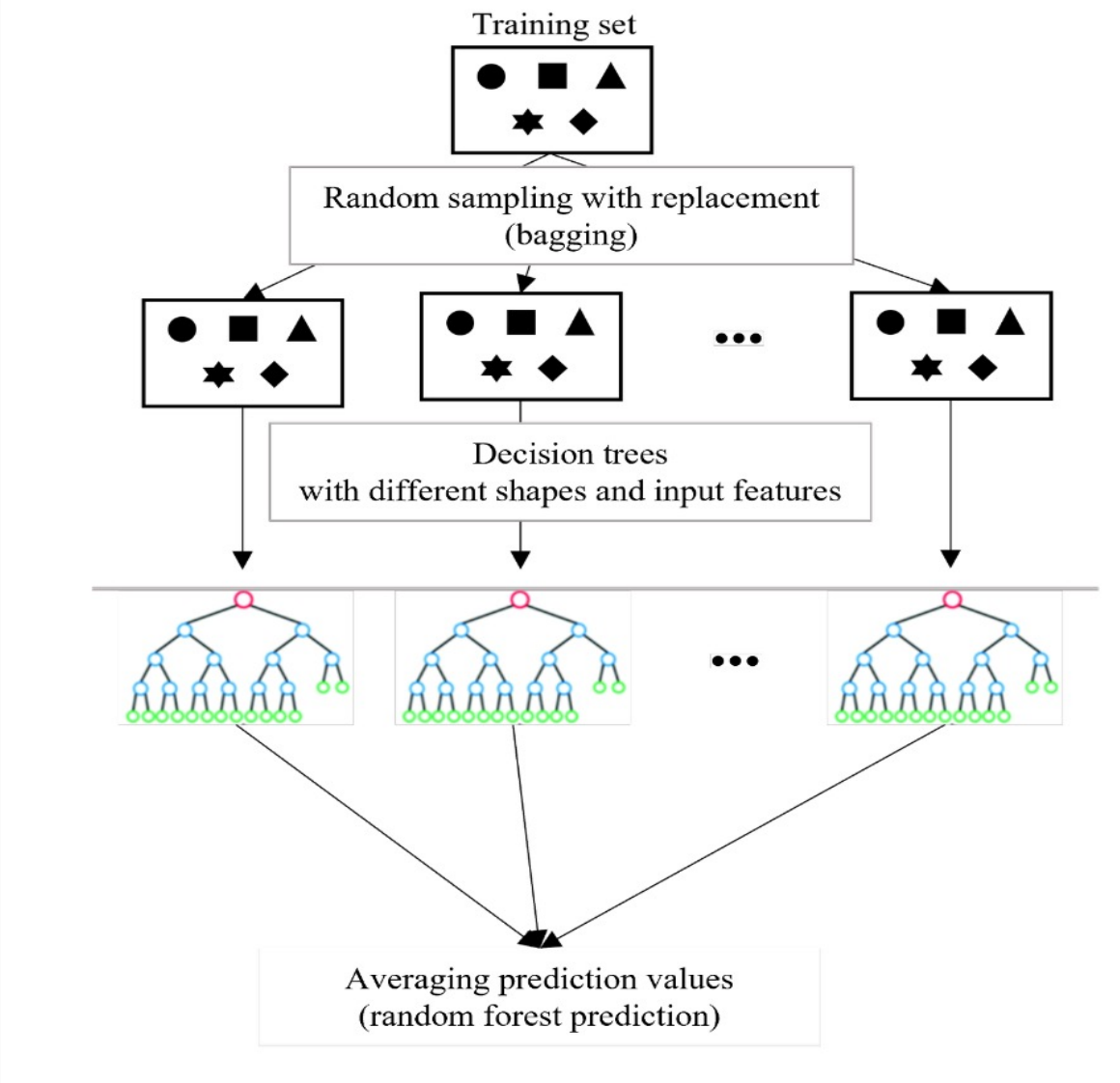
Decision Tree 하나만으로도 머신러닝을 할 수 있지만 Decision Tree의 단점은 훈련 데이터에 오버피팅이 되는 경향이 있다는 것이다.

여러 개의 Decision Tree 를 통해 Random Forest 를 만들면 오버피팅 되는 단점을 해결할 수 있다.

Random Forest는 의사결정나무에 Bagging이라는 앙상블 학습을 적용한 모델이고, Bagging은 분산을 줄이기 위해 사용된다.

Random Forest 는 오버피팅을 해소하고 분산을 감소시켜 정확도가 높다는 장점이 있지만, 계산 비용이 높고 규칙이 많아 추론 로직을 설명하기 어렵다는 단점도 있다.

Random Forest



Random Forest

Ex)
건강의 위험도를 예측하기 위해서 성별, 키, 몸무게, 지역, 운동량, 흡연유무, 음주 여부, 혈당, 근육량, 기초 대사량 등 수많은 요소가 필요하다. 이렇게 수많은 요소(Feature)를 기반으로 건강의 위험도(Label)를 예측한다면 분명 오버피팅이 일어날 것이다.

Feature가 30개라고 하자. 30개의 Feature를 기반으로 하나의 결정 트리를 만든다면 트리의 가지가 많아질 것이고, 이는 오버피팅의 결과를 야기할 것이다.

하지만 30개의 Feature 중 랜덤으로 5개의 Feature만 선택해서 하나의 결정 트리를 만들고, 또 30개 중 랜덤으로 5개의 Feature를 선택해서 또 다른 결정 트리를 만들고, 이렇게 계속 반복하여 여러 개의 결정 트리를 만들 수 있다.

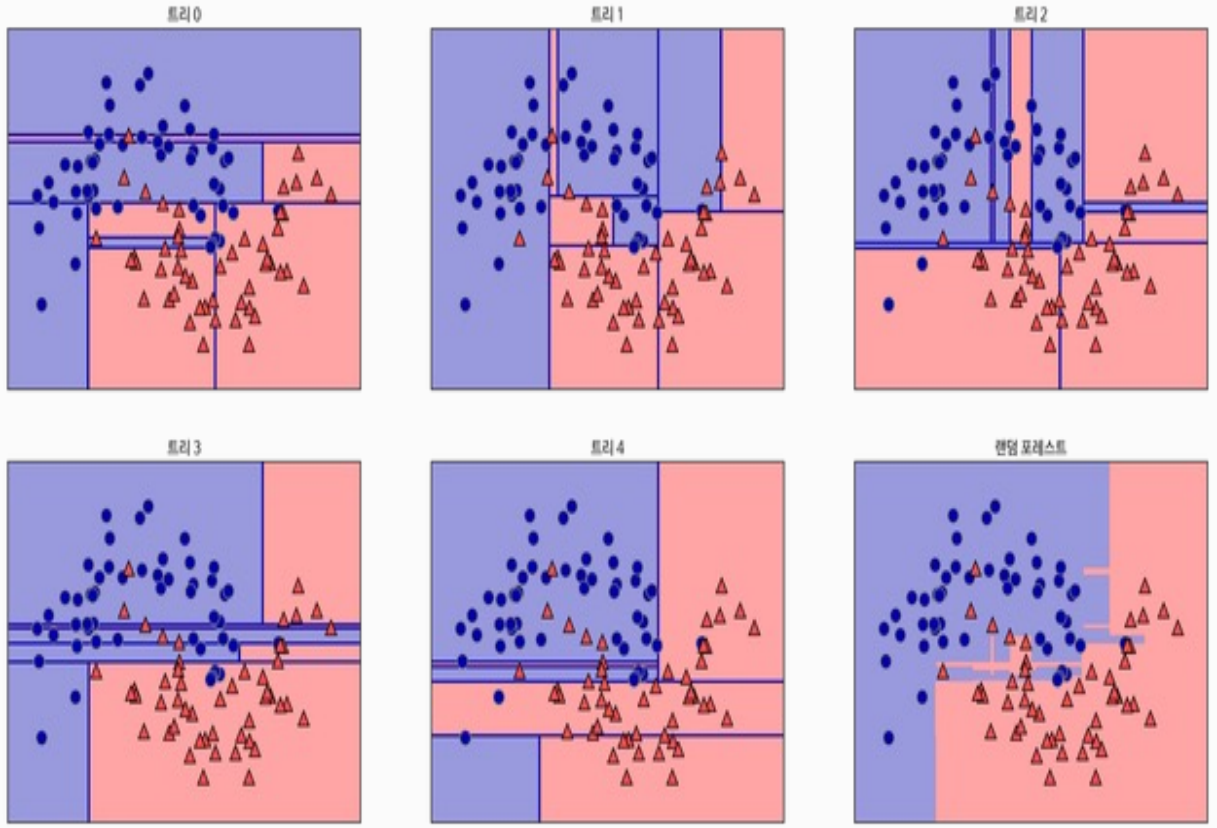
결정 트리 하나마다 예측 값을 내놓으면 여러 결정 트리들이 내린 예측 값들 중 가장 많이 나온 값을 최종 예측값으로 정하게 된다. 즉, 다수결의 원칙에 따르는 것이다.

Random Forest

즉, 하나의 거대한 (깊이가 깊은) 결정 트리를 만드는 것이 아니라 여러 개의 작은 결정 트리를 만드는 알고리즘이다.

여러 개의 작은 결정 트리가 예측한 값들 중 가장 많은 값(분류일 경우) 혹은 평균값(회귀일 경우)을 최종 예측 값으로 정하는 것이다.

문제를 풀 때도 한 명의 똑똑한 사람보다 100 명의 평범한 사람이 더 잘 푸는 원리이다.



Random Forest 실습

n_estimators: 랜덤 포레스트 안의 결정 트리 개수

n_estimators는 클수록 좋다. 결정 트리가 많을수록 더 깔끔한 Decision Boundary가 나오기 때문이다. 하지만 그만큼 메모리와 훈련 시간이 증가한다.

max_features: 무작위로 선택할 Feature의 개수

max_features가 전체 Feature 개수와 같으면 전체 Feature 모두를 사용해 결정 트리를 만든다. 또한 기본값인 bootstrap=True이면 전체 Feature에서 복원 추출해서 트리를 만든다.

max_features 값이 크면 Random Forest의 Tree들이 서로 매우 비슷해지고, 가장 두드러진 특성에 맞게 예측을 할 것이다. max_features 값이 작으면 랜덤 포레스트의 트리들이 서로 매우 달라진다. (오버피팅이 줄어드는 효과가 있는 것) 기본값을 사용하는 것이 가장 좋다.

```python
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# 데이터 로드
iris = load_iris()
X, y = iris.data, iris.target

# 학습 데이터와 테스트 데이터로 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 랜덤 포레스트 모델 생성 및 학습
clf = RandomForestClassifier(n_estimators=100, max_depth=3, random_state=42)
clf.fit(X_train, y_train)

# 예측 및 평가
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# 중요 특징 시각화
feature_importances = clf.feature_importances_
features = iris.feature_names
importance_df = pd.DataFrame({'Feature': features, 'Importance': feature_importances})

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importance in Random Forest')
plt.show()
```

4

Gradient Boost

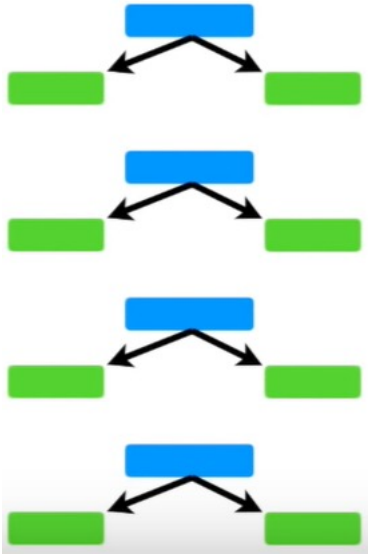
AdaBoost
Process of Gradient Boost

AdaBoost

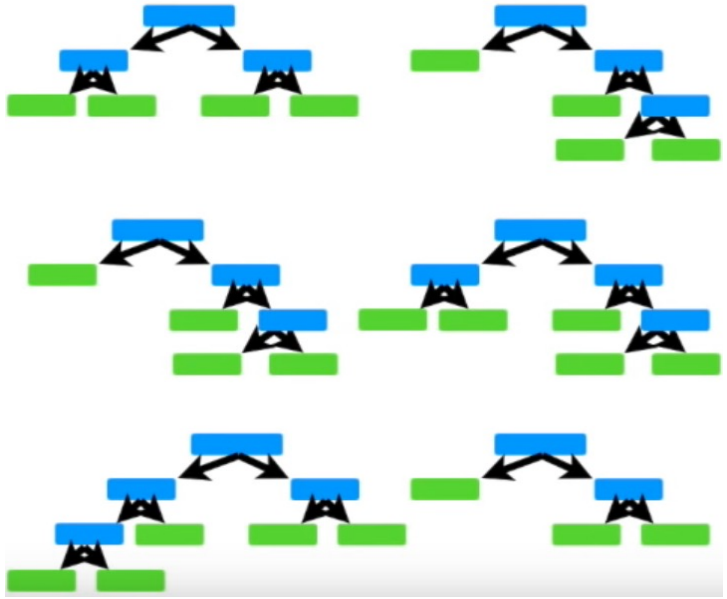
먼저 Boosting 기법의 가장 기본이 되는 AdaBoost에 대해 알아보자.

Stump
두 개의 leaf를 지닌 Tree를 Stump라고 한다.

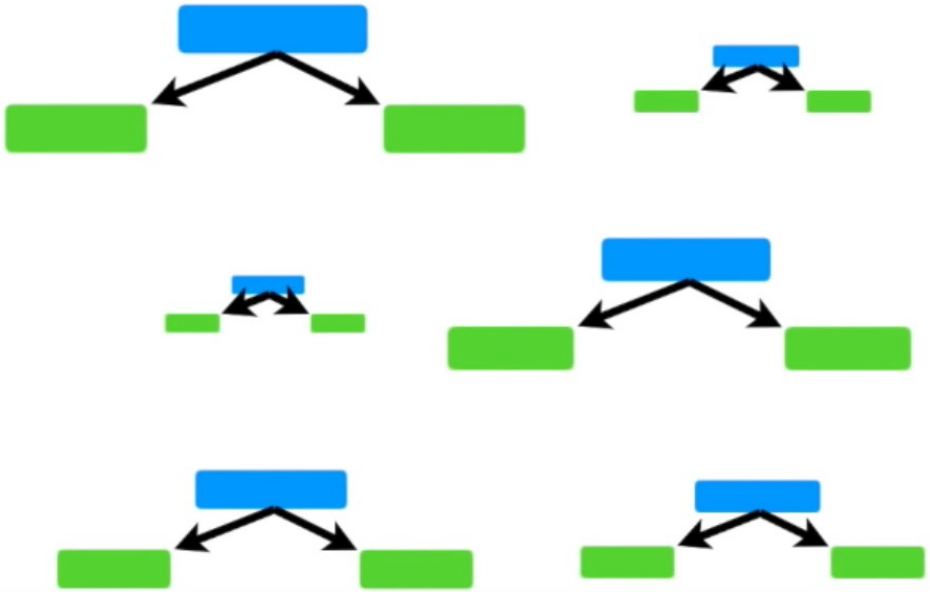
AdaBoost는 여러 개의 Stump로 구성되어 있고, 이러한 구조를 Forest of Stump라고 한다.
Tree와 다르게 질문이 하나밖에 없기 때문에 정확한 분류를 하지 못한다. (weak learner)



AdaBoost



Random Forest는 여러 트리들이 동등한 가중치를 가지고 있다.



그에 반해 AdaBoost에서는 특정 Stump가 다른 Stump보다 더 가중치가 높다.
이를 결과에 미치는 영향이 크다는 의미로 Amount of Say가 높다고 표현한다.

또한 순차적으로 Stump의 error가 다음 Stump의 결과에 영향을 준다.

Gradient Boost

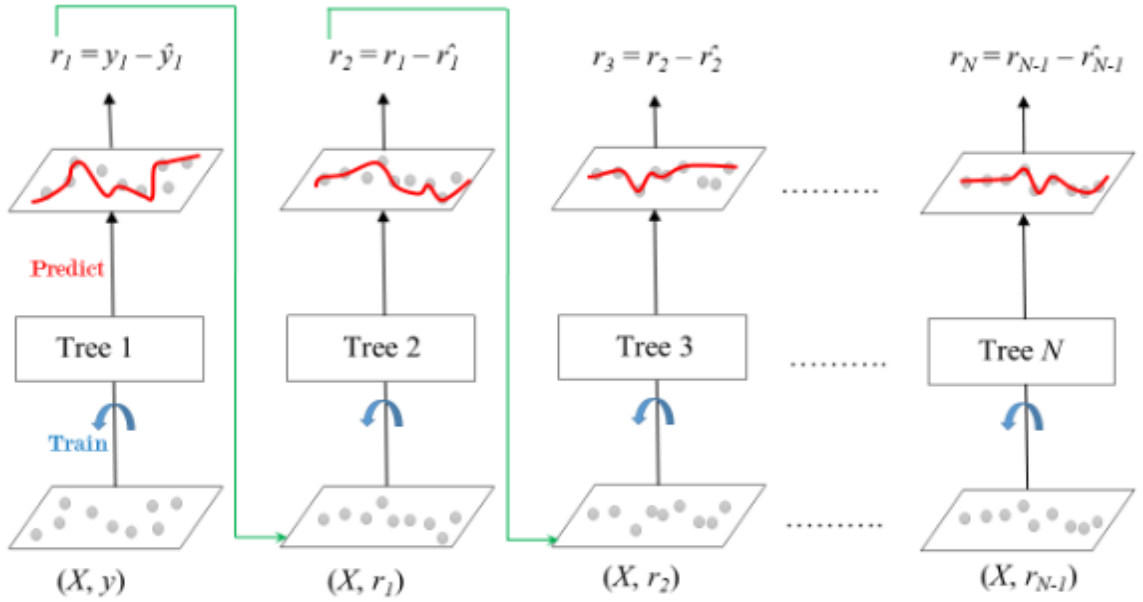
순차적으로 Decision Tree를 추가하여 예측 성능을 향상시키는 앙상블 학습 방법

각 tree는 오차를 줄이기 위해 학습되며, 손실 함수를 최소화 하는 방향으로 트리를 추가한다.

기본적으로 Gradient Boost 는 무작위성이 없고, 대신 사전 가지치기가 사용된다.

따라서 Gradient Boosting Tree는 보통 하나에서 다섯 정도의 깊지 않은 트리를 사용하므로 메모리를 적게 사용하고, 예측도 빠르다.

Gradient Boost 의 근본 아이디어는 이러한 얕은 Tree 같은 간단한 모델 (weak learner)들을 많이 연결하는 것이다.



Gradient Boost

AdaBoost는 Stump로 구성되어 있다. 하나의 Stump에서 발생한 error가 다음 stump에 영향을 주며 최종 결과를 도출하게 된다.

반면, Gradient Boost는 Stump나 Tree가 아닌 하나의 Leaf 부터 시작한다.

Leaf는 초기 추정 값을 나타내고 이를 보통 평균으로 정한다. 그 다음은 AdaBoost와 동일하게 이전 Tree의 error는 다음 Tree에 영향을 준다. 하지만 AdaBoost와 다르게 Stump가 아닌 Tree로 구성되어 있다.

보통 Leaf가 8개에서 32개 되는 Tree로 구성한다.

Process of Gradient Boost

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

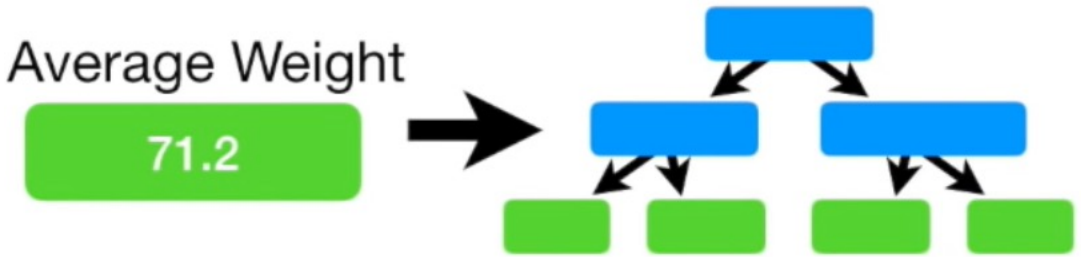
키, 좋아하는 색, 성별을 기반으로 몸무게를 예측해보자.

Gradient Boost는 Leaf부터 시작하며, 그 Leaf 모델이 예측하는 타겟 추정 값은 모든 타겟 값의 평균이다.

(88 + 76 + 56 + 73 + 77 + 57) / 6 = 71.2

따라서, 여기서 학습을 멈추었을 때 Leaf로 몸무게를 예측한다면 모든 사람은 71.2kg이라고 할 것이다.

Process of Gradient Boost



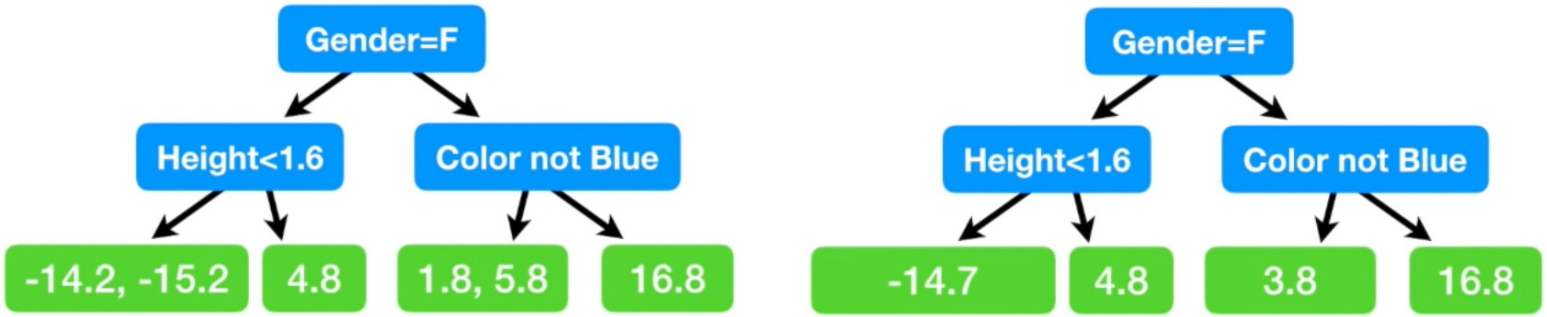
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

이제 Leaf에서 예측한 값과 실제 값의 error를 반영한 새로운 트리를 만들어야 한다.

먼저, Weight – Average Weight 로 Pseudo Residual을 구해준다.

다음으로 키, 좋아하는 색, 성별로 몸무게가 아닌 몸무게의 Residual을 예측해보자.

Process of Gradient Boost



성별로 첫 번째 노드를 나누고, Leaf에 값이 2개 이상이면 평균 값으로 바꾸어 준다.

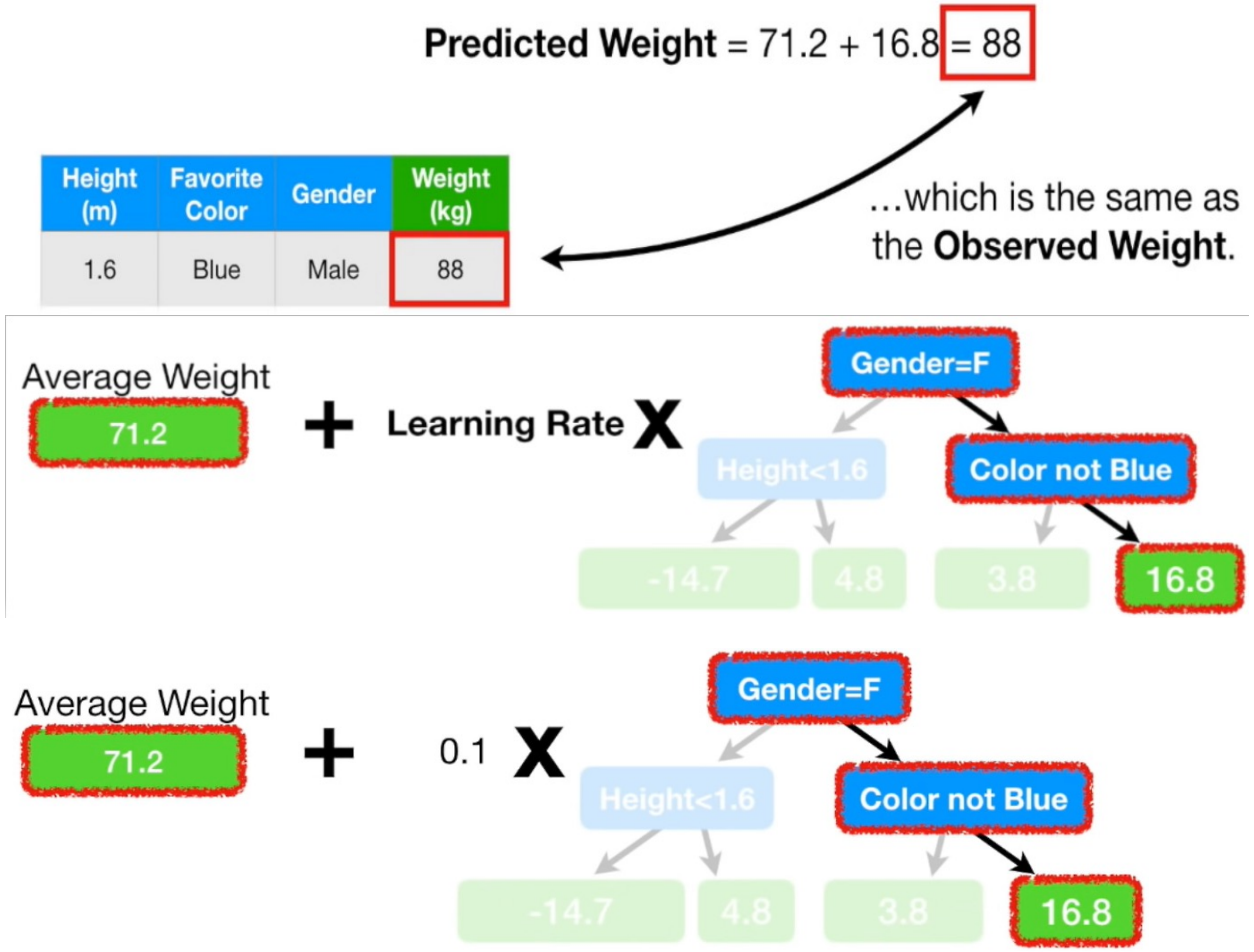
초기에 구한 Tree와 두번째로 구한 트리를 조합하여 몸무게를 예측해보자.

두번째 트리에서 성별이 남자고, 좋아하는 색이 파란색면 residua을 16.8로 예측했다.

따라서 성별이 남자고, 좋아하는 색이 파란색이면 그 사람의 몸무게는 88kg라고 예측을 한 것이다.

…so the **Predicted Weight** = 71.2 + 16.8 = 88

Process of Gradient Boost



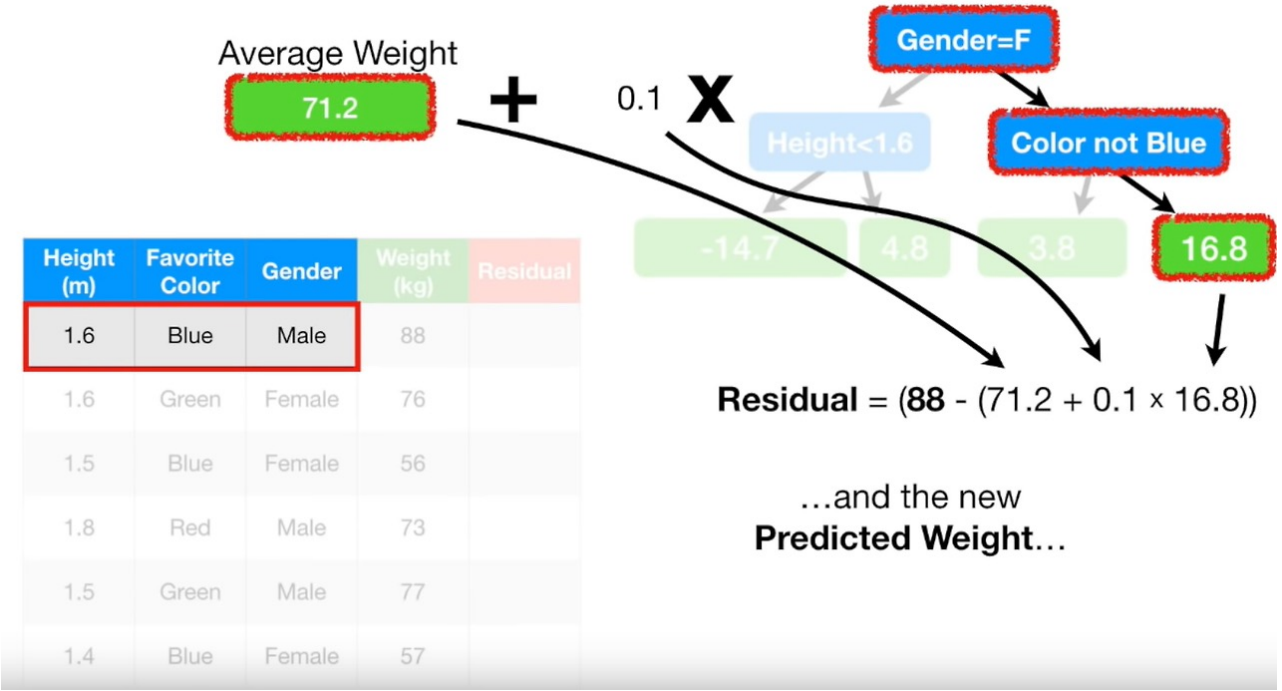
실제 몸무게와 예측이 일치한다! 사실 놀랄게 아니라 오버피팅인 상황이다.
Bias는 작지만, Variance는 클 수 있는 것이다.

따라서 훈련속도나 오버피팅을 조절하기 위해 학습률(Learning Rate)라는 값을 활용해야 하고, 보통 0.1~0.001의 값을 가진다.

학습률이 0.1인 경우 모델이 예측한 몸무게는 72.9kg이다. 처음 예측한 71.2kg보다 실제 값 88kg에 더 가까워졌다.

Gradient Boost 모델은 이런식으로 실제 값에 조금씩 가까워지는 방향으로 학습을 한다.

Process of Gradient Boost



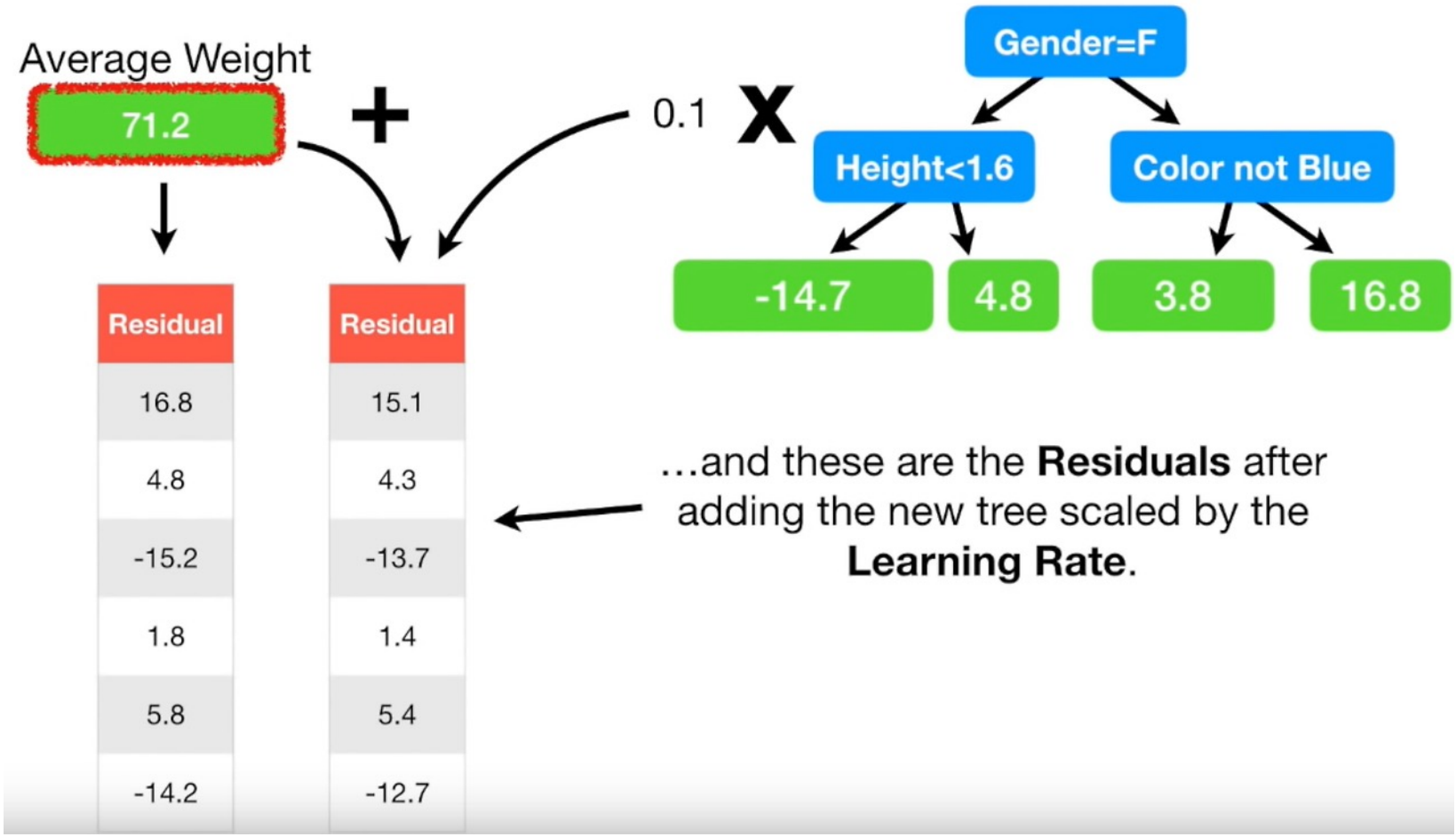
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	
1.6	Green	Female	76	
1.5	Blue	Female	56	
1.8	Red	Male	73	
1.5	Green	Male	77	
1.4	Blue	Female	57	

올바른 방향으로 한 걸음 더 다가가보자.

지금까지 만든 모델로 Pseudo Residual을 다시 구한다.

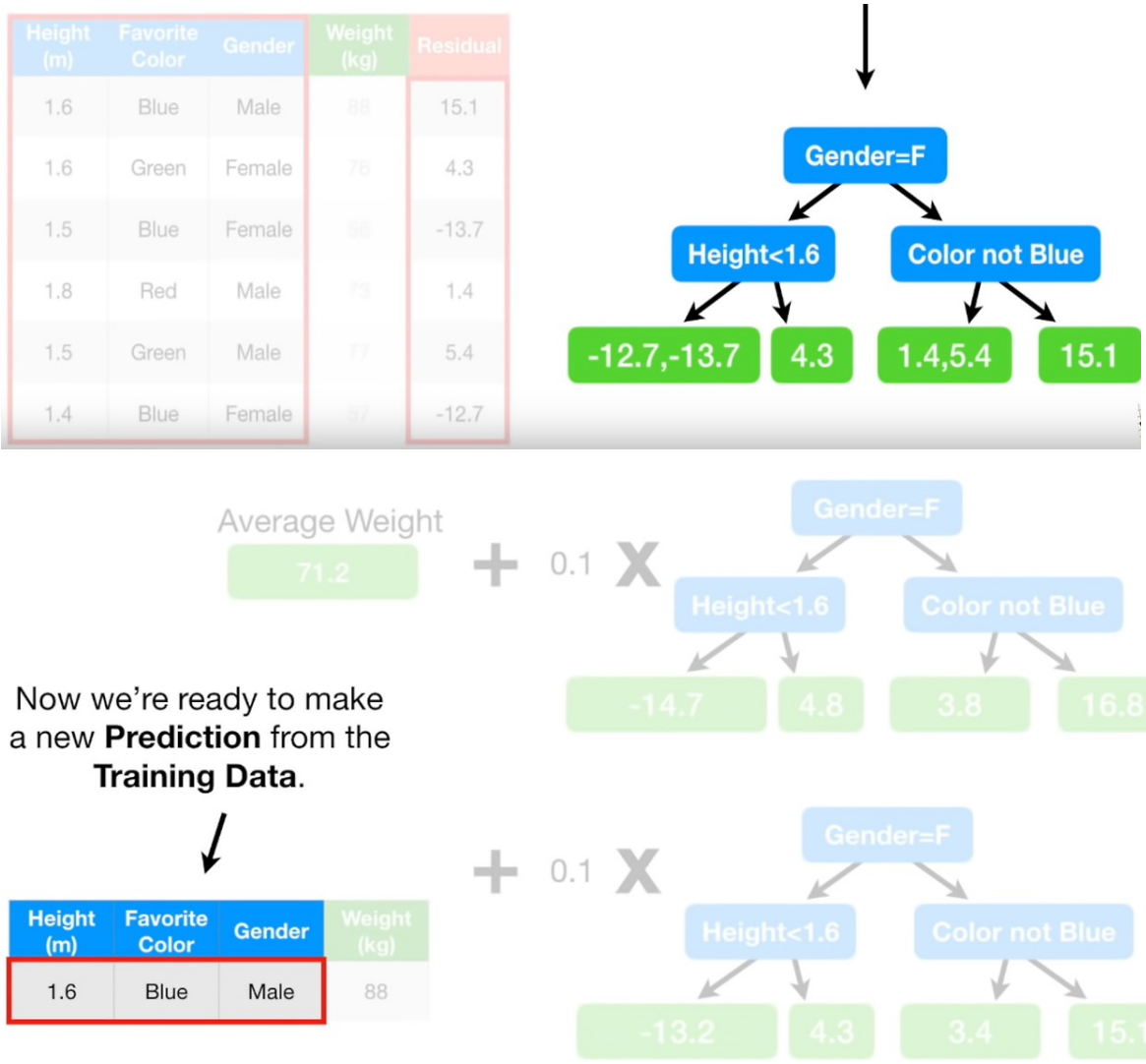
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	4.3
1.5	Blue	Female	56	-13.7
1.8	Red	Male	73	1.4
1.5	Green	Male	77	5.4
1.4	Blue	Female	57	-12.7

Process of Gradient Boost



초기 Residual과 비교해 보았을 때 새로운 Residual 값이 모두 이전보다 작아졌음을 확인할 수 있다. 즉, 실제 값에 다가간 것이다.

Process of Gradient Boost



계속해서 새로운 Residual에 대해서도 모델을 적용해보자.

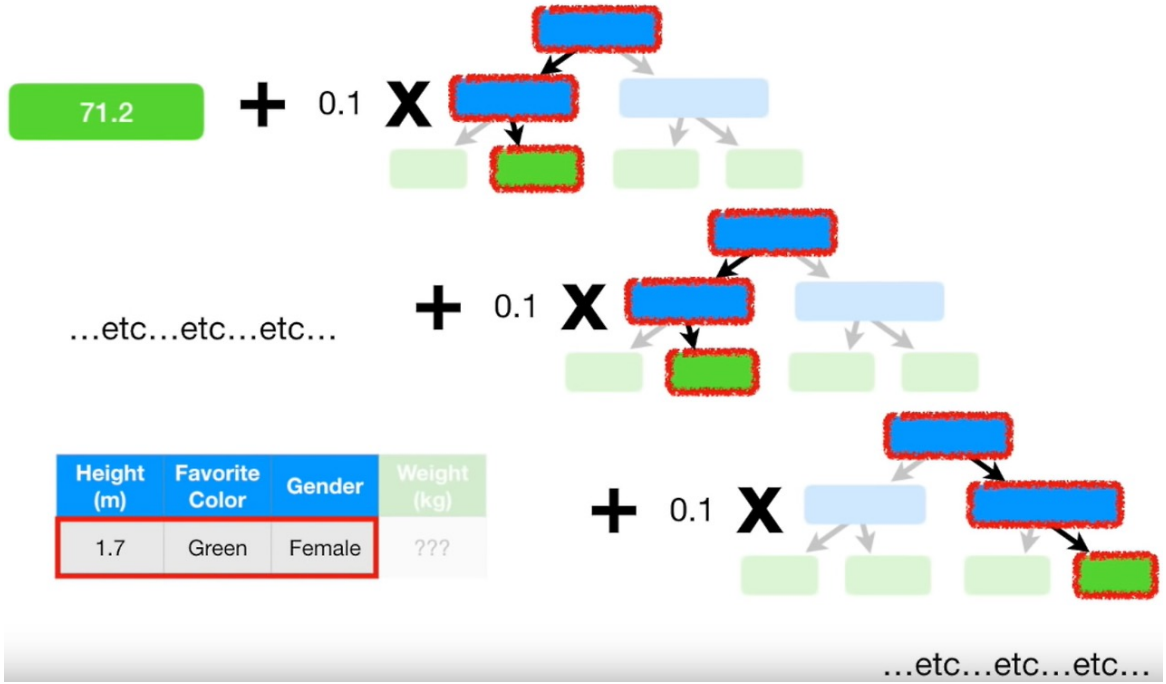
초기 Leaf Tree +(학습률 X 첫번째 Residual 예측 Tree) + (학습률 X 두번째 Residual 예측 Tree)

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

71.2 + (0.1 x 16.8) + (0.1 x 15.1) = 74.4

역시 실제 값에 가까워졌다. 각각의 새로운 Residual 또한 전보다 모두 감소하였다.

Process of Gradient Boost



사전에 하이퍼 파라미터로 정해 놓은 Iteration 횟수에 도달하거나 더 이상 Residual이 작아지지 않을 때까지 이것을 반복하면 최종적으로 Gradient Boost 모델이 구축된 것이다.

새로운 test data가 주어지면 이 모델로 몸무게를 예측할 수 있다.

Gradient Boost 모델은 이런 방식으로 학습한다.

Gradient Boost

왜 Gradient Boost 일까?

먼저 손실함수 (Loss Function)을 squared error로 설정하여 다음과 같이 정의한다.

$$j(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$$

Gradient Boost의 학습과정에는 Residual이 중요하게 사용 되는데, 이 Residual이 바로 Loss Function의 Negative gradient 이다.

$$\frac{\partial j(y_i, f(x_i))}{\partial f(x_i)} = \frac{\partial\left[\frac{1}{2}(y_i - f(x_i))^2\right]}{\partial f(x_i)} = f(x_i) - y_i$$

Negative gradient는 pseudo-residual이라고도 하는데, 이것은 어떤 데이터 포인트에서 Loss Function이 줄어들기 위해 f(x)가 가려고하는 방향을 뜻한다.

따라서 요약하면 Loss Function을 줄이기 위해 순차적인 다음 모델에 Negative gradient를 더해가는 Gradient descent + boosting 모델이므로 이를 줄여 Gradient Boosting 모델이 되었다.

Gradient Boost 실습

Gradient Boost는 지도 학습에서 가장 강력하고 널리 사용되는 모델 중 하나이다. 가장 큰 단점은 매개변수를 잘 조정해야 한다는 것과 훈련 시간이 길다는 것이다.

중요 매개변수는 트리의 개수를 지정하는 n_estimators와 학습률 learning_rate이다. Learning_rate를 낮추면 비슷한 복잡도의 모델을 만들기 위해서 더 많은 트리를 추가해야 한다.

n_estimators가 클수록 좋은 Random Forest와 달리 n_estimators를 크게 하면 모델이 복잡해지고 오버피팅의 위험이 있다. 따라서 가용시간과 메모리 한도에서 n_estimators를 맞추고, 적절한 learning_rate를 찾아야 한다.

중요한 또 다른 매개변수는 각 Tree에 복잡도를 낮추는 max_dapth이다. Gradient Boost에서는 깊이가 5보다 깊어지지 않게 해야 한다.

Gradient Boost 실습

```
# 데이터 로드
iris = load_iris()
X, y = iris.data, iris.target

# 학습 데이터와 테스트 데이터로 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 그래디언트 부스팅 모델 생성 및 학습
clf = GradientBoostingClassifier(n

_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
clf.fit(X_train, y_train)

# 예측 및 평가
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

Gradient Boost

Empirical evidence shows that taking lots of small steps in the right direction results in better predictions with a testing dataset, i.e. lower variance

- Jerome Friedman, Gardient Boost inventor -

"올바른 방향으로 조금씩 다가가는 식으로 학습된 모델은 테스트 데이터에서 좋은 성능을 보여준다."

감사합니다