

# 수요예측 - part 2

2021.03.12

# 선형 모형 / ML 모형 계열

- 현업에서 필요한 예측기법은 인과형 예측기법이라고 생각됨
  - 여러가지 독립변수를 통해 시계열이 설명 될 수 있어야 하고,
  - 현업의 데이터는 보통 정상성을 만족하지 않고 (그래서 차분을 통해 정상성 확보)
  - 대내외 수많은 이벤트들이 있기 때문에, 경험상 Box-Jenkins 모형은 특히 잘 맞지 않음
- 선형, ML, DL 모두 인과형 예측기 기법에 활용할 수 있음
  - 선형 모형 - 회귀분석, 계량경제 모형
  - ML 모형 - SVR, XGBoost 등 Boosting 모형, Random Forest 등 Bagging 모형
  - DL 모형 - LSTM, DeepAR, N-BEAT(Meta 사용 불가), Informer, Autoformer
- 단 ML 모형은 extrapolation이 불가능하므로, 데이터 변형(time delay embedding)이 필요함

# 선형 모형

# 선형 모형 (Econometric approach)

- 가장 기본적인 인과형 예측기법으로 회귀분석을 사용할 수 있음

$$y_t = \beta_0 + \alpha_1 \times T + \sum_{i=1}^p \beta_i x_{it} + \epsilon_t$$

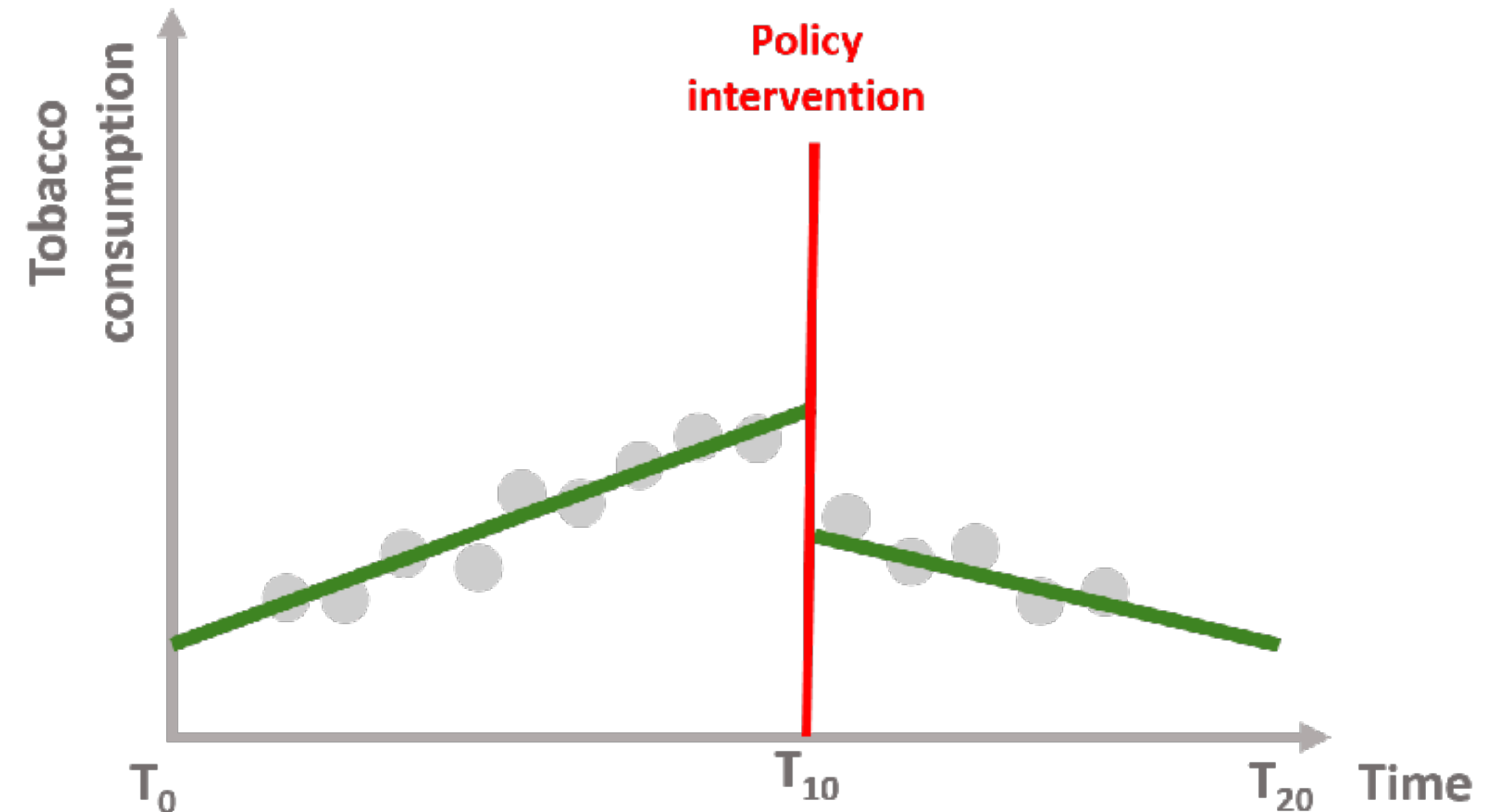
- 시점을 나타내는 변수로  $T$ 라는 변수를 사용
- 시간에 따른 패턴이 비선형 모형인 경우에는  $T$ 의 다항식을 사용

# 선형 모형 (Econometric approach)

- 시계열에 변경을 주는 이벤트들은 회귀 분석에서 indicator 형태로 반영
  - 예) 시간에 따라 패턴이 급격하게 달라지는 경우에도 indicator을 사용

$$y_t = (\beta_0 + \alpha_2 \times I) + (\alpha_1 + \alpha_2 \times I) \times T + \sum_{i=1}^p \beta_i x_{it} + \epsilon_t$$

여기서 만약  $T > T_{10}$ 이면  $I = 1$ , 그렇지 않으면  $I = 0$



# 선형 모형 (Econometric approach)

- 시계열 데이터에 선형 모형을 적용하기 위해서는 다음의 가정들이 만족되어야 함
  - 시계열 예측 변수에 대한 선형적 반응을 보임
  - 입력 변수 들 간에 선형독립임
  - 오차에 대한 자기상관 함수 그래프는 어떠한 패턴도 띄지 않음
  - 오차의 분산은 시간으로부터 독립적임
- 추가적인 패턴이 있는지 확인하기 위해서는 잔차를 확인

# 선형 모형 (Econometric approach)

- 시계열을 설명하기 위해 다른 시계열을 사용할 때에는 허구적 회귀 현상에 유의해야 함

$$y_t = \beta_0 + \alpha_1 \times T + \sum_{i=1}^p \beta_i x_{it} + \epsilon_t$$

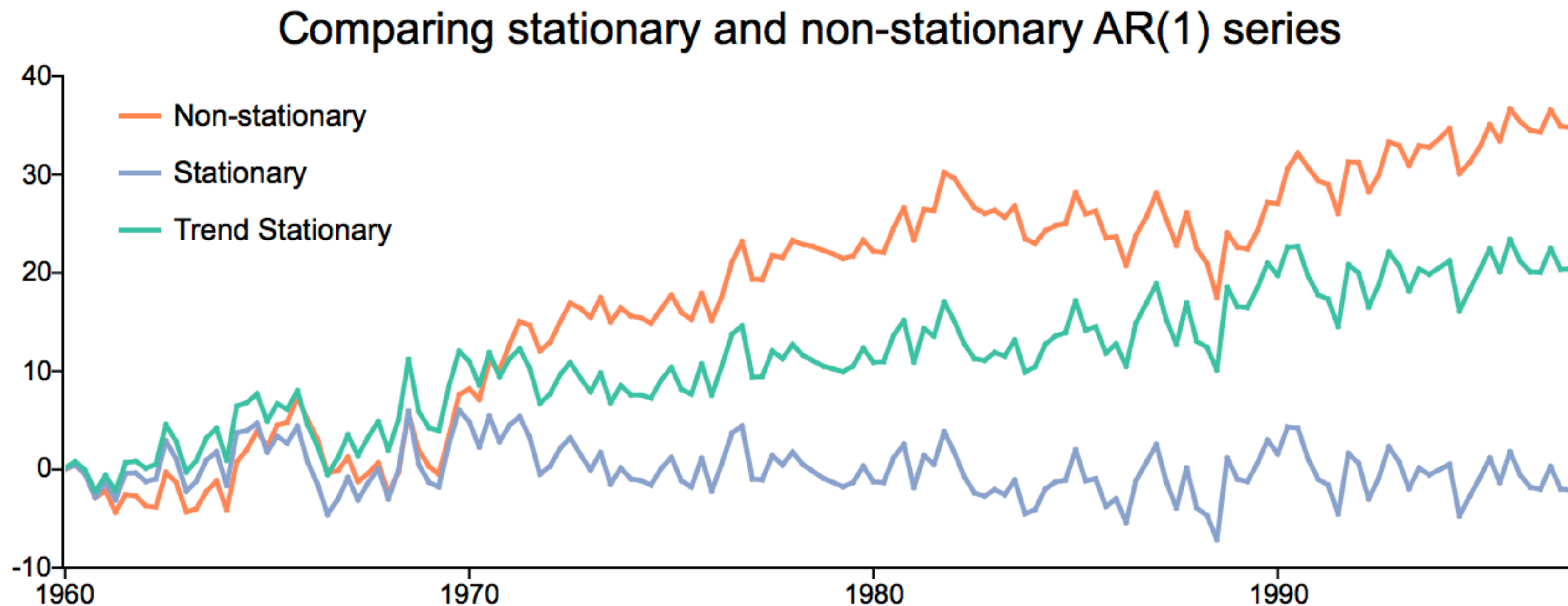
- 위의 식에서  $x_{it}$  들 중 하나가 또다른 시계열인 경우 발생 ( $x_{it}$ 는 메타정보일수 있음)
- 허구적 회귀 현상은 시계열에 단위근이 존재하는 경우에 발생
  - 단위근이 존재하는 불안정적인 시계열을 그대로 사용하면 표본 수가 증가함에 따라 회귀계수의 t-값도 증가하여 상관관계가 없는 변수간에도 매우 강한 상관관계가 있는 것으로 나타나는 가성회귀(허구적 회귀 : spurious regression)의 문제가 발생

# 선형 모형 (Econometric approach)

\* 다른 테스트도 있으나, ADF가 가장 표준적임

- Elliott-Rothenberg-Stock Test,
- Schmidt-Phillips Test
- Phillips-Perron (PP) Test
- Zivot-Andrews test

- 단위근의 존재 여부는 눈으로 잘 판단할 수 없으므로, ADF test를 활용
- Correlation이 시간이 지나감에 따라 줄어들지 않으므로, 오차가 누적됨
- 추세가 있는 정상시계열(단위근을 가지지 않는 시계열)과 눈으로 구분할 수 없음





# 선형 모형 (Econometric approach)

NOTE: Nelson and Plosser(1982)의 연구를 기점으로 그 이후 에 발표된 실증적×이론적 연구들에 의해 다수의 경제변수들이 안정적인 시계열(stationary time series)보다는 단위근(또는 확률적 추세)을 가지는 불안정적인 시계열임이 알려지게 됨

# 선형 모형 (Econometric approach)

- Nelson and Plosser(1982)의 연구를 기점으로 그 이후 에 발표된 실증적×이론적 연구들에 의해 다수의 경제변수들이 안정적인 시계열(stationary time series)보다는 단위근(또는 확률적 추세)을 가지는 불안정적인 시계열임이 알려지게 되었음
- 설명변수로 사용되는 시계열이 단위근을 가지더라도, 공적분 상태라면 허구적 회귀 현상이 나타나지 않음
  - 공적분 검정법으로는 최근에 다변량 시계열 분석에 의한 요한슨 공적분 검정(Johansen's cointegration test)이 다른 공적분 검정법보다 우월한 것으로 인정되어 널리 사용되고 있음

# Markridakis Competition

- The **Makridakis Competitions** (also known as the **M Competitions** or **M-Competitions**) are a series of open competitions to evaluate and compare the accuracy of different **time series** forecasting methods. They are organized by teams led by forecasting researcher **Spyros Makridakis** and were first held in 1982.

# M4 - 2018

- M4 는 보다 많고 다양한 시계열 데이터를 사용해 이전 세번의 경진대회 결과를 보다 다양한 방법들을 대상으로 다시 한번 입증했음
- M4 경진대회는 각각의 경우에 가장 적합한 예측 방법을 찾아내려고 했음
- 정확하고 설득력 있는 정답을 얻기 위해 M4는 100,000여개의 실제 시계열 데이터와 인공지능, ML과 전통적인 통계방법론까지 모든 주요한 예측기법을 모두 적용한 방대한 실험을 진행

# M4가 전해 준 주요 시사점

1. 방법론들의 조합이 가장 우수한 성능을 보임. 정확도 기준 상위 17개의 모형 중, 12개 모형은 통계적인 방법론에 기반한 조합이었음. 이 중 1, 2위 모형은 불확실성을 과소평가하지 않으면서도 95% PI를 정교하게 제시
2. 정확하게 가장 놀라운 점은 통계적인 방법과 ML을 섞어서 사용하는 “hybrid” 접근법이었음. Uber의 data scientist의 경우 sMAPE 관점에서 Comb benchmark 대비 10% 가량 더욱 정확해짐 (18년 실행된 M3에서는 4% 개선에 그침)
3. 2위는 통계적 방법과 ML의 조합을 사용한 모형이 차지 (Spain 대학, Australia의 Monash 대학 협업). 평균 낼때의 weight를 계산할 때 ML을 적용하였음.
4. 6개의 ML 방법들은 성능이 좋지 않았음. Comb benchmark보다 좋은 성능을 내는 모형은 없었으며, Naïve2보다는 1개 모형이 나았을 뿐임. PLOS ONE (Makridakis, et al., 2018)의 결과와 같은 맥락의 결론을 보여줌

# M5 - 2020

- M5는 20년 3월 3일부터 7월 1일까지 개최됨
- Kaggle platform에서 Walmart가 제공한 실 데이터로 진행
- 총 US\$100,000의 상금
- The data 구성
  - 약 4만 2천개의 일별 hierarchical daily timeseries로 구성. SKU 단위부터 지역별 수요까지 다양한 스케일의 데이터로 구성
  - 판매 데이터 외에도 가격 정보, 광고/홍보활동, 재고 수준의 데이터 제공

# M5에서 얻은 새로운 발견 - 시계열 모형에 닥친 ML의 물결

1. M5의 좀더 방대한 dataset에서는 ML 방법론이 1위를 차지함
  - 1~50위의 방법들이 ML 기반
2. 다재다능한 **LightGBM**(시계열 예측용)의 발견과, **Amazon's DeepAR** and **N-BEATS**의 등장이 눈에 띈다. 특히 N-BEATS는 M4의 1위 모형 대비 3%의 개선을 보임





Research Prediction Competition


# Ventilator Pressure Prediction

## Google Brain - Ventilator Pressure Prediction

Simulate a ventilator connected to a sedated patient's lung

\$7,500

Prize Money

 Google Brain에서 기획한 Kaggle competition

- 1. 실제 시계열 데이터 문제를 해결하기 위한 딥러닝의 중요성을 확인하고자 함
- 2. 대회의 목적은 컨트롤 input의 변화에 따른 인공호흡기의 압력 변화를 예측하고자 함
- 3. 개별 training instance는 각각이 시계열이므로, multiple time series 문제임
- 4. 우승팀은 LSTM network와 Transformer block을 포함한 multi-level deep architecture를 제시한 팀





Research Prediction Competition

# Ventilator Pressure Prediction

## Google Brain - Ventilator Pressure Prediction

Simulate a ventilator connected to a sedated patient's lung

\$7,500

Prize Money



1. 이번 대회에서는 시계열 분석에 있어서 다음의 측면들이 강조됨

- M-competition의 역할로 실용적인 측면이 부각되기는 했지만, 연구적인 성격이 강했음
- ML로 인해 성능은 많이 개선되었지만, 설명력이 떨어짐

Late Submission



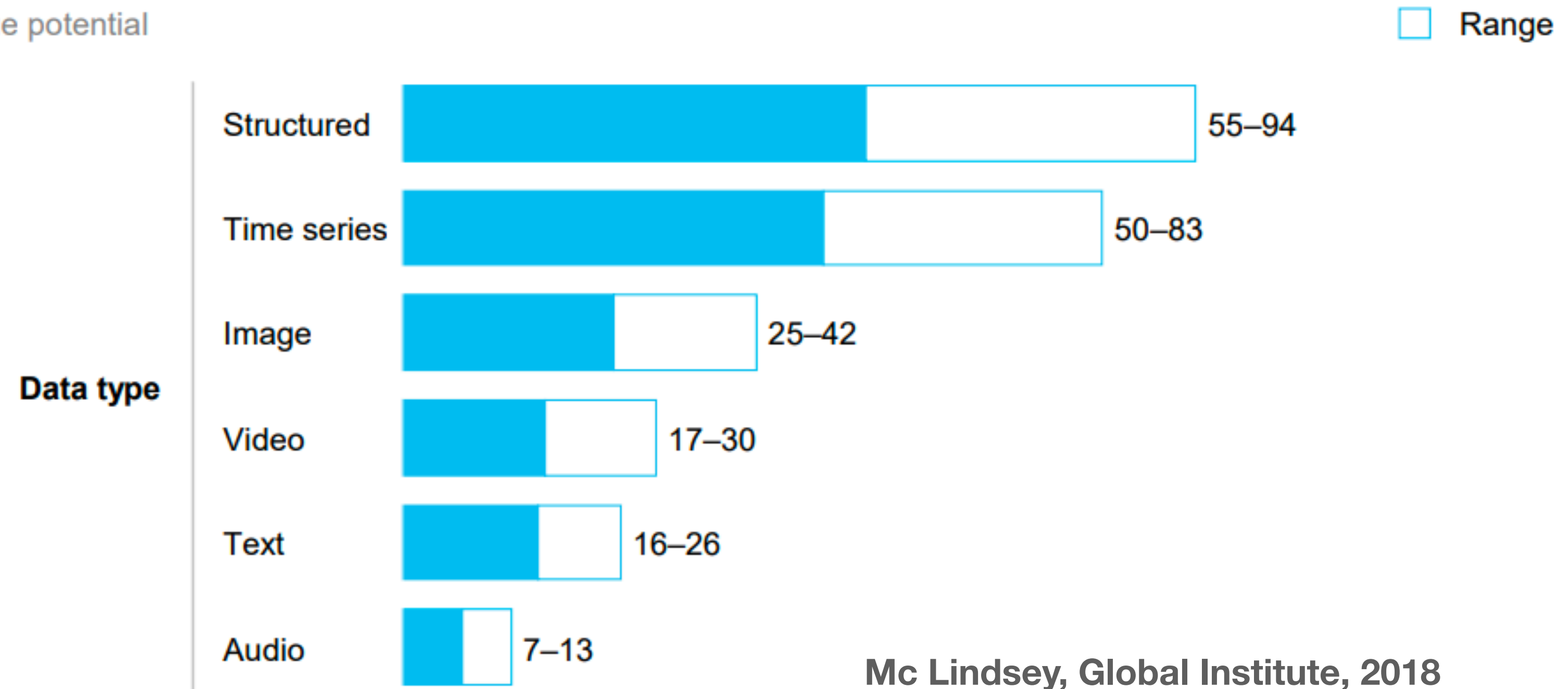
- **Versatility:** 여러 상황의 데이터에도 작동하는 다재다능함
- **MLOps:** Production에서도 사용될 수 있는 가능성
- **Interpretability and explainability:** 설명가능성

# 시계열에서의 ML 모형

- 결국 시계열에 사용되는 모형들도 loss를 최소화하는 방식으로 학습이 진행됨
  - 시계열 모형은 오차항의 상관관계를 고려한 Loss를 고려
  - 회귀 분석은 오차항의 상관관계가 없다는 가정 하에 다양한 meta 변수를 고려한 loss를 최소화
- 결국 머신러닝 기법들 역시 시계열 상황을 고려한 loss를 구할 수 있다면 시계열 모형에 적용 가능
  - 머신러닝 모형들은 훨씬 많은 interaction 및 비선형 관계를 고려하므로
  - 선형모형 대비 뛰어난 예측결과를 보여줄 수 있음
  - 다만 데이터에 시계열 정보를 담을 수 있도록 변환 작업이 필요하고,
  - 모형 선택 과정에서 시점을 고려하여 신중하게 진행해야 함

# 시계열 예측은 현업 관점에서는 아주 중요한 needs

- AI의 도입으로 인해 얻을 수 있는 데이터 타입별 잠재 가치에서 시계열 데이터는 2위
- 최근 vision, 혹은 자연어처리 및 음성 처리 등이 주목을 받고는 있지만,
- 시계열 예측은 모든 산업 영역에 있어서 중요한 영역
- 어떻게 보면, 시계열 예측이 충분히 발전하지 않아서 사용을 못했던 측면이 강함



<https://www.mckinsey.com/~/media/mckinsey/featured%20insights/artificial%20intelligence/notes%20from%20the%20ai%20frontier%20applications%20and%20value%20of%20deep%20learning/notes-from-the-ai-frontier-insights-from-hundreds-of-use-cases-discussion-paper.ashx>

<https://medium.com/daria-blog/machine-learning-for-time-series-forecasting-part-1-6e97661c9773>



# ML 모형

- Ensemble 모형의 기초 -

# ML 모형 - Tree 기반 Ensemble

- Ensemble 기법은 약한 여러개의 learner를 여러개 사용하여 예측 성능을 높이는 방법
  - 데이터 타입에 따라, classifier와 regressor로 나뉨
  - Weak learner는 어떤 모형도 될 수 있음
  - 하지만, 현업에서 사용하는 package들은 대부분 tree 기반의 모형을 사용
  - 이 수업에서는 tree를 weak learner로 가정하여 진행
- Ensemble 기법은 다음과 같이 weak learner의 선형 결합으로 나타낼 수 있음

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F}, \text{ 여기서 } \mathcal{F} \text{ 은 모든 CART 모형의 모임}$$

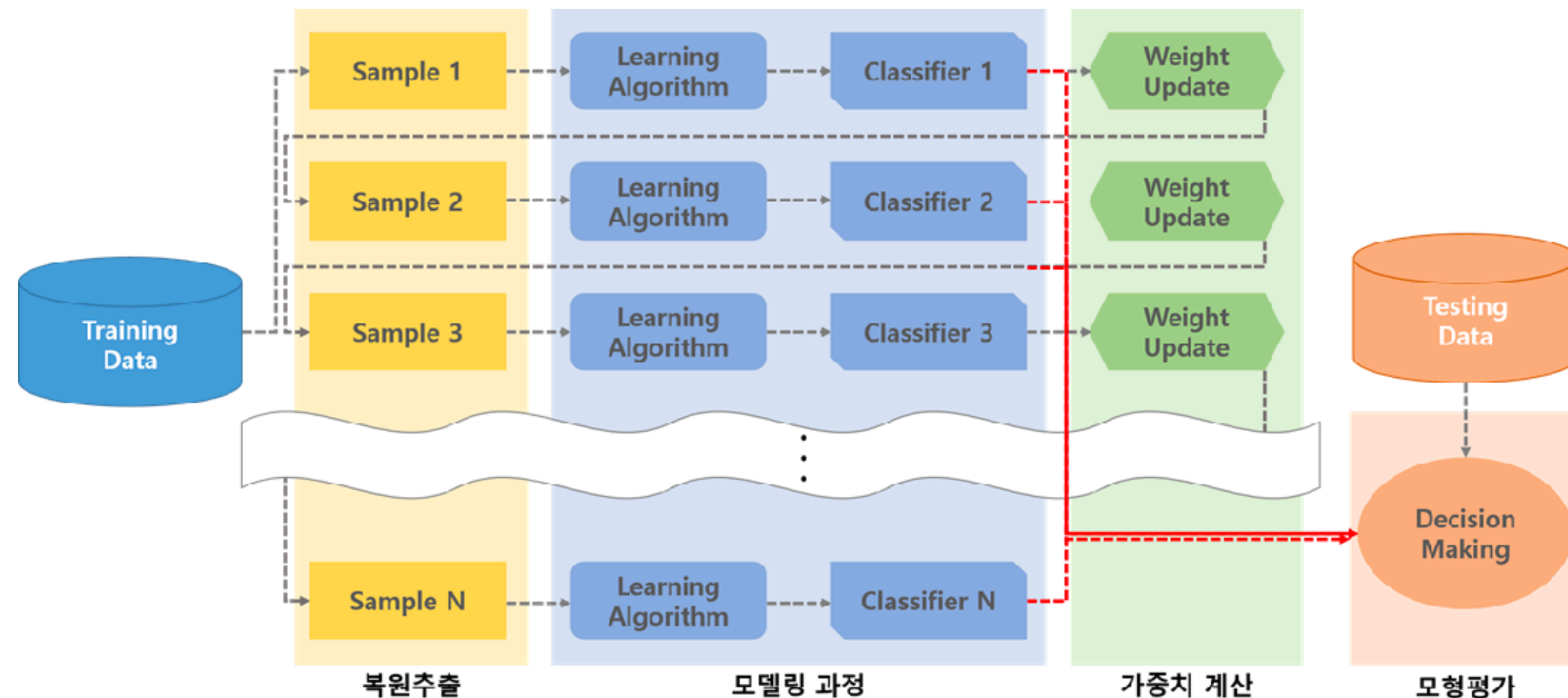
- Weak learner를 탐색하고 구성하는 방법에 따라 Bagging과 Boosting으로 크게 나뉨

# Bagging (Bootstrap Aggregating)

- 데이터에서 다수의 Bootstrap sample을 생성하고, 각각을 모델링한 후 결합하여, 최종 예측 모형 구축
  - Bootstrap : 주어진 자료에서 단순 random 복원추출 방법을 활용하여 동일한 크기의 표본을 여러 개 생성하는 sampling 방법
- 개별 모형을 평균함으로써 예측시 분산을 줄여, 새로운 데이터에 대한 일반화 능력을 향상
- Decision tree와는 다르게, 개별 tree는 가지치기를 하지 않음으로 해서, overfitting 시킴
  - 그렇지 않으면 개별 tree들이 압도적인 feature만 계속 사용할 가능성이 있음
  - 개별 tree를 서로 다르게 함으로써, 예측치와 예측 에러의 독립성을 확보

# Boosting

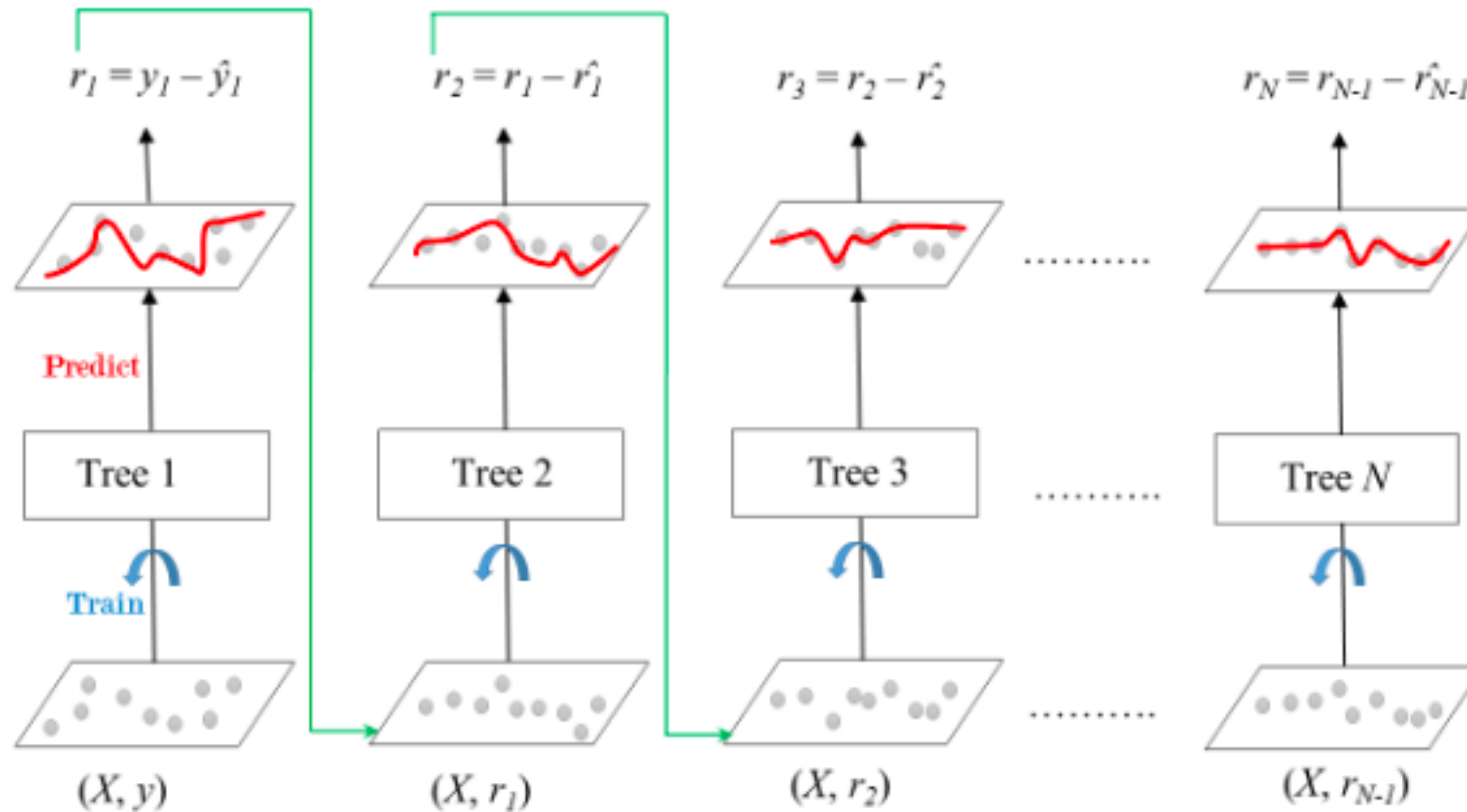
- 순차적으로 잘못 분류된 개체들에 가중치를 적용, 새로운 분류 규칙을 만들고, 이 과정을 반복해 최종 모델을 구축
- 잘 분류된 데이터의 가중치를 낮추고, 오분류된 데이터의 가중치를 높임



# Gradient Boosting



# Gradient Boosting



# Gradient Boosting

- "Gradient Boosting": [Greedy Function Approximation: A Gradient Boosting Machine, by Friedman] 논문에서 처음 나온 용어
- 이방법에서는 이전 단계의 residual을 다음단계의 데이터로 활용하는데, residual은 squared loss의 gradient값이라고 생각할 수 있음
- Weight값을 구하는 단계가 별도로 따로 있음

# Gradient Boosting

Input: training set  $\{(x_i, y_i)\}_{i=1}^n$ , a differentiable loss function  $L(y, F(x))$ , number of iterations  $M$ .

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma).$$

2. For  $m = 1$  to  $M$ :

1. Compute so-called *pseudo-residuals*:

$$r_{im} = - \left[ \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \dots, n.$$

2. Fit a base learner (or weak learner, e.g. tree) closed under scaling  $h_m(x)$  to pseudo-residuals, i.e. train it using the training set  $\{(x_i, r_{im})\}_{i=1}^n$ .
  3. Compute multiplier  $\gamma_m$  by solving the following [one-dimensional optimization](#) problem:

$$\gamma_m = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)).$$

4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output  $F_M(x)$ .

# Parameter (Tree 관련 parameter)

- **max\_depth**

- 트리의 최대 깊이
- default : 3
- 깊이가 깊어지면 과적합될 수 있으므로 적절히 제어 필요

- **min\_samples\_split**

- 노드를 분할하기 위한 최소한의 샘플 데이터수
  - 과적합을 제어하는데 사용됩니다. 값이 작을수록 분할노드가 많아져 과적합 가능성 증가
- default : 2

- **min\_samples\_leaf**

- 리프노드가 되기 위한 최소한의 샘플 데이터수
  - 과적합을 제어하는데 사용됩니다. 값이 작을수록 과적합 가능성 증가
- default : 1
- 불균형 데이터의 경우 특정 클래스 데이터가 극도로 적을 수 있으므로 작은 값으로 설정 필요

# Parameter (Tree 관련 parameter)

- **max\_features**

- 최적의 분할을 위해 고려할 피처의 최대 갯수
- default : None
- int형으로 지정 → 피처 갯수
- float형으로 지정 → 전체 갯수의 일정 비율만큼 사용
- sqrt 또는 auto → 전체 피처 중  $\sqrt{(\text{피처 개수})}$  만큼 선정
- log2 : 전체 피처 중  $\log_2(\text{전체 피처 개수})$  만큼 선정

- **max\_leaf\_nodes**

- 리프노드의 최대 갯수
- default : None → 제한없음

# Parameter (Boosting 관련 parameter)

- **loss**

- 경사하강법에서 사용할 loss function 지정
- default : deviance

- **n\_estimators**

- 생성할 트리의 갯수를 지정
- default : 100
- 많을수록 성능을 좋아질 수 있지만 시간이 오래 걸림

- **learning\_rate**

- 학습을 진행할 때마다 적용하는 학습률(0~1 사이의 값)
- 약한 학습기가 순차적으로 오류값을 보정해나갈 때 적용하는 계수
- default : 0.1
- 낮은만큼 최소 손실값을 찾아 예측 성능이 높아질 수 있음
- 하지만 그 만큼 많은 수의 트리가 필요하고 시간이 많이 소요

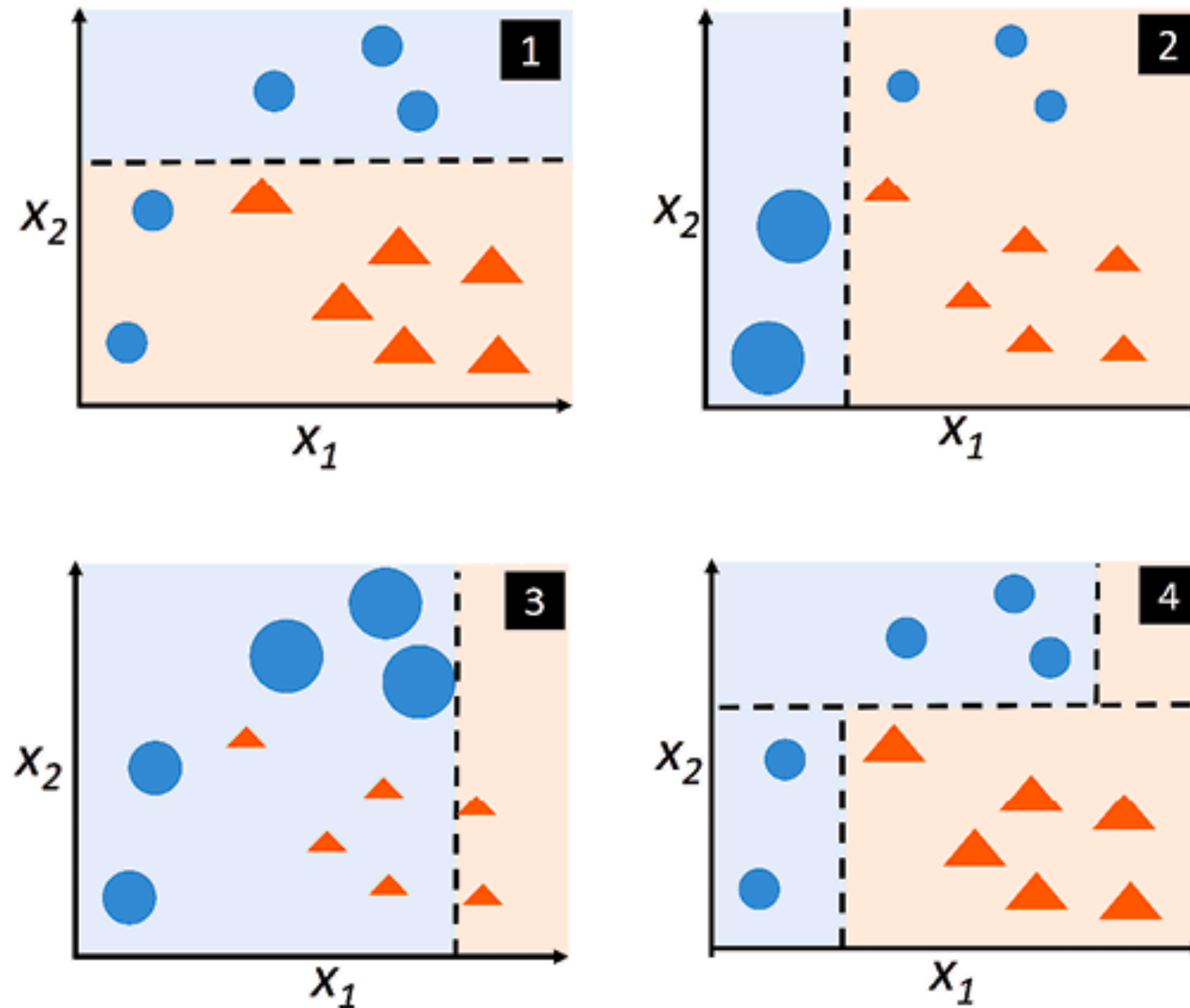
- **subsample**

- 개별트리가 학습에 사용하는 데이터 샘플링 비율(0~1 사이의 값)
- default : 1 (전체 데이터 학습)
- 이 값을 조절하여 트리 간의 상관도를 줄일 수 있음

# AdaBoost

# AdaBoost

- 이전 단계의 분류기에서 잘못 분류한 sample의 가중치를 조정 후 순차적으로 모델을 추가해 나가는 방법





# Gradient Boosting

- 알고리즘을 잘 이해하기 위해서는 가중치를 조정하는 방식을 이해해야 함
- $m$  step에서의 learner는 다음과 같이 정의

$$C_m(x_i) = C_{(m-1)}(x_i) + \alpha_m k_m(x_i)$$

- Exponential loss를 사용하면 다음과 같은 loss를 얻을 수 있음

$$E = \sum_{i=1}^N e^{-y_i C_m(x_i)} = \sum_{i=1}^N e^{-y_i C_{(m-1)}(x_i)} e^{-y_i \alpha_m k_m(x_i)} = \sum_{i=1}^N w_i^{(m)} e^{-y_i \alpha_m k_m(x_i)}, \text{ 여기서 } w_i^{(1)} = 1, w_i^{(m)} = e^{-y_i C_{(m-1)}(x_i)}$$

- 위의 Weighted loss를 최소화하는  $\alpha_m$ 은 다음과 같이 결정됨

$$\alpha_m = \frac{1}{2} \ln \left( \frac{\sum_{y_i=k_m(x_i)} w_i^{(m)}}{\sum_{y_i \neq k_m(x_i)} w_i^{(m)}} \right)$$

# Gradient Boosting

---

## Algorithm 10.1 *AdaBoost.M1*.

---

1. Initialize the observed weights  $w_i = 1/N, i = 1, 2, \dots, N$ .
    - 학습에 사용하는 알고리즘
    - default : None  $\rightarrow$  DecisionTreeClassifier(max\_depth = 1) 이 적용
  2. For  $m = 1$  to  $M$ :
    - n\_estimators
    - 생성할 약한 학습기의 개수를 지정
    - (a) Fit a classifier  $G_m$  to the training data using weights  $w_i$ .
      - default : 50
      - learning\_rate
    - (b) Compute  $err_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}$ .
      - 학습을 진행할 때마다 적용하는 학습률(0~1 사이의 값)
      - 약한 학습기가 순차적으로 오류값을 보정해나갈 때 적용하는 계수
      - default : 1
      - 낮은만큼 최소 손실값을 찾아 예측 성능이 높아질 수 있음
      - 하지만 그 만큼 많은 수의 트리가 필요하고 시간이 많이 소요
    - (c) Compute  $\alpha_m = \log((1 - err_m)/err_m)$ .
    - (d) Set  $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))], i = 1, 2, \dots, N$ .
  3. Output  $G(x) = \text{sign} \left[ \sum_{m=1}^M \alpha_m G_m(x) \right]$ .
-

# Parameter

- **base\_estimators**

- 학습에 사용하는 알고리즘
- default : None → DecisionTreeClassifier(max\_depth = 1) 이 적용

- **n\_estimators**

- 생성할 약한 학습기의 개수를 지정
- default : 50

- **learning\_rate**

- 학습을 진행할 때마다 적용하는 학습률(0~1 사이의 값)
- 약한 학습기가 순차적으로 오류값을 보정해나갈 때 적용하는 계수
- default : 1
- 낮은만큼 최소 손실값을 찾아 예측 성능이 높아질 수 있음
- 하지만 그 만큼 많은 수의 트리가 필요하고 시간이 많이 소요

# XGBoost

# XG-Boost의 개요

- "Extreme Gradient Boosting"을 의미
- XGBoost는 Gradient Boosting 방법 중 한 가지로 표시
- 효율성과 유연성, 휴대성(efficient, flexible, portable)이 뛰어나도록 최적화된 distributed gradient boosting 라이브러리로도 제공
- XGBoost는 많은 데이터 과학 문제를 빠르고 정확하게 해결하는 parallel tree boosting(GBDT, GBM이라고도 함)을 제공
- 동일한 코드가 주요 분산 환경(하둡, SGE, MPI)에서 실행되며 다양한 문제에 적용되는 다재다능한 알고리즘임

# XG-Boost (특장점)

- Over fitting(과적합)을 방지할 수 있음
- 신경망에 비해 시각화가 쉽고, 이해하기보다 직관적임
- 자원(CPU, 메모리)이 많으면 많을수록 빠르게 학습하고 예측할 수 있음
- Cross validation을 지원
- 높은 성능: 실제로 Kaggle에서 XGBoost를 쓴 결과들이 상위권을 휩쓰는 결과

# XG-Boost 모형 상세

- 데이터 셋  $m \times n$ 이 주어져 있다고 가정하면, 즉,

$$\mathcal{D} = \{(x_i, y_i) \mid |\mathcal{D}| = n, x_i \in \mathbb{R}^n, y_i \in \mathbb{R}\}$$

- Tree Boost 기법에서는 일반적으로 다음과 같은 구조의 모형을 통해 예측을 진행

$$\hat{y} = \phi(x) = \sum_{k=1}^K f_k(x), \quad f \in \mathcal{F}$$

$$\mathcal{F} = \{f(x) = w_{q(x)}, q : \mathbb{R}^m \rightarrow \{1, \dots, T\}, w \in \mathbb{R}^T\}$$

# XG-Boost 모형 상세

- Model Complexity (Gradient boosting에는 없는 추가된 개념)
  - 결국 Tree 모형에서 classifier는 다음과 같은 vector로 정의할 수 있음

$$f_t(x) = w_{q(x)}, \quad w \in R^T, \quad q : R^d \rightarrow 1, 2, \dots, T$$

$T$ :  $t$ 번째 tree의 leaf node 갯수

$w$ :는 Leaf에서의 score가 저장되어 있는  $T$ 차원 vector

$q$ : 개별 데이터 포인트를  $T$ 개의 leaf node로 나누는 tree classifier

- 위와 같이 classifier를 정의하면 다음과 같이 regularization term을 표현할 수 있음

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$



# XG-Boost 모형 상세

- Complexity가 고려된 목적함수는 다음과 같음

$$L(\theta) = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k), \quad \Omega(f_k) = \gamma T + \frac{1}{2} \lambda \sum_{i=1}^T w_i^2$$

- Loss 함수가 작아질 수록 목적함수는 작아짐
- 모형이 작을 수록, 복잡도가 낮을 수록 penalty를 적게 받게 됨
- Regularization term이 없으면 gradient boosting과 같아짐

# XG-Boost 모형 상세

- 목적함수를 최소화 하는 K개의 tree를 찾는 것은 현실적으로 불가능
  - NP Complete 문제로 알려져 있음
- XG-Boost에서는 매 iteration마다 순차적으로 Tree의 가지를 추가해 가는 Additive training 전략을 사용

$$\begin{aligned}\hat{y}_i^{(0)} &= 0 \\ \hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\ \hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\ &\vdots\end{aligned}$$

- 매 단계의 Loss function에 주목하고 이를 최소화하기 위한 알고리즘 개발

# XG-Boost 모형 상세

- XG Boost에서는 regularization part에 주목해서 식을 전개
  - 이전의 tree 모형에서는 그렇게 중요하게 생각하지 않았던 요소.
  - 그동안의 tree 모형들은 Impurity를 개선하기 위한 알고리즘 개발에 주력
  - 상대적으로 그 중요도가 덜 강조되었고,
  - Model complexity는 stopping rule 등을 사용한 pruning을 통해서 주로 해소하였음
  - XGBoost는 Regularization term을 명확하게 정의, 모형에 대한 이해를 높이고
  - 알고리즘에 의해 정량적인 방법으로 pruning을 진행

# XG-Boost 모형 상세

- Objective function

$$\begin{aligned} L^{(t)} &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \gamma T + \frac{1}{2} \lambda \sum_{i=1}^T w_i^2 \\ &\propto \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \gamma T + \frac{1}{2} \lambda \sum_{i=1}^T w_i^2 \\ &\propto \sum_{i=1}^n \left[ l(y_i, y_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{i=1}^T w_i^2 \end{aligned}$$

(Taylor Expansion, 2nd order approx.)

$$\text{Where, } g_i = \frac{\partial}{\partial \hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}), \quad h_i = \frac{\partial^2}{\partial \hat{y}_i^{(t-1)2}} l(y_i, \hat{y}_i^{(t-1)})$$

- Objective function은 오로지  $g_i, h_i$ 에만 의존함: XGBoost가 custom loss function을 지원할 수 있는 이유 (Gradient boosting에서는 지원하지 않음)

# Loss + Complexity

- 여기에서는  $y_i$ 를 T 차원의 one-hot vector로 간주

$$L^{(t)} \approx \sum_{i=1}^n \left[ g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{i=1}^T w_i^2 \quad (g_i w_{q(x_i)} \text{와 } h_i w_{q(x_i)}^2 \text{ 모두 행렬 곱에 의한 scalar})$$
$$= \sum_{j=1}^T \left[ G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T \quad (\text{관측치 관점이 아니라 Leaf node 관점에서의 sum})$$

여기서.  $H_j = \sum_{i \in I_j} h_i, \quad G_j = \sum_{i \in I_j} g_i, \quad I_j = \{i \mid q(x_i) = j\}$

- j번째 node에 속하는 관측치의 결과값  $w_j$ 는 모두 동일함

# Derived Goodness of Fit measure

- 앞에서 도출한 목적함수를 최소화 하는  $w_j^*$ 는 다음과 같이 표현할 수 있음

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

결국 목적함수의 최소값은

$$obj^* = -\frac{1}{2} \sum_{j=1}^T \frac{G_j}{H_j + \lambda} + \gamma T$$

- Tree structure를 나타내는  $q(x)$ 의 우월성을 측정하는 measure 역할

# Information gain

- 목표는 information gain을 극대화하는 split point를 찾는 것임
- 순차적으로 Tree를 탐색해 나갈 때, 다음의 information gain을 바탕으로 탐색

$$Gain = \frac{1}{2} \left[ \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

만약 Gain이  $\gamma$ 보다 작으면, 더이상 가지치기를 하지 않음: 기존의 pruning에 해당

# Parameter

- 일반 parameter: Booster type, n\_jobs, verbosity
- Booster parameter: Learning\_rate, n\_estimators, max\_depth, min\_child\_weight, gamma, lambda, alpha, subsample, colsample\_bytree
- Learning Parameter: objective, eval\_metric, seed



# 일반 Parameter

Parameter	Option	초기값	의미
Booster Type	gbtree	○	높을 수록 과적합하기 쉬움
	gblinear		
	dart		커질 수록 모델의 복잡도가 커짐
n_jobs	1		병렬작업의 개수
Verbosity	0		로그 없음
	1	○	경고 출력
	2		정보 출력
	3		Debug용 error 출력
Valid_parameters	FALSE		Parameter간의 관계 검증
disable_default_eval_metric	0	○	Custom metric 사용시 설정
	1		
num_pbuff			마지막 부스팅 단계에서 결과 저장 공간 크기, 자동설정
num_feature			사용할 feature의 갯수, 자동설정

# Booster Parameter

Parameter	의미	초기값	추천값
Learning rate	Learning rate	0.3	높을 수록 과적합하기 쉬움
n_estimators	Weak learner의 갯수	100	
max_depth	개별 tree의 max depth	6	커질 수록 모델의 복잡도가 커짐
min_child_weight	관측치에 대한 가중치 합의 최소	1	높을 수록 과적합 방지
gamma	Leaf node의 추가분할을 결정할 최소손실값	0	높을 수록 과적합 방지
subsample	Weak learner가 학습에 사용하는 데이터 샘플링 비율	1	낮을 수록 과적합 방지
subsample_method	데이터 샘플링 방법 (uniform, gradient_based)	Uniform	
colsample_bytree	각 step tree별로 사용된 feature의 %	1	낮을 수록 과적합 방지
colsample_bylevel	각 tree의 depth level별로 사용될 feature %	1	
colsample_bynode	각 node별로 사용될 feature %	1	
lambda (reg_lambda)	가중치에 대한 L2 regularization 값	1	클수록 과적합 감소
alpha (reg_alpha)	L1 regularization 적용값	0	클수록 과적합 감소
scale_pos_weight	Data의 label 불균형 정도를 알려줌	1	Imbalance 정도를 입력

# Learning Parameter

Parameter	Option	초기값	비고
base score		0.5	초기편향값
Objective	reg: squared error	0	높을 수록 과적합하기 쉬움
	Binary: logistic error		
	Multi:softmax		num_class를 지정해야 함
	Multi:softprob		
	Count: Poisson		
seed	0		결과 재현을 위한 seed
eval_metric	rmse		
	mae	0	
	logloss		
	Error		
	mlogloss		
	auc		ROC auc를 의미. FPR과 TPR에 민감한 경우 사용
	aucpr		Pre/Recall auc를 의미. f-score나 recall에 민감할 때 사용
	map		
number_boost_round			몇회의 Step을 반복할지 결정
early_stopping_rounds			early_stopping_rounds 동안 metric이 개선되지 않을 때 조기종료

# LightGBM

# LightGBM

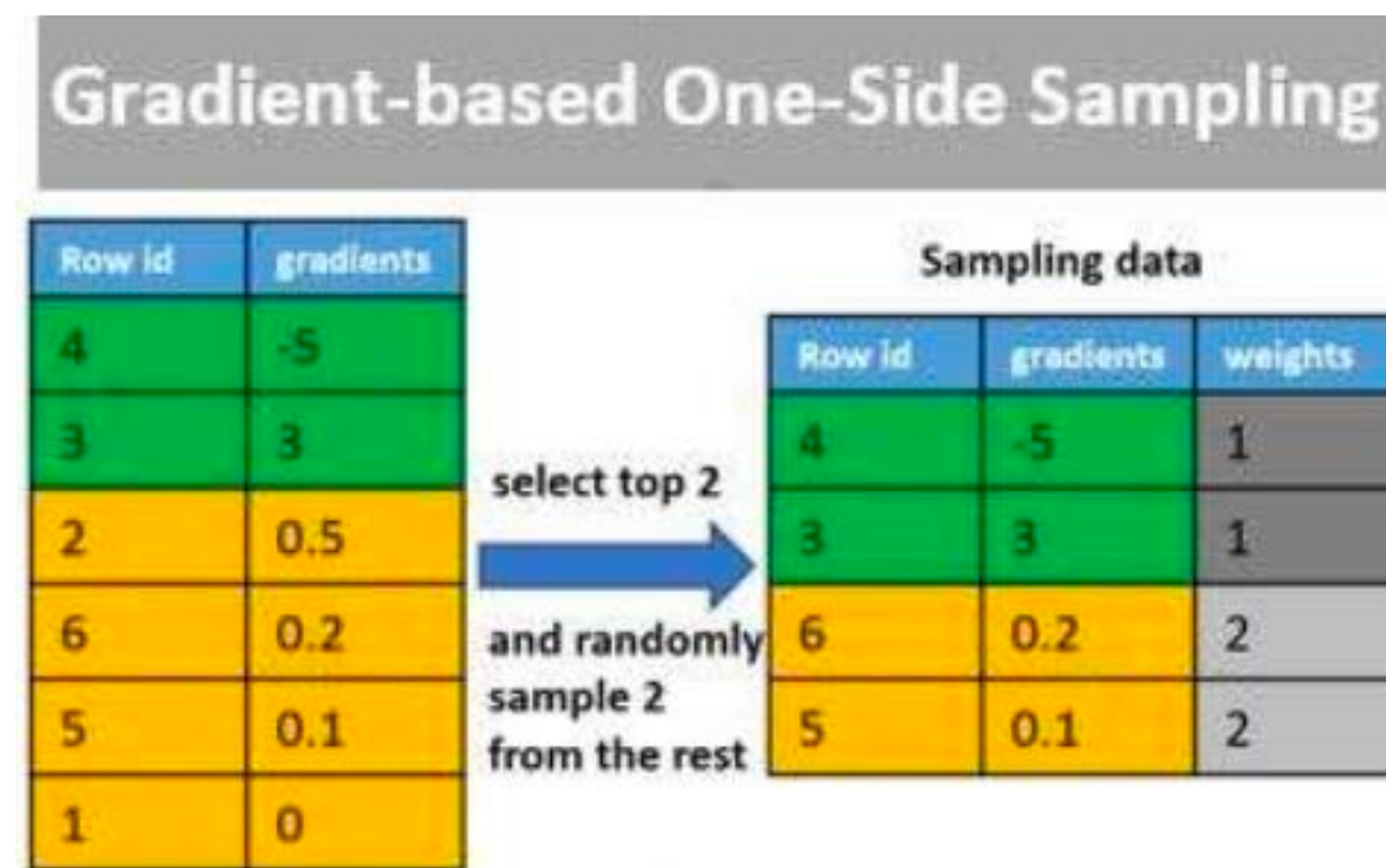
- GBM은 Decision Tree를 만드는 과정에 Gradient Boosting을 활용하는 것
- 트리가 가지를 늘려가는 과정에서 Gradient(MSE loss에서는 단순히 residual)를 계산하여 손실함수를 가장 큰 폭으로 줄일 수 있는 부분에서 leaf를 분리
- 다른 말로는, 매 단계마다 이전 단계에서 잘 학습하지 못했던 데이터를 보다 집중적으로 학습하는 트리를 만든다는 것
- GBDT를 만드는 과정 중에서 가장 많은 시간과 자원이 소요되는 부분은 **split 지점을 찾는 부분**
  - 해당 Feature의 모든 값을 정렬하고, 가능한 모든 split 지점 중에서 어떤 부분이 가장 잘 데이터를 분류하는지 찾아야 하기 때문
- 이렇게 모든 split 지점을 확인하는 것은 매우 비효율적이기 때문에, 효율적으로 GBDT를 구성하기 위해 많은 방법들이 제시되고 있음(e.g. XGBoost, LightGBM, CatBoost).

# Gradient-based One-Side Sampling (GOSS)

- 기본 가정은 instance의 gradient가 클수록 영향력이 크고 덜 학습된 instance라는 것
- instance의 gradient가 크다는 것은, 해당 instance를 구성하는 값(element)들의 변화가 손실함수에 큰 변화를 가져올 수 있다는 것
- LightGBM에서는 이런 instance를 **모델이 충분히 학습하지 못한 *instance*** 라고 보고 있고, 따라서 이러한 instance를 주로 포함하면서 데이터셋의 샘플 수를 줄이려고 시도

# Gradient-based One-Side Sampling (GOSS)

- $i$ 번째 step의 모델을 이용해서 예측한 후
- Negative gradient의 크기 순서대로 전체 데이터를 나열
- $A$ 집합(추정에 도움되는 set): 처음  $a \times 100$  (%)개의 관측치
- $B$ 집합(도움되지 않는 set):  $A^c$ 에서 Random하게  $b \times 100$  (%) 추출
- $B$ 집합에 속하는 관측치에  $\frac{1-a}{b}$ 만큼의 가중치(fact)를 적용하여 모형
- 전체 데이터 대신  $a + b$ 개의 데이터만 사용하여 계산량을 줄임  
과 동시에 의미있는 데이터에 더욱 집중





# Gradient-based One-Side Sampling (GOSS)

---

**Algorithm 2:** Gradient-based One-Side Sampling

---

**Input:**  $I$ : training data,  $d$ : iterations

**Input:**  $a$ : sampling ratio of large gradient data

**Input:**  $b$ : sampling ratio of small gradient data

**Input:**  $loss$ : loss function,  $L$ : weak learner

$models \leftarrow \{\}$ ,  $fact \leftarrow \frac{1-a}{b}$

$topN \leftarrow a \times \text{len}(I)$ ,  $randN \leftarrow b \times \text{len}(I)$

**for**  $i = 1$  **to**  $d$  **do**

$preds \leftarrow models.predict(I)$

$g \leftarrow loss(I, preds)$ ,  $w \leftarrow \{1, 1, \dots\}$

$sorted \leftarrow \text{GetSortedIndices}(\text{abs}(g))$

$topSet \leftarrow sorted[1:topN]$

$randSet \leftarrow \text{RandomPick}(sorted[topN:\text{len}(I)],$   
     $randN)$

$usedSet \leftarrow topSet + randSet$

$w[randSet] \times = fact$   $\triangleright$  Assign weight  $fact$  to the  
    small gradient data.

$newModel \leftarrow L(I[usedSet], -g[usedSet],$   
     $w[usedSet])$

$models.append(newModel)$

---



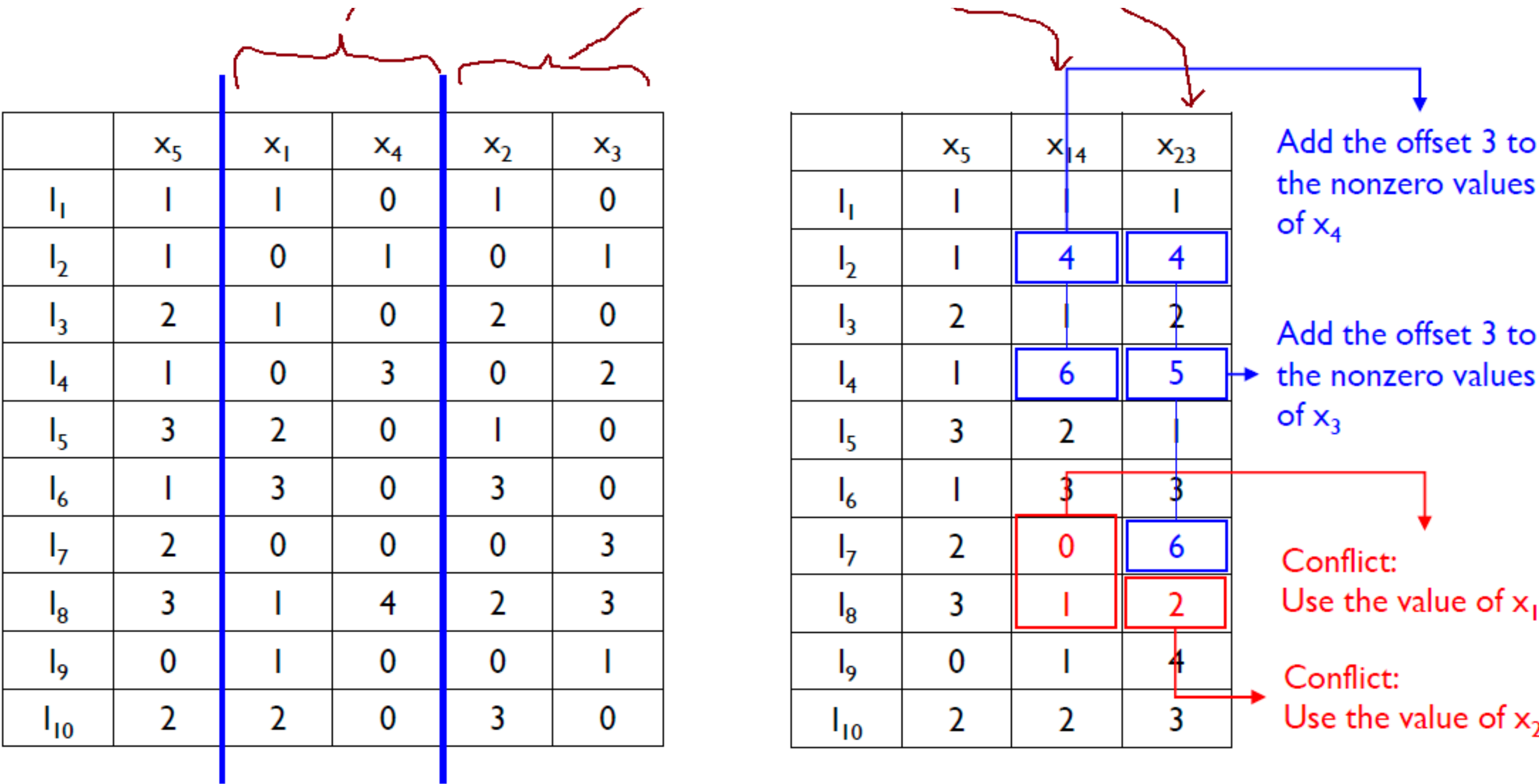
# EFB, Exclusive Feature Bundling

- EFB는 데이터셋의 *Feature* 수를 줄이는 알고리즘
- 실제 categorical data 등 고차원 데이터는 sparse 함
- 데이터는 sparse하다는 가정 아래, 상호 배타적인 변수들을 하나의 bundle로 묶어서 Feature 수를 줄임
  - "상호배타적인 변수들"은 동시에 0이 아닌 값을 갖지 않는 변수
- 상호배타적인 변수들을 묶어서 최소의 bundle 로 만드는 문제는 NP-complete 문제
  - Graph 색칠 문제와 같은 문제 -> 다항시간 내 해를 찾을 수 없음
- 따라서, Greedy 알고리즘을 활용

# EFB (Exclusive Feature Building)

- Feature의 갯수를 효과적으로 줄여 계산 속도를 개선하고자 함

예) 5개의 원래 feature를 3개로 줄이는 경우 (max conflict count  $K = 2$ )



# EFB (Exclusive Feature Building)

- Bundle을 찾는 데에는 Green algorithm이 사용됨
- Bundle 내의 변수를 하나의 변수로 합하는 과정도 필요
  - conflict 한 경우는 기존 변수 값 그대로 가져감
  - conflict가 아닌 (상호배타적인) 경우 기존 변수의 최대값 + 다른 변수 값을 취함

---

**Algorithm 3: Greedy Bundling**

---

**Input:**  $F$ : features,  $K$ : max conflict count

Construct graph  $G$

$searchOrder \leftarrow G.sortByDegree()$

$bundles \leftarrow \{\}$ ,  $bundlesConflict \leftarrow \{\}$

**for**  $i$  **in**  $searchOrder$  **do**

$needNew \leftarrow \text{True}$

**for**  $j = 1$  **to**  $len(bundles)$  **do**

$cnt \leftarrow \text{ConflictCnt}(bundles[j], F[i])$

**if**  $cnt + bundlesConflict[i] \leq K$  **then**

$bundles[j].add(F[i])$ ,  $needNew \leftarrow \text{False}$   
            **break**

**if**  $needNew$  **then**

        Add  $F[i]$  as a new bundle to  $bundles$

**Output:**  $bundles$

---

---

**Algorithm 4: Merge Exclusive Features**

---

**Input:**  $numData$ : number of data

**Input:**  $F$ : One bundle of exclusive features

$binRanges \leftarrow \{0\}$ ,  $totalBin \leftarrow 0$

**for**  $f$  **in**  $F$  **do**

$totalBin += f.numBin$

$binRanges.append(totalBin)$

$newBin \leftarrow \text{new Bin}(numData)$

**for**  $i = 1$  **to**  $numData$  **do**

$newBin[i] \leftarrow 0$

**for**  $j = 1$  **to**  $len(F)$  **do**

**if**  $F[j].bin[i] \neq 0$  **then**

$newBin[i] \leftarrow F[j].bin[i] + binRanges[j]$

**Output:**  $newBin, binRanges$

---

# Merge Exclusive Features

- bundle 내에서 기준 변수 설정
- 모든 instance에 대해서, conflict 한 경우는 기준 변수 값 그대로 가져감
- 모든 instance에 대해서, conflict가 아닌 (상호 배타적인) 경우 기준 변수의 최대값 + 다른 변수 값을 취함

---

**Algorithm 4:** Merge Exclusive Features

---

**Input:**  $numData$ : number of data

**Input:**  $F$ : One bundle of exclusive features

$binRanges \leftarrow \{0\}$ ,  $totalBin \leftarrow 0$

**for**  $f$  **in**  $F$  **do**

$totalBin += f.numBin$

$binRanges.append(totalBin)$

$newBin \leftarrow new\ Bin(numData)$

**for**  $i = 1$  **to**  $numData$  **do**

$newBin[i] \leftarrow 0$

**for**  $j = 1$  **to**  $len(F)$  **do**

**if**  $F[j].bin[i] \neq 0$  **then**

$newBin[i] \leftarrow F[j].bin[i] + binRanges[j]$

**Output:**  $newBin, binRanges$

---



# 특장점

## 빠른 속도

Table 2: Overall training time cost comparison. LightGBM is lgb\_baseline with GOSS and EFB. EFB\_only is lgb\_baseline with EFB. The values in the table are the average time cost (seconds) for training one iteration.

	xgb_exa	xgb_his	lgb_baseline	EFB_only	LightGBM
Allstate	10.85	2.63	6.07	0.71	<b>0.28</b>
Flight Delay	5.94	1.05	1.39	0.27	<b>0.22</b>
LETOR	5.55	0.63	0.49	0.46	<b>0.31</b>
KDD10	108.27	OOM	39.85	6.33	<b>2.85</b>
KDD12	191.99	OOM	168.26	20.23	<b>12.67</b>

Table 3: Overall accuracy comparison on test datasets. Use AUC for classification task and NDCG@10 for ranking task. SGB is lgb\_baseline with Stochastic Gradient Boosting, and its sampling ratio is the same as LightGBM.

	xgb_exa	xgb_his	lgb_baseline	SGB	LightGBM
Allstate	0.6070	0.6089	0.6093	$0.6064 \pm 7e-4$	<b><math>0.6093 \pm 9e-5</math></b>
Flight Delay	0.7601	0.7840	0.7847	$0.7780 \pm 8e-4$	<b><math>0.7846 \pm 4e-5</math></b>
LETOR	0.4977	0.4982	0.5277	$0.5239 \pm 6e-4$	<b><math>0.5275 \pm 5e-4</math></b>
KDD10	0.7796	OOM	0.78735	$0.7759 \pm 3e-4$	<b><math>0.78732 \pm 1e-4</math></b>
KDD12	0.7029	OOM	0.7049	$0.6989 \pm 8e-4$	<b><math>0.7051 \pm 5e-5</math></b>



# 특장점

빠른 수렴 속도

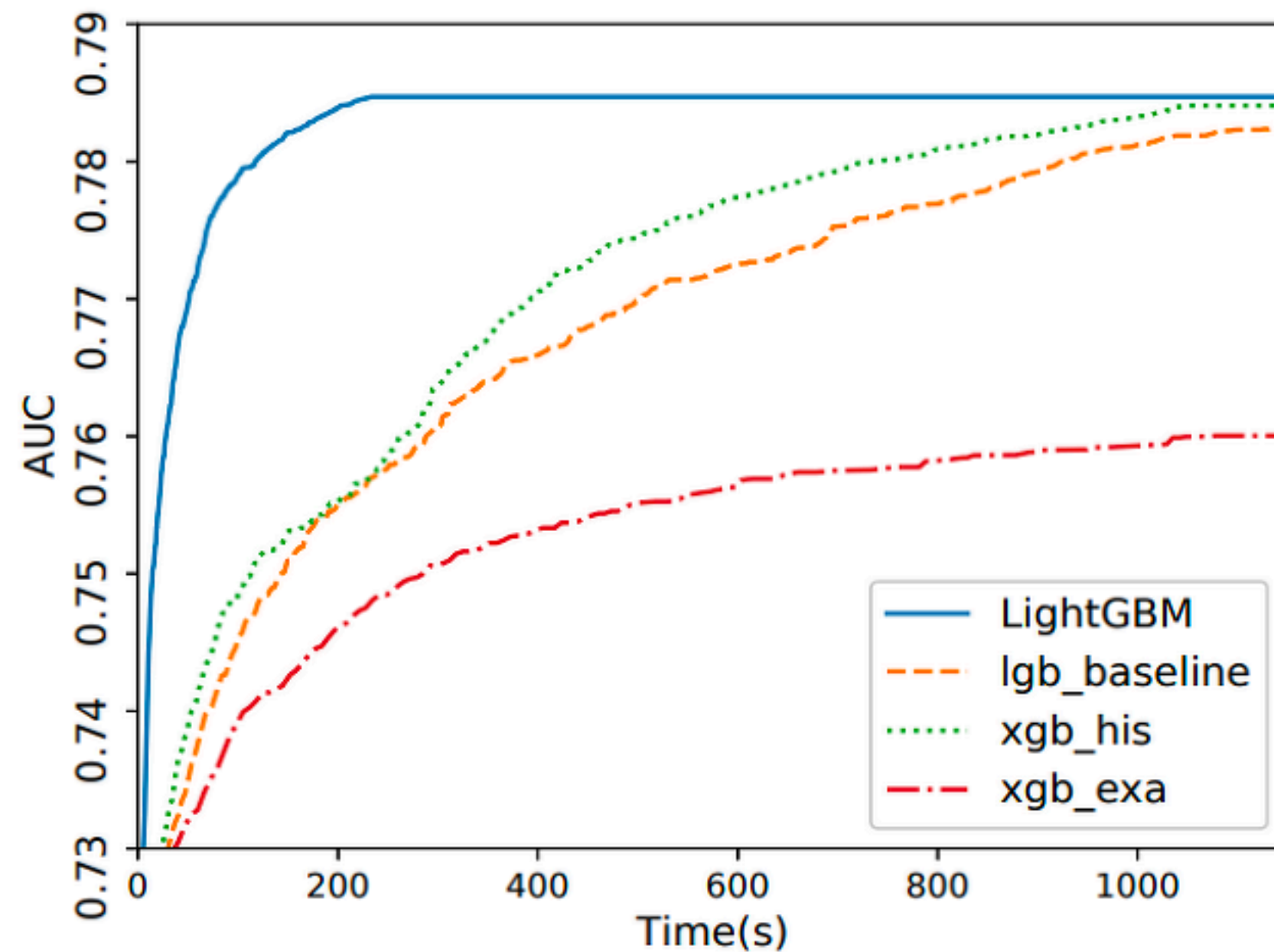


Figure 1: Time-AUC curve on Flight Delay.

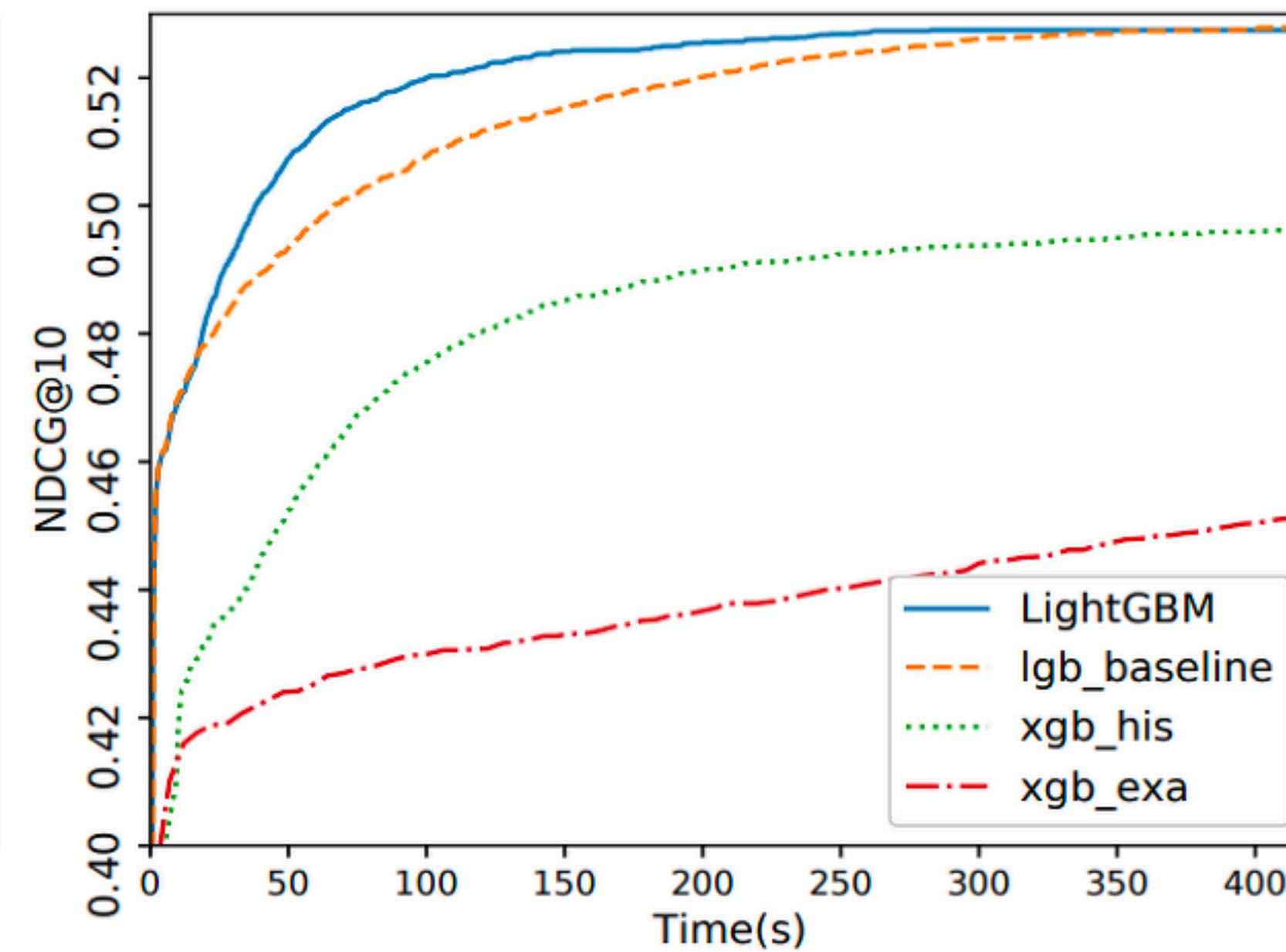


Figure 2: Time-NDCG curve on LETOR.

# Random Forest

# ML 모형 - Random Forest 개요

- 랜덤 포레스트(Random Forest)는 배깅과 부스팅보다 더 많은 무작위성\*을 주어 약한 학습기들을 생성한 후 이를 선형 결합하여 최종 학습기를 만드는 방법
- 배깅을 하되, 개별 tree에서 사용하는 feature를 subset을 활용함으로써, tree의 다양성을 확보하고, 동시에 예측치의 독립성을 확보
- 무작위성(Randomness) : 사건에 패턴이나 예측 가능성이 없는 것, 인위적인 요소가 없는 성질
- 랜덤 포레스트는 정확도와 예측력 측면에서 좋은 성과를 보여줍니다.
- 하지만, 이론적 설명이나 최종 결과에 대한 해석이 어렵다는 단점이 있습니다.



# Parameter

포레스트 크기	<ul style="list-style-type: none"><li>- 총 포레스트를 몇 개의 트리로 구성할지를 결정하는 매개변수</li><li>- 포레스트가 작을 때 : Overfitting</li><li>- 포레스트가 클 때 : Generalization이 우수</li></ul>
최대 허용 깊이	<ul style="list-style-type: none"><li>- 하나의 트리에서 루트 노드부터 종단 노드까지 최대 몇 개의 노드 결정</li><li>- 최대 허용 깊이가 작으면 과소 적합(Under-fitting),</li><li>- 깊으면 과대 적합이 일어나기 때문에 적절한 값 설정 필요</li></ul>
임의성 정도	<ul style="list-style-type: none"><li>- 임의성 정도에 따라 비상관화 수준 결정</li></ul>