Introduction to Ensemble Learning
앙상블 학습에 대한 이해

6주차 스터디 세션

2023.12.15(금) 09:30

김설진/안정현/이승용/이지흔/이혜준

Index

학습 목차

부스팅 모델 확장

1) Gradient Boosting Machine
2) XGBoost
3) LightGBM
4) Catboost

8.1 객체와 앙상블

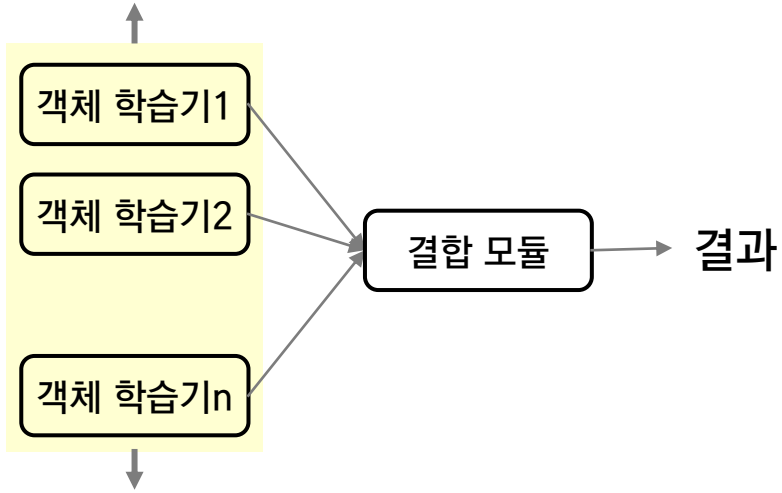
앙상블의 핵심 아이디어

다수의 약한 학습기의 결합을 통해 우수한 **일반화 성능**을 얻는 방법론

일반화 성능이 랜덤 학습기(50%)보다 조금 더 좋은 수준

과적합을 줄이면서 예측 성능↑

동질적(Homogeneous)인 경우
객체 학습기들이 트리 또는 신경망 알고리즘
한 종류로만 구성



이질적(Heterogeneous)인 경우
객체 학습기들이 트리, 신경망 등 **여러 알고리즘**으로
구성

좋은 앙상블 모델의 조건

객체 학습기는 일정 수준의 **'정확성'**이 있어야 하고 동시에 **'다양성'**도 가져야 함

- 단, 기초 학습기 사이의 오차가 상호독립적
- 실질적으로 객체 학습기의 정확성과 독립성(다양성)은 서로 모순됨
  - 같은 학습 데이터를 사용하기 때문
  - 다양성 ↑ 정확성 ↓

8.1 객체와 앙상블

앙상블 방법론

앙상블 학습 연구의 핵심

'좋지만 다른' 객체 학습기를

어떻게 생성하고 결합하는가?

8.2 부스팅

부스팅의 핵심 아이디어

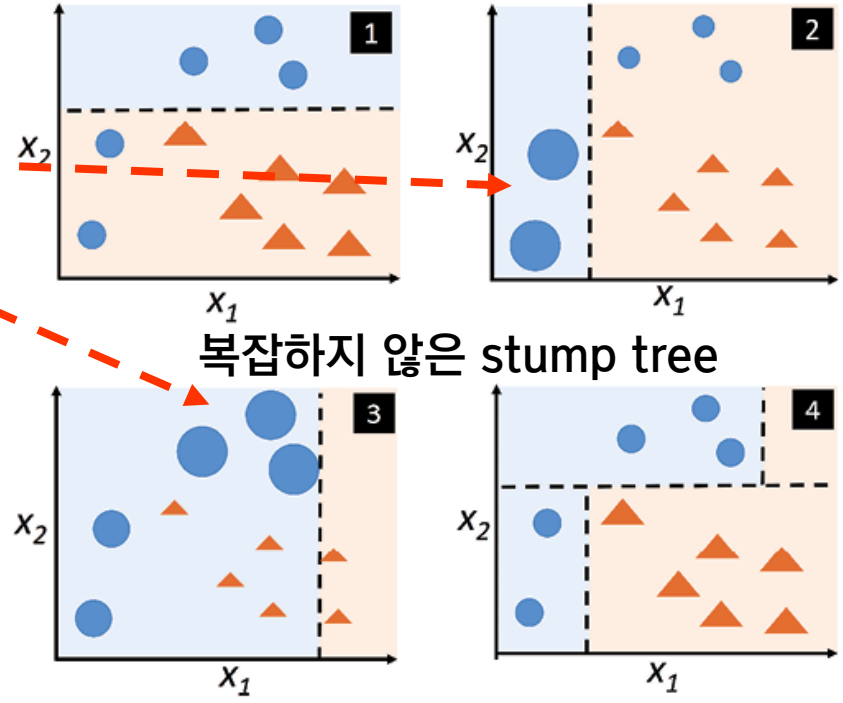
잘못 분류한 샘플에 대해서 집중적으로 훈련하는 학습 방법

알고리즘 절차

1. 초기 훈련 세트로부터 기초 학습기를 훈련시킴
2. 앞선 기초 학습기에서 오류를 범했던 훈련 샘플에 대해 주의하도록 설정 (재가중치 re-weighting)
3. 조정 후의 샘플 분포를 기반으로 다음 기초 학습기를 훈련
4. 사전에 설정해 놓은 기초 학습기 개수 T에 도달할 때까지 위 절차(2~3) 반복
5. 최종적으로 얻은 T개의 기초 학습기에 대한 가중 결합 진행

Boosting에서는 오류를 바탕으로 샘플 분포를 갱신하는 것이 중요

대표 알고리즘 AdaBoost



복잡하지 않은 stump tree

Bagging과 달리 병렬 처리 불가

8.2.1 AdaBoost

AdaBoost의 의사코드

Algorithm 2 Adaboost

Input: Required ensemble size T 총 T개의 개별 학습기 생성

Input: Training set $S = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, where $y_i \in \{-1, +1\}$

Define a uniform distribution $D_1(i)$ over elements of S. 처음 각 샘플이 추출될 확률을 균등 분포로 정의

for $t = 1$ to T **do** 개별 모델은 Stump tree

Train a model h_t using distribution D_t.

Calculate $\epsilon_t = P_{D_t}(h_t(x) \neq y)$ 현재 학습된 데이터에 대한 error rate 계산

If $\epsilon_t \geq 0.5$ break Error rate은 최소한 0.5 이상은 되어야 다음 절차 진행

T번 반복 Set $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ ← 지수손실함수의 도함수에서 도출 → Error rate = 0.5(random guessing) -〉 알파는 0
Error rate = 0 -〉 알파는 무한대
=〉 분류기의 성능이 좋을수록 높은 가중치를 부여함

Update $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$

where Z_t is a normalization factor so that D_{t+1} is a valid distribution.

end for

For a new testing point (x', y'),

$H(x') = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x')\right)$ T개의 분류기에 대한 예측결과를 가중 결합하여 최종 예측 결과를 출력

8.2.1 AdaBoost

샘플 분포 업데이트

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$

$$= \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

t+1시점의 sampling distribution

t시점의 sampling distribution

알파(a)에 대한 해석

1. 분류기의 성능이 좋으면(알파가 크면), 다음 단계에서의 선택 확률을 낮추고
2. 분류기의 성능이 안 좋으면(알파가 작으면), 다음 단계에서의 선택 확률을 높임

y x h(x)에 대한 해석

1. 실제값(y)와 예측값(h(x))가 같으면 1
2. 실제값(y)와 예측값(h(x))가 다르면 -1

즉, 해당 분류기가 정답을 맞춘 개체에 대해서는 선택 확률을 낮추고, 못 맞춘 개체에 대해서는 선택 확률을 높임

8.2.1 AdaBoost

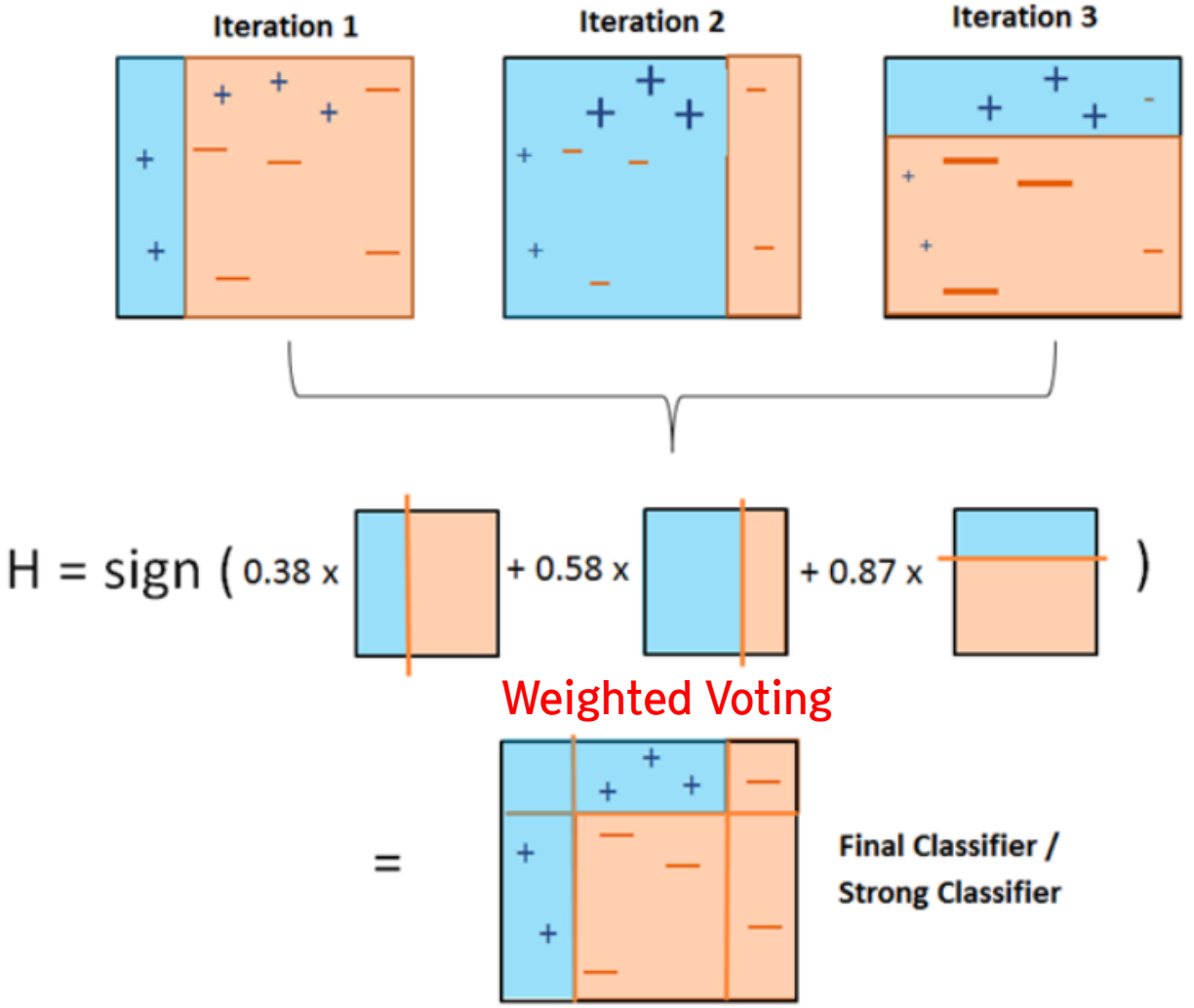
AdaBoost 정리

Boosting

- 재가중(re-weighting) 방법을 통해 학습 진행
- 가중치를 가질 수 없는 샘플에 대해서는 리샘플링 (re-sampling) 진행
- 리샘플링을 통해 랜덤 예측(0.5)보다 성능이 좋지 않아 학습이 조기 종료되는 상황을 방지
- parallel processing 불가

Bagging

- 모든 샘플의 sampling 확률이 균등하다는 조건에서 복원추출(bootstrapping) 진행
- parallel processing 가능

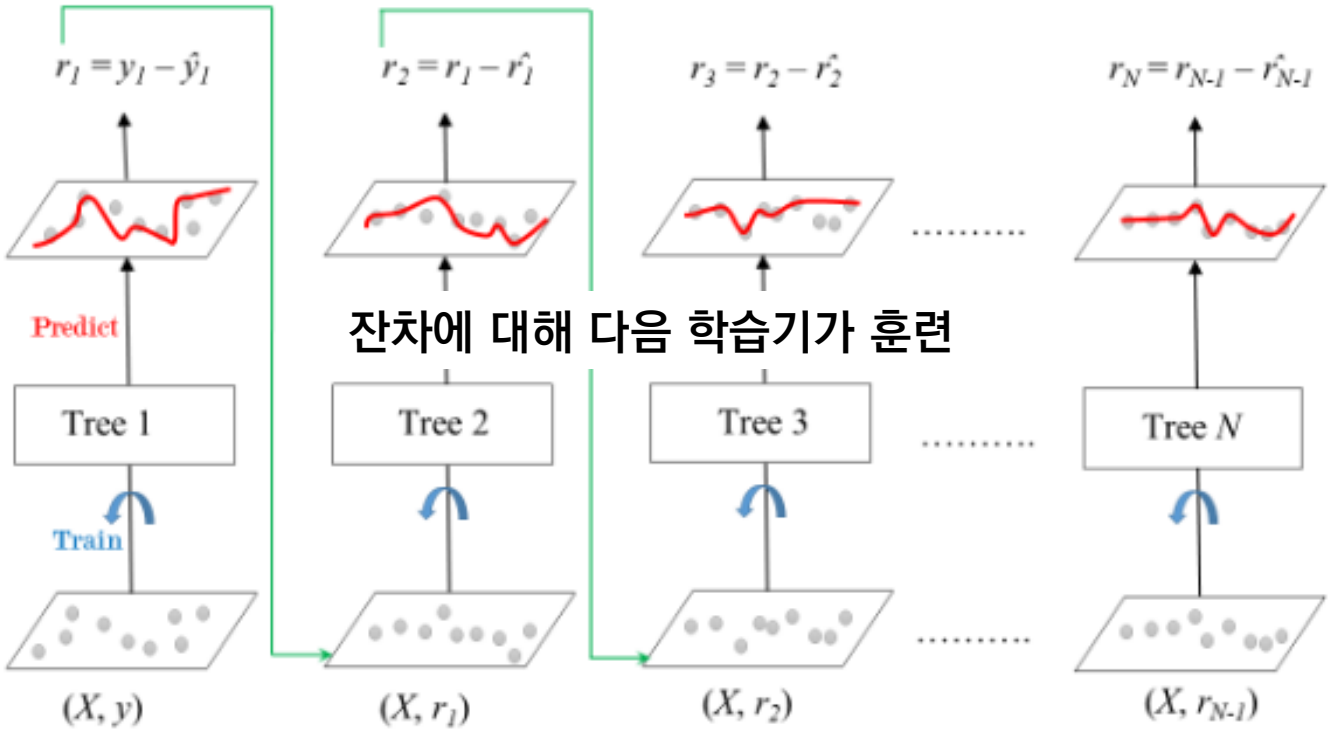


8.2.2 GradientBoostingMachine

GBM의 핵심 아이디어

이전 학습기의 잔차에 대해 다음 학습기가 훈련하는 방법으로 Gradient Descent 방식과 유사

알고리즘 절차



잔차에 대해 다음 학습기가 훈련

경사하강법과의 유사성

최소자승법에 대한 Loss function

$$\min L = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2$$

f(x)에 대한 편미분

$$\frac{\partial L}{\partial f(\mathbf{x}_i)} = f(\mathbf{x}_i) - y_i$$

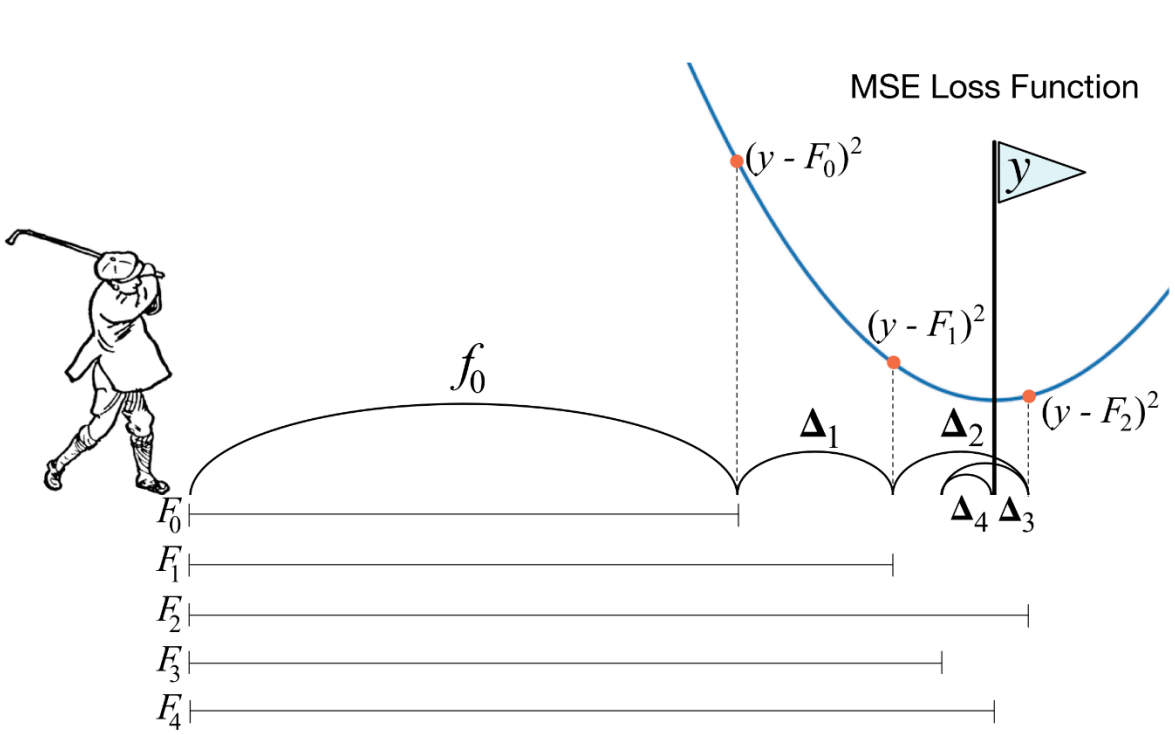
gradient

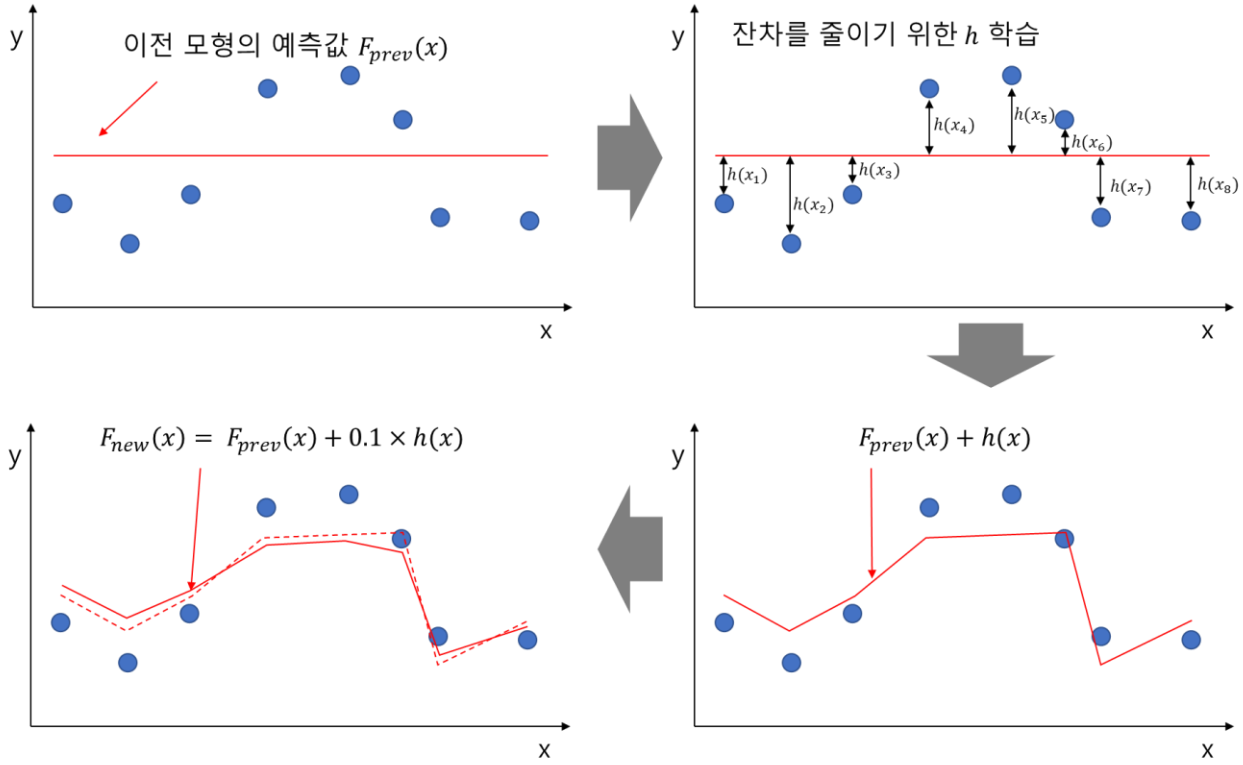
$$y_i - f(\mathbf{x}_i) = -\frac{\partial L}{\partial f(\mathbf{x}_i)}$$

잔차(residual) Gradient의 반대 방향

8.2.2 GradientBoostingMachine

GBM과 Gradient Descent와의 유사성





8.2.2 GradientBoostingMachine

Gradient Boosting Algorithm (Regressor 기준)

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$. 초기 학습기는 상수 y=r로 하는 f(x) 지정

2. For $m = 1$ to M:

   (a) For $i = 1, 2, \ldots, N$ compute N개의 샘플에 대해 반복

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$

유사 잔차(Pseudo Residual) 계산

   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.

   (c) For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$

   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.

M번 반복

앞서의 잔차를 종속변수로 하는 트리 학습기 학습

각 샘플의 loss를 최소화할 수 있는 상수 r 계산

새로 학습된 학습기로 업데이트

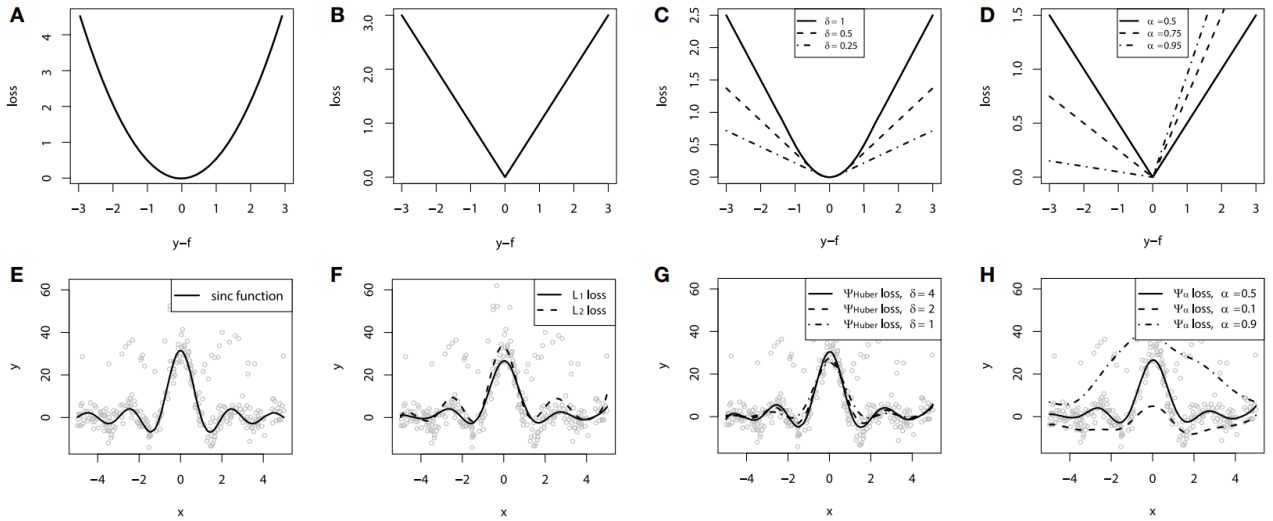
3. Output $\hat{f}(x) = f_M(x)$. 최종 모델 출력

8.2.2 GradientBoostingMachine

Gradient Boosting의 특징

장점

- 잔차를 줄여나가는 방식으로 정확도가 높은 편
- DecisoinTree 외에도 다양한 학습기를 base learner로 사용할 수 있고, 다양한 손실함수를 활용할 수 있기 때문에 유연함 (huber loss, quantile loss, Bernoulli loss 등)



단점

- 노이즈까지 학습하게 되어 과적합 발생 가능성이 큼
- 각 학습기가 복잡한 편이기 때문에 iteration이 많을수록 메모리 문제가 발생
- 각 입력 변수에 대해 출력 변수가 어떻게 변하는지 해석이 어려움

- Subsampling : 각 iteration마다 전체가 아닌 일부 데이터(random)에 대해서만 학습을 진행
- Shrinkage : 가중치를 부여하여, 추가적으로 학습된 학습기에 대한 영향력을 줄임
- EarlyStopping : 조기학습 종료 조건을 지정하여 과적합 방지

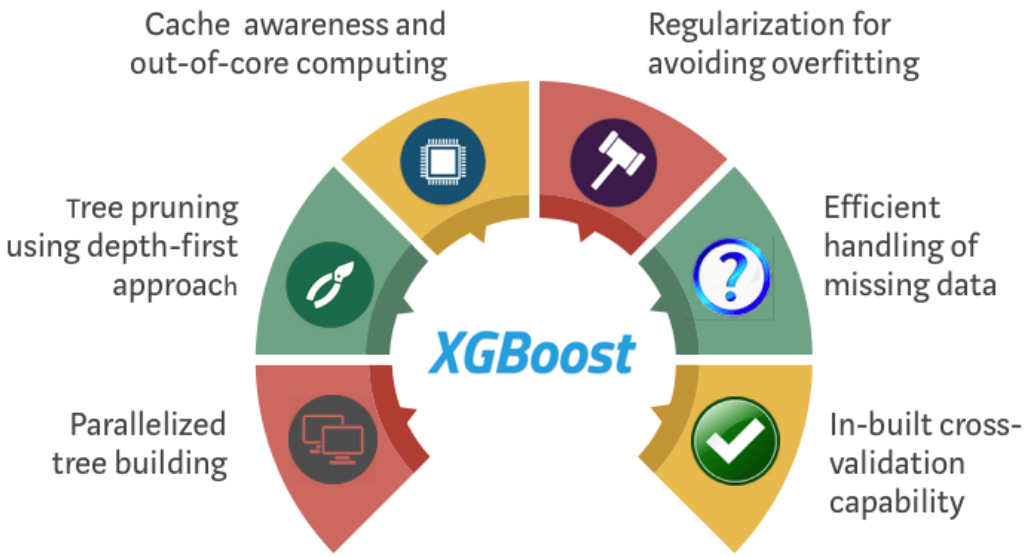
8.2.3 XGBoost

XGBoost의 핵심 아이디어

분할 지점을 근사적으로 찾아내고, 이를 병렬처리화함으로써 GBM을 최적화한 방법

Not Greedy But Aproximate

A Scalable Tree Boosting System



1. 계산 속도가 빠른 cache 메모리를 사용하여 처리 속도 향상
2. 과적합을 방지하기 위해 규제가 더해진 손실 함수 사용
3. 결측값 처리에 효과적인 알고리즘 사용
4. 병렬적으로 트리 생성 (기존 부스팅 모델에서는 불가능)
5. 내장된 교차검증 기능
6. Greedy하지 않은 깊이 우선 탐색을 통해 최적해를 근사화

HW + SW 각각의 개선을 통해 속도 향상

8.2.3 XGBoost

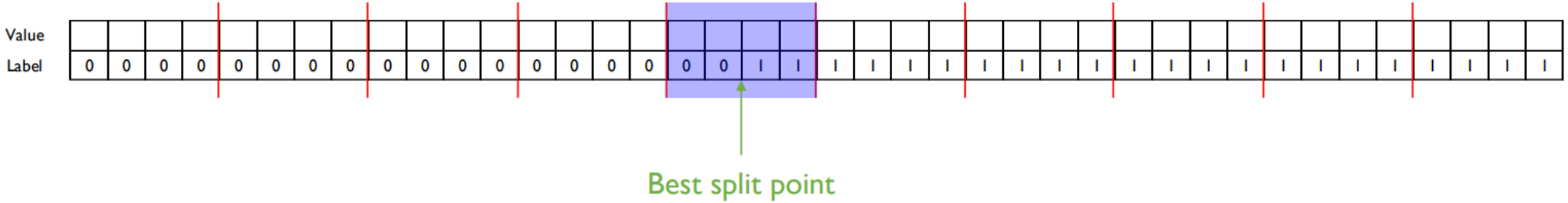
Split Finding Algorithm

Exact greedy algorithm

- 모든 경우의 수를 확인하여, split point를 탐색하는 알고리즘
- 항상 최적의 분기점을 찾을 수 있기 때문에, 높은 정확도와 성능을 기대할 수 있음
- 하지만, 데이터의 크기가 메모리를 초과할 경우에는 탐색을 진행하지 못하고 또한 분산처리가 불가능

Approximate algorithm

- 전체 데이터를 분할하고 각 분할된 영역(bucket) 내에서 gradient를 계산하여 찾아낸 split point를 best point로 approximate
- 탐색 횟수를 줄이면서 병렬처리가 가능하기 때문에 메모리와 시간 효율성을 극대화할 수 있음
- Bucket의 크기와 사이즈는 하이퍼파라미터(epsilon)을 통해 조정할 수 있으며, global, local 방식 중 선택할 수 있음



8.2.3 XGBoost

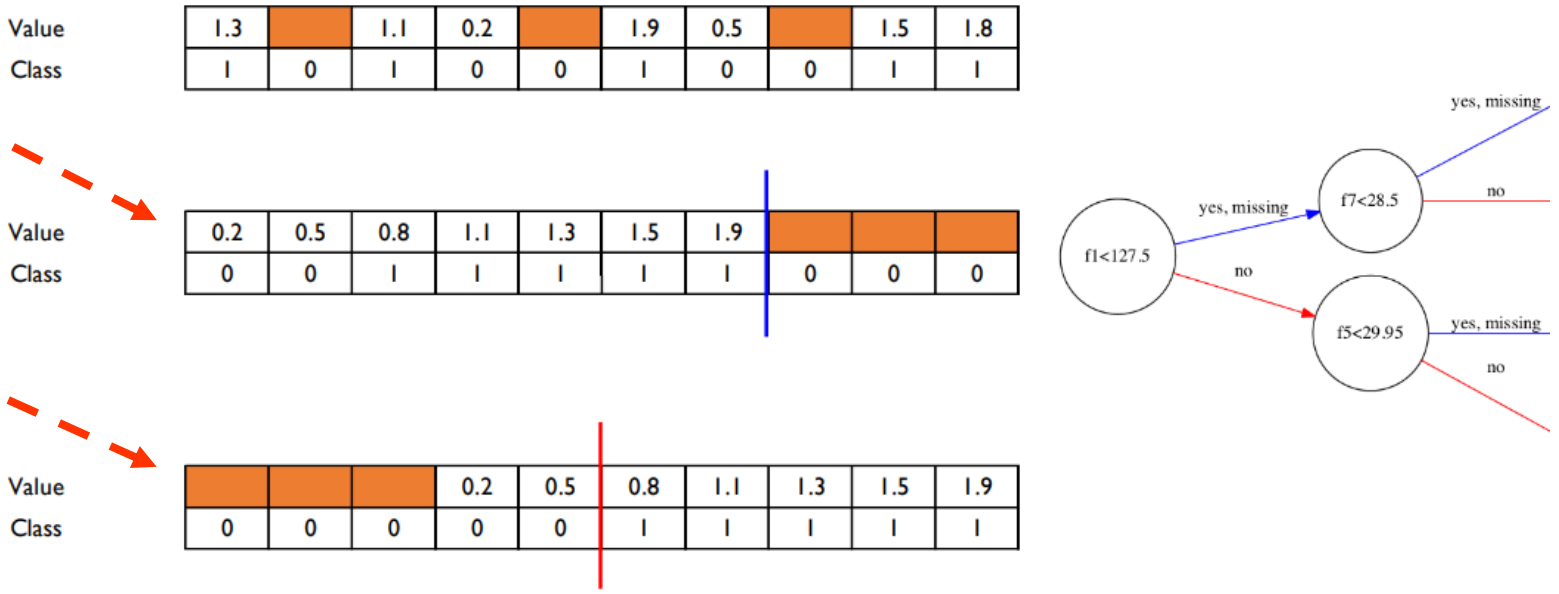
Sparsity-Aware Split Finding

Algorithm 3: Sparsity-aware Split Finding

Input: I, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
Input: d, feature dimension
Also applies to the approximate setting, only collect statistics of non-missing entries into buckets
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  // enumerate missing value goto right
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted*$(I_k,$ *ascent order by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
  // enumerate missing value goto left
  $G_R \leftarrow 0,\ H_R \leftarrow 0$
  for j *in sorted*$(I_k,$ *descent order by* $\mathbf{x}_{jk})$ **do**
    $G_R \leftarrow G_R + g_j,\ H_R \leftarrow H_R + h_j$
    $G_L \leftarrow G - G_R,\ H_L \leftarrow H - H_R$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split and default directions with max gain

Sparsity-aware split finding

- XGBoost는 결측치에 대한 효율적인 처리가 가능
- missing value에 해당하는 부분들을 한쪽 방향(왼쪽 또는 오른쪽)으로 몰아 넣은 다음
- 각각의 gradient를 계산하여 gradient가 가장 큰 경우의 방향(direction)을 default direction으로 하여 결측값을 처리



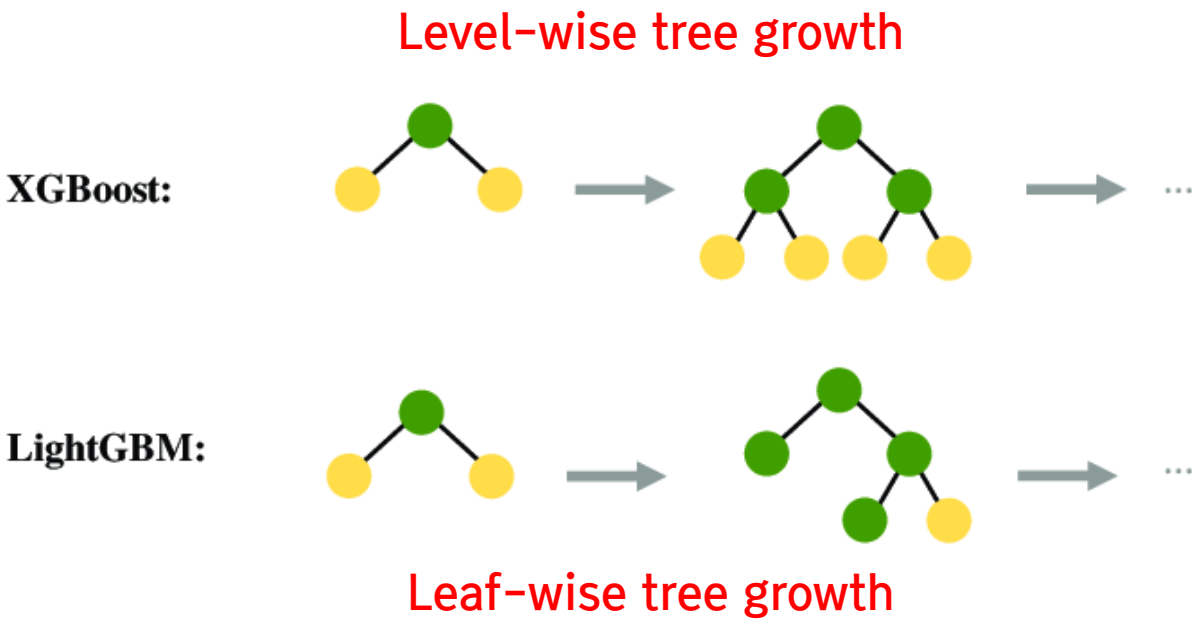
8.2.4 LightGBM

LightGBM의 핵심 아이디어

성능은 유지시키면서 학습 시간을 단축시킨 효율적인 GBM 알고리즘

데이터 수가 충분히 큰 경우에 성능 보장 일부 데이터와 변수만을 사용

- 기존의 Gradient Boosting Machine(GBM)의 경우, 모든 feature에 대해서 데이터의 모든 개체(instance)들을 탐색
- 하지만, 데이터셋의 크기가 커지면 커질수록 탐색해야 하는 경우의 수는 기하급수적으로 증가
- 이러한 단점을 극복하기 위해 탐색하는 데이터의 수와 특징 수를 줄이는 방법론이 바로 LightGBM
  - GOSS (데이터 선택 방식)
  - EFB (특성 선택 방식)



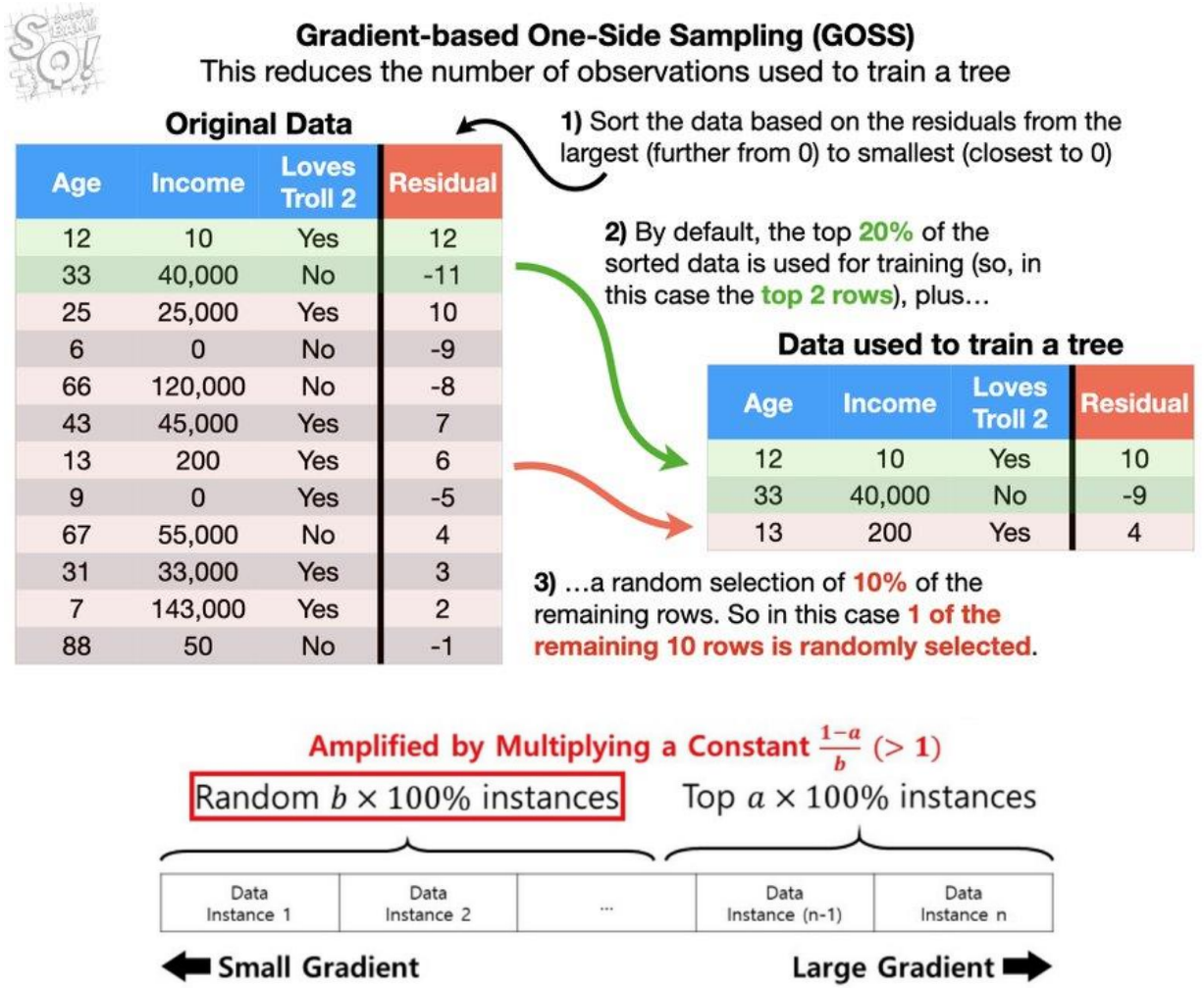
8.2.4 LightGBM

Gradient-based One-Side Sampling(GOSS)

Gradient-based One-Side Sampling

- Gradient값이 클수록 영향력은 커질 것이고, 이러한 점을 이용하여 불필요한 개체들은 제외하고 중요한 개체들만 가져가는 것이 LightGBM의 데이터 선택 방식

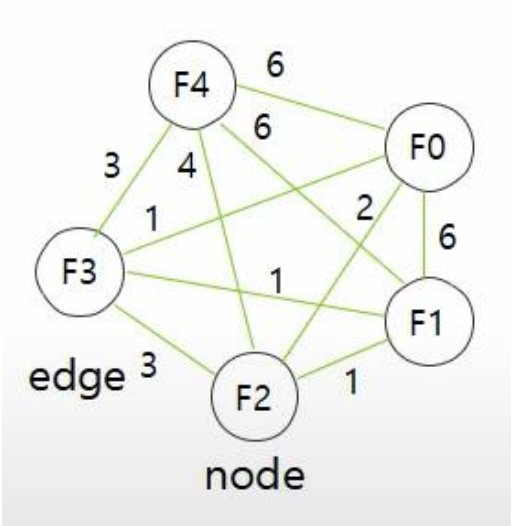
- 그림을 통해 시각적으로 확인해보면 각 데이터 개체(instance)별로 gradient(residual)를 계산하고, 이 중 top 20%(상위 2개 선택)와 나머지 중 랜덤하게 10%(나머지 10개 중 임의로 1개 선택)를 사용하여 tree를 학습



8.2.4 LightGBM

Exclusive Feature Bundling(EFB)

- 희소 특성 벡터 공간(sparse feature space)에서 수많은 변수(feature)들은 거의 대부분 배타적(exclusive)
- LightGBM은 이러한 exclusive한 변수들을 하나의 묶음(Bundle)으로 처리하여 변수의 수를 축소



Greedy Bundling(변수들끼리 묶는 작업)

- graph coloring problem으로 계산
- 각 특성(feature)들을 노드, exclusive한 관계(두 노드 사이에 동시에 0이 아닌 정도)를 엣지로 표현
- 두 노드 사이에 강한 엣지일 경우 같은 bundle로 묶음

Merge Exclusive Features(묶인 변수들끼리 통합하는 작업)

- 오른쪽 예시를 보면 Feature1(기준 변수)에 0이 포함될 경우, Feature2에 offset(기준 변수의 최대값)인 4를 더해 새로운 Bundled Feature을 생성하고, 반대로 Feature2에 0이 있을 경우 그대로 기준 feature인 Feature1의 값을 가져감

Feature 1	Feature 2	Bundled Feature
0	2	6
0	1	5
0	2	6
1	0	1
2	0	2
3	0	3
4	0	4

8.2.5 CatBoost

CatBoost의 핵심 아이디어

기존 GBM 모델들이 가진 문제를 해결하고 **카테고리 변수에 최적화**된 방법론

학습에 y값을 사용하는 문제

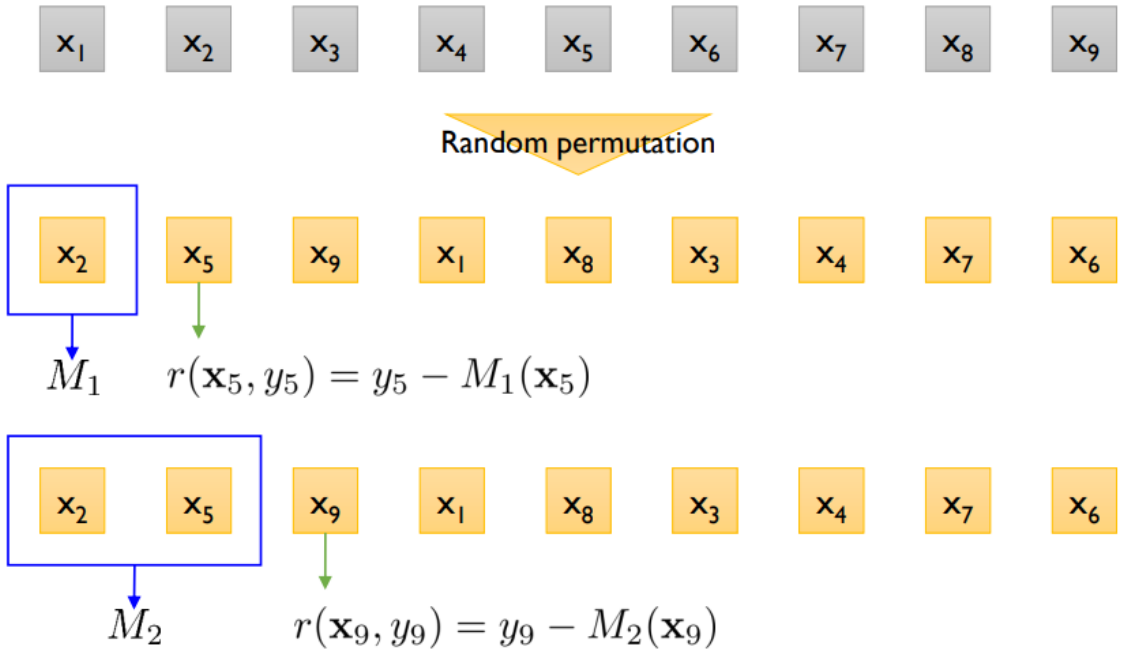
기존 GBM의 문제 1

Train과 test 사이에

Prediction shift 발생

Ordered Boosting

- 모든 train 데이터에 대해 잔차를 계산하는 것이 아닌 일부 데이터에 대해서만 잔차를 계산
- 랜덤 permutation 후 순차적으로 잔차를 계산하며 모델 개선



8.2.5 CatBoost

CatBoost의 핵심 아이디어

기존 GBM 모델들이 가진 문제를 해결하고 카테고리 변수에 최적화된 방법론

학습에 y값을 사용하는 문제

기존 GBM의 문제 2

y값을 통해 카테고리 변수에 대한 수치 변형을 시도했던 기존의 방법론(target statistics)들에는 target leakage 문제가 존재

	…	x^i	…	TS	y
I_1	…	A	…	0.000	I
I_2	…	B	…	I.000	I
I_3	…	C	…	I.000	I
I_4	…	A	…	I.000	0
I_5	…	B	…	0.977	I
I_6	…	C	…	0.982	I
I_7	…	B	…	0.992	0
I_8	…	C	…	0.986	I
I_9	…	C	…	0.992	0
I_{10}	…	C	…		I

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

$$= \frac{3 + 0.1 \times 0.75}{3 + 0.1} = 0.992$$

Ordered Target Encoding

- 시퀀스가 없는 데이터에 인위적인 시간을 부여하여 이전 시점의 y값만을 가지고 인코딩 진행 (cf 원-핫인코딩)

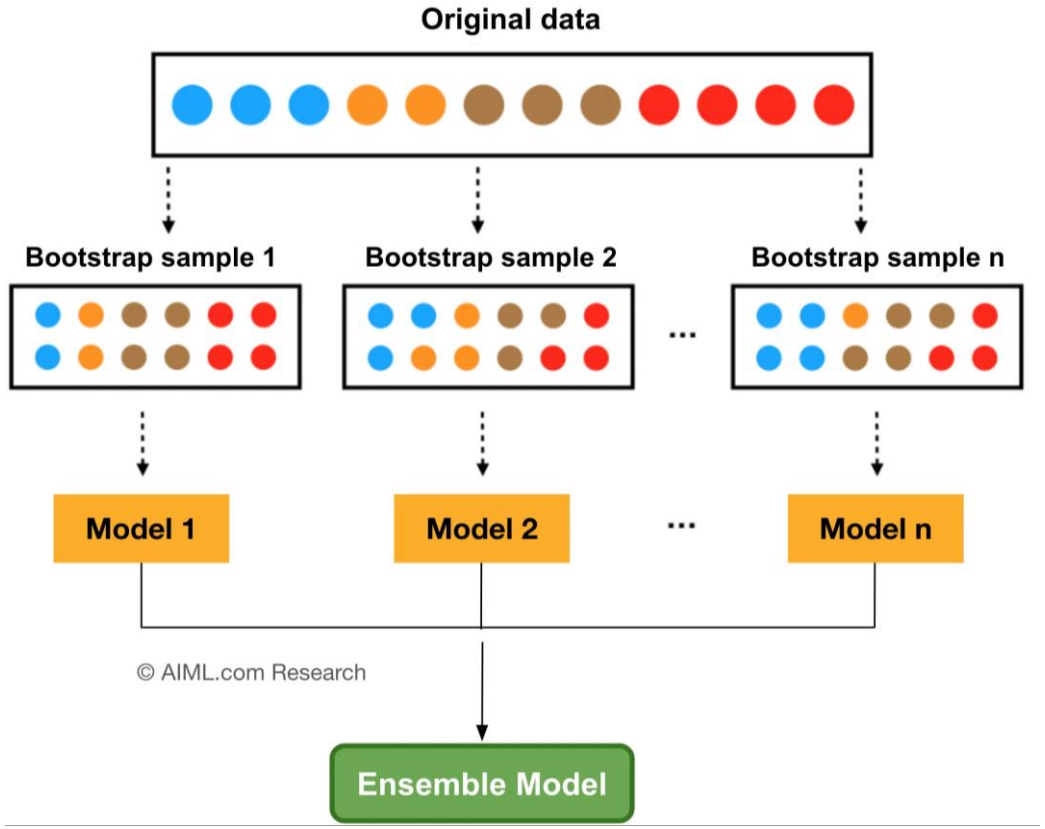
8.3.1 배깅

배깅의 핵심 아이디어

Bootstrapping을 통해 학습한 여러 학습기 결과를 Aggregating하는 전략

전체 데이터셋 중 일부(63.2%)만 샘플링하여 학습

단순 투표(분류) 또는 단순 평균(회귀)로 집계



- Bootstrapping 복원 추출 시행
- 샘플링/평균 집계에 따른 계산 복잡도는 단일 학습기와 비슷한 수준 (효율적인 모델)
- 다양한 모델(DecisionTree, SVM, Logistic)들로 병렬 처리 학습 가능
- 배깅은 분산을 줄이는데 초점이 맞춰져 있어, 과적합 가능성이 높은 학습기에 효과적

OOB 데이터

- 부트스트래핑 결과 남은 36.8%의 샘플 데이터를 검증에 활용
- 사후 가지치기(pruning), 조기 종료(early stopping)에 활용
- OOB에 대한 일반화 오차

$$\mathbf{e}^{oob} = \frac{1}{|D|} \sum_{(\mathbf{x},\mathbf{y}) \in D} I\left(H^{oob}(x) \neq y\right)$$

8.3.2 랜덤 포레스트

랜덤 포레스트의 핵심 아이디어

배깅에 랜덤 속성 선택을 더해 일반화 성능을 높인 알고리즘

다양성 증대

앙상블의 다양성 증가

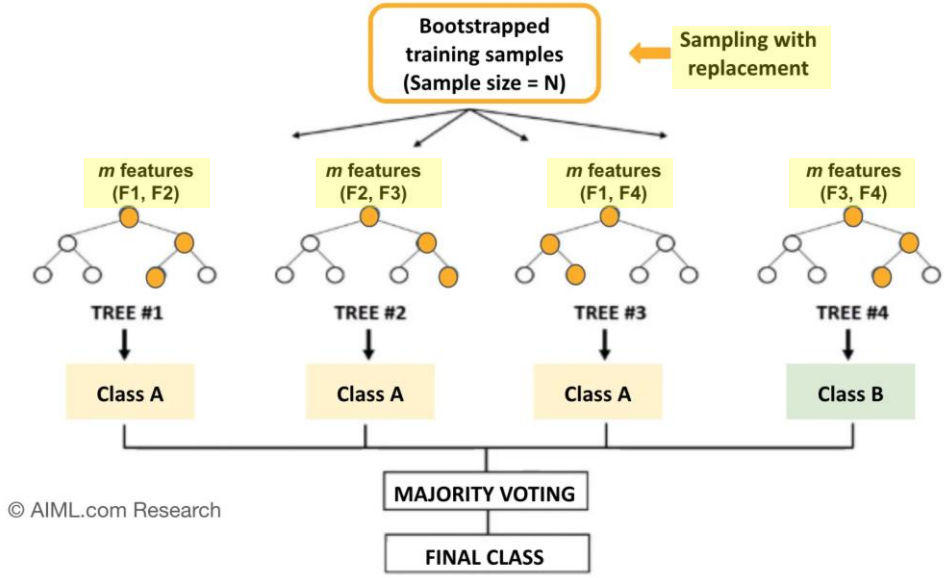
1. 부트스트래핑
2. 랜덤 속성 선택

일반화 성능 향상

데이터와 속성값에 대한 샘플링 전략을 통해 일반화 오차를 획기적으로 줄일 수 있음

$$Generalization\ Error \leqq \frac{\bar{\rho}(1-s^2)}{s^2}$$

- $\bar{\rho}$: 개별 tree 간의 상관 계수의 평균값 (다양성)
- s^2 : 정답과 오답 사이의 평균적인 차이 (정확도)
- 개별 모델의 정확도가 높을수록, 모델 간 상관 관계가 낮을수록 일반화 오차는 작아짐



- 전체 d개의 feature 중 k개의 feature 부분 집합 선택
  - 1 <= k <= d
  - 최적의 k는 log2d

8.3.2 랜덤포레스트

랜덤 포레스트의 의사코드

1. For $b = 1$ to B: 전체 샘플 중 N개의 부트스트랩 샘플 추출
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables. 전체 속성 p개 중 m개를 랜덤하게 선택
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$. Regression은 평균으로 집계

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$. Classification은 다수결로 집계

8.3.3 변수 중요도

Feature Importance

변수 중요도(Feature Importance) 계산 과정

1. 부트스트랩 샘플에 활용되지 않은 OOB에 대한 에러(e)를 계산

2. 변수 x를 permutation시켰을 때의 OOB에러(p)를 계산

$$d_i^m = p_i^m - e_i^m$$

3. 모든 tree들에 대해서 p-i의 평균과 분산을 계산

$$\bar{d}_i = \frac{1}{m}\sum_{i=1}^{m} d_i^m, \quad s_i^2 = \frac{1}{m-1}\sum_{i=1}^{m}(d_i^m - \bar{d}_i)^2$$

4. 비율을 통해 변수 중요도 산출

$$v_i = \frac{\bar{d}_i}{s_i}$$

Validation Data Before Shuffling

Feature #1 (original)	Feature #2 (original)	Label
5	4	1
5	4	1
2	4	0
2	4	0

Model Accuracy: 100%

Validation Data After Shuffling

Feature #1 (scrambled)	Feature #2 (original)	Label
2	4	1
5	4	1
2	4	0
5	4	0

Model Accuracy: 50%

변수 x가 해당 tree에서 분할에 사용되지 않았다면
- p와 e는 유사
- 즉, x가 데이터를 설명하는데 크게 중요하지 않았다는 것을 의미

변수 x가 해당 tree에서 분할에 사용되었다면
- p와 e 사이에 차이가 발생 (e < p)
- x가 분할에 더 자주 사용될수록 그 차이는 증가
- 즉, 해당 변수 x가 데이터를 설명하는데 중요한 역할 수행

EOD