# Random Forest

서로 다른 방향으로 과대적합된 트리를 많이 만들고 그 결과를 평균 내어 과대적합된 양을 줄일 수 있다.

For it

1. 결정 트리를 많이 만들어야 한다.
2. 각 트리는 예측을 잘햐야 한다.
3. 각 트리는 구별되어야 한다.

For #3: 트리 생성 시 무작위성을 주입하여 모든 트리가 서로 달라지록 한다

방법 1: 트리 만들 때 사용하는 데이터를 무작위로 선택한다.

방법 2: 분할 테스트에서 특성을 무작위로 선택한다.

각 노드에서 전체 특성을 대상을 최선의 특성을 찾는 Decision Tree와 달리 Random Forest는 무작위로 선택한 특성 후보 중에서 최적의 특성을 찾는다

→ 트리르 더욱 다양해지고 편향을 손해 보는 대신 분산을 낮춘다

## 만들기

1. 트리 개수 정하기: **n_estimators**
2. 부트스트랩 샘플(Bootstrap Sample) 생성: **bootstrap**

   **n_samples**개의 데이터 포인트 중에서 무작위로 데이터를

   **n_samples** 횟수만큼 중복을 허용하여 반복 추출
3. 만든 데이터셋으로 결정 트리 만들기

   ※ 각 노드에서 전체 특성을 대상으로 최선의 테스트 찾기 X

   Yes: 각 노드에서 후보 특성을 무작위로 선택한 후

   이 후보들 중에서 최선의 테스트 찾기

   → **max_feature** 매개변수로 고를 특성 개수 결정

   후보 특성을 고르는 것은 매 노드마다 반복되므로

   트리의 각 노드는 다른 후보 특성들을 사용하여 테스트를 만든다

## 매개변수

몇 가지 예외는 있지만 (트리 성장 조절을 위한) Decision Tree의 매개변수와 앙상블 자체를 제어하는데 필요한 bagging의 매개변수를 모두 가지고 있다.

1. n_estimators: 트리의 개수

   클수록 좋다

더 많은 트리를 평균하면 과적합을 줄일 수 있다
but 더 많은 트리는 더 많은 메모리와 긴 훈련 시간

2. max_feature 각 트리가 얼마나 무작위가 될지
= n_feature: 특성 선택에 무작위성x but 샘플링의 무작위성 아직o
= 1 : 테스트할 특성 고를 필요x 무작위로 선택한 특성의 임계값 찾기
작은 값이면 과대적합을 줄인다. 일반적으로 기본값을 쓴다

4. n_jobs: 사용할 코어수 지정
※ 코어 개수보다 크게 지정하는 것은 도움 안 됨
모든 코어를 사용하려면 n_jobs=-1

3. random_state

max_feature을 크게 하면 트리들은 비슷해지고 가장 두드러진 특성을 이용해 데이터에 잘 맞춰진다. 낮추면 트리는 많이 달라지고 각 트리는 데이터에 맞추기 위해 깊이가 깊어진다.

## 장점

1. 성능이 뛰어나고 매개변수 튜닝을 많이 하지 않아도 잘 작동하며 스케일을 맞출 필요 없다.
2. 단일 트리의 단점을 보완하고 장점은 그대로
3. 매우 큰 데이터셋에도 잘 작동한다
4. 여러 cpu 코어로 간단하게 병렬화하여 훈련할 수 있다
5. 특성 중요도를 측정하기 쉽다: 어떤 특성을 사용한 노드가 평균적으로 불순도를 얼마나 감소시키는지 확인하여 특성의 중요도를 측정한다. 정확히 말하면 가중치 평균이며 각 노드의 가중치는 연관된 훈련 샘플 수와 같다

## 단점

1. 트리 각각을 자세히 분석하기 어렵고
2. 랜덤포레스트의 트리는 결정트리보다 더 깊어지는 경향(특성의 일부만 사용하므로)
3. 매우 차원이 높고 희소한 데이터에는 잘 작동하지 않는다
4. 선형 모델보다 많은 메모리를 사용하며 훈련과 예측이 느리다

## Gradient Boosting

부스팅: 약한 학습기(깊지 않은 트리)를 여러 개 연결하여 이전 모델을 보완하는 방식으로 강한 학습기를 만드는 앙상블 방법

1. 에이다부스트: 이전 트리에서 과소적합한 샘플의 가중치 높여서 다음 모델 훈련
   반복마다 샘플의 가중치 수정
2. 그래디언트 부스팅: 이전 오차를 보정하도록 순차적으로 트리(에측기) 만들기
   이전 예측기가 만든 잔여 오차에 새로운 예측기 학습

근본 아이디어: 얕은 트리 같은 간단한 모델(약한 학습기)를 많이 연결

1. 각각 트리는 데이터의 일부에 대해서만 예측을 잘 수행할 수 있어서 트리가 많이 추가될수록 성능이 좋아진다
2. 무작위성을 없으나 강력한 가지치기가 사용된다
3. 랜덤 포레스트 보다는 매개변수 설정에 조금 더 민감하지만 잘 조정하면 더 높은 정확도 제공

일반적 관례로 가용한 시간과 메모리 한도에서 **n_estimators**를 맞추고 나서 적절한 **learning_rate**를 찾는다

비슷한 종류의 데이터에서 Random Forest와 Gradient Boosting 둘 다 잘 작동하지만 보통 더 안정적인 Random Forest를 먼저 적용한다.
예측 시간이 더 중요하거나 모델의 마지막 성능까지 쥐어짜야 할 때 Gradient Boosting을 사용하면 도움이 된다.

### 매개변수

1. learning_rate: 이전 트리의 오차를 얼마나 강하게 보정할 것인지 제어
   학습률 크면 보정을 강하게 하여 복잡한 모델 만든다
   낮추면 비슷한 복잡도의 모델을 만들기 위해 더 많은 트리를 추가해야 한다
2. n_estimators: 트리 개수
   키우면 앙상블에 트리가 많이 추가되어 모델의 복잡도가 커지고 훈련 세트에서의 실수를 바로 잡을 기회가 더 많아진다
   랜덤 포레스트와 달리 그레디언트 부스팅에서는 n_estimators를 크게하면 모델이 복잡해지고 과대적합될 가능성이 높아진다.
3. validation_fraction: 검증 데이터 비율
4. n_iter_no_change: 훈련 조기 종료 기준

훈련 데이터에서 validation_fraction(기본값 0.1) 비율만큼 검증 데이터로 사용하여 n_iter_no_change 반복 동안 검증 점수가 향상되지 않으면 훈련이 종료된다. n_iter_no_change 기본값이 none이면 조기 종료 사용하지 않는다

5. max_depth 또는 max_leaf_nodes

통상 그레디언트 부스팅 모델에서는 max_depth를 매우 작게 설정하며 트리의 깊이가 5보다 깊어지지 않게 한다

## 장점

1. 특성의 스케일을 조정하지 않아도 된다.
2. 이진 특성이나 연속적인 특성에서도 잘 작동한다
3. 깊지 않은 트리를 사용하므로 메모리를 적게 사용하고 예측도 빠르다

## 단점

1. 매개변수 설정에 조금 민감하여 잘 조정해야한다
2. 훈련 시간이 길다
3. 희소한 고차원 데이터에는 잘 작동하지 않는다

## XGBoost

Exteme Gradient Boosting

대용량 분산 처리를 위한 그레디언트 부스팅 오픈 소스 라이브러리

목표: 매우 빠른 속도, 확장성, 이식성

## 배깅

Bagging(Bootstrap aggregating)

## 엑스트라 트리

Extra-Tree

극단적으로 무작위한 트리의 랜덤 포레스트: 편향이 늘어나지만 분산을 낮춘다

## 에이다부스트

AdaBoos(Adaptive Boosting)