**[앙상블(Ensemble)]**

앙상블은 조화 또는 통일을 의미합니다.

어떤 데이터의 값을 예측한다고 할 때, 하나의 모델을 활용합니다.

하지만 여러 개의 모델을 조화롭게 학습시켜 그 모델들의 예측 결과들을 이용한다면 더 정확한 예측 값을 구할 수 있을 것이다.

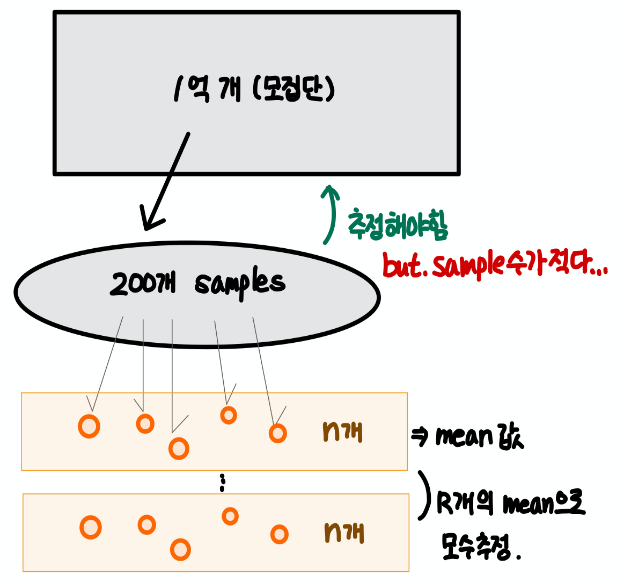
앙상블 학습은 여러 개의 결정 트리(Decision Tree)를 결합하여 하나의 결정 트리보다 더 좋은 성능을 내는 머신러닝 기법이다. 앙상블 학습의 핵심은 여러 개의 약 분류기 (Weak Classifier)를 결합하여 강 분류기(Strong Classifier)를 만드는 것입니다. 그리하여 모델의 정확성이 향상됩니다.

앙상블 학습법에는 세 가지 배깅(Bagging)과 부스팅(Boosting), 스태킹(Stacking)있다.

Ref: [tistory.귀퉁귀퉁 앙상블](https://bkshin.tistory.com/entry/%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-11-%EC%95%99%EC%83%81%EB%B8%94-%ED%95%99%EC%8A%B5-Ensemble-Learning-%EB%B0%B0%EA%B9%85Bagging%EA%B3%BC-%EB%B6%80%EC%8A%A4%ED%8C%85Boosting)

**[부트스트랩(Bootstrap)]**

모수의 분포를 추정하는 방법은 현재 있는 표본에서 추가적으로 표본을 복원 추출하고 각 표본에 대한 통계량을 다시 계산하는 것입니다. 이러한 절차를 부트스트랩이라고 합니다.

부트스트랩은 데이터가 정규분포를 따라야 한다는 가정이 꼭 필요하지 않습니다. 1억 개의 모집단에서 뽑은 200개의 표본이 있다고 합시다. 200개로만 통계량을 구하는 것이 아니라 200개를 기준으로 복원 추출하여 새로운 통계량을 구하는 것입니다.

**<부트스트랩으로 신뢰구간을 구하는 절차> >>>통념적 기준(n,r)**

1. 200개의 표본 중 하나를 뽑아 기록하고 다시 제자리에 둡니다.

2. 이를 n번 반복합니다.

3. n번 재표본추출한 값의 평균을 구합니다.

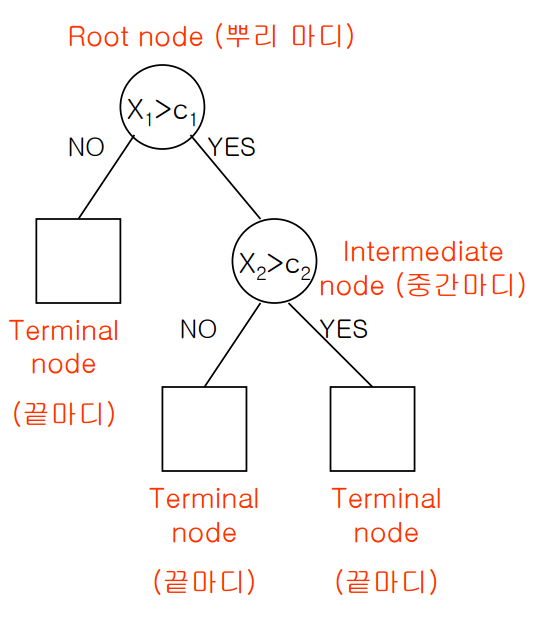
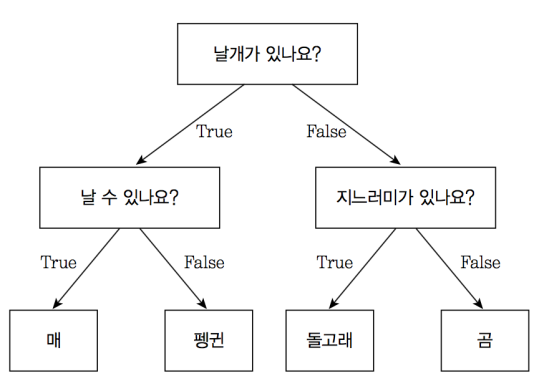
4. 1~3 단계를 R번 반복합니다. (R: 부트스트랩 반복 횟수)

5. 평균에 대한 결과 R개를 사용하여 신뢰구간을 구합니다.

이 방법을 사용하면 표본이 200개밖에 없을지라도 부트스트랩을 통해 200개 보다 더 많은 통계량을 구할 수 있습니다. 따라서 부트스트랩을 활용하면 모수를 더 정확히 추정할 수 있습니다. 특히, R이 클수록 신뢰구간에 대한 추정은 더 정확해진다. 표본의 수가 많을 수록(n이 클수록) 신뢰구간은 작아지고, 신뢰수준(Confidence Level)이 클수록 (ex. 95% -> 99%) 신뢰구간은 커집니다.

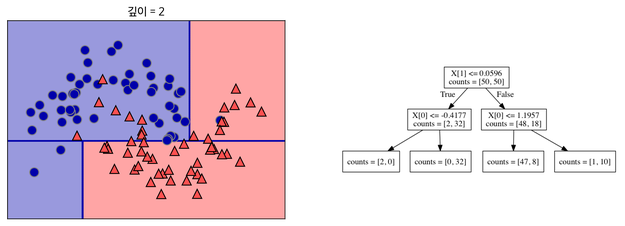
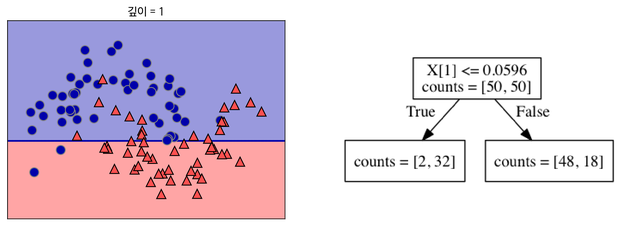
**[결정 트리(Decision Tree)]**

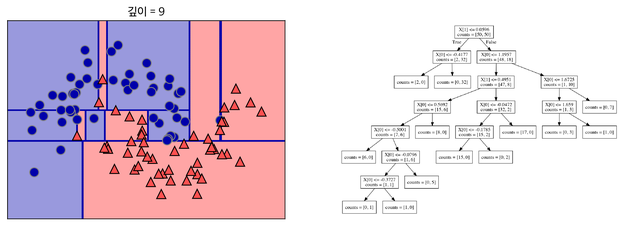
결정 트리(Decision Tree, 의사결정트리, 의사결정나무라고도 함)는 분류(Classification)와 회귀(Regression) 모두 가능한 지도 학습 모델 중 하나입니다. 결정 트리는 스무고개 하듯이 예/아니오 질문을 이어가며 학습합니다.



이렇게 특정 기준(질문)에 따라 데이터를 구분하는 모델을 결정 트리 모델이라고 합니다. 한번의 분기 때마다 변수 영역을 두 개로 구분합니다. 결정 트리에서 질문이나 정답을 담은 네모 상자를 노드(Node)라고 합니다. 맨 처음 분류 기준 (즉, 첫 질문)을 Root Node라고 하고, 맨 마지막 노드를 Terminal Node 혹은 Leaf Node라고 합니다.

**<프로세스>**



 깊이(depth)가 깊어지면 오버피팅 될 확률이 커진다.

**<가지치기(Pruning)>**

오버피팅을 막기 위한 전략으로 가지치기(Pruning)라는 기법이 있습니다. 트리에 가지가 너무 많다면 오버피팅이라 볼 수 있습니다.

**사전 가지치기**: 최대 깊이나 터미널 노드의 최대 개수, 혹은 한 노드가 분할하기 위한 최소 데이터 수를 제한하는 것입니다. min\_sample\_split = 10이면 한 노드에 10개의 데이터가 있다면 그 노드는 더 이상 분기를 하지 않습니다. 또한, max\_depth를 통해서 최대 깊이를 지정해줄 수도 있습니다. max\_depth = 4이면, 깊이가 4보다 크게 가지를 치지 않습니다.

**사후 가지치기**: 결과를 본 후 노드를 지우거나 깊이를 조정하는 것. 가지치기는 사전 가지치기와 사후 가지치기가 있지만 sklearn에서는 사전 가지치기만 지원합니다.

**<엔트로피>**

불순도(Impurity)란 해당 범주 안에 서로 다른 데이터가 얼마나 섞여 있는지를 뜻합니다.

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

엔트로피가 높다는 것은 불순도가 높다는 뜻이고, 엔트로피가 낮다는 것은 불순도가 낮다는 뜻입니다. 엔트로피가 1이면 불순도가 최대입니다. 즉, 한 범주 안에 서로 다른 데이터가 정확히 반반 있다는 뜻입니다. 엔트로피가 0이면 불순도는 최소입니다. 한 범주 안에 하나의 데이터만 있다는 뜻입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

**<정보획득>**

엔트로피가 1인 상태에서 0.7인 상태로 바뀌었다면 정보 획득(information gain)은 0.3입니다. 분기 이전의 엔트로피에서 분기 이후의 엔트로피를 뺀 수치를 정보 획득이라고 합니다.

결정 트리 알고리즘은 정보 획득을 최대화하는 방향으로 학습이 진행됩니다

따라서 첫 분기점을 정보 획득을 최대화하는 기준으로 잡습니다. 이런식으로 max\_depth나 min\_sample\_split으로 설정한 범위까지 분기를 하게 됩니다. 이것이 바로 결정트리의 전체적인 알고리즘입니다.

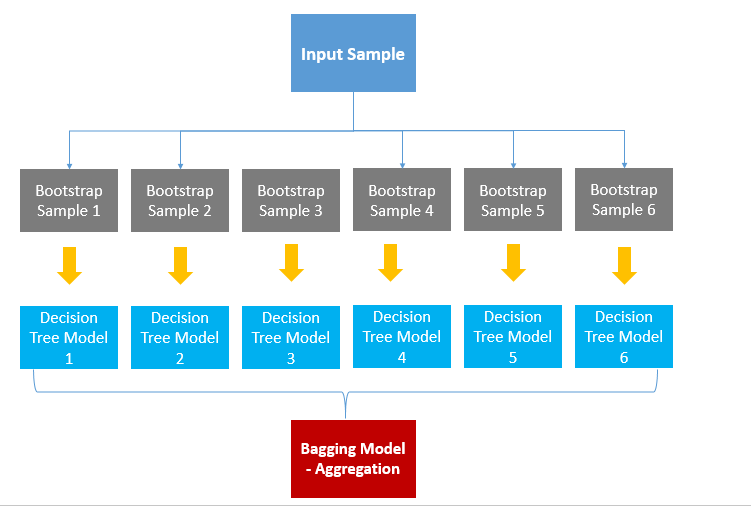
Ref: [tistory 귀퉁귀퉁 결정트리](https://bkshin.tistory.com/entry/%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-4-%EA%B2%B0%EC%A0%95-%ED%8A%B8%EB%A6%ACDecision-Tree?category=1057680)

**[배깅 앙상블(Bagging Ensemble)]**

Bagging은 Bootstrap Aggregation의 약자입니다. 배깅은 샘플을 여러 번 뽑아(Bootstrap) 각 모델을 학습시켜 결과물을 집계(Aggregration)하는 방법입니다.

우선, 데이터로부터 부트스트랩을 합니다. (복원 랜덤 샘플링) 부트스트랩한 데이터로 모델을 학습시킵니다. 그리고 학습된 모델의 결과를 집계하여 최종 결과 값을 구합니다.

Categorical Data는 투표 방식(Votinig)으로 결과를 집계하며, Continuous Data는 평균으로 집계합니다.



Categorical Data일 때, 투표 방식으로 한다는 것은 전체 모델에서 예측한 값 중 가장 많은 값을 최종 예측값으로 선정한다는 것입니다. 6개의 결정 트리 모델이 있다고 합시다. 4개는 A로 예측했고, 2개는 B로 예측했다면 투표에 의해 4개의 모델이 선택한 A를 최종 결과로 예측한다는 것입니다.

Hard Voting : 개수로 Class 구별

Soft Voting : 확률로 Class 구별

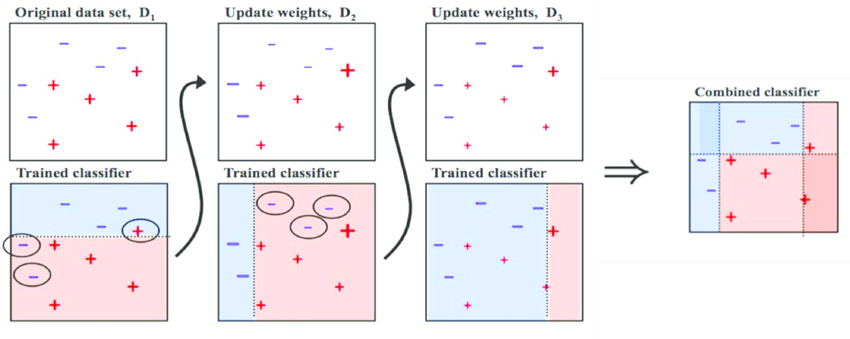
Weighted Voting: 모델의 Score에 비례하여 결과값에 가중치 부여(hard나 soft에 옵션으로 적용)

평균으로 집계한다는 것은 말 그대로 각각의 결정 트리 모델이 예측한 값에 평균을 취해 최종 Bagging Model의 예측값을 결정한다는 것입니다. 배깅 기법을 활용한 모델이 바로 랜덤 포레스트입니다.

Ref: [Voting](https://devkor.tistory.com/entry/Soft-Voting-%EA%B3%BC-Hard-Voting)

**[부스팅 앙상블(Boosting Ensemble)]**

배깅은 여러 개의 독립적인 결정 트리가 각각 값을 예측한 뒤, 그 결과 값을 집계해 최종 결과 값을 예측하는 방식입니다. 하지만 부스팅은 처음 모델이 예측을 하면 그 예측 결과에 따라 데이터에 가중치가 부여되고, 부여된 가중치가 다음 모델에 영향을 줍니다. 잘못 분류된 데이터에 집중하여 새로운 분류 규칙을 만드는 단계를 반복합니다.



**<배깅 vs 부스팅>**

배깅은 병렬로 학습하는 반면, 부스팅은 순차적으로 학습합니다. 한번 학습이 끝난 후 결과에 따라 가중치를 부여합니다. 그렇게 부여된 가중치가 다음 모델의 결과 예측에 영향을 줍니다.

오답에 대해서는 높은 가중치를 부여하고, 정답에 대해서는 낮은 가중치를 부여합니다. 따라서 오답을 정답으로 맞히기 위해 오답에 더 집중할 수 있게 되는 것입니다.

부스팅은 배깅에 비해 error가 적습니다. 즉, 성능이 좋습니다. 하지만 속도가 느리고 오버 피팅이 될 가능성이 있습니다. 그렇다면 실제 사용할 때는 배깅과 부스팅 중 어떤 것을 선택해야 할까요? 상황에 따라 다르다고 할 수 있습니다. 개별 결정 트리의 낮은 성능이 문제라면 부스팅이 적합하고, 오버 피팅이 문제라면 배깅이 적합합니다.

Ref: [tistory.귀퉁귀퉁 앙상블](https://bkshin.tistory.com/entry/%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-11-%EC%95%99%EC%83%81%EB%B8%94-%ED%95%99%EC%8A%B5-Ensemble-Learning-%EB%B0%B0%EA%B9%85Bagging%EA%B3%BC-%EB%B6%80%EC%8A%A4%ED%8C%85Boosting)

**[스태킹 앙상블(Stacking Ensemble)]**

개별 모델이 예측한 데이터를 다시 training set으로 사용해서 학습하는 것이다.

스태킹 앙상블을 할 때는 개별 모델들(복수), 최종 모델(하나)가 필요하다.

>원본 데이터의 train, test가 존재

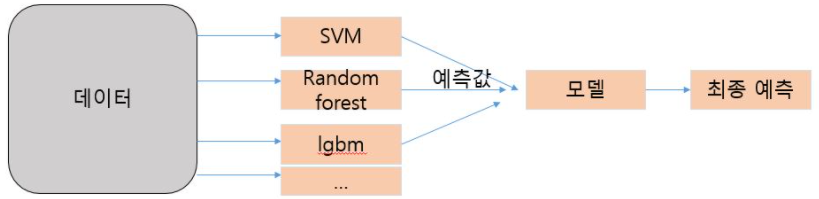
>원본 training data를 3개의 머신러닝 모델이 학습

>각 모델마다 X\_test를 넣어서 예측 후 predict를 뽑아냄(3개의 predict된 값)

>3개의 predict를 다시 학습 데이터로 사용

>최종 model을 하나 선정해 학습

>최종 평가



Ex) 개별 모델: SVM, Random forest, logistic regression

최종 모델: LightGBM(3개 predict, y\_test로 학습)

이러한 기본적인 스태킹 앙상블은 과적합(overfitting)의 문제점이 있어 사용하지 않는다. 보통 CV기반(KFold 등)의 스태킹 앙상블을 사용한다.

**<Cross Validation(Kfold)기반 stacking ensemble>**

CV기반 스태킹은 각 모델들이 교차 검증으로 최종 모델을 위한 학습용 데이터를 생성한다.

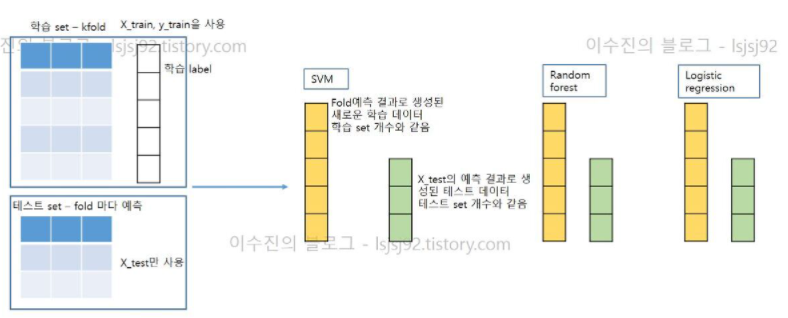
또한, 예측을 위한 테이스용 데이터도 생성하여 이를 기반으로 최종 모델이 학습을 진행한다.

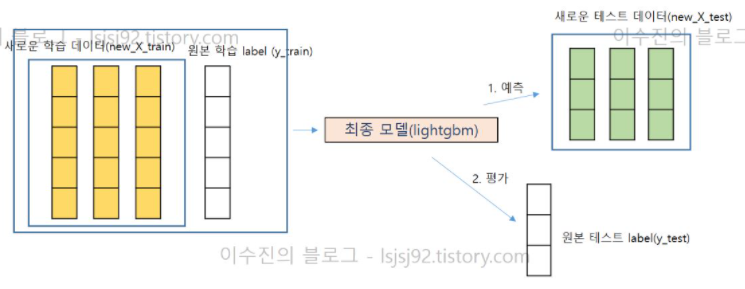
1. 데이터를 Fold로 나눔
2. 각 모델 별로 Fold로 나누어진 데이터를 기반으로 훈련을 진행(X\_train, y\_tain)사용

* 각 Fold마다 뽑아진 훈련 데이터로 모델을 훈련하고 검증 데이터를 활용해 예측 후 값을 저장
* 또 각 Fold마다 나온 model을 기반으로 원본 X\_test 데이터를 훈련하여 저장

(추후 Average)

1. 2까지 진행해서 나온 각 모델별 예측 데이터(2-1)를 모두 stacking하여 최종 모델의 훈련 데이터로 사용(2-1, y\_train)
2. 2-2에서 나온 데이터로 예측을 수행하여 pred 값을 뽑아냄
3. 4에서 나온 pred와 y\_test 값을 비교해서 최종 모델 평가





텍스트이(가) 표시된 사진

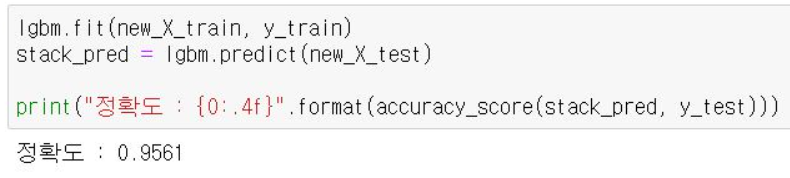
자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

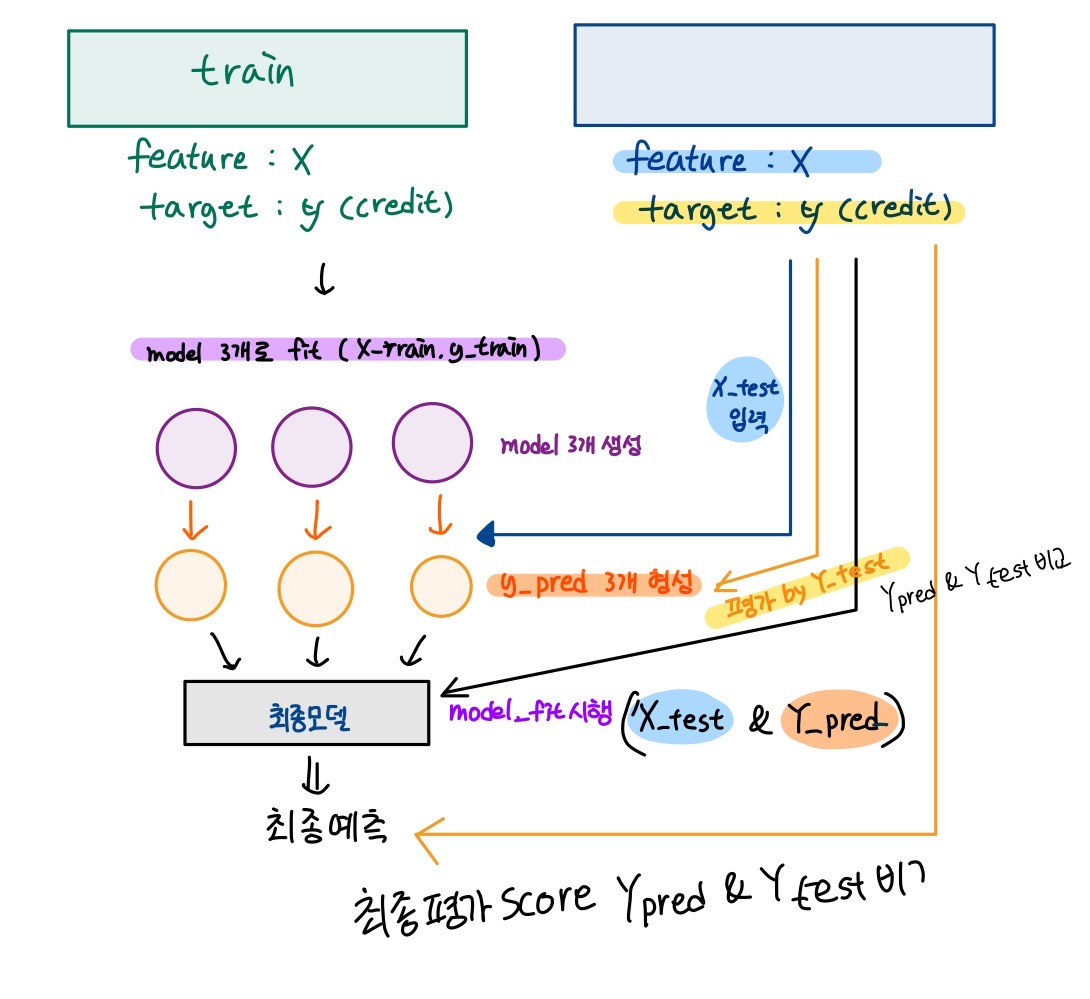
텍스트이(가) 표시된 사진

자동 생성된 설명



텍스트이(가) 표시된 사진

자동 생성된 설명



Ref: [stacking ensemble](https://lsjsj92.tistory.com/558)

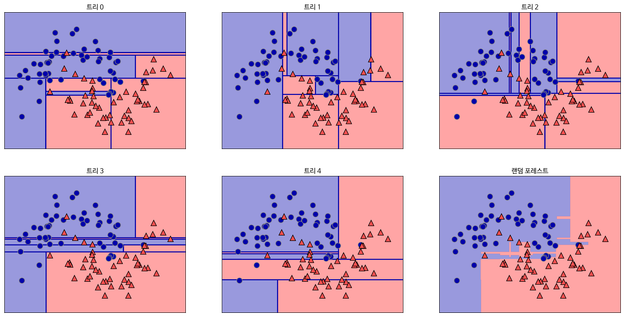
[cv\_stacking ensemble](https://lsjsj92.tistory.com/559?category=853217)

[cv\_stacking\_설명좋음](https://techblog-history-younghunjo1.tistory.com/103)

**[랜덤 포레스트(Random Forest)]**

랜덤 포레스트의 포레스트는 숲(Forest), 결정 트리는 나무(Tree)입니다. 나무가 모여 숲을 이룹니다. 즉, 결정 트리(Decision Tree)가 모여 랜덤 포레스트(Random Forest)를 구성합니다. 결정 트리 하나만으로도 머신러닝을 할 수 있습니다. 하지만 결정 트리의 단점은 훈련 데이터에 오버피팅이 되는 경향이 있다는 것입니다. 여러 개의 결정 트리를 통해 랜덤 포레스트를 만들면 오버피팅 되는 단점을 해결할 수 있습니다.

예를 들어 Feature가 30개라고 합시다. 30개의 Feature를 기반으로 하나의 결정 트리를 만든다면 트리의 가지가 많아질 것이고, 이는 오버피팅의 결과를 야기할 것입니다. 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 있습니다. 결정 트리 하나마다 예측 값을 내놓겠죠. 여러 결정 트리들이 내린 예측 값들 중 가장 많이 나온 값을 최종 예측값으로 정합니다. 다수결의 원칙에 따르는 것입니다. 이렇게 의견을 통합하거나 여러 가지 결과를 합치는 방식을 앙상블(Ensemble)이라고 합니다. 즉, 하나의 거대한 (깊이가 깊은) 결정 트리를 만드는 것이 아니라 여러 개의 작은 결정 트리를 만드는 것입니다. 여러 개의 작은 결정 트리가 예측한 값들 중 가장 많은 값(분류일 경우) 혹은 평균값(회귀일 경우)을 최종 예측 값으로 정하는 것입니다.



**n\_estimators: 랜덤 포레스트 안의 결정 트리 갯수**

n\_estimators는 클수록 더 깔끔한 Decision Boundary가 나온다.

**max\_features: 무작위로 선택할 Feature의 개수**

max\_features 값이 작으면 랜덤 포레스트의 트리들이 서로 매우 달라집니다. 오버피팅이 줄어드는 효과가 있겠죠. max\_features는 일반적으로 Defalut 값을 씁니다.

**[Gradient Boosting]**

**[XGBoost]**

**[LightGBM]**

**[CatBoost]**