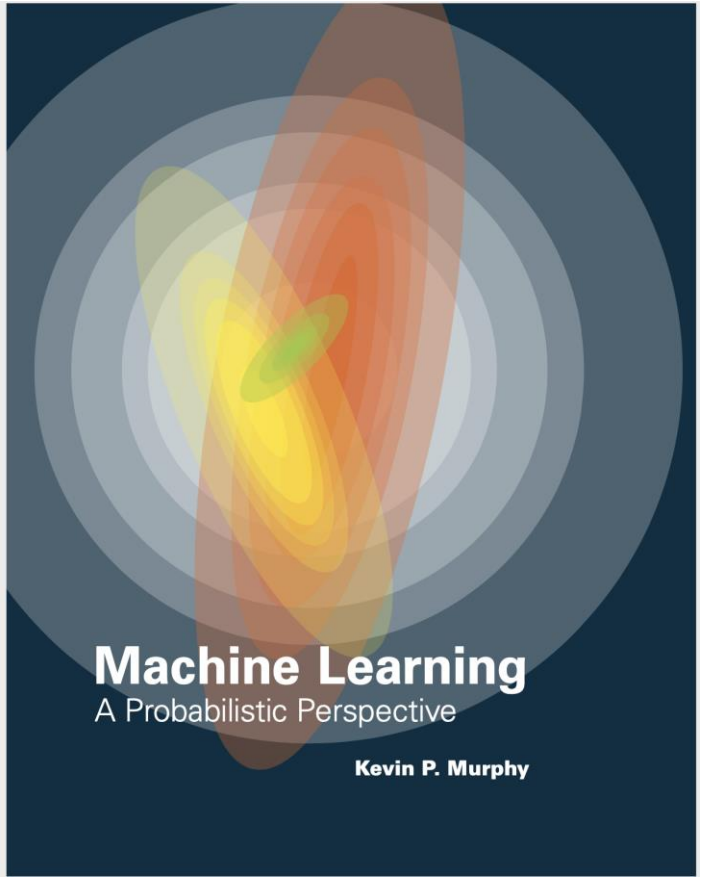
머신러닝 강의자료

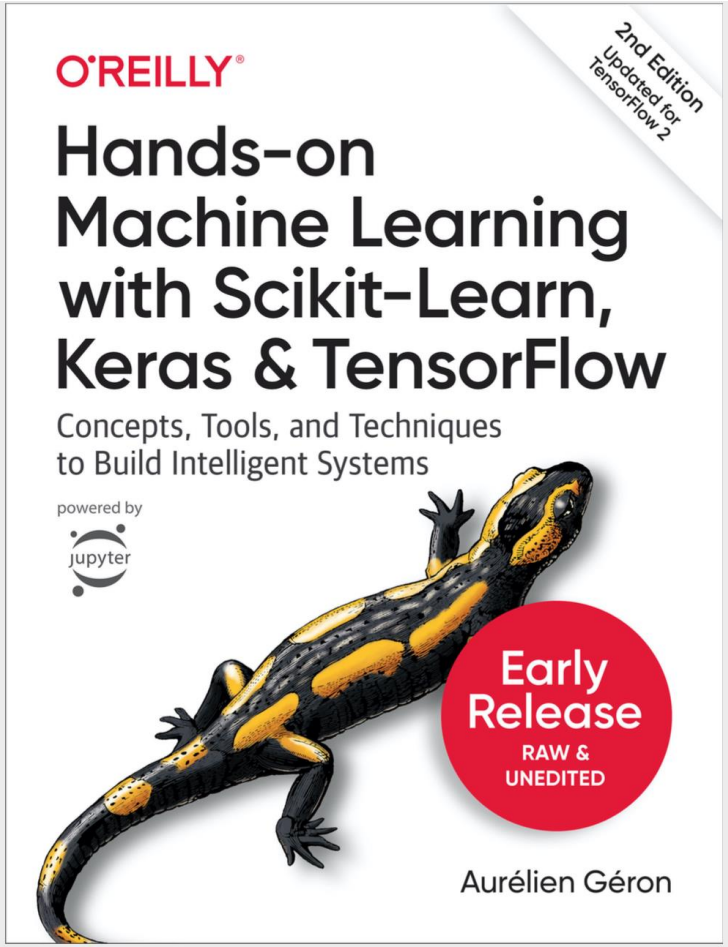
7강 – 앙상블 학습과 랜덤 포레스트

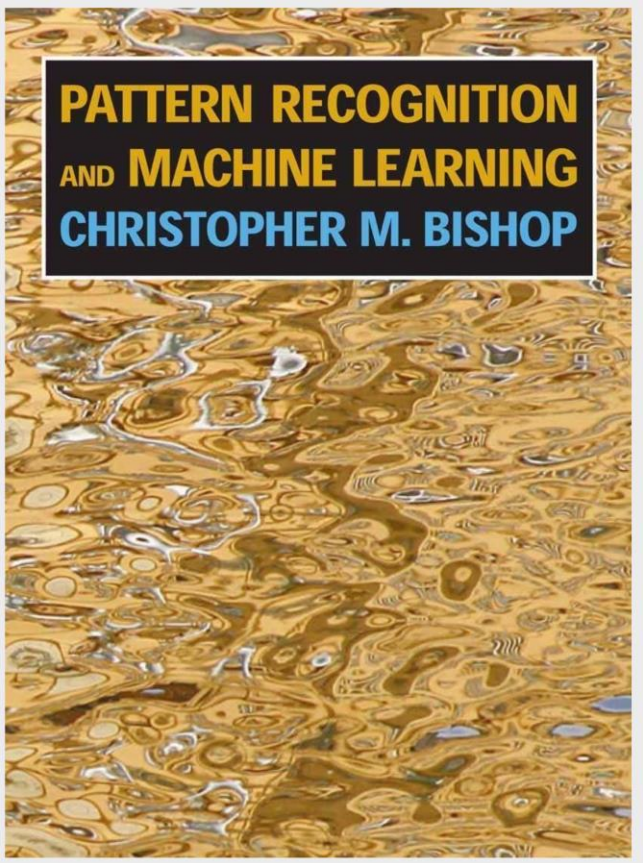
닥터윌컨설팅 딥러닝 R&D 책임연구원
고려대학교 인공지능대학원 박사과정

류회성(Hoe Sung Ryu)

들어가기 앞서

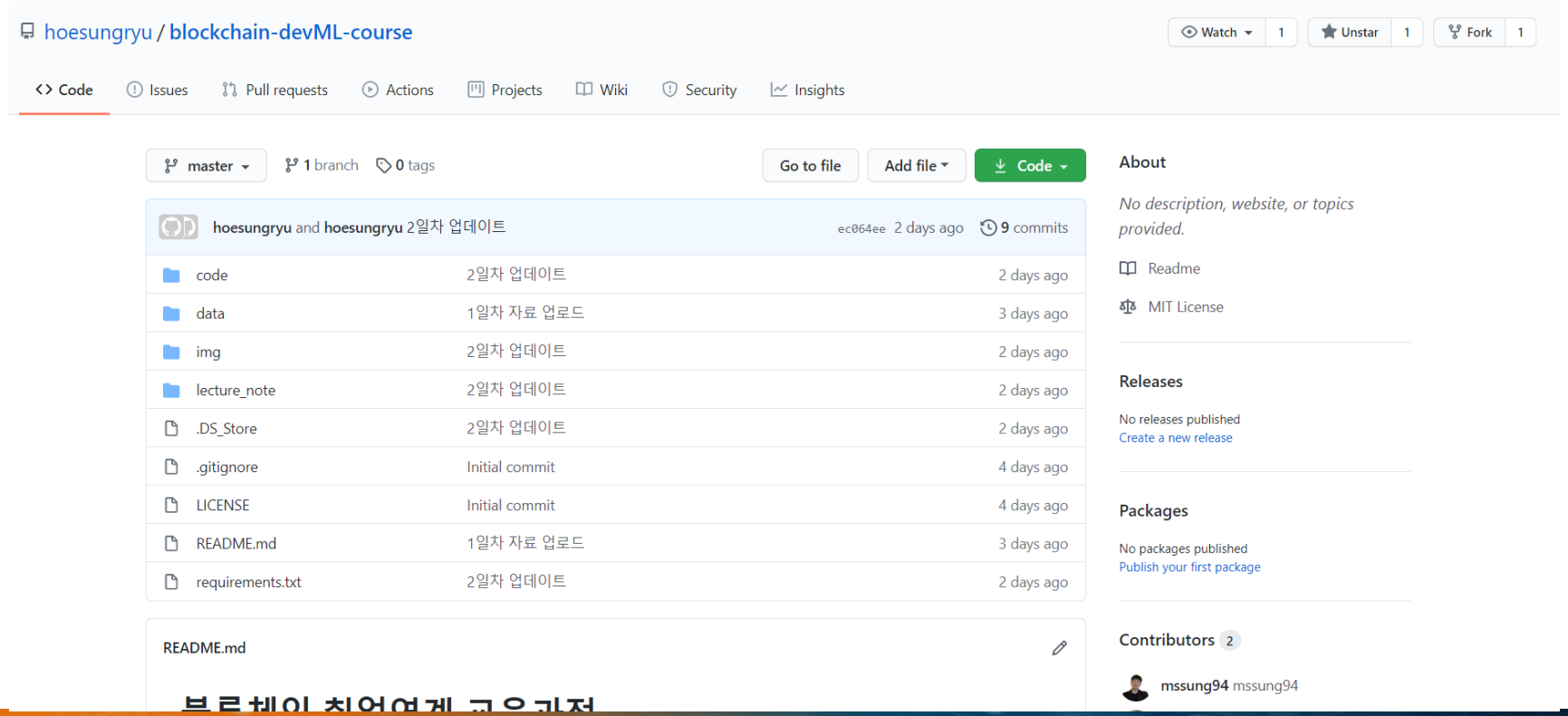






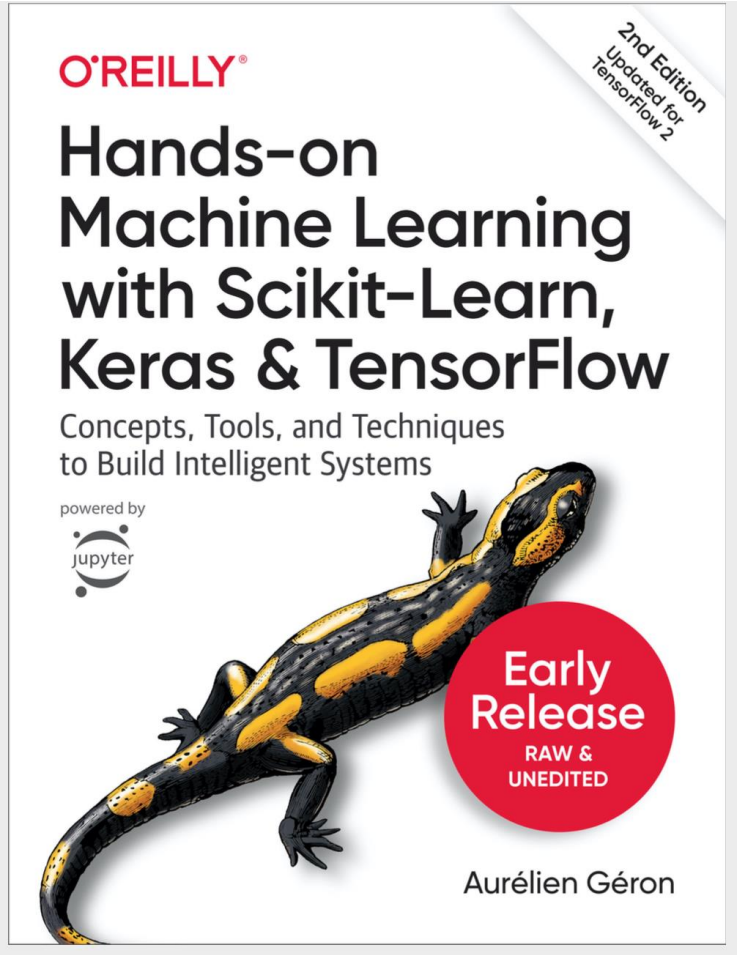
들어가기 앞서

- GitHub
  - https://github.com/hoesungryu/blockchain-devML-course



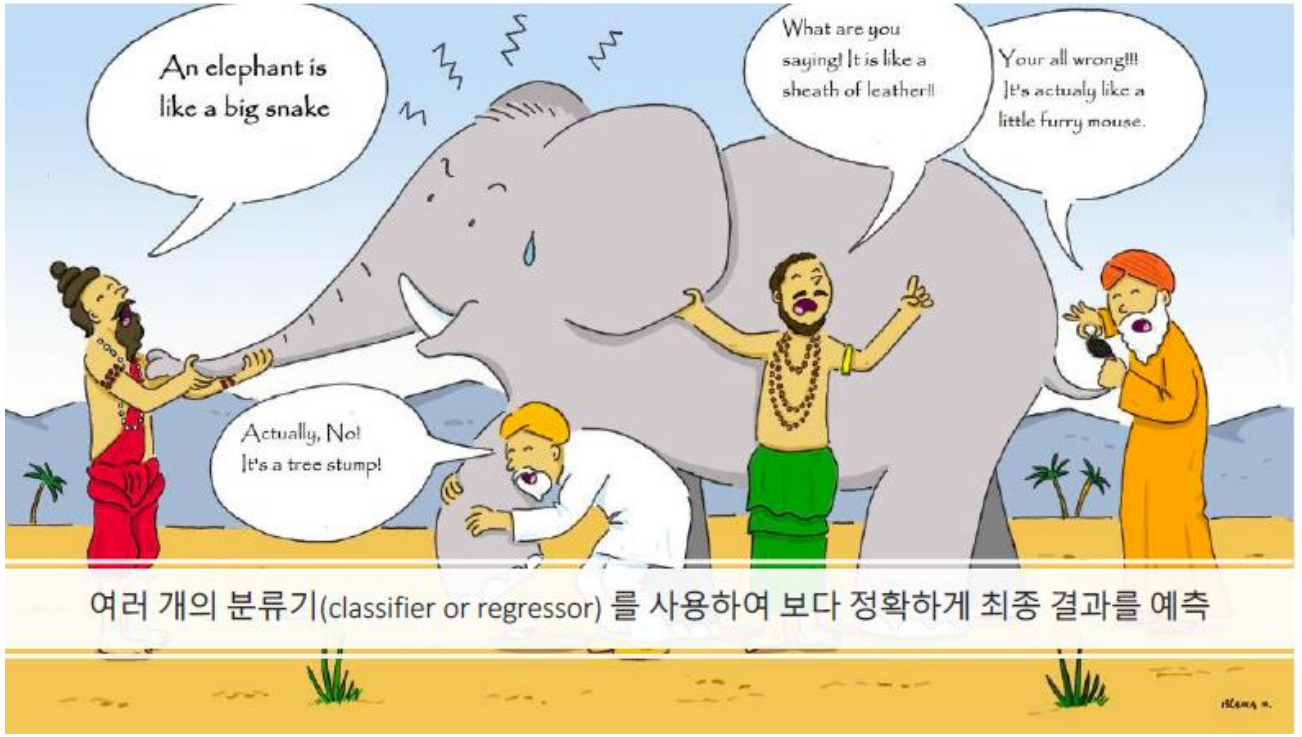
강의내용

- 앙상블 학습과 랜덤 포레스트
  - 보팅
  - 배깅과 페이스팅
  - 랜덤 패치와 랜덤 서브스페이스
  - 랜덤 포레스트
  - 부스팅
  - 스태킹

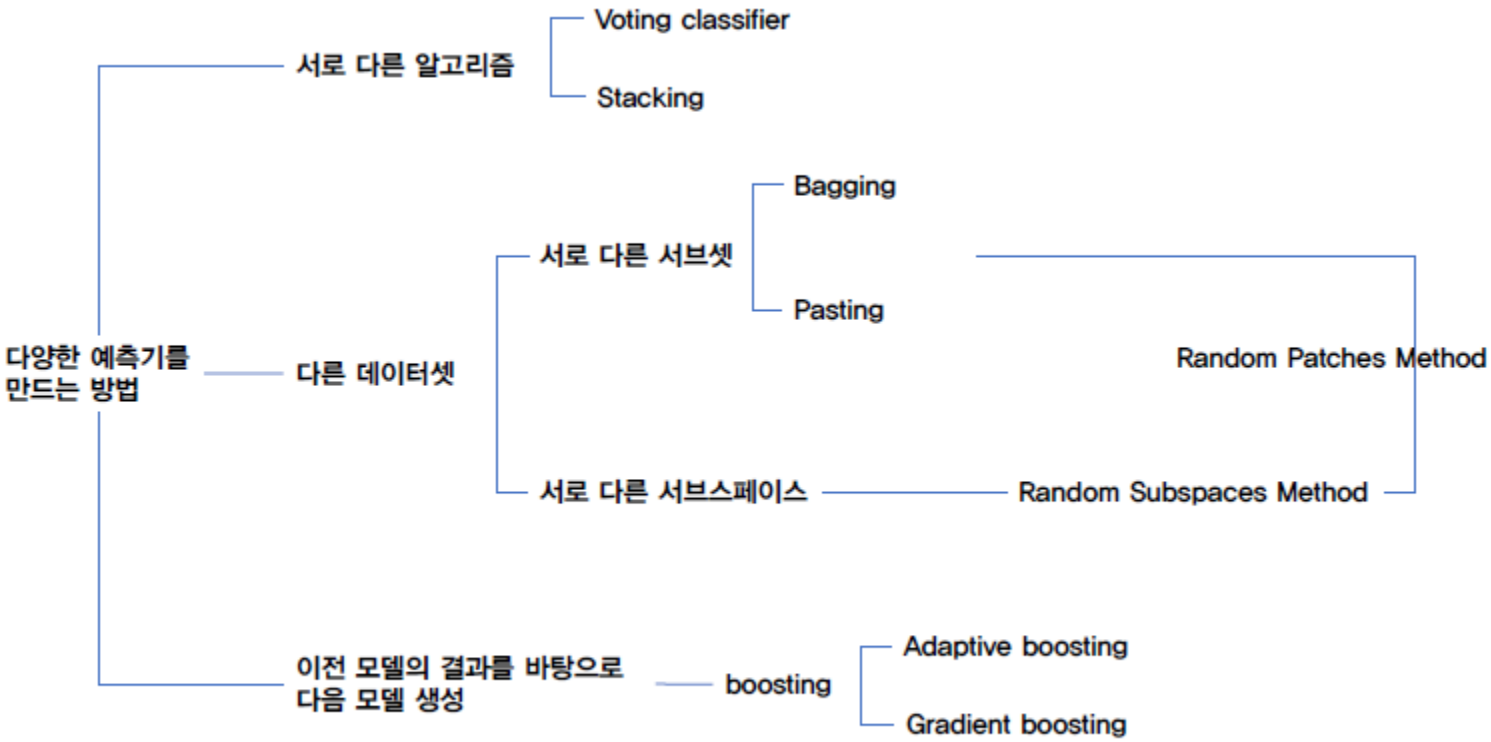


앙상블 학습

- 앙상블 학습이란?
  - 다양한 모델의 예측을 수집하여 가장 좋은 예측을 얻는 것
  - 다양한 예측기를 만드는 방법
    - 보팅(Voting)
    - 스태킹(Stacking)
    - 배깅(Bagging)
    - 부스팅(Boosting)
    - 페이스팅(Pasting)



앙상블 학습



[보팅]

보팅

- **기본 아이디어**
  - 서로 다른 훈련 알고리즘으로 하나의 데이터셋을 예측한 후 각 알고리즘의 다수결 투표(Voting)로 예측
- **약한 학습기의 앙상블은 강한 학습기를 만든다**
  - 큰 수의 법칙

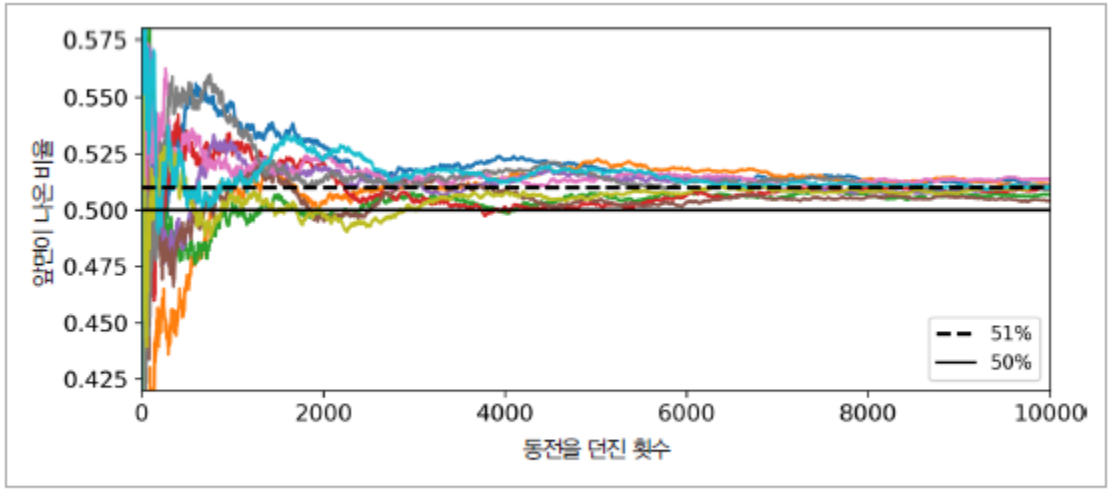


그림 7-3 큰 수의 법칙

보팅

- 보팅의 종류
  - 간접 투표(Soft voting): 예측의 평균, 분류기들의 분류 확률의 평균
  - 직접 투표(Hard voting): 예측의 최빈값, 분류기 간 다수결

보팅

- 사이킷런에서의 보팅

sklearn.ensemble.VotingClassifier¶

class sklearn.ensemble.**VotingClassifier**(*estimators, voting='hard', weights=None, n_jobs=None, flatten_transform=True*) [source]

estimators : *list of (str, estimator) tuples*

Invoking the `fit` method on the `VotingClassifier` will fit clones of those original estimators that will be stored in the class attribute `self.estimators_`. An estimator can be set to `'drop'` using `set_params`.

Deprecated since version 0.22: Using `None` to drop an estimator is deprecated in 0.22 and support will be dropped in 0.24. Use the string `'drop'` instead.

voting : *str, {'hard', 'soft'} (default='hard')*

If 'hard', uses predicted class labels for majority rule voting. Else if 'soft', predicts the class label based on the argmax of the sums of the predicted probabilities, which is recommended for an ensemble of well-calibrated classifiers.

보팅

- 사이킷런에서의 보팅

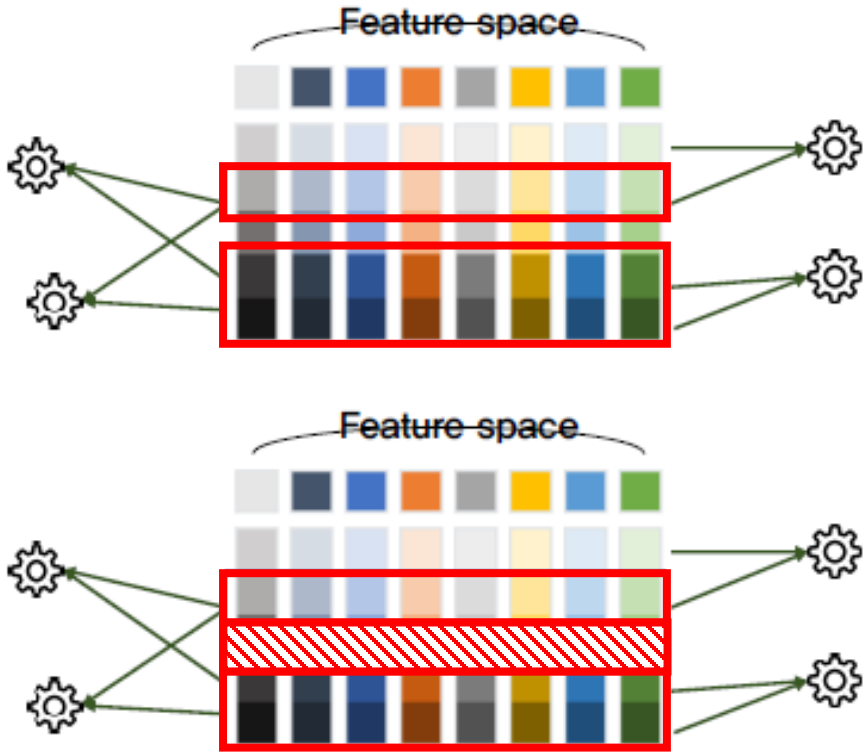
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
voting_clf.fit(X_train, y_train)
```

```
>>> from sklearn.metrics import accuracy_score
>>> for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
...     clf.fit(X_train, y_train)
...     y_pred = clf.predict(X_test)
...     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
...
LogisticRegression 0.864
RandomForestClassifier 0.894
SVC 0.888
VotingClassifier 0.904
```

[배깅과 페이스팅]

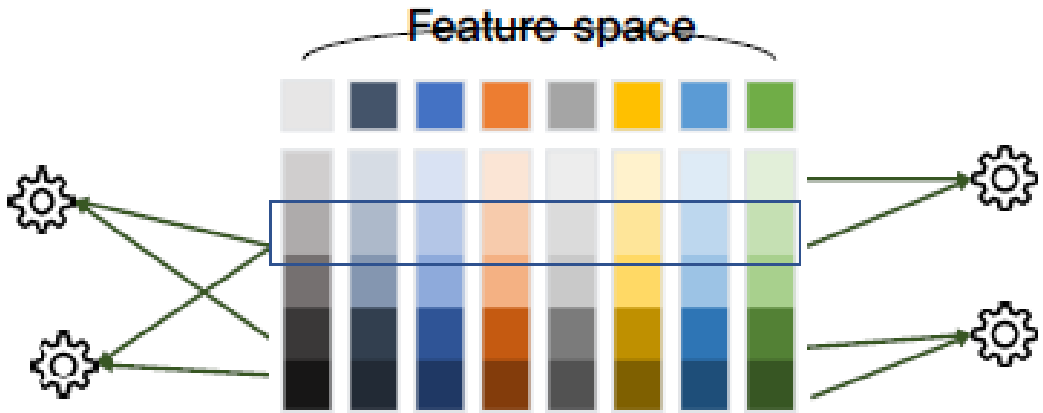
배깅과 페이스팅

- 기본 아이디어
  - 알고리즘이 같아도 **훈련 세트가 다르다면** 서로 다른 예측기를 만들 수 있다
- 배깅(Bagging) vs 페이스팅(Pasting)
  - 모두 훈련 세트의 **샘플을 샘플링**하는 방법
  - 배깅(Bagging): **중복을 허용하여** 샘플링
  - 페이스팅(Pasting): **중복을 허용하지 않고** 샘플링



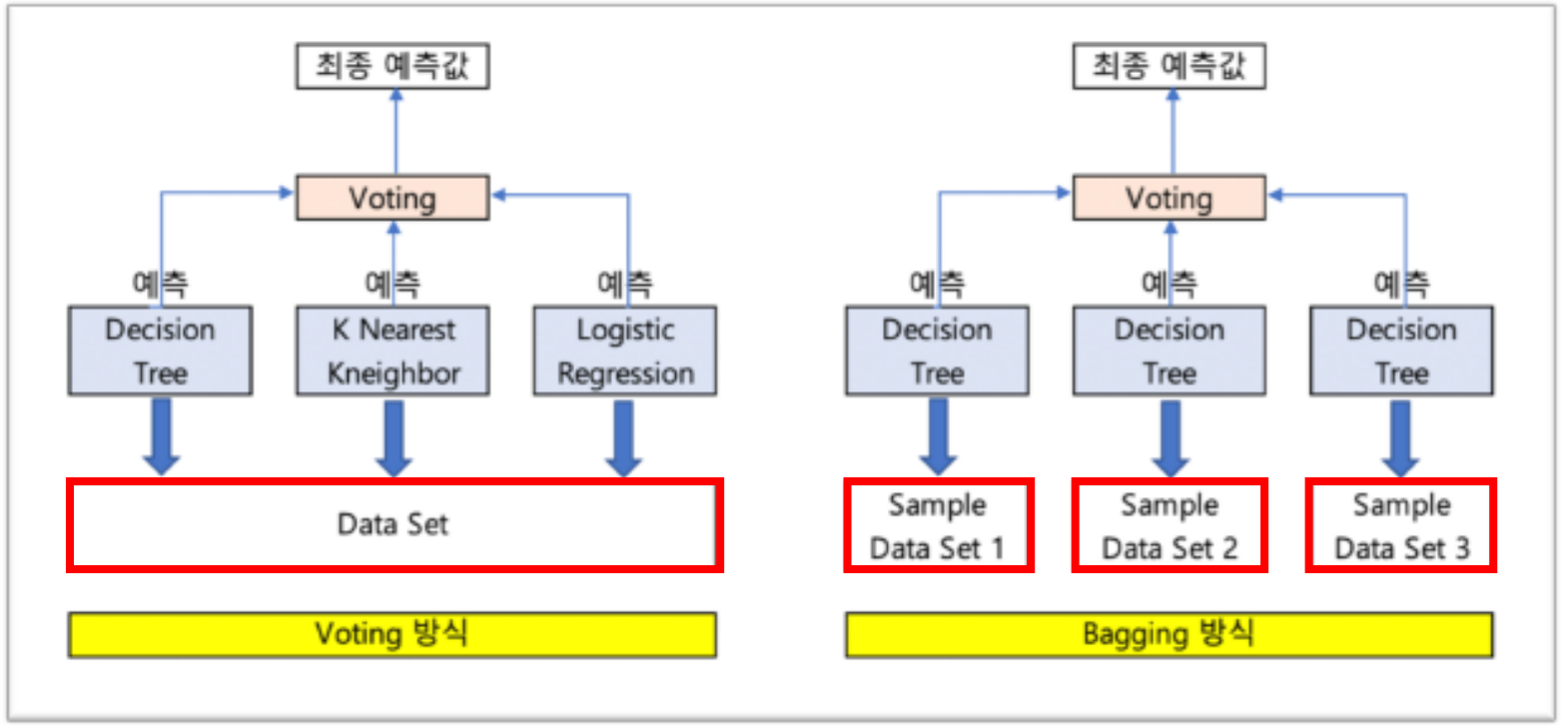
배깅과 페이스팅

- 수집함수
  - 예측기의 예측을 취합하는 함수
    - 분류: 통계적 최빈값
    - 회귀: 예측의 평균
- 편향은 비슷하지만 분산은 줄어듬
  - 오버피팅 방지할 수 있음
- OOB(Out of bag) 평가
  - 배깅의 경우 선택되지 않은 훈련세트로 모델을 평가가능



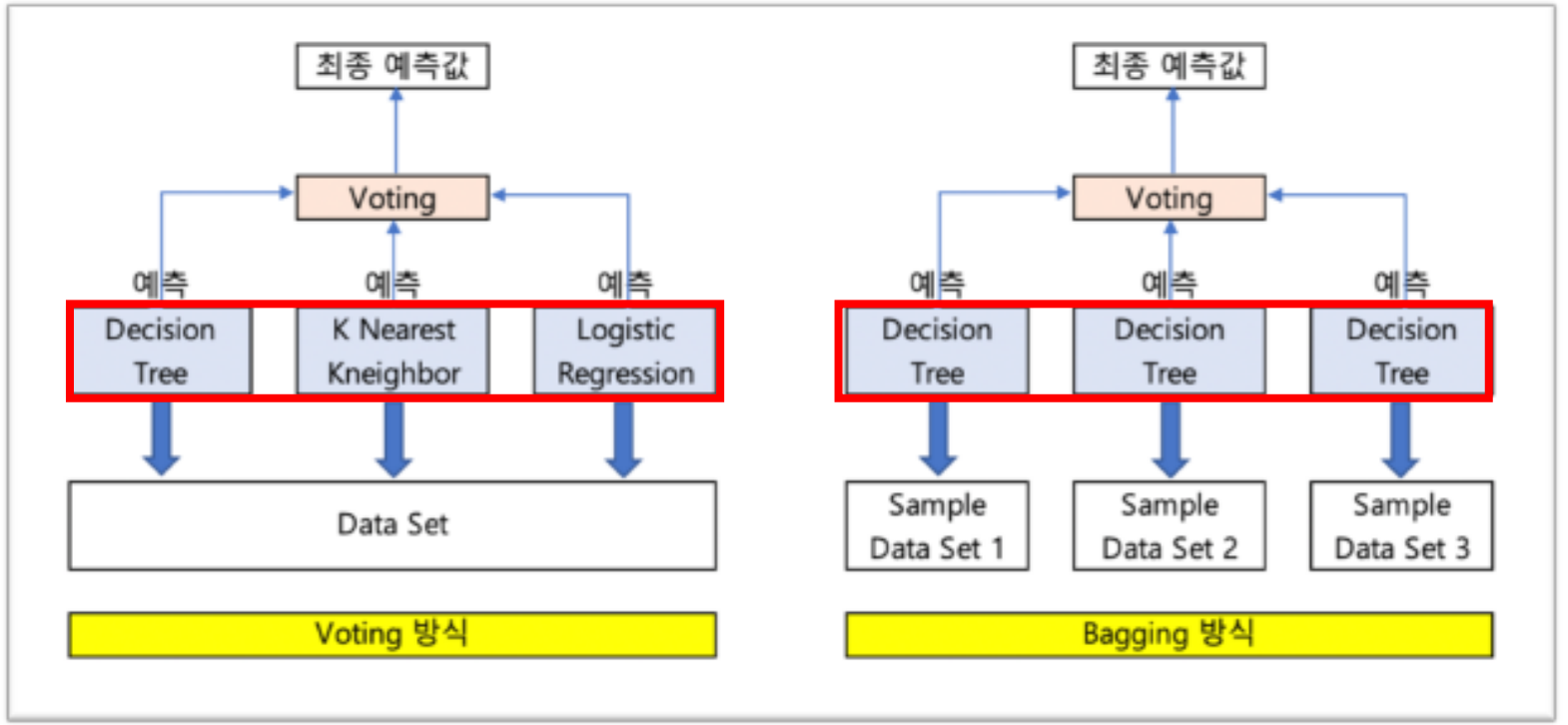
배깅과 페이스팅

- 배깅과 보팅의 차이점
  - 훈련 세트를 어떻게 선택하느냐



배깅과 페이스팅

- 배깅과 보팅의 차이점
  - 동일한 분류기를 이용하는지 동일하지 않은 분류기를 이용하는지



배깅과 페이스팅

- 사이킷런에서의 배깅

sklearn.ensemble.BaggingClassifier

class sklearn.ensemble.**BaggingClassifier**(*base_estimator=None, n_estimators=10, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0*)

[source]

Parameters:

base_estimator : *object or None, optional (default=None)* 알고리즘
The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a decision tree.

n_estimators : *int, optional (default=10)* 생성할 예측기 갯수
The number of base estimators in the ensemble.

max_samples : *int or float, optional (default=1.0)* 샘플 개수 or 비율 → bagging, pasting
The number of samples to draw from X to train each base estimator.

- If int, then draw `max_samples` samples.
- If float, then draw `max_samples * X.shape[0]` samples.

max_features : *int or float, optional (default=1.0)* 특성 개수 or 비율 → random subspace
The number of features to draw from X to train each base estimator.

- If int, then draw `max_features` features.
- If float, then draw `max_features * X.shape[1]` features.

bootstrap : *boolean, optional (default=True)* 샘플을 고를 때 중복허용 여부 → True : bagging, False : pasting
Whether samples are drawn with replacement. If False, sampling without replacement is performed.

bootstrap_features : *boolean, optional (default=False)* 특성을 고를 때 중복 허용 여부
Whether features are drawn with replacement.

oob_score : *bool, optional (default=False)* Oob score 사용 여부
Whether to use out-of-bag samples to estimate the generalization error.

Quiz1

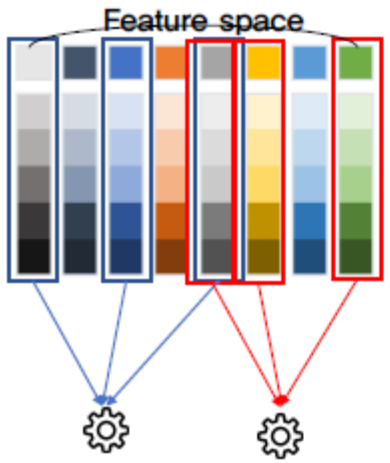
- Q1. 머신러닝에서 보팅 이란?
  - A1.
- Q2. 보팅과 배깅 차이점
  - A2.
- Q3. 배깅과 패스팅 차이점
  - A3.

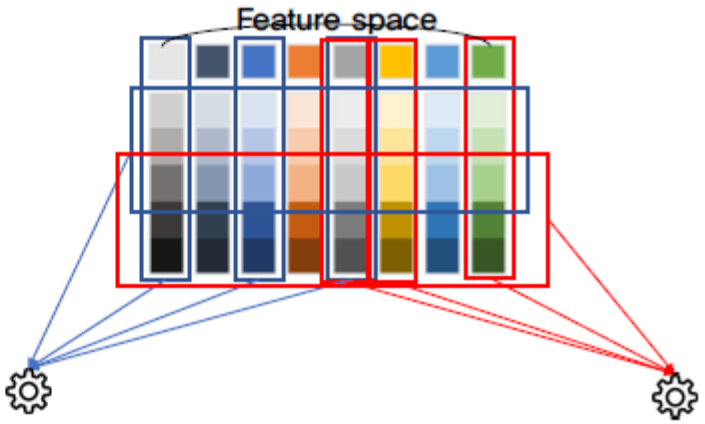
(1/3)

[랜덤 패치와 랜덤 서브스페이스]

랜덤 패치와 랜덤 서브스페이스

- **기본 아이디어**
  - 알고리즘이 같아도 훈련 세트가 다르다면 서로 다른 예측기를 만들 수 있음
  - 특성을 샘플링하는 방법으로도 다양한 훈련 세트를 만들 수 있음
- **랜덤 패치와 랜덤 서브 스페이스**
  - 랜덤 패치(Random Patches): 샘플과 특성을 모두 샘플링
  - 랜덤 서브스페이스(Random Subspace): 샘플은 모두 사용, 특성만 샘플링
- **편향은 늘리는 대신 분산은 낮춤**
  - 오버피팅 방지할 수 있음

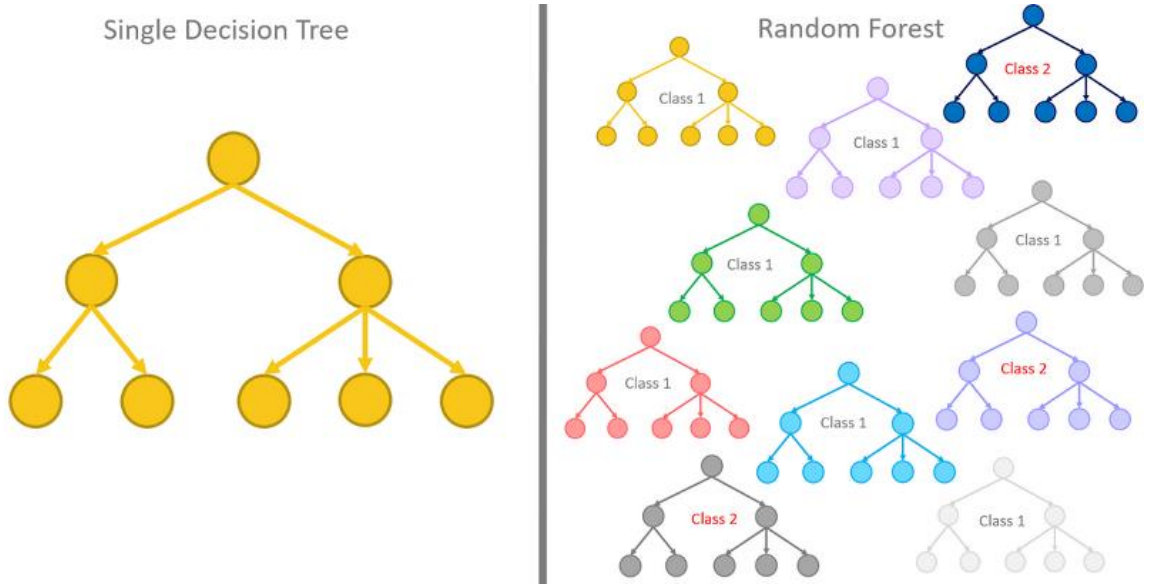




[랜덤 포레스트]

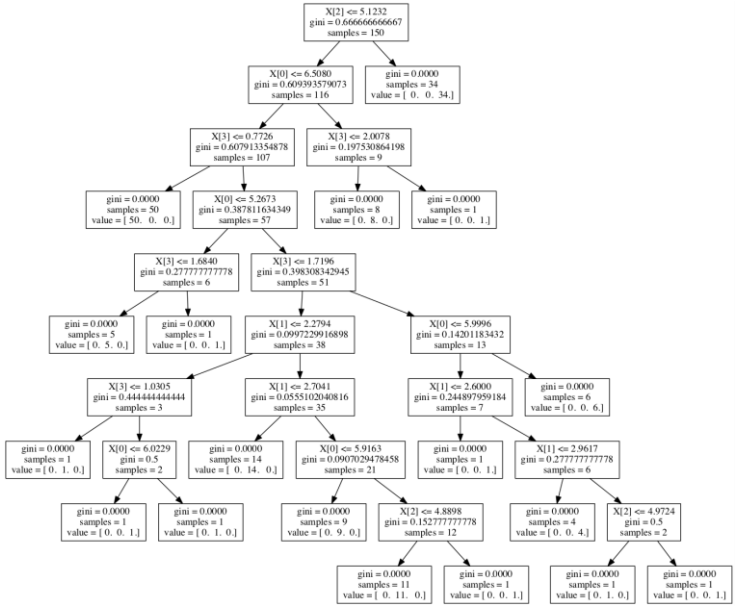
랜덤 포레스트

- 기본 아이디어
  - 배깅과 페이스팅을 적용한 결정트리의 앙상블
  - 노드를 분할할 때 최적의 특성을 찾는 대신 무작위로 선택한 특성 후보 중에서 최적의 특성을 찾음
  - 편향을 손해보는 대신 분산을 낮춤



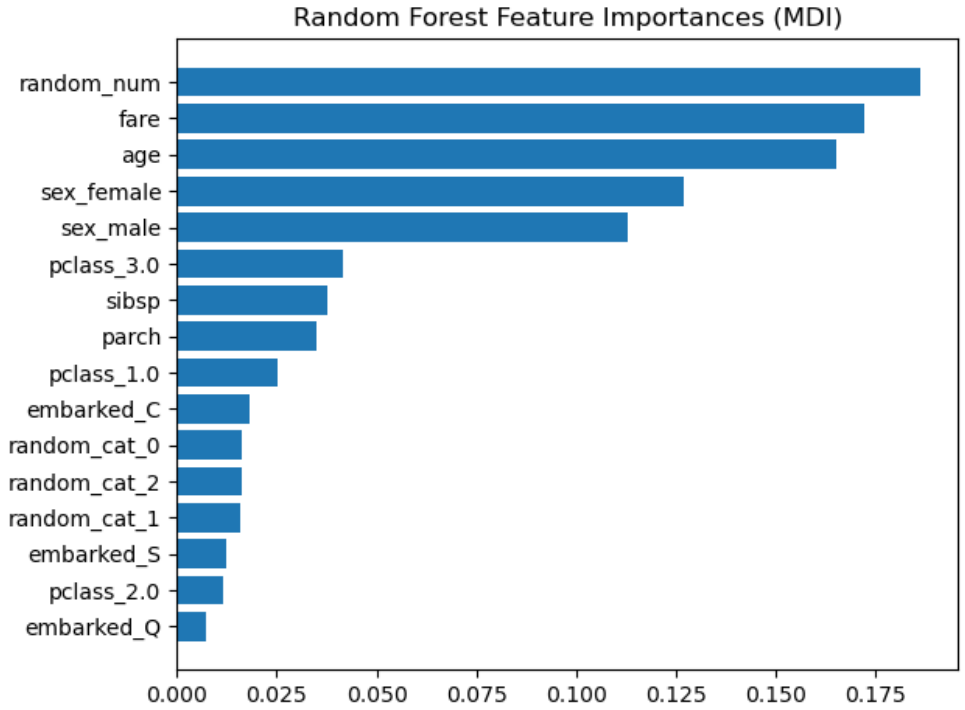
랜덤 포레스트

- 엑스트라 트리
  - 극단적으로 무작위한 트리의 앙상블
  - 노드를 분할하는 기준 특성을 찾는 대신 무작위로 분할한 후, 최상의 분할을 선택



랜덤 포레스트

- 특성 중요도
  - 불순도를 많이 감소시키는 특성일수록 중요도가 큼
    - 클래스를 잘 나누었다는 걸 뜻하므로
  - 어떤 특성을 사용한 노드가 불순도를 감소시킨 정도의 평균
    - 가중치 평균이며 노드의 가중치는 훈련 샘플 수와 같음
    - 즉, 훈련 샘플 수가 많은 노드를 잘 분류하는 특성일수록 특성 중요도가 높음



랜덤 포레스트

- **특성 중요도**
  - MNIST 데이터셋에 랜덤 포레스트 분류기를 훈련시키고 각 픽셀의 중요도를 그래프로 나타내보자
  - 분류기가 어디를 중요하게 선택하는지 확인할 수 있음

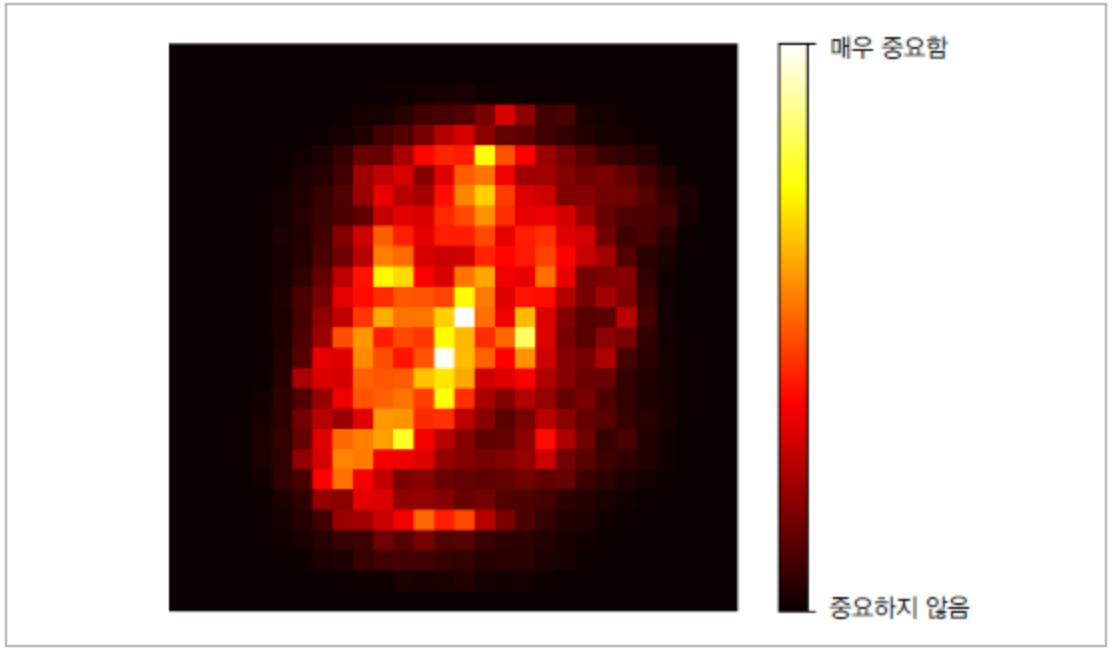


그림 7-6 (랜덤 포레스트 분류기에서 얻은) MNIST 픽셀 중요도

랜덤 포레스트

- 노드 분할 방식 비교
  - 노드를 분할할 기준 특성을 찾음
    - 최적의 특성을 찾음 → 결정 트리
    - 임의로 선택한 특성 중 최적의 특성을 찾음 → 랜덤 포레스트
  - 노드를 분할할 기준 특성을 찾지 않음
    - 임의로 여러 번 분할한 다음 그 중 최적의 분할을 찾음 → 엑스트라 트리

	Decision Tree	Random Forest	Extra Trees
Number of trees	1	Many	Many
No of features considered for split at each decision node	All Features	Random subset of Features	Random subset of Features
Boostrapping(Drawing Sampling without replacement)	Not applied	Yes	No
How split is made	Best Split	Best Split	Random Split

랜덤 포레스트

- 사이킷런에서의 랜덤 포레스트와 엑스트라 트리

3.2.4.3.1. sklearn.ensemble.RandomForestClassifier

class sklearn.ensemble.**RandomForestClassifier**(*n_estimators=100, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

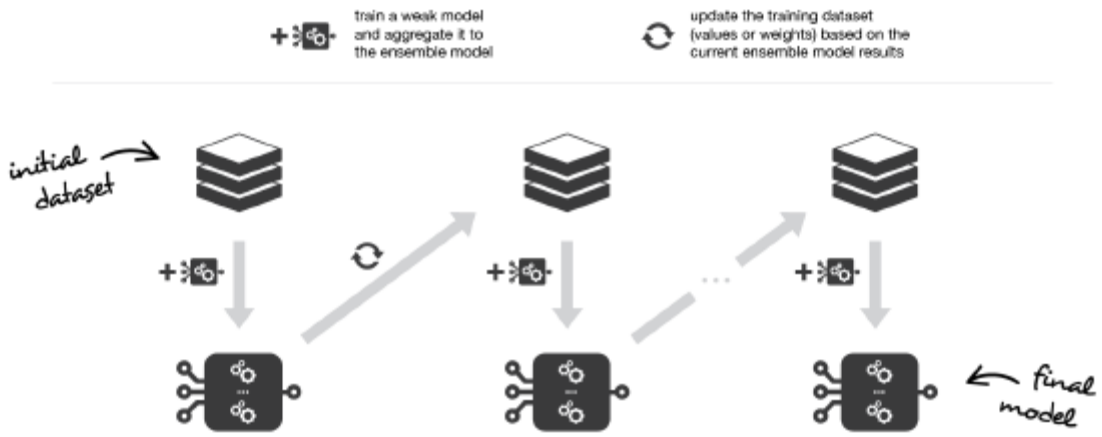
3.2.4.3.3. sklearn.ensemble.ExtraTreesClassifier

class sklearn.ensemble.**ExtraTreesClassifier**(*n_estimators=100, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=False, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

[부스팅]

부스팅

- 기본 아이디어
  - 이전 모델의 훈련 결과를 바탕으로 일련의 예측기를 학습 시키는 방법
  - 여러 개의 분류기가 순차적으로 학습을 수행
    - 단 앞서 학습한 분류기의 예측이 틀린 데이터에 대해 가중치를 부여 후 다음 분류기의 학습과 예측을 수행



부스팅

- 부스팅의 종류 및 특성
  - Adapative Boosting
  - Gradient Boosting
  - XGBoost
  - Light GBM

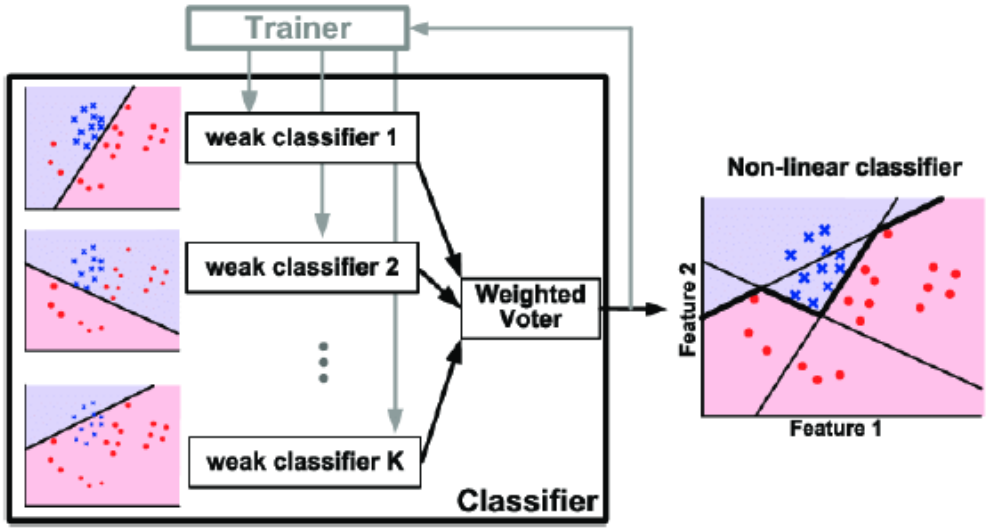
알고리즘	특징
AdaBoost	• 다수결을 통한 정답 분류 및 오답에 가중치 부여
GBM	• Loss Function의 gradient를 통해 오답에 가중치 부여
Xgboost	• GBM 대비 성능향상 • 시스템 자원 효율적 활용 (CPU, Mem) • Kaggle을 통한 성능 검증 (많은 상위 랭커가 사용)
Light GBM	• Xgboost 대비 성능향상 및 자원소모 최소화 • Xgboost가 처리하지 못하는 대용량 데이터 학습 가능 • Approximates the split (근사치의 분할)을 통한 성능 향상

Scikit-learn 에서 제공

- Scikit-learn에서 제공X
- 별도의 라이브러리 설치
- 최근 가장 많이 활용되는 boosting 방법들

부스팅

- Adaptive Boosting
  - 이전 예측기가 잘 예측하지 못하는 훈련 샘플의 가중치를 높여서 다음 예측기를 훈련시키는 방법
  - 예측기들의 투표를 통해 예측이 결정되지만, 예측기마다 부여되는 가중치가 다름



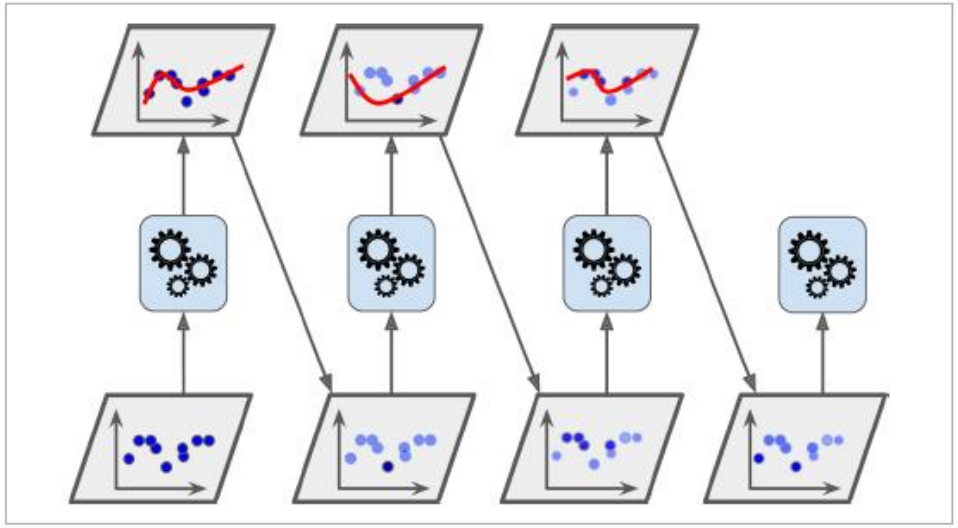


그림 7-7 샘플의 가중치를 업데이트하면서 순차적으로 학습하는 에이다부스트

부스팅

- 사이킷런에서의 AdaBoost

sklearn.ensemble.AdaBoostClassifier

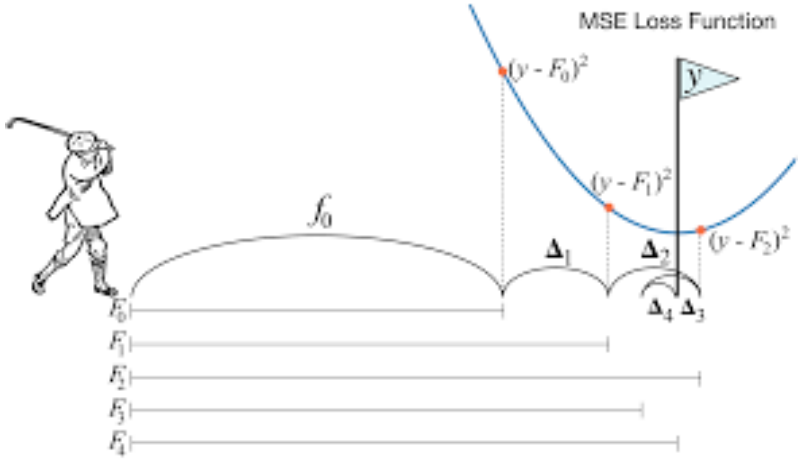
class sklearn.ensemble.**AdaBoostClassifier**(*base_estimator=None, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None*) [source]

sklearn.ensemble.AdaBoostRegressor

class sklearn.ensemble.**AdaBoostRegressor**(*base_estimator=None, n_estimators=50, learning_rate=1.0, loss='linear', random_state=None*) [source]

부스팅

- Gradient Boosting
  - 예측값의 오차를 안다면 더욱 예측을 보완할 수 있음
    - 예측값 + 오차 = 정답이니까!
    - 오차를 알면 정답에 가까워질 수 있음!
  - 이전 예측기의 오차를 다음 예측기의 훈련 데이터로 사용
  - 생성된 예측기의 예측 결과의 합으로 결과를 취합



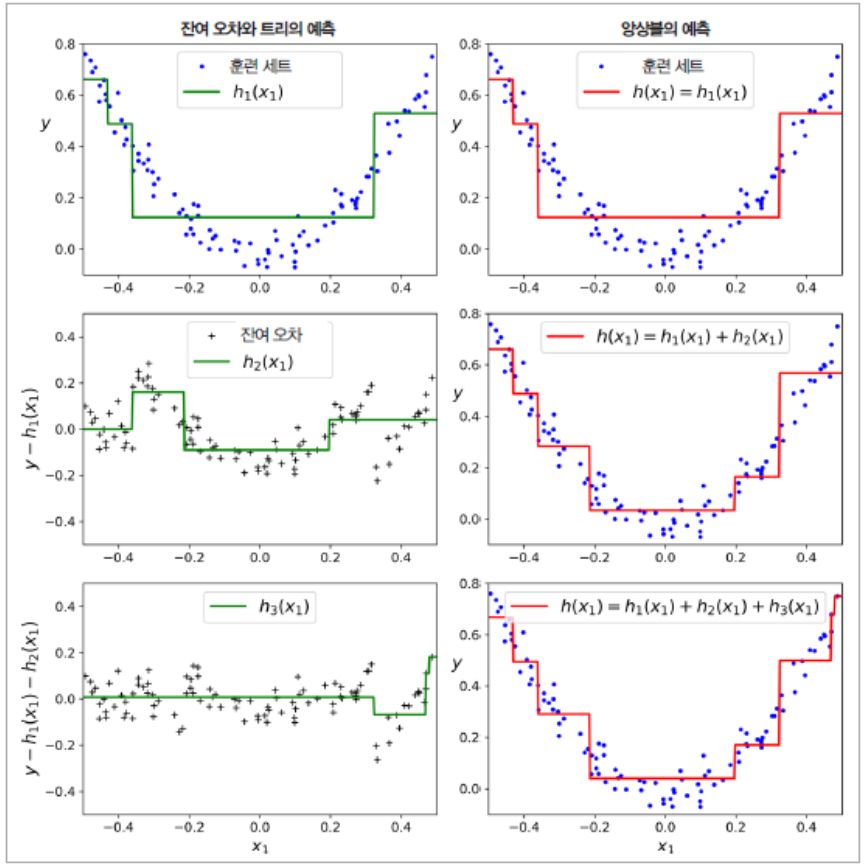


그림 7-9 이 그레이디언트 부스팅 그래프에서 첫 번째 예측기(왼쪽 위)가 평소와 같이 훈련됩니다. 그다음 연이은 예측기(왼쪽 중간, 왼쪽 아래)가 이전의 예측기의 잔여 오차에서 훈련됩니다. 오른쪽 열은 만들어진 앙상블의 예측을 보여줍니다.

부스팅

- 사이킷런에서의 Gradient Boosting

3.2.4.3.5. sklearn.ensemble.GradientBoostingClassifier¶

```
class sklearn.ensemble.GradientBoostingClassifier(loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0,
criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3,
min_impurity_decrease=0.0, min_impurity_split=None, init=None, random_state=None, max_features=None, verbose=0,
max_leaf_nodes=None, warm_start=False, presort='deprecated', validation_fraction=0.1, n_iter_no_change=None, tol=0.0001,
```

3.2.4.3.6. sklearn.ensemble.GradientBoostingRegressor

각 트리의 기여도

```
class sklearn.ensemble.GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=100, subsample=1.0,
criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3,
min_impurity_decrease=0.0, min_impurity_split=None, init=None, random_state=None, max_features=None, alpha=0.9, verbose=0,
max_leaf_nodes=None, warm_start=False, presort='deprecated', validation_fraction=0.1, n_iter_no_change=None, tol=0.0001,
```

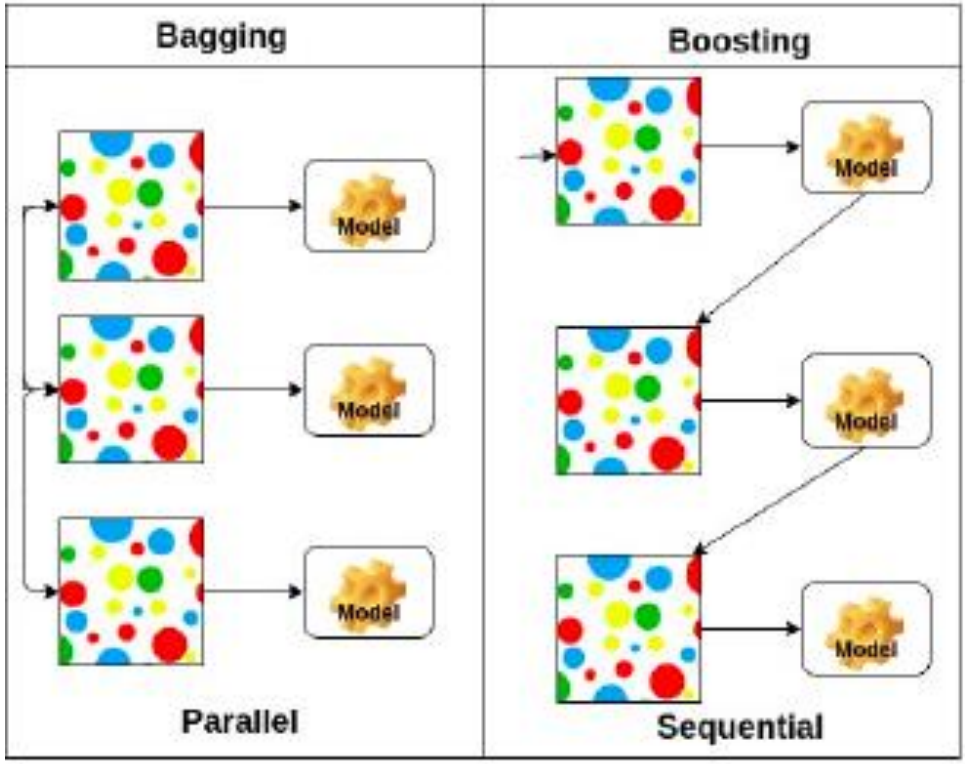
부스팅

- 랜덤 포레스트와 그레디언트 부스팅 비교

	Random Forest (Bagging)	GBM (Boosting)
장점	✓ 모델 성능이 뛰어남. ✓ 파라미터 최적화를 많이 하지 않아도 훌륭한 성능을 보장 ✓ 병렬 처리 ✓ 데이터 스케일을 맞출 필요도 없음	✓ 지도 학습에서 가장 강력하고 널리 사용되는 모델 중 하나 ✓ 데이터 스케일을 맞출 필요도 없음 ✓ 파라미터 최적화 시, 매우 훌륭한 성능
단점	✓ 메모리 사용량이 많음.	✓ 파라미터 최적화가 필수. (미 최적화 시, 성능 보장 X) ✓ 학습 시간이 김. ✓ 병렬 처리가 어려움

부스팅

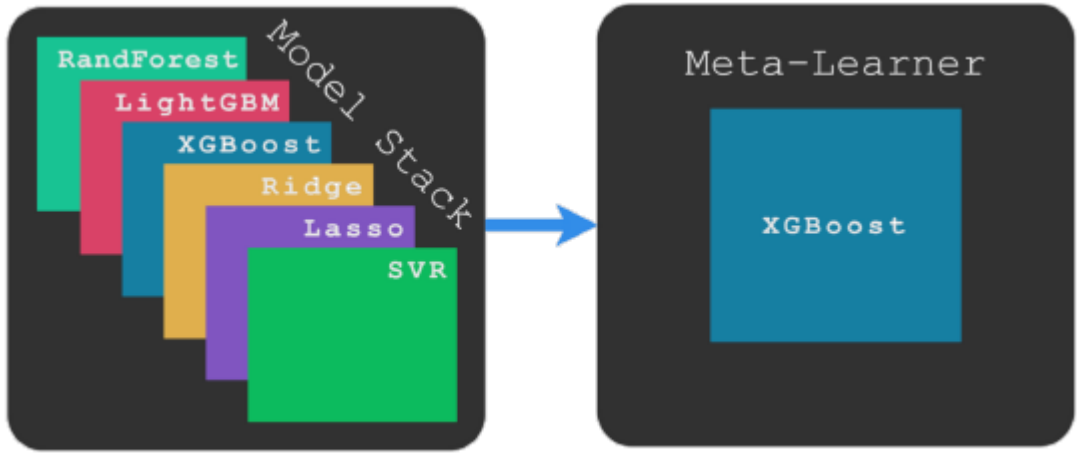
- 배깅과 부스팅의 차이점



[스태킹]

스태킹

- 기본 아이디어
  - 앙상블의 예측을 취합하는 함수를 훈련시킴
  - 여러가지 다른 모델의 예측 결과값을 다시 학습 데이터로 만들고 재 학습시킨 결과를 예측하는 방법
  - 마지막 예측기를 블랜더 혹은 메타 학습기라고 함



스태킹

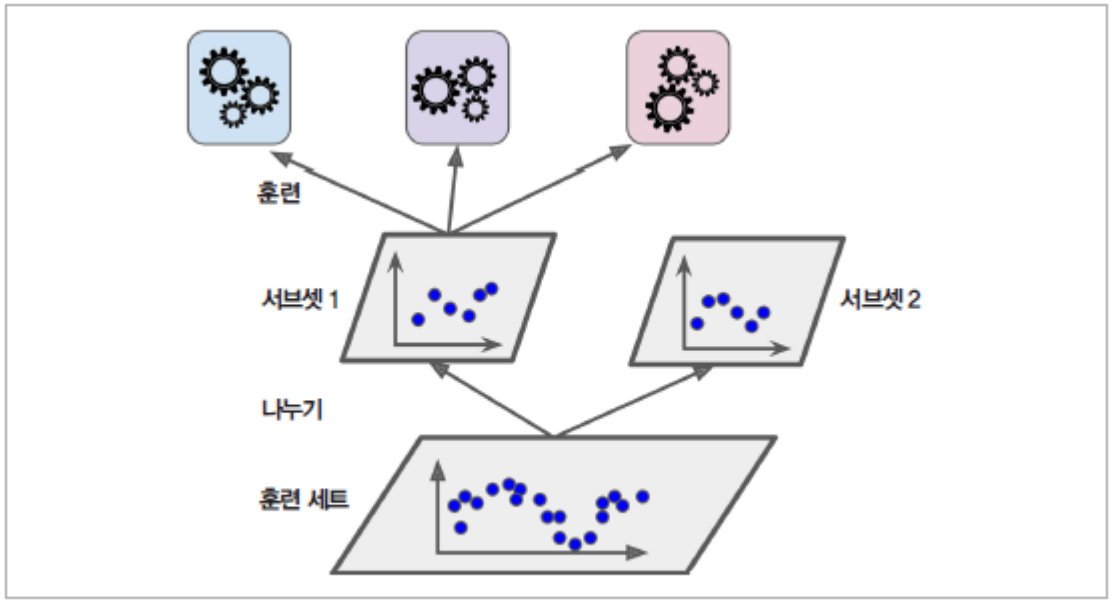


그림 7-13 첫 번째 레이어 훈련하기

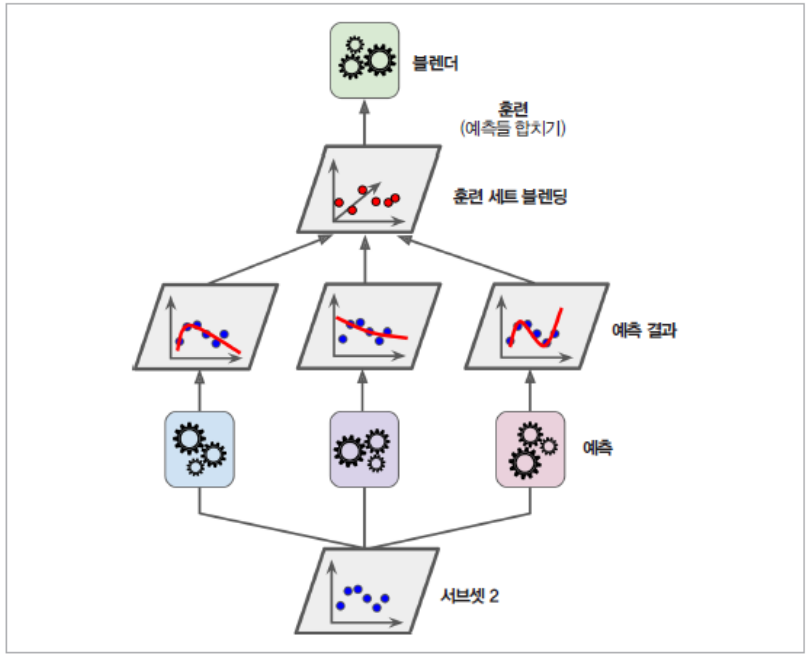
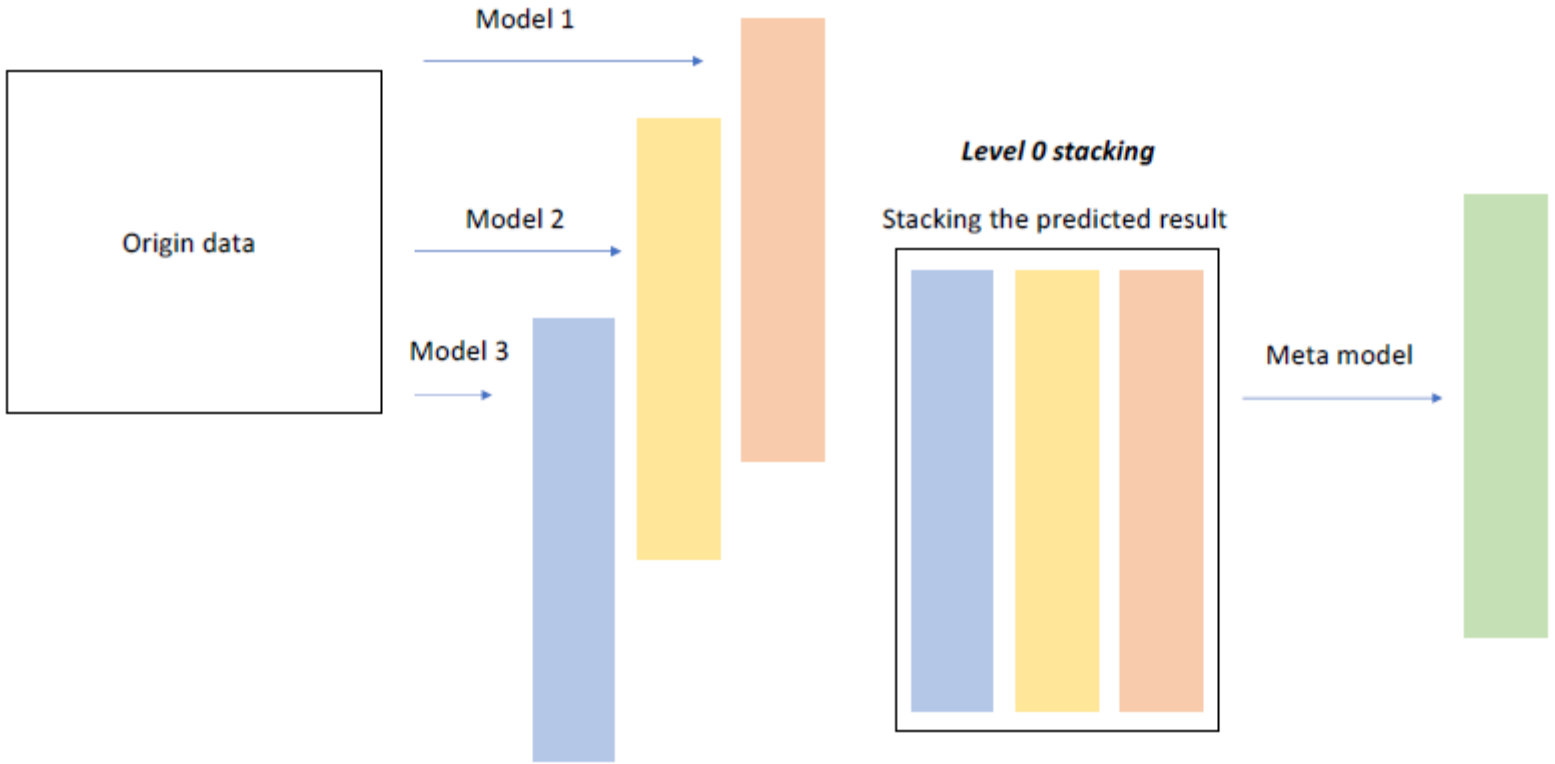


그림 7-14 블렌더 훈련

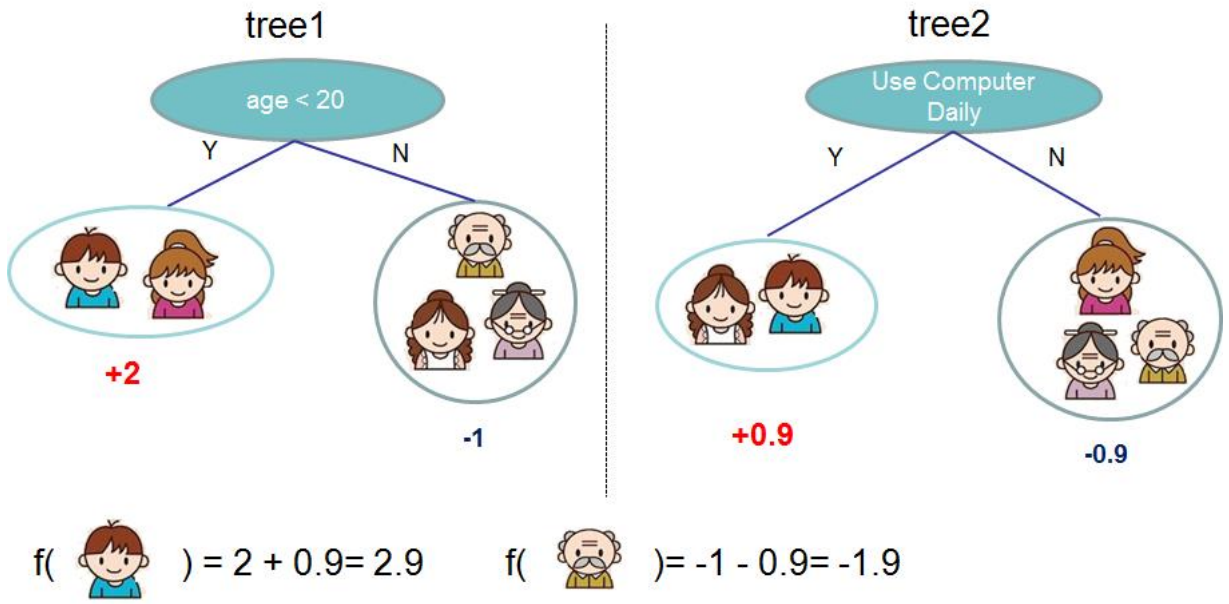
스태킹



[XGBoost]

XGBoost

- 트리 기반의 앙상블 학습에서 최근 가장 각광받고 있는 알고리즘
  - 캐글에서 상위 데이터 과학자들이 많이 활용
  - GBM 기반으로 구현



XGBoost

- XGBoost의 장점

뛰어난 예측 성능	- 분류와 회귀 영역에서 뛰어난 예측 성능 발휘
GBM 대비 빠른 수행 시간	- 일반적인 GBM은 순차적으로 weak learne가 가중치를 증감하는 방법으로 학습하기 때문에 병렬 수행이 불가능하여 학습 속도가 느림 - XGBoost는 병렬 수행 및 다양한 기능으로 빠른 학습 성능 보장. (GBM 대비 빠른 것이지, 결정트리, 랜덤포레스트 등 타 ML 알고리즘에 비해서는 느림)
과적합 규제	- 일반적인 GBM은 과적합 규제 기능 X
나무 가지치기 (Tree Pruning)	- 일반적인 GBM은 분할 시 부정 손실이 발생하면, 분할 수행 X. (부정 손실이란 트리를 분할하면 오히려 엔트로피나 지니지수와 같은 정보 이득이 줄어듬을 의미함.) - XGBoost는 max_depth까지 진행한 뒤 loss function 에서의 개선이 일정 threshold에 못미칠 경우까지 역방향으로 pruning과정을 수행.
자체 내장된 교차 검증	- 반복 수행 시마다 내부적으로 교차검증을 수행 - 조기 중단 기능 있음. (지정된 반복 횟수가 아니라 교차검증을 통해 최적화되면 반복을 멈춤)
결손값 자체 처리	결손값을 자체 처리할 수 있는 기능

XGBoost

- XGBoost의 설치 후 사용

Note

Pre-built binary wheel for Python

If you are planning to use Python, consider installing XGBoost from a pre-built binary wheel, available from Python Package Index (PyPI). You may download and install it by running

```
# Ensure that you are downloading one of the following:
#   * xgboost-{version}-py2.py3-none-manylinux1_x86_64.whl
#   * xgboost-{version}-py2.py3-none-win_amd64.whl
pip3 install xgboost
```

```
import xgboost as xgb
# read in data
dtrain = xgb.DMatrix('demo/data/agaricus.txt.train')
dtest = xgb.DMatrix('demo/data/agaricus.txt.test')
# specify parameters via map
param = {'max_depth':2, 'eta':1, 'objective':'binary:logistic' }
num_round = 2
bst = xgb.train(param, dtrain, num_round)
# make prediction
preds = bst.predict(dtest)
```

Quiz3

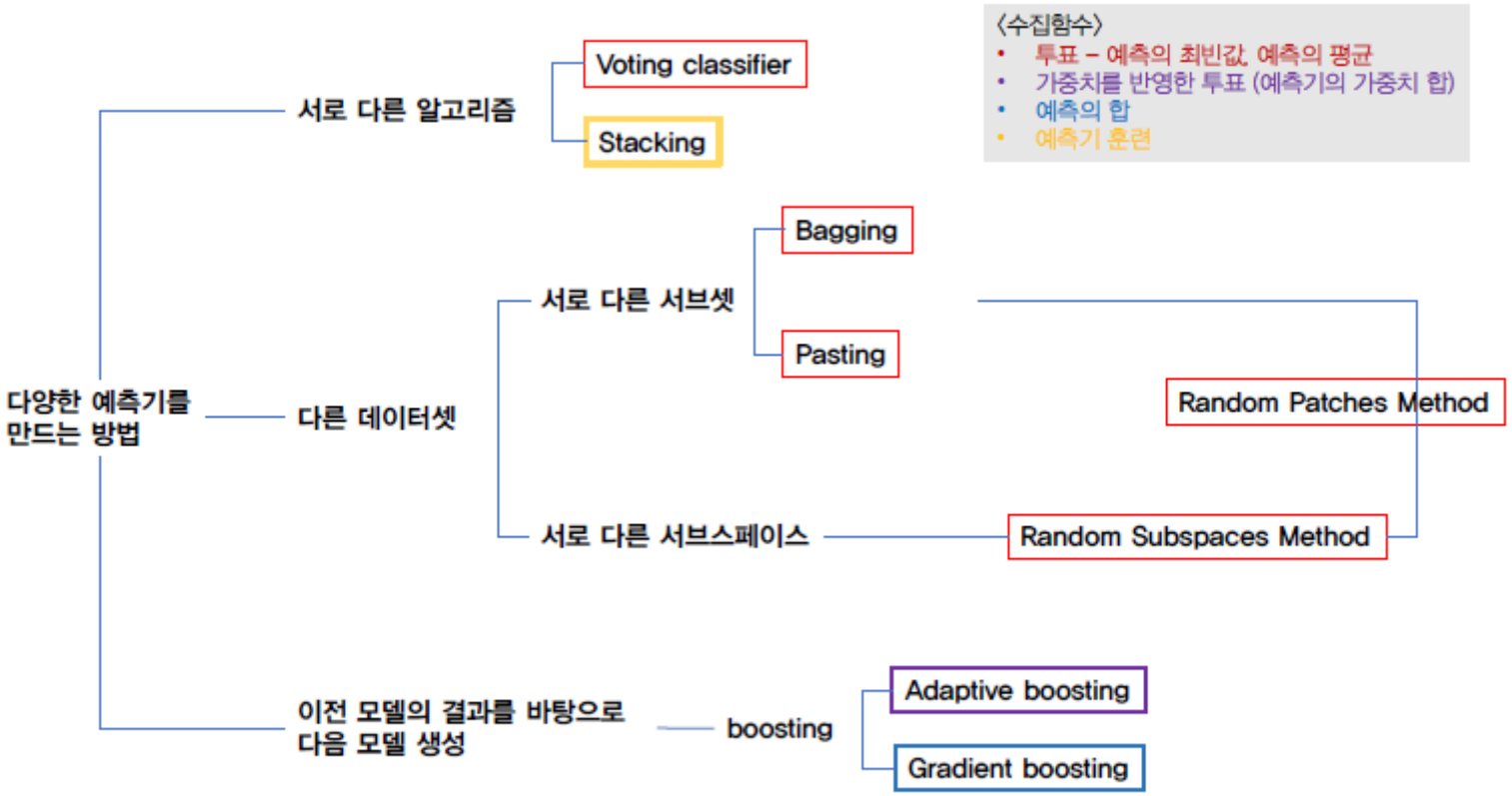
- Q1. 배깅과 부스팅 차이점
  - A1.
- Q2. 스태킹에서 메타 모델이란?
  - A2.
- Q3. 모델의 중요도를 볼 수 있는가?
  - A3.

(2/3)

[실습]

[요약]

요약



-vlog 끝-