5. 앙상블

1 앙상블의 종류

앙상블이란?

앙상블은 기본 모델(base model)이라 불리는 여러 작은 모델을 개별적으로 학습하여 종합한 모델입니다.

공짜 점심 이론: 모든 문제에 대해 다른 모델보다 항상 우월한 모델은 없음 → 앙상블 도입의 배경

앙상블 모델은 서로 다른 여러 모델을 바탕으로 집단 지성을 표방함

캐글에서 가장 많이 사용되는 알고리즘이 XGBoost와 LightGBM 등의 앙상블 모델임

앙상블 종류 개요

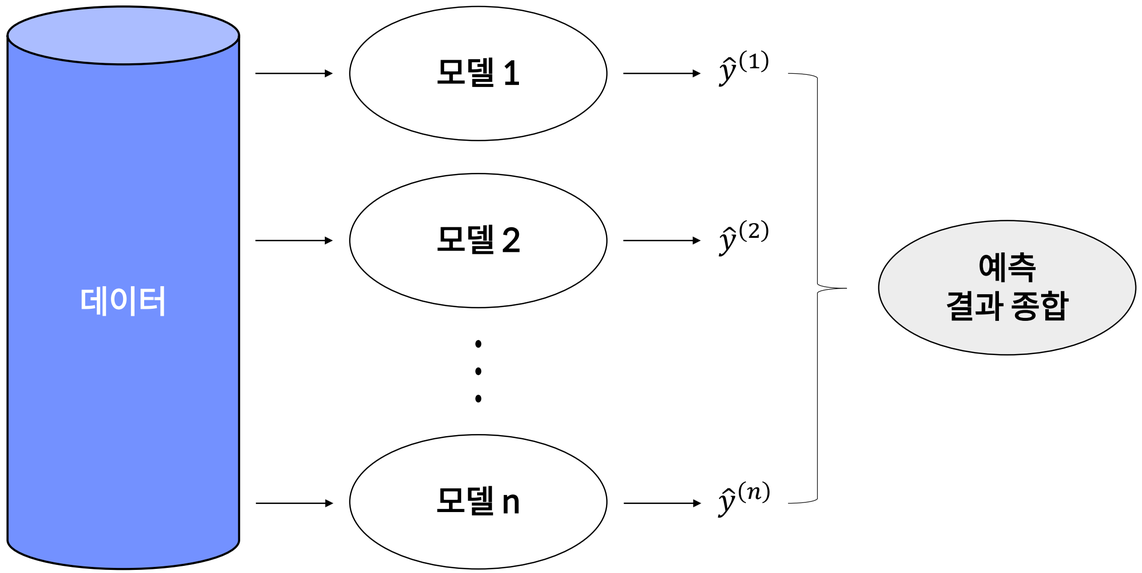
앙상블은 여러 모델을 학습하는 방법과 예측 결과를 종합하는 방법에 따라 앙상블을 구분할 수 있습니다.

앙상블의 종류

앙상블	학습 방법	종합 방법
투표(voting)	전체 데이터를 사용하여 독립적으로 학습	예측 결과의 평균 및 최빈값으로 종합
배깅(bagging)	붓스트랩한 데이터를 사용하여 독립적으로 학습	예측 결과의 평균 및 최빈값으로 종합
부스팅(boosting)	전체 데이터를 사용하여 순차적으로 학습	예측 결과의 평균 및 최빈값으로 종합
스태킹(stacking)	전체 데이터를 사용하여 독립적으로 학습	각 모델의 예측 결과를 특징으로 하는 모델을 사용하여 최종 예측 결과를 종합

투표 모델

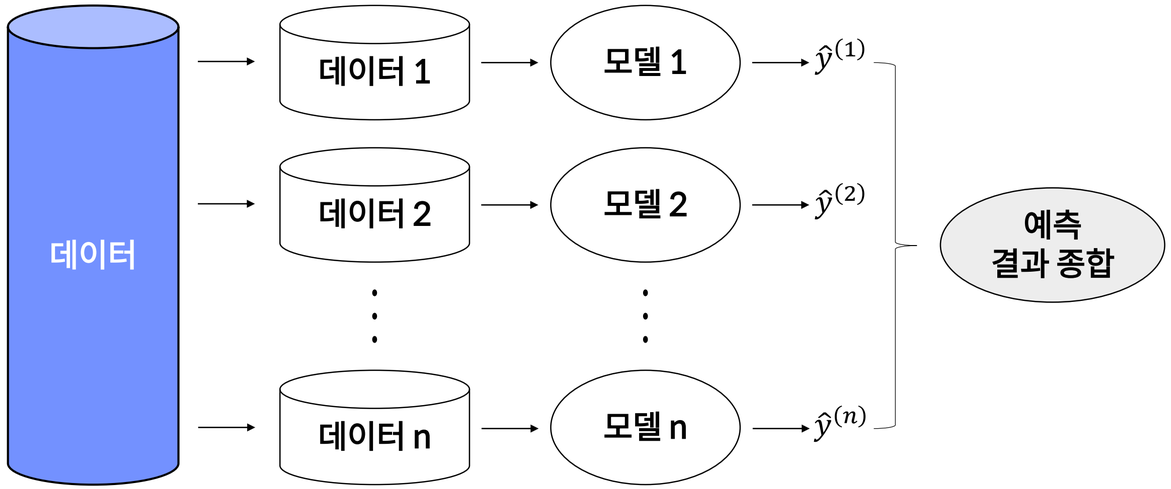
투표 모델은 전체 데이터를 사용해 독립적으로 학습한 여러 모델의 예측 결과를 평균 및 최빈값으로 종합하는 모델입니다.



- 같은 데이터로 여러 모델을 만들기 때문에 각 모델의 예측 결과가 비슷할 수 있음
- 여러 모델의 예측 결과를 종합한 것과 하나의 모델의 예측 결과에 큰 차이가 없어 여러 모델을 학습하는 게 의미가 없을 수 있음
- 투표 모델을 사용할 땐 서로 크게 다른 모델 여러 개를 사용해야 함

배깅

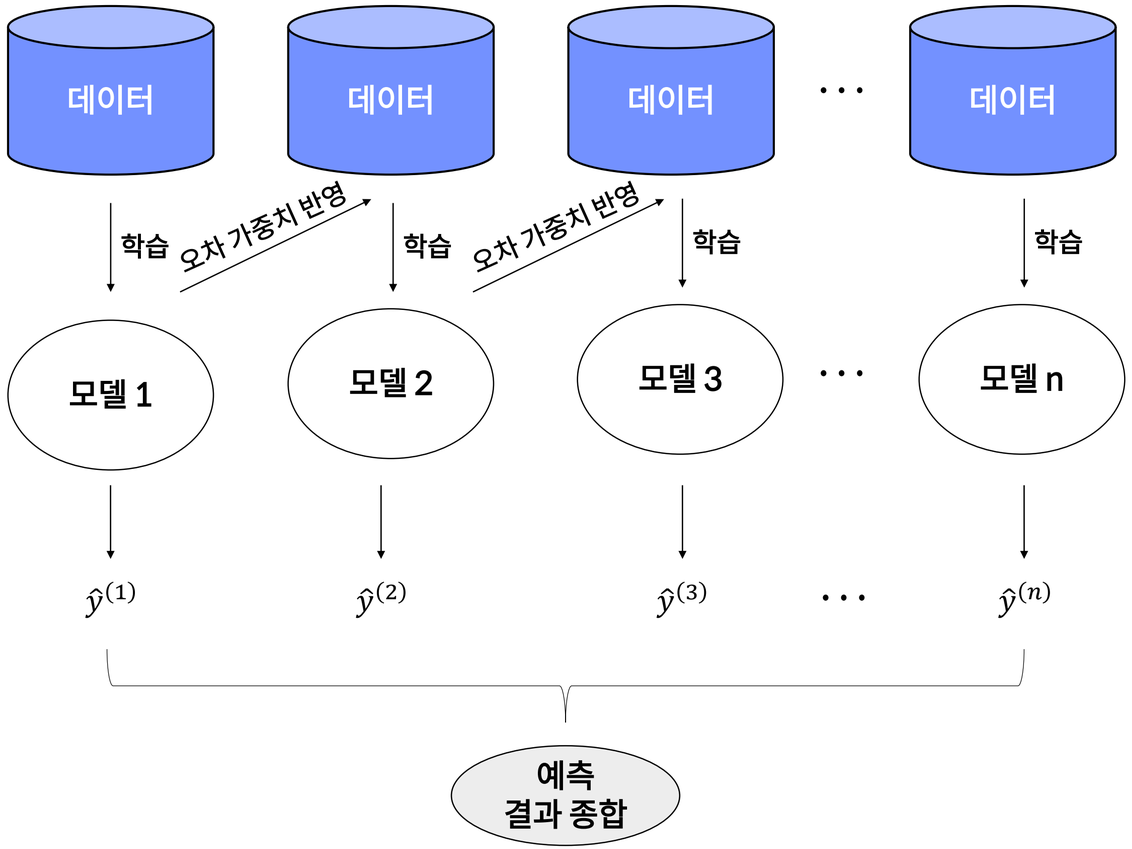
배깅은 투표 모델과 비슷하지만 붓스트랩(bootstrap)으로 샘플링한 데이터로 각각의 모델을 학습한다는 데서 차이가 있습니다.



- 붓스트랩이란 복원을 허용하는 랜덤 샘플링(random sampling)으로 원 데이터에서 행(샘플)과 열(특징)을 임의로 골라 작은 데이터를 샘플링함
- 복원을 허용하기에 같은 샘플 혹은 특징이 둘 이상의 샘플링한 데이터에 포함될 수 있음

부스팅

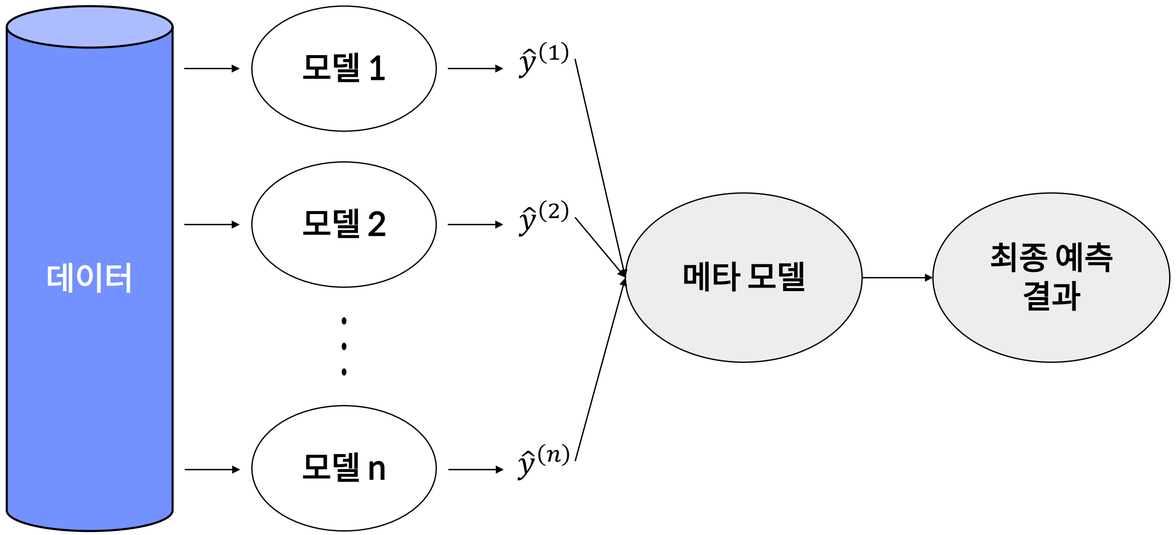
부스팅은 한 모델을 학습하고 계산한 오차를 보정하는 모델을 순차적으로 학습합니다.



- 여기서 i번째 모델은 (i-1)번째 모델이 제대로 예측 못한 샘플에 가중치를 부여한 데이터로 학습함
- 컴퓨팅 자원만 충분하다면 동시에 여러 모델을 학습할 수 있는 배깅과 투표 모델, 스태킹 모델과 다르게, 부스팅 모델은 이전 모델이 학습돼야만 다음 모델을 학습할 수 있으므로 학습 시간이 오래 걸림
- 이러한 단점을 보완하기 위해 대표적인 부스팅 모델인 XGBoost와 LightGBM은 결정 나무를 학습하는 과정에서 병렬 연산(parallel computing)을 사용함

스태킹

스태킹은 개별 모델이 예측한 결과를 입력으로 하는 메타 모델(meta model)을 사용해 예측 결과를 종합합니다.



- 메타 모델은 개별 모델의 예측값을 특징으로 사용하고 데이터의 라벨을 그대로 라벨에 사용해 학습함

5. 앙상블

2 결정 나무 기반의 앙상블 모델

왜 결정 나무인가?

결정 나무는 앙상블 모델의 기본 모델로 많이 사용됩니다.

결정 나무는 대표적인 앙상블 모델인 랜덤 포레스트, XGBoost, LightGBM의 기본 모델임

얕은 결정 나무는 적은 데이터로 학습하더라도 과적합될 가능성이 매우 작음

배깅은 붓스트랩된 작은 데이터로 모델을 학습하므로 과적합 위험이 적은 모델을 사용해야 함

시드가 다르다면 같은 데이터로 학습한 결정 나무일지라도 다르게 분지될 수 있어, 다양한 조건으로 데이터 공간을 탐색할 수 있음

랜덤포레스트

랜덤 포레스트는 기본 모델이 결정 나무인 배깅 모델입니다.

랜덤 포레스트의 주요 하이퍼 파라미터는 기본 모델의 개수와 붓스트랩 관련 하이퍼파라미터를 제외하면 결정 나무의 하이퍼 파라미터와 같음

기본 모델의 개수가 많을수록 모델이 복잡해지지만 각 모델이 학습에 사용하는 데이터 크기는 비슷하므로 과적합 위험이 커진다고 보긴 힘듦

그러나 작은 학습 데이터로 개별 모델을 학습하므로 나무의 깊이는 5 이하로 낮게 설정해주는 것이 적절함

XGboost

XGBoost는 부스팅 방식으로 여러 개의 결정 나무를 병렬로 학습하는 앙상블 모델입니다.

손실 함수

$$\mathcal{L} = \sum_{i=1}^{n} E_i + \sum_{k=1}^{K} \phi(f_k)$$

- f_k $(k = 1, 2, \cdots, K)$: k번째 결정 나무
- E_i: 샘플 i에 대한 오차
- $\phi(f_k)$: 잎 노드의 개수 등으로 계산하는 f_k의 모델 복잡도
- 각 모델을 학습할 때 이전 모델의 오차를 얼마나 반영할지 나타내는 하이퍼파라미터인 학습률이 있으며, 학습률이 클수록 이전 트리의 오차를 과하게 반영하므로 과적합 위험이 높아집니다.

예측 과정

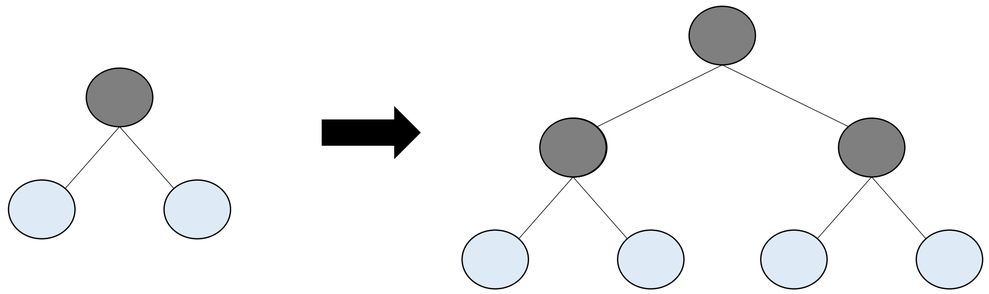
$$\hat{y}_{k,i} = \hat{y}_{k-1,i} + f_k\left(\boldsymbol{x}^{(i)}\right)$$

- $f_k\left(\boldsymbol{x}^{(i)}\right)$: f_k가 샘플 i의 라벨을 예측한 값
- $\hat{y}_{k-1,i}$: 이전 $k-1$ 개의 모델로 예측한 값을 종합한 값

LightGBM

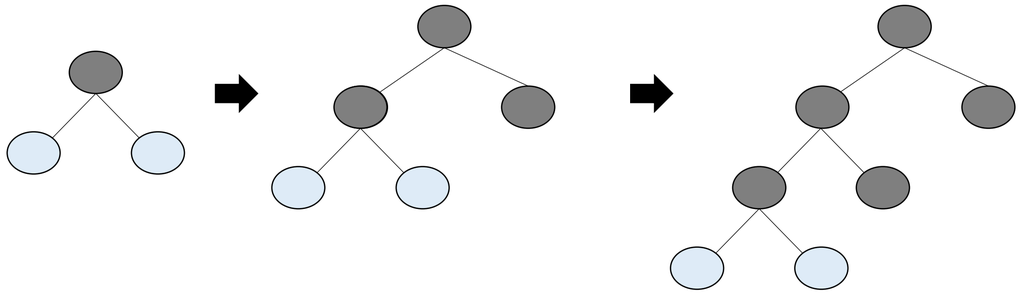
LightGBM은 XGBoost와 비슷하지만 잎 노드 중심 분할(leaf wise) 방식을 사용해 깊고 비대칭적인 트리를 만듭니다.

XGBoost: 균형 분할 방식



- 파란색 노드: 더 분지돼야 할 노드를
- 검은색 노드: 이미 분지가 완료된 노드
- n번째 층에 $2^{(n-1)}$개의 노드가 있도록 분할

LightGBM: 잎 노드 중심 분할 방식



- LightGBM은 잎 노드 중심의 분할을 통해 불필요한 분지를 하지 않으므로 예측 오차가 적고 모델이 가벼움
- 결정 나무, 랜덤 포레스트, XGBoost와 다르게 잎 노드가 깊이에 비례해서 많은 것은 아니므로 최대 깊이보다는 최대 잎 노드 개수를 설정하는 것이 좋음

5. 앙상블

3 실습

예제 데이터 불러오기

앙상블 모델을 학습할 예제 데이터를 불러옵니다.

예제 데이터 불러오기

```
1  import pandas as pd
2  from sklearn.model_selection import train_test_split
3
4  df1 = pd.read_csv("../../data/classification/optdigits.csv") # 다중 분류 데이터
5  df2 = pd.read_csv("../../data/regression/baseball.csv") # 회귀 데이터
6
7  X1 = df1.drop('y', axis = 1)
8  y1 = df1['y']
9  X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, random_state = 2022)
10
11 X2 = df2.drop('y', axis = 1)
12 y2 = df2['y']
13 X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, random_state = 2022)
```

랜덤 포레스트 모델 학습

랜덤 포레스트는 sklearn.ensemble 모듈의 RandomForestClassifier와 RandomForestRegressor를 이용해 구현할 수 있습니다.

주요 인자

인자	설명	기본 값
n_estimators	결정 나무의 개수	100
max_depth	최대 깊이	None
max_features	개별 결정 나무를 학습하는 데 사용하는 특징 개수 혹은 비율	전체 특징을 사용

랜덤 포레스트와 결정 나무 비교

랜덤 포레스트와 결정 나무를 비교해보겠습니다.

랜덤 포레스트와 결정 나무 비교 (분류)

```
1  from sklearn.tree import DecisionTreeClassifier as DTC, DecisionTreeRegressor as DTR
2  from sklearn.ensemble import RandomForestClassifier as RFC, RandomForestRegressor as RFR
3  from sklearn.metrics import *
4  dtc = DTC(max_depth = 10, random_state = 2022).fit(X1_train, y1_train)
5  rfc = RFC(max_depth = 5, random_state = 2022).fit(X1_train, y1_train)
6  dtc_pred = dtc.predict(X1_test)
7  rfc_pred = rfc.predict(X1_test)
8  dtc_acc = accuracy_score(y1_test, dtc_pred)
9  rfc_acc = accuracy_score(y1_test, rfc_pred)
10 print(dtc_acc, rfc_acc)
```

- **라인 1:** DecisionTreeClassifier와 DecisionTreeRegressor를 각각 DTC와 DTR로 축약하여 불러왔습니다. 이처럼 둘 이상의 함수를 축약해서 불러올 때는 import 함수1 as 축약어1, 함수2 as 축약어와 같이 쉼표(,)로 연결하여 불러옵니다.

- **라인 4:** 최대 깊이가 10인 결정 나무를 학습합니다.
- **라인 5:** 최대 깊이가 5인 결정 나무를 100개(기본값) 학습합니다.

```
0.8811387900355871      0.9423487544483986
```

랜덤 포레스트와 결정 나무 비교 (계속)

랜덤 포레스트와 결정 나무를 비교해보겠습니다.

랜덤 포레스트와 결정 나무 비교 (분류)

```
1  print(rfc[0])
2  print(rfc[50])
```

DecisionTreeClassifier(max_depth=5, max_features='auto', random_state=40194941)
DecisionTreeClassifier(max_depth=5, max_features='auto', random_state=101951500)

- 0번째 요소와 50번째 요소 모두 DecisionTreeClassifer임을 알 수 있음
- random_state를 2022로 설정했는데도 개별 모델의 random_state는 다름. 그 이유는 사이킷런에서는 다양한 모델로 구성된 랜덤 포레스트를 학습하기 위해 랜덤 포레스트의 시드를 바탕으로 결정 나무의 시드를 난수로 정하기 때문
- 즉, 랜덤 포레스트의 시드가 개별 결정 나무의 시드를 결정하는 것이지, 랜덤 포레스트의 시드와 개별 결정 나무의 시드가 같은 것은 아닙니다.

랜덤 포레스트와 결정 나무 비교 (계속)

랜덤 포레스트와 결정 나무를 비교해보겠습니다.

랜덤 포레스트와 결정 나무 비교 (회귀)

```
1  dtr = DTR(max_depth = 10, random_state = 2022).fit(X2_train, y2_train)
2  rfr = RFR(max_depth = 5, random_state = 2022).fit(X2_train, y2_train)
3
4  dtr_pred = dtr.predict(X2_test)
5  rfr_pred = rfr.predict(X2_test)
6
7  dtr_mae = mean_absolute_error(y2_test, dtr_pred)
8  rfr_mae = mean_absolute_error(y2_test, rfr_pred)
9  print(dtr_mae, rfr_mae)
```

```
579.9513012477719    530.4179945238225
```

XGBoost 모델 학습

xgboost 모듈의 XGBClassifier와 XGBRegressor를 이용하면 각각 XGBoost 방식의 분류 모델과 회귀 모델을 학습할 수 있습니다.

xgboost 설치

```
$ pip install xgboost
```

주요 인자

인자	설명
n_estimators	결정 나무의 개수
max_depth	최대 깊이
learning_rate	학습률

XGBoost 모델 학습 (계속)

xgboost 모듈의 XGBClassifier와 XGBRegressor를 이용하면 각각 XGBoost 방식의 분류 모델과 회귀 모델을 학습할 수 있습니다.

XGBoost와 결정 나무 비교

```
1  from xgboost import XGBClassifier as XGBC, XGBRegressor as XGBR
2  xgbc = XGBC(random_state=2022, learning_rate = 0.1, max_depth = 5).fit(X1_train, y1_train)
3  xgbr = XGBR(random_state=2022, learning_rate = 0.1, max_depth = 5).fit(X2_train, y2_train)
4  xgbc_pred = xgbc.predict(X1_test)
5  xgbr_pred = xgbr.predict(X2_test)
6
7  acc = accuracy_score(xgbc_pred, y1_test)
8  mae = mean_absolute_error(xgbr_pred, y2_test)
9  print(acc, mae)
```

```
0.9779359430604982   561.5312666949104
```

- 결정 나무의 정확도(0.8811)와 MAE(579.9513)보다 더 좋은 성능을 보임

LightGBM 모델 학습

lightgbm 모듈의 LGBMClassifier와 LGBMRegressor를 이용하면 각각 LightGBM 방식의 분류 모델과 회귀 모델을 학습할 수 있습니다.

lightgbm 설치

```
$ pip install lightgbm
```

주요 인자

인자	설명
n_estimators	결정 나무의 개수
max_depth	최대 깊이
num_leaves	최대 잎 노드 개수
learning_rate	학습률

LightGBM 모델 학습 (계속)

lightgbm 모듈의 LGBMClassifier와 LGBMRegressor를 이용하면 각각 LightGBM 방식의 분류 모델과 회귀 모델을 학습할 수 있습니다.

LightGBM과 결정 나무 비교

```
1  from lightgbm import LGBMClassifier as LGBC, LGBMRegressor as LGBR
2  lgbc = LGBC(random_state=2022, learning_rate=0.1, num_leaves=32).fit(X1_train, y1_train)
3  lgbr = LGBR(random_state=2022, learning_rate=0.1, num_leaves=32).fit(X2_train, y2_train)
4  lgbc_pred = lgbc.predict(X1_test)
5  lgbr_pred = lgbr.predict(X2_test)
6
7  acc = accuracy_score(lgbc_pred, y1_test)
8  mae = mean_absolute_error(lgbr_pred, y2_test)
9  print(acc, mae)
```

```
0.9814946619217082 572.6819576553671
```

- 분류 성능은 LightGBM이 가장 좋지만, 회귀 성능은 랜덤 포레스트와 XGBoost보다는 약간 떨어지고 결정 나무보다는 좋음
- 그러나 데이터에 따라 결과가 다를 수 있으므로 이 결과만 보고 모델과 하이퍼파라미터를 선택하는 것은 매우 위험합니다.