2. MyAutoML2

1 문제 정의 및 클래스 설계

탐색 공간

모델 및 하이퍼파라미터 튜닝 문제의 탐색 공간은 실험을 통해서 설정하며, 이 시스템에서 고려하는 초기 탐색 공간 다음과 같습니다.

하이퍼파라미터	탐색 공간	랜덤 포레스트	XGBoost	LightGBM
결정 나무 개수	$\{10, 11, \cdots, 500\}$	n_estimators	n_estimators	n_estimators
학습률	$\{x \mid 0.01 \leq x \leq 0.31\}$	-	eta	learning_rate
나무의 최대 깊이	$\{2, 3, 4, \cdots, 10\}$	max_depth	max_depth	max_depth
최대 잎 나무 개수	$\{8, 9, \cdots, 256\}$	max_leaf_nodes	max_leaves	num_leaves
행 샘플링 비율	$\{x \mid 0.1 \leq x \leq 0.95\}$	max_samples	subsample	subsample
열 샘플링 비율	$\{x \mid 0.3 \leq x \leq 0.9\}$	max_features	colsample_bytree	colsample_bytree

스태킹 앙상블

랜덤 서치를 사용해 탐색을 수행하며, 성능이 우수한 num_base_models 개의 모델로 스태킹 앙상블을 학습합니다. 스태킹 앙상블은 sklearn.ensemble의 StackingRegressor를 사용해 학습합니다.

인자	설명
estimators	스태킹 앙상블을 구성하는 기본 모델 목록으로 요소가 (모델명, 모델 인스턴스)인 리스트
final_estimator	베이스 모델을 병합하는 데 사용할 분류 모델

클래스 설계
:인자 정의

MyAutoml_2는 다음과 같은 인자를 갖는 파이썬 클래스로 개발합니다.

인자	설명	기본값
seed	각 모델의 시드	2022
cv	폴드 개수	5
scoring	회귀 평가 척도	"mean_absolute_error"
summarize_score	평가 결과 요약 방법	“mean”
num_iter	탐색 횟수	1000
num_base_models	앙상블을 구성하는 모델 개수	100
fit_stacking	스태킹 앙상블 모델을 학습할 지 여부	True

클래스 설계
:메서드 정의

MyAutoml_2는 다음과 같은 메서드를 갖습니다.

메서드	설명
fit	특징 X와 라벨 y를 입력받아 랜덤 서치를 사용해 모델과 하이퍼파라미터를 탐색합니다. 또한, 최고 점수의 모델 인스턴스를 전체 데이터로 재학습해 model 속성에 저장합니다.
show_leaderboard	모델 및 파라미터 탐색 결과인 리더보드를 출력합니다. 이때 리더보드는 모델, 하이퍼파라미터, 성능 지표로 구성된 데이터프레임입니다.
predict	fit 메서드에서 저장한 model로 새로 입력된 특징의 라벨을 예측합니다.

2. MyAutoML2

2 하이퍼 파라미터 범위 설정

모듈 불러오기

하이퍼 파라미터 설정을 위한 실험을 진행하는데 필요한 모듈 및 패키지를 불러옵니다.

모듈 불러오기

```
1  import pandas as pd
2  from sklearn.model_selection import ParameterGrid, KFold, cross_val_score
3  from sklearn.ensemble import RandomForestRegressor as RFR
4  from xgboost import XGBRegressor as XGBR
5  from lightgbm import LGBMRegressor as LGBR
6  from sklearn.metrics import mean_absolute_error as MAE
7  import numpy as np
8  from sklearn.tree import DecisionTreeRegressor
9  from sklearn.tree import export_text
10 from scipy.stats import randint, uniform
11 from tqdm import tqdm
```

샘플링 함수 정의

실험에 사용할 샘플링 함수를 정의합니다.

샘플링 인스턴스 생성

```
1  num_trees_rv = randint(10, 500)
2  learning_rate_rv = uniform(0.01, 0.3)
3  max_depth_rv = randint(2, 10)
4  num_leaves_rv = randint(8, 256)
5  row_ratio_rv = uniform(0.1, 0.85)
6  col_ratio_rv = uniform(0.3, 0.6)
```

- **라인 2:** 결정 나무 개수를 10과 500 사이의 정수로 샘플링하는 인스턴스 num_trees_rv를 randint를 사용해 생성합니다.
- **라인 3:** 학습률을 0.01과 0.31 사이의 실수로 샘플링하는 인스턴스 learning_rate_rv를 uniform을 사용해 생성합니다.

샘플링 함수 정의

```
1  def sampling():
2      num_trees = num_trees_rv.rvs()
3      learning_rate = learning_rate_rv.rvs()
4      max_depth = max_depth_rv.rvs()
5      num_leaves = num_leaves_rv.rvs()
6      row_ratio = row_ratio_rv.rvs()
7      col_ratio = col_ratio_rv.rvs()
8      return num_trees, learning_rate, max_depth, num_leaves, row_ratio, col_ratio
```

- 이 함수는 num_trees_rv부터 col_ratio_rv까지 rvs 메서드를 사용해 샘플링한 값을 튜플(num_trees, learning_rate, max_depth, num_leaves, row_ratio, col_ratio)로 반환합니다.

데이터 준비

실험에 사용할 다섯 개 데이터를 불러오고 특징과 라벨로 분리하여 experiment_data_list에 저장하겠습니다.

데이터 준비

```
1  experiment_data_list = []
2  for file_name in ["ele-1", "ele-2", "friedman",  "puma32h",  "wizmir"]:
3      df = pd.read_csv("../../data/regression/{}.csv".format(file_name))
4      X = df.drop('y', axis = 1)
5      y = df['y']
6      experiment_data_list.append((X, y))
```

실험 데이터 생성

모델별로 1만 번씩 하이퍼파라미터를 샘플링하고 k-겹 교차 검증을 통해 평가하는 방식으로 실험 데이터를 생성하겠습니다.

실험 데이터 생성

```
1 RFR_result = []
2 XGB_result = []
3 LGB_result = []
4 for _ in tqdm(range(10000)):
5    data_idx = np.random.choice(range(len(experiment_data_list)))
6    X, y = experiment_data_list[data_idx]
7    num_trees, learning_rate, max_depth, num_leaves, row_ratio, col_ratio = sampling()
8
9    RFR_model = RFR(
10       n_estimators=num_trees,
11       max_depth=max_depth,
12       max_leaf_nodes=num_leaves,
13       max_samples=row_ratio,
14       max_features=col_ratio,
15   )
16
...
34
```

- **라인 1~3:** 각각의 실험 결과를 담을 RFR_result, XGB_result, LGB_result를 빈 리스트로 초기화합니다.
- **라인 5~6:** 임의로 선택한 인덱스를 바탕으로 실험 데이터를 고릅니다.
- **라인 7:** sampling 함수를 이용해 하이퍼파라미터를 샘플링합니다.
- **라인 9~15:** 샘플링한 하이퍼파라미터를 갖는 랜덤 포레스트 인스턴스 RFR_model을 생성합니다.

실험 데이터 생성 (계속)

모델별로 1만 번씩 하이퍼파라미터를 샘플링하고 k-겹 교차 검증을 통해 평가하는 방식으로 실험 데이터를 생성하겠습니다.

실험 데이터 생성

```
35  RFR_score_list = cross_val_score(
36      RFR_model, X, y, cv=5, scoring="neg_mean_absolute_error"
37  )
38  RFR_score = (-RFR_score_list).mean()
...
46  LGB_score = (-LGB_score_list).mean()
47
48  RFR_record = [num_trees, max_depth, num_leaves, row_ratio, col_ratio, RFR_score]
...
58  LGB_record = [
...
66  ]
67
68  RFR_result.append(RFR_record)
69  XGB_result.append(XGB_record)
70  LGB_result.append(LGB_record)
```

실험 데이터 생성
(계속)

모델별로 1만 번씩 하이퍼파라미터를 샘플링하고 k-겹 교차 검증을 통해 평가하는 방식으로 실험 데이터를 생성하겠습니다.

실험 데이터 생성

```
RFR_cols = ["num_trees", "max_depth", "num_leaves", "row_ratio", "col_ratio", "score"]
XGB_cols = [
    "num_trees",
...
    "score",
]

RFR_result = pd.DataFrame(RFR_result, columns=RFR_cols)
XGB_result = pd.DataFrame(XGB_result, columns=XGB_cols)
LGB_result = pd.DataFrame(LGB_result, columns=XGB_cols)
```

```
RFR_result.to_csv("MyAutoML2_RFR_실험결과.csv", index = False)
XGB_result.to_csv("MyAutoML2_XGB_실험결과.csv", index = False)
LGB_result.to_csv("MyAutoML2_LGB_실험결과.csv", index = False)
```

모델별 MAE 분포 확인

모델별 MAE의 분포를 다음과 같이 확인하겠습니다.

실험 데이터 생성

```
1  RFR_MAE_dist = RFR_result["score"].describe()
2  XGB_MAE_dist = XGB_result["score"].describe()
3  LGB_MAE_dist = LGB_result["score"].describe()
4  MAE_dist = pd.concat([RFR_MAE_dist, XGB_MAE_dist, LGB_MAE_dist], axis = 1)
5  MAE_dist.columns = ["RFR", "XGB", "LGB"]
6  display(MAE_dist)
```

- **라인 1:** RFR_result의 score 칼럼의 통계량을 RFR_MAE_dist에 저장합니다.
- **라인 4~5:** RFR_MAE_dist, XGB_MAE_dist, LGB_MAE_dist를 열 방향으로 병합하고 칼럼 명을 ["RFR", "XGB", "LGB"]로 설정합니다.

	RFR	XGB	LGB
count	10000.000000	10000.000000	10000.000000
mean	125.287492	128.788634	112.990791
std	177.680552	223.506338	187.934021
min	0.006287	0.006035	0.005951
25%	0.998655	0.962118	0.947524
50%	2.084521	1.497201	1.246464
75%	236.530071	85.982339	91.620124
max	542.994830	1373.506491	1133.407707

- 랜덤 포레스트: 1사분위수, 중위수, 3사분위수가 가장 크고 최댓값이 가장 작음
- XGBoost: 3사분위수가 가장 작고 최댓값이 가장 크며 나머지 지표는 중간임
- LightGBM: 평균, 중위수, 최솟값이 가장 작음

➡ 각 모델이 선택될 확률: 랜덤 포레스트: 20%, XGBoost: 30%, LightGBM: 50%

모델별 결정 나무 학습 및 해석

결정 나무 해석에 필요한 함수 text_to_rule_list와 extract_float를 이전과 같이 정의하겠습니다.

```
def text_to_rule_list(r):
    node_list = []
    leaf_node_list = []

    for i, node in enumerate(r.split("\n")[:-1]):
        rule = node.split('- ')[1]
        indent = node.count(' ' * 3)
        if 'value' in rule:
            leaf_node_list.append([i, rule, indent])
        node_list.append([i, rule, indent])

    prediction_rule_list = []
    for leaf_node in leaf_node_list:
        prediction_rule = []
        idx, decision, indent = leaf_node
        for indent_level in range(indent-1, -1, -1):
            for node_idx in range(idx, -1, -1):
                node = node_list[node_idx]
                rule = node[1]
                if node[2] == indent_level and "value" not in node[1]:
                    prediction_rule.append(rule)
                    break
        prediction_rule_list.append([prediction_rule, decision])

return prediction_rule_list
```

```
def extract_float(output):
    output = output.split('[')[1]
    output = float(output[:-1])
    return output
```

랜덤 포레스트의 하이퍼파라미터 범위 설정

결정 나무를 활용하여 랜덤 포레스트의 하이퍼파라미터 범위를 결정하겠습니다.

```
RFR_X = RFR_result.drop('score', axis = 1)
RFR_y = RFR_result['score']
model = DecisionTreeRegressor(max_depth = 5, random_state = 2022).fit(RFR_X, RFR_y)
result = text_to_rule_list(export_text(model, feature_names = list(RFR_X.columns)))
result = pd.DataFrame(result, columns = ["condition", "output"])

result['condition'] = result['condition'].apply(' & '.join)
result['output'] = result['output'].apply(extract_float)
result.sort_values(by = "output", inplace = True)

display(result.head(10))
```

랜덤 포레스트의 하이퍼파라미터 범위 설정 (계속)

결정 나무를 활용하여 랜덤 포레스트의 하이퍼파라미터 범위를 결정하겠습니다.

	condition	output
5	num_trees > 495.50 & col_ratio <= 0.82 & num_trees > 494.50 & max_depth <= 2.50 & max_depth <= 3.50	0.74
16	col_ratio <= 0.30 & col_ratio <= 0.30 & num_leaves <= 36.50 & max_depth > 3.50	2.07
4	num_trees <= 495.50 & col_ratio <= 0.82 & num_trees > 494.50 & max_depth <= 2.50 & max_depth <= 3.50	2.62
28	num_trees > 493.50 & max_depth <= 8.50 & col_ratio > 0.50 & num_leaves > 36.50 & max_depth > 3.50	54.40
15	col_ratio > 0.53 & num_leaves > 206.50 & col_ratio > 0.44 & max_depth > 2.50 & max_depth <= 3.50	81.71
12	row_ratio <= 0.22 & num_leaves <= 206.50 & col_ratio > 0.44 & max_depth > 2.50 & max_depth <= 3.50	87.36
30	num_leaves > 51.50 & max_depth > 8.50 & col_ratio > 0.50 & num_leaves > 36.50 & max_depth > 3.50	87.45
27	num_trees <= 493.50 & max_depth <= 8.50 & col_ratio > 0.50 & num_leaves > 36.50 & max_depth > 3.50	112.44
9	row_ratio > 0.73 & col_ratio <= 0.41 & col_ratio <= 0.44 & max_depth > 2.50 & max_depth <= 3.50	115.26
24	row_ratio > 0.10 & col_ratio <= 0.50 & col_ratio <= 0.50 & num_leaves > 36.50 & max_depth > 3.50	118.39

- 주로 등장하는 하이퍼파라미터는 max_depth(10회), col_ratio(10회), num_leaves(7회)임
- num_trees와 row_ratio는 각각 4회와 3회만 등장함
  → 따라서 num_trees와 row_ratio는 튜닝하지 않음
- max_depth와 num_leaves는 서로 영향을 주는 하이퍼파라미터임
- max_depth와 num_leaves의 관계를 염두에 두고 해석하면 인덱스가 30인 행을 제외하고 깊이가 3에서 8 사이일 때 좋은 결과를 낼 수 있음을 알 수 있음 → num_leaves는 튜닝하지 않고 max_depth를 3과 8 사이에서 튜닝함
- col_ratio는 인덱스가 16, 9, 24인 행을 제외하면 0.44보다 크거나 같게 설정하고 0.82보다 작거나 같게 설정하는 것이 좋음

→

- max_depth: {3,4,5,6,7,8}
- col_ratio: $\{x \mid 0.44 \leq x \leq 0.82\}$

XGBoost의
하이퍼파라미터 범위 설정

결정 나무를 활용하여 XGBoost의 하이퍼파라미터 범위를 결정하겠습니다.

```
XGB_X = XGB_result.drop('score', axis = 1)
XGB_y = XGB_result['score']
model = DecisionTreeRegressor(max_depth = 5, random_state = 2022).fit(XGB_X, XGB_y)
result = text_to_rule_list(export_text(model, feature_names = list(XGB_X.columns)))
result = pd.DataFrame(result, columns = ["condition", "output"])

result['condition'] = result['condition'].apply(' & '.join)
result['output'] = result['output'].apply(extract_float)
result.sort_values(by = "output", inplace = True)

display(result.head(10))
```

XGBoost의 하이퍼파라미터 범위 설정 (계속)

결정 나무를 활용하여 XGBoost의 하이퍼파라미터 범위를 결정하겠습니다.

	condition	output
3	col_ratio > 0.38 & col_ratio > 0.36 & col_ratio <= 0.57 & learning_rate <= 0.04 & num_trees <= 23.50	8.13
29	max_depth > 5.50 & row_ratio <= 0.16 & row_ratio > 0.16 & col_ratio > 0.50 & num_trees > 23.50	26.85
15	learning_rate > 0.17 & row_ratio > 0.71 & learning_rate > 0.13 & learning_rate > 0.04 & num_trees <= 23.50	29.39
2	col_ratio <= 0.38 & col_ratio > 0.36 & col_ratio <= 0.57 & learning_rate <= 0.04 & num_trees <= 23.50	31.33
23	col_ratio > 0.50 & row_ratio > 0.73 & col_ratio > 0.50 & col_ratio <= 0.50 & num_trees > 23.50	35.87
27	learning_rate > 0.31 & learning_rate > 0.31 & row_ratio <= 0.16 & col_ratio > 0.50 & num_trees > 23.50	42.16
30	col_ratio <= 0.50 & row_ratio > 0.16 & row_ratio > 0.16 & col_ratio > 0.50 & num_trees > 23.50	72.85
9	num_leaves > 25.00 & row_ratio <= 0.32 & learning_rate <= 0.13 & learning_rate > 0.04 & num_trees <= 23.50	75.59
25	num_trees > 44.50 & learning_rate <= 0.31 & row_ratio <= 0.16 & col_ratio > 0.50 & num_trees > 23.50	90.46
12	row_ratio <= 0.54 & row_ratio <= 0.71 & learning_rate > 0.13 & learning_rate > 0.04 & num_trees <= 23.50	102.09

- 주로 등장하는 하이퍼파라미터는 num_trees(10회), row_ratio(8회), learning_rate(7회), col_ratio(7회)임
- max_depth와 num_leaves는 겨우 한 번만 등장함 → max_depth와 num_leaves는 튜닝하지 않음
- num_trees는 MAE가 가장 작은 10개의 잎 노드 모두에서 23.5를 기준으로 분지됐지만 모두 같은 값으로 분지돼 중요하지 않은 하이퍼파라미터라고 생각할 수 있으나, 모든 잎 노드에 등장했다는 것 자체만으로 분지에 중요한 역할을 한다고 볼 수 있음. 그렇지만 원래 num_trees의 탐색 공간인 {10, 11, 12, ⋯, 500}을 고려하면 23.5는 충분히 작은 수치이므로 튜닝 범위를 줄여도 될 것으로 판단됨
- row_ratio는 인덱스가 15와 23인 행을 제외하면 상대적으로 적은 값을 가짐을 알 수 있으나, 특별한 규칙이 보이지 않으므로 탐색 공간을 조금 줄이겠음
- col_ratio는 주로 0.5 부근에서 MAE가 작고 learning_rate는 특별한 패턴을 찾기 어려우므로 원래 범위로 튜닝하겠음

→

- num_trees: $\{10,20,30,40,50,\cdots,120\}$
- row_ratio: $\{x \mid 0.15 \leq x \leq 0.60\}$
- col_ratio: $\{x \mid 0.40 \leq x \leq 0.60\}$
- learning_rate: $\{x \mid 0.01 \leq x \leq 0.31\}$

LightGBM의 하이퍼파라미터 범위 설정

결정 나무를 활용하여 LightGBM의 하이퍼파라미터 범위를 결정하겠습니다.

```
LGB_X = LGB_result.drop('score', axis = 1)
LGB_y = LGB_result['score']
model = DecisionTreeRegressor(max_depth = 5, random_state = 2022).fit(LGB_X, LGB_y)
result = text_to_rule_list(export_text(model, feature_names = list(LGB_X.columns)))
result = pd.DataFrame(result, columns = ["condition", "output"])

result['condition'] = result['condition'].apply(' & '.join)
result['output'] = result['output'].apply(extract_float)
result.sort_values(by = "output", inplace = True)

display(result.head(10))
```

LightGBM의 하이퍼파라미터 범위 설정 (계속)

결정 나무를 활용하여 LightGBM의 하이퍼파라미터 범위를 결정하겠습니다.

	condition	output
0	max_depth <= 3.50 & num_leaves <= 172.50 & col_ratio <= 0.57 & learning_rate <= 0.04 & num_trees <= 27.50	0.02
22	learning_rate <= 0.16 & row_ratio <= 0.10 & row_ratio <= 0.16 & col_ratio > 0.39 & num_trees > 27.50	0.46
10	col_ratio > 0.78 & num_leaves <= 14.00 & learning_rate > 0.04 & num_trees <= 27.50	1.82
6	row_ratio <= 0.55 & col_ratio > 0.70 & col_ratio > 0.57 & learning_rate <= 0.04 & num_trees <= 27.50	3.09
1	max_depth > 3.50 & num_leaves <= 172.50 & col_ratio <= 0.57 & learning_rate <= 0.04 & num_trees <= 27.50	4.53
27	max_depth > 5.50 & row_ratio <= 0.16 & row_ratio > 0.16 & col_ratio > 0.39 & num_trees > 27.50	27.05
18	learning_rate > 0.22 & col_ratio > 0.32 & num_leaves <= 10.50 & col_ratio <= 0.39 & num_trees > 27.50	32.07
14	row_ratio > 0.68 & learning_rate > 0.16 & num_leaves > 14.00 & learning_rate > 0.04 & num_trees <= 27.50	52.96
25	num_trees > 44.50 & row_ratio > 0.10 & row_ratio <= 0.16 & col_ratio > 0.39 & num_trees > 27.50	81.59
11	row_ratio <= 0.35 & learning_rate <= 0.16 & num_leaves > 14.00 & learning_rate > 0.04 & num_trees <= 27.50	89.98

- 주로 등장하는 하이퍼파라미터는 num_trees(10회), col_ratio(7회), learning_rate(7회), row_ratio(6회), num_leaves(6회)임
- max_depth는 겨우 한 번만 등장함 → max_depth는 튜닝하지 않음
- num_trees는 상위 10개 잎 노드 모두에서 작은 값을 기준으로 분지됐으므로 num_trees의 탐색 공간을 줄이겠음
- col_ratio는 인덱스가 10과 6인 행을 제외하면 주로 0.3과 0.6 사이의 값을 가짐
- learning_rate는 인덱스가 18인 행을 제외하면 주로 0.2 이하의 값을 가짐
- row_ratio는 14번 행을 제외하면 주로 0.55 이하의 값을 가짐
- num_leaves는 14번 행과 11번 행을 제외하면 전부 이하(<=) 규칙이며 또 172.5 미만임

→

- num_trees: $\{10,20,30,40,50,\cdots,120\}$
- col_ratio: $\{x \mid 0.30 \leq x \leq 0.60\}$
- learning_rate: $\{x \mid 0.01 \leq x \leq 0.20\}$
- row_ratio: $\{x \mid 0.10 \leq x \leq 0.55\}$
- num_leaves: $\{10,20,30,40,50,\cdots,120,130,140,150,160,170\}$

2. MyAutoML2

3 시스템 구현

생성자 정의
(1) 인자 정의

클래스 생성자의 인자를 정의합니다.

클래스 생성자 입력 정의

```
class MyAutoML2:
    ## 생성자
    def __init__(
        self,
        seed=None,
        cv=5,
        scoring="mean_absolute_error",
        summarize_scoring="mean",
        num_iter=1000,
        num_base_models=100,
        fit_stacking = True
    ):

```

생성자 정의
(2) 인자값 검사

각 인자가 잘못 입력되면 적절한 문구와 함께 ValueError를 띄우고, 정상적으로 입력되면 self의 속성으로 저장하겠습니다.

인자값 검사: seed

```
14      # self.seed 정의
15      if (type(seed) != int) and (seed is not None):
16          raise ValueError("seed는 int 형 혹은 None이어야 합니다.")
17      self.seed = seed
18
```

인자값 검사: cv

```
19      # self.cv 정의
20      if type(cv) != int:
21          raise ValueError("cv는 int 형이어야 합니다.")
22      if cv < 2:
23          raise ValueError("cv는 2보다는 커야 합니다.")
24      self.cv = cv
25
```

생성자 정의
(2) 인자값 검사 (계속)

각 인자가 잘못 입력되면 적절한 문구와 함께 ValueError를 띄우고, 정상적으로 입력되면 self의 속성으로 저장하겠습니다.

인자값 검사: scoring

```
26      # self.scoring 정의
27      scoring_dict = {
28          "mean_absolute_error": mean_absolute_error,
29          "mean_squared_error": mean_squared_error,
30
31      }
32
33      if scoring not in scoring_dict.keys():
34              msg = "scoring은 {}중 하나여야 합니다.".format(scoring_dict.keys())
35              raise ValueError(msg)
36      self.scoring = scoring_dict[scoring]
37
```

생성자 정의
(2) 인자값 검사 (계속)

각 인자가 잘못 입력되면 적절한 문구와 함께 ValueError를 띄우고, 정상적으로 입력되면 self의 속성으로 저장하겠습니다.

인자값 검사: num_iter

```
46      # self.num_iter 정의
47      if type(num_iter) != int:
48          raise ValueError("num_iter는 int 자료형이어야 합니다.")
49      elif num_iter <= 0:
50          raise ValueError("num_iter는 0보다 커야 합니다.")
51      self.num_iter = num_iter
52
```

- **라인 47~51:** num_iter는 자연수여야 하므로 int 자료형이 아니거나 0보다 크지 않으면 오류를 발생시킵니다.

생성자 정의
(2) 인자값 검사 (계속)

각 인자가 잘못 입력되면 적절한 문구와 함께 ValueError를 띄우고, 정상적으로 입력되면 self의 속성으로 저장하겠습니다.

인자값 검사: num_base_models

```
53    # self.num_base_models 정의
54    if type(num_base_models) != int:
55        raise ValueError("num_base_models는 int 자료형이어야 합니다.")
56    elif num_base_models <= 0:
57        raise ValueError("num_base_models는 0보다 커야 합니다.")
58    elif num_base_models > num_iter:
59        raise ValueError("num_base_models는 num_iter보다 작거나 같아야 합니다.")
60    self.num_base_models = num_base_models
61
```

- **라인 58~59:** 학습한 전체 모델보다 앙상블 모델의 기본 모델이 더 많을 수 없으므로 num_base_models가 num_iter보다 크면 오류를 발생시킵니다.

인자값 검사: fit_stacking

```
62    # self.fit_stacking 정의
63    if type(fit_stacking) != bool:
64        raise ValueError("fit_stacking는 bool 자료형이어야 합니다.")
65    self.fit_stacking = fit_stacking
66
```

sampling 메서드

sampling 메서드는 하이퍼파라미터가 설정된 모델을 샘플링합니다.

sampling 메서드

```
67  ## 샘플링 정의
68  def sampling(self):
69      model_choice = np.random.multinomial(1, [0.2, 0.3, 0.5]).argmax()
70      if model_choice == 0: # 랜덤 포레스트
71          max_depth = np.random.choice(range(3, 9))
72          col_ratio = round(uniform(0.44, 0.82-0.44).rvs(), 3)
73          model = RFR(max_depth=max_depth,
74                      max_features=col_ratio)
75
76      elif model_choice == 1: # XGBoost
77          num_trees = np.random.choice(range(10, 130, 10))
...
85
86      elif model_choice == 2: # LightGBM
...
98      return model
99
```

- **라인 69:** 랜덤 포레스트, XGBoost, LightGBM을 각각 0.2, 0.3, 0.5의 확률로 선택합니다. 즉, 각각이 선택될 확률이 [0.2, 0.3, 0.5]인 다항 분포에서 하나의 값을 샘플링한 값을 model_choice에 저장합니다. 이 값이 0이라면 랜덤 포레스트가, 1이라면 XGBoost가, 2라면 LightGBM이 선택됐다고 간주합니다.
- **라인 71:** np.random.choice를 이용해 [3, 4, 5, 6, 7, 8, 9] 중 하나를 골라 max_depth에 저장합니다.
- **라인 72:** 0.44와 0.82 사이의 값을 임의로 선택한 뒤 소수 셋째짜리까지 반올림한 값을 col_ratio에 저장합니다.
- **라인 77:** np.random.choice를 이용해 [10, 20, 30, ⋯, 120] 중 하나를 골라 num_trees에 저장합니다.

fit 메서드
(1) 입력값 변환

특징 벡터 X와 라벨 y가 입력됐을 때 랜덤 서치를 바탕으로 최적의 모델과 하이퍼파라미터를 탐색하는 fit 메서드를 구현해보겠습니다. 먼저, 입력된 자료형에 따라 다른 코드를 사용하는 것을 방지하기 위해 X와 y를 ndarray로 변환합니다.

fit 메서드: 입력값 변환

```
100   ## fit 메서드
101   def fit(self, X, y):
102       # X, y 포맷 변경
103       if isinstance(X, pd.DataFrame):
104           X = X.values
105       elif isinstance(X, list) or isinstance(X, tuple):
106           X = np.array(X)
107       if isinstance(y, pd.Series):
108           y = y.values
109       elif isinstance(y, list) or isinstance(y, tuple):
110           y = np.array(y)
111
```

fit 메서드
(2) 랜덤 서치

랜덤 서치를 사용해 최적의 모델을 찾겠습니다.

```
113     best_score = np.inf
114     self.leaderboard = []
115     self.model_list = []
116     for _ in range(self.num_iter):
117         while True:
118             model = self.sampling()
119             if model not in self.model_list:
120                 break
121
122         self.model_list.append(model)
123         kf = KFold(n_splits=self.cv, shuffle=True, random_state=self.seed)
124         fold_score_list = []
125         for train_index, test_index in kf.split(X):
126             X_train, X_test = X[train_index], X[test_index]
127             y_train, y_test = y[train_index], y[test_index]
128             model.fit(X_train, y_train)
129             y_pred = model.predict(X_test)
130             fold_score = self.scoring(y_test, y_pred)
131             fold_score_list.append(fold_score)
132         # 현재까지 찾은 최고의 해 및 리더보드 업데이트
133         score = self.summarize_scoring(fold_score_list)
134         if best_score > score:
135             best_score = score
136             best_model = model
137         self.leaderboard.append([str(model).replace('\n', ''), score])
138
139     self.leaderboard = pd.DataFrame(self.leaderboard,
140                                     columns=["모델", "점수"])
```

- **라인 116:** 사용자가 입력한 num_iter만큼 다음 내용을 반복합니다.
- **라인 117~120:** sampling 메서드를 사용해 하이퍼파라미터가 설정된 모델 인스턴스를 샘플링합니다. 단, 동일한 모델을 재평가하는 것을 막기 위해 self.model_list에 샘플링한 model이 없을 때까지 샘플링을 반복합니다.
- **라인 128:** 모델 인스턴스를 학습합니다. 같은 인스턴스더라도 fit을 수행할 때마다 새로 학습되므로 이전 학습 정보가 누적되진 않습니다.
- **라인 134:** 대상 모델이 회귀 모델이므로 점수가 작을수록 좋습니다.

fit 메서드
(3) 스태킹 앙상블 학습

랜덤 서치를 이용해 탐색한 결과를 바탕으로 스태킹 모델을 학습합니다.

```
144     if self.fit_stacking:
145         top_model_idx_list=self.leaderboard.sort_values(by="점수").index[:self.num_base_models]
146         top_model_list = [self.model_list[idx] for idx in top_model_idx_list]
147         top_model_tuple_list = [("model_{}".format(idx+1, model), model) for idx, model in enumerate(top_model_list)]
148 
149         stacking_model = StackingRegressor(top_model_tuple_list)
```

- **라인 144:** fit_stacking이 True이면 스태킹 앙상블을 학습합니다.
- **라인 145:** leaderboard를 점수를 기준으로 오름차순으로 정렬한 인덱스에서 self.num_base_models 개만큼의 모델 인덱스를 가져옵니다. 즉, 점수가 큰 모델의 인덱스를 top_model_idx_list에 저장합니다.
- **라인 146:** top_model_idx_list의 요소를 바탕으로 self.model_list에서 모델을 가져와 top_model_list에 저장합니다.
- **라인 147:** StackingRegressor의 입력으로 사용할 top_model_tuple_list를 정의합니다. 이 리스트의 i번째 요소는 0번째 요소가 "model_i"이고 1번째 요소가 성능 기준 상위 i번째 모델 인스턴스인 튜플입니다.
- **라인 149:** 스태킹 앙상블 모델 인스턴스를 정의합니다.

fit 메서드
(4) 스태킹 앙상블 평가

학습한 스태킹 모델을 평가합니다.

```
151        # 스태킹 모델 평가
152        fold_score_list = []
153        for train_index, test_index in kf.split(X):
154                X_train, X_test = X[train_index], X[test_index]
155                y_train, y_test = y[train_index], y[test_index]
156                stacking_model.fit(X_train, y_train)
157                y_pred = stacking_model.predict(X_test)
158                fold_score = self.scoring(y_test, y_pred)
159                fold_score_list.append(fold_score)
160
161        score = self.summarize_scoring(fold_score_list)
162        new_row = [str(stacking_model).replace('\n', ''), score]
163        new_row = pd.Series(new_row, index = self.leaderboard.columns)
164        self.leaderboard = self.leaderboard.append(new_row, ignore_index=True)
165
166        if best_score > score:
167                best_model = stacking_model
168
169    self.model = best_model.fit(X, y)
```

- **라인 151~161:** 스태킹 앙상블 모델도 다른 모델과 마찬가지로 평가합니다.
- **라인 162~163:** self.leaderboard의 맨 마지막 행으로 추가할 수 있도록 스태킹 앙상블에 관한 리더보드를 시리즈로 변환합니다. 데이터프레임의 append는 새로운 행(시리즈)을 추가하는 메서드로, 시리즈의 인덱스와 데이터프레임의 열 인덱스가 일치해야 정상적으로 추가됩니다.
- **라인 166~167:** 만약 스태킹 모델의 점수가 현재까지의 최고 점수보다 작다면 best_model을 업데이트합니다. 최고 점수는 더 이상 사용하지 않으므로 따로 업데이트하지 않았습니다.

predict와 show_leaderboard 메서드

fit 메서드에서 저장한 self.model과 self.leaderboard를 사용해 각각 predict 메서드와 show_leaderboard 메서드를 구현합니다.

predict 메서드

```
171  ## predict 메서드
172  def predict(self, X):
173          return self.model.predict(X)
174
```

show_leaderboard 메서드

```
175  ## show_leaderboard 메서드
176  def show_leaderboard(self):
177          return self.leaderboard
```

2. MyAutoML2

4 시스템 활용

활용 예제 1

데이터 불러오기

```
1  # 데이터 불러오기
2  df = pd.read_csv("../../data/regression/laser.csv")
3  X = df.drop('y', axis = 1)
4  y = df['y']
```

머신러닝 자동화

```
1  aml = MyAutoML2()
2  aml.fit(X, y)
3  result = aml.show_leaderboard()
4  display(result.sort_values(by = "점수", ascending = True))
```

	모델	점수
607	RandomForestRegressor(max_depth=8, max_feature...	3.545854
0	RandomForestRegressor(max_depth=8, max_feature...	3.597390
723	RandomForestRegressor(max_depth=8, max_feature...	3.601368
171	RandomForestRegressor(max_depth=8, max_feature...	3.627731
634	RandomForestRegressor(max_depth=8, max_feature...	3.649754
...	...	...
405	LGBMRegressor(colsample_bytree=0.375, learning...	34.252825
160	XGBRegressor(base_score=0.5, booster='gbtree',...	38.764955
161	XGBRegressor(base_score=0.5, booster='gbtree',...	39.856808
779	XGBRegressor(base_score=0.5, booster='gbtree',...	42.470008
356	XGBRegressor(base_score=0.5, booster='gbtree',...	43.973228

1001 rows × 2 columns

활용 예제 1
(계속)

스태킹 모델 출력

```
1 display(result.iloc[-1])
```

```
모델    StackingRegressor(estimators=[('model_1',    ...
점수                              4.907067
Name: 1000, dtype: object
```