3. MyAutoML3

1 문제 정의 및 클래스 설계

탐색 공간: 모델

탐색 공간이 지나치게 넓어지는 것을 방지하기 위해 모델은 신경망으로 고정합니다.

하이퍼파라미터	탐색 분포
h1	$Poisson(15)$
h2	$Poisson(3)$
h3	$\begin{cases} Poisson(2), & \text{if } h_2 > 0 \\ 0, & \text{if } h_2 = 0 \end{cases}$
h4	$\begin{cases} Poisson(1), & \text{if } h_3 > 0 \\ 0, & \text{if } h_3 = 0 \end{cases}$
max_iter	$LogUniform(100, 10000)$
learning_rate_init	$LogUniform(0.0001, 0.1)$

<신경망 구조>

- h1~h4는 각각 첫 번째부터 네 번째 은닉 노드 수를 나타냄
- 즉, (h1, h2, h3, h4)는 MLPClassifier 클래스의 은닉층 구조를 나타내는 인자 hidden_layer_sizes임
- $Poisson$은 푸아송 분포를 나타내며, h3과 h4는 각각 h2와 h3이 0이면 0을 갖도록 정의함

<최대 에티레이션 횟수>

- max_iter는 최대 이터레이션 횟수를 나타냄
- LogUniform은 로그 유니폼 분포를 나타냄
- 최대 이터레이션 횟수는 자연수로 정의돼야 하므로 샘플링한 뒤 int 자료형으로 변환해야 함

<learning_rate_init>

- learning_rate_init은 초기에 설정한 학습률을 나타냄
- 그런데 이터레이션에 따른 학습률 변화를 설정하는 인자인 learning_rate의 기본값은 ‘constant’임
- learning_rate는 기본값을 사용하므로 learning_rate_init 자체가 학습률이라고 봐도 무방함

탐색 공간: 전처리

고려하는 전처리는 스케일링과 재샘플링이며, 두 방법에 대한 탐색 공간은 다음과 같이 정의합니다.

(s_1, s_2, s_3)	스케일링 (s_1)	재샘플링 (s_2, s_3)
(0, 0, 0)	하지 않음 (0)	하지 않음 (0, 0)
(0, 1, 0)	하지 않음 (0)	SMOTE (1, 0)
(0, 0, 1)	하지 않음 (0)	NearMiss (0, 1)
(1, 0, 0)	최소-최대 스케일링 (1)	하지 않음 (0, 0)
(1, 1, 0)	최소-최대 스케일링 (1)	SMOTE (1, 0)
(1, 0, 1)	최소-최대 스케일링 (1)	NearMiss (0, 1)

메타 모델의 입력으로 사용할 수 있도록 재샘플링과 관련된 변수를 더미화했음

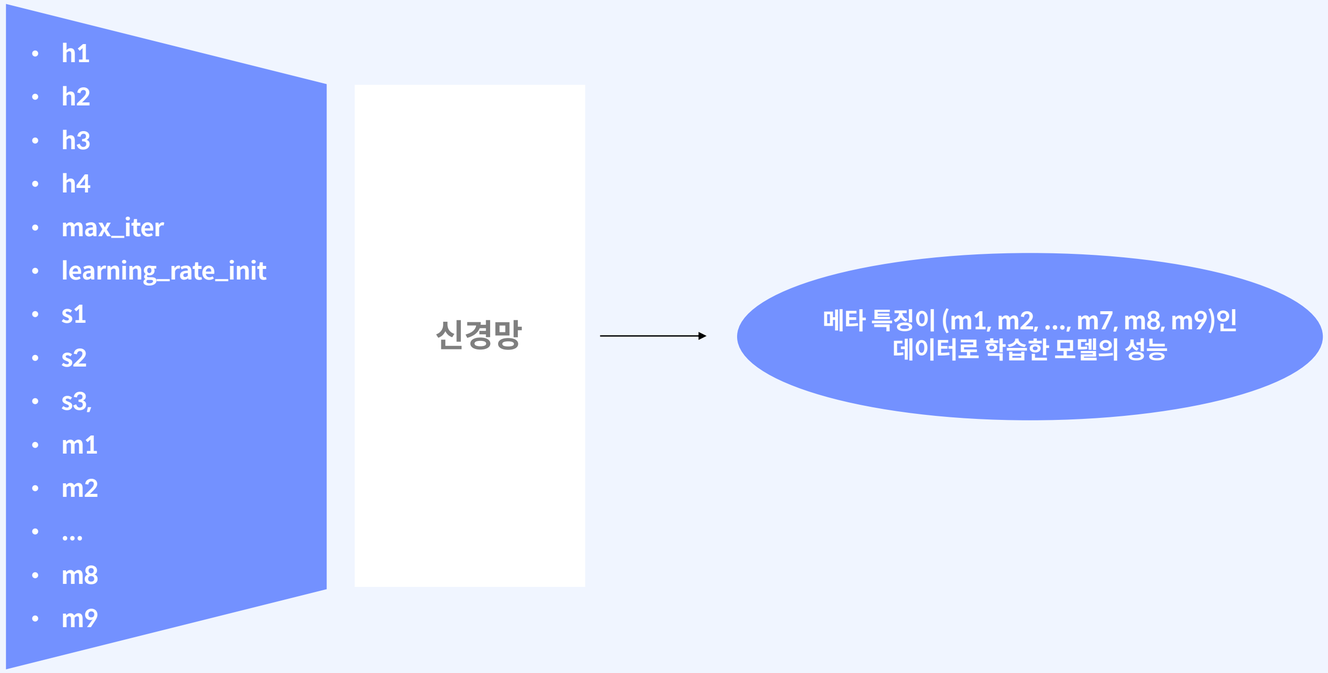
메타 모델

메타 모델은 앞서 제시한 탐색 공간을 랜덤 서치한 결과로 학습하고 베이지안 최적화의 초기 해를 생성하는 데 사용합니다.

MyAutoML3의 메타 특징 목록

메타 특징	설명
m1	샘플 개수
m2	특징 개수
m3	클래스 불균형 비율
m4	샘플 대비 특징 비율
m5	정수형 특징 비율
m6	실수형 특징 비율
m7	특징별 범위의 최댓값
m8	특징별 범위의 최솟값
m9	모든 값이 양수인 특징 비율

메타 모델의 구성



베이지안 최적화의 구성

베이지안 최적화를 이용해 하이퍼 파라미터를 튜닝합니다.

구분	값
대체 모델	가우시안 프로세스
커널	Matern 커널
초기 후보 해 개수	num_candidate_init_samples
초기 해 개수	num_init_samples
샘플러	모델과 전처리의 탐색 공간을 참고하여 설계
획득 함수	개선 기대

클래스 설계
:인자 정의

MyAutoml_3은 다음과 같은 메서드를 파이썬 클래스로 개발합니다.

인자	설명	기본값
seed	시드	2022
cv	폴드 개수	5
scoring	분류 평가 척도	“accuracy”
summarize_score	평가 결과 요약 방법	“mean”
num_iter	베이지안 최적화의 이터레이션 횟수	1,000
num_candidate	각 이터레이션마다 생성할 후보 해의 개수	100
num_init	초기 해 개수	10
num_sample	각 이터레이션마다 선택할 해의 개수	1

클래스 설계
:메서드 정의

MyAutoml_3은 다음과 같은 메서드를 갖습니다.

메서드	설명
_extract_meta_features	메타 특징을 추출합니다.
_sampling	베이지안 최적화의 샘플러입니다.
_solution_evaluate	해를 평가합니다. 즉, 입력된 신경망 및 전처리 하이퍼파라미터에 부합하는 모델을 학습하고 평가한 결과를 반환합니다.
_EI	베이지안 최적화의 획득 함수인 개선 기대 함수입니다.
fit	메타 모델과 베이지안 최적화를 사용해 최적의 신경망의 하이퍼파라미터와 전처리 하이퍼파라미터를 탐색합니다.
show_leaderboard	학습한 결과를 하이퍼파라미터와 성능 지표로 구성된 리더보드로 반환합니다.
predict	학습한 모델 가운데 가장 좋은 성능을 갖는 모델로 예측합니다.

3. MyAutoML3

2 메타 모델 학습

모듈 불러오기

메타 모델을 학습하는데 필요한 모듈을 불러옵니다.

모듈 불러오기

```
import pandas as pd
import numpy as np
from scipy.stats import poisson, loguniform
from sklearn.neural_network import MLPClassifier as MLPC
from sklearn.neural_network import MLPRegressor as MLPR
from sklearn.model_selection import *
from sklearn.metrics import *
from sklearn.preprocessing import MinMaxScaler
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
import pickle
import warnings
from sklearn.gaussian_process import GaussianProcessRegressor as GPR
from sklearn.gaussian_process.kernels import Matern
from scipy.stats import norm
from tqdm import tqdm
warnings.filterwarnings("ignore")
```

메타 특징 추출 함수 정의

데이터에서 메타 특징을 추출하는 extract_meta_features 함수를 다음과 같이 정의합니다.

메타 특징 추출 함수 정의

```
def extract_meta_features(X, y):
    m1, m2 = X.shape # 샘플 개수, 특징 개수
    y_vc = y.value_counts() # 라벨 분포
    m3 = y_vc.iloc[0] / y_vc.iloc[-1] # 클래스 불균형 비율
    m4 = m2 / m1 # 샘플 대비 특징 비율
    m5 = sum(X.dtypes == float) / m2 # 실수형 특징 비율
    m6 = sum(X.dtypes == int) / m2 # 정수형 특징 비율
    m7 = (X.max() - X.min()).max() # 특징별 범위의 최댓값
    m8 = (X.max() - X.min()).min() # 특징별 범위의 최솟값
    m9 = sum(X.min() > 0) / m2 # 모든 값이 양수인 비율

    return [m1, m2, m3, m4, m5, m6, m7, m8, m9]
```

샘플링 함수 정의

신경망의 하이퍼파라미터와 전처리 관련 하이퍼파라미터를 샘플링하는 함수 sampling을 다음과 같이 정의합니다.

샘플링 함수 정의

```
1   def sampling():
2       h1 = poisson(15).rvs()
3       h2 = poisson(3).rvs()
4       h3 = poisson(2).rvs() if h2 > 0 else 0
5       h4 = poisson(1).rvs() if h3 > 0 else 0
6       max_iter = int(loguniform(100, 10000).rvs())
7       learning_rate_init = loguniform(0.0001, 0.1).rvs()
8       s1 = np.random.choice([0, 1])
9       s2 = np.random.choice([0, 1])
10      s3 = np.random.choice([0, 1]) if s2 == 0 else 0
11
12      return [h1, h2, h3, h4, max_iter, learning_rate_init, s1, s2, s3]
```

- **라인 1:** possion 클래스의 rvs 메서드를 이용하여 파라미터가 15인 푸아송 분포에서 샘플링한 값을 h1에 저장합니다.
- **라인 4:** h2가 0보다 크다면 h3에 파라미터가 2인 푸아송 분포에서 샘플링한 값을 h3에 저장합니다. 이처럼 조건에 따라 변수를 정의할 때 if-else문을 한 줄에 쓸 수 있습니다.
- **라인 5:** max_iter를 로그 유니폼 분포에서 샘플링합니다. 단, 값이 정수가 아닐 수 있으므로 int 형으로 바꿉니다.
- **라인 8~9:** s1과 s2를 0과 1중 하나로 임의로 설정합니다.
- **라인 10~11:** s2와 s3 모두 1이 될 수 없으므로, s2가 0이면 s3를 0과 1중 하나로 설정하고, s2가 1이면 s3를 0으로 설정합니다.

모델 평가 함수 정의

모델 인스턴스, 데이터, 전처리 관련 하이퍼파라미터를 k-겹 교차 검증 방식으로 평가하는 model_test 함수를 다음과 같이 정의합니다.

모델 평가 함수 정의

```
1 def model_test(model, X, y, s1, s2, s3):
2   kf = KFold(n_splits = 5, shuffle = True, random_state = 2022)
3   # 데이터 타입 변경 (pandas -> numpy)
4   if isinstance(X, pd.DataFrame):
5         X = X.values
6   if isinstance(y, pd.Series):
7         y = y.values
8
9   # 모델 학습
10  score = 0
11  for train_index, test_index in kf.split(X):
12    X_train, X_test = X[train_index], X[test_index]
13    y_train, y_test = y[train_index], y[test_index]
14
15    if s1 == 1:
16      scaler = MinMaxScaler().fit(X_train)
17      X_train = scaler.transform(X_train)
18      X_test = scaler.transform(X_test)
19
20    if s2 == 1:
21      X_train, y_train = SMOTE(k_neighbors = 3,
22                               random_state=2022).fit_resample(X_train, y_train)
23    elif s3 == 1:
24      X_train, y_train = NearMiss().fit_resample(X_train, y_train)
25
26    model.fit(X_train, y_train)
27    y_pred = model.predict(X_test)
28    score += f1_score(y_test, y_pred) / 5
29
30  return score
```

- **라인 3~7:** X와 y를 모두 ndarray로 변환합니다.
- **라인 15~18:** s1이 1이면, X_train과 X_test를 스케일링합니다.
- **라인 20~22:** s2가 1이면 SMOTE를 이용하여 X_Train과 y_train을 오버샘플링합니다.
- **라인 23~24:** s3가 1이면, NearMiss를 이용하여 X_Train과 y_train을 언더샘플링합니다.

학습 데이터 불러오기

메타 모델을 학습할 데이터를 불러옵니다.

학습 데이터 불러오기

```
meta_file_name_list = [
    "shuttle-c2-vs-c4",
    "iris0",
    "glass-0-1-6_vs_5",
    "glass-0-1-6_vs_2",
    "sonar",
    "glass0",
    "glass-0-1-2-3_vs_4-5-6",
    "glass1",
    "glass2",
    "glass5",
    "glass6",
    "new-thyroid1",
    "ecoli-0_vs_1",
    "spectfheart",
    "heart",
    "haberman",
    "bupa",
    "ionosphere",
    "monk-2",
    "page-blocks-1-3_vs_4",
    "wdbc",
    "vehicle0",
    "vehicle2",
    "vehicle3",
    "yeast-1-2-8-9_vs_7",
    "vowel0"
]
```

```
meta_data_list = []
for file_name in meta_file_name_list:
    df = pd.read_csv("../../data/classification/{}.csv".format(file_name))
    X = df.drop('y', axis = 1)
    y = df['y']
    meta_features = extract_meta_features(X, y)
    meta_data_list.append((X, y, meta_features))
```

메타 모델 학습 데이터 생성

데이터 선택, 메타 특징 추출, 샘플링, 평가를 1만 번 반복하여 메타 모델을 학습할 데이터를 생성하겠습니다.

메타 모델 학습 데이터 생성

```
1   meta_X = []
2   meta_y = []
3
4   for _ in tqdm(range(10000)):
5       idx = np.random.choice(range(len(meta_file_name_list)))
6       X, y, meta_features = meta_data_list[idx]
7       h1, h2, h3, h4, max_iter, learning_rate_init, s1, s2, s3 = sampling()
8
9       if h2 == 0:
10          hidden_layer_sizes = (h1, )
11      elif h3 == 0:
12          hidden_layer_sizes = (h1, h2)
13      elif h4 == 0:
14          hidden_layer_sizes = (h1, h2, h3)
15      else:
16          hidden_layer_sizes = (h1, h2, h3, h4)
17
18      model = MLPC(hidden_layer_sizes = hidden_layer_sizes,
19                   max_iter = max_iter,
20                   learning_rate_init = learning_rate_init,
21                   random_state = 2022)
22
23      score = model_test(model, X, y, s1, s2, s3)
24      record = [h1, h2, h3, h4, max_iter, learning_rate_init, s1, s2, s3]
25      record += meta_features
26
27      meta_X.append(record)
28      meta_y.append(score)
```

- **라인 1~2:** 메타 데이터의 특징과 라벨인 meta_X와 meta_y를 빈 리스트로 초기화합니다.
- **라인 5~6:** meta_data_list에서 데이터를 임의로 선택합니다. np.random.choice는 1차원 배열만 입력받으므로 인덱스를 기준으로 선택했습니다.
- **라인 7:** sampling 함수를 사용하여 신경망 및 전처리 하이퍼파라미터를 샘플링합니다.
- **라인 9~16:** h1, h2, h3, h4를 MLPClassifier의 인자인 hidden_layer_sizes로 사용할 수 있게 변환합니다. 특정 은닉층의 노드 개수가 0이라면 이전 층까지가 은닉층이 되도록 값을 설정했습니다.
- **라인 18~21:** 샘플링한 하이퍼파라미터를 갖는 신경망 인스턴스 model을 정의합니다.
- **라인 23:** model_test 함수를 이용해 샘플링한 하이퍼파라미터를 평가합니다.
- **라인 24~27:** 샘플링한 하이퍼파라미터와 메타 특징을 record에 담고, record를 meta_X에 추가합니다.

메타 모델 학습 데이터 생성 (계속)

리스트인 meta_X와 meta_y를 열 방향으로 병합하여 데이터프레임으로 바꾸겠습니다.

메타 모델 학습 데이터 생성

```
1  meta_X_cols = ["h1", "h2", "h3", "h4",
2                 "max_iter", "learning_rate_init",
3                 "s1", "s2", "s3",
4                 "m1", "m2", "m3", "m4", "m5", "m6", "m7", "m8", "m9"]
5  meta_X = pd.DataFrame(meta_X, columns = meta_X_cols)
6  meta_y = pd.Series(meta_y)
7  meta_y.name = "y"
8  meta_df = pd.concat([meta_X, meta_y], axis = 1)
9  meta_df.to_csv("MyAutoML3_메타모델_학습데이터.csv", index = False)
```

- **라인 1~5:** meta_X를 데이터프레임으로 변환합니다.
- **라인 6~7:** meta_y를 시리즈로 변환합니다. 이때, name 속성을 "y"로 정의한 이유는 데이터프레임과 시리즈가 열 방향으로 병합되면 시리즈에 해당하는 칼럼 이름이 name 속성이 되기 때문입니다.
- **라인 8~9:** meta_X와 meta_y를 열 방향으로 병합해 meta_df를 생성합니다. 또한, 메타 모델 학습 데이터를 만드는 시간이 오래 걸리므로 meta_df를 파일로 저장합니다.

메타 모델 학습

meta_X와 meta_y를 이용해 메타 모델을 학습하겠습니다. 하이퍼파라미터는 랜덤 서치를 이용해 튜닝합니다.

메타 모델 학습

```
1   best_score = np.inf
2   for _ in range(1000):
3       h1, h2, h3, h4, max_iter, learning_rate_init, s1, s2, s3 = sampling()
4       if h2 == 0:
5           hidden_layer_sizes = (h1,)
6       elif h3 == 0:
7           hidden_layer_sizes = (h1, h2)
8       elif h4 == 0:
9           hidden_layer_sizes = (h1, h2, h3)
10      else:
11          hidden_layer_sizes = (h1, h2, h3, h4)
12      model = MLPR(
13                          hidden_layer_sizes=hidden_layer_sizes,
14                          max_iter=max_iter,
15                          learning_rate_init=learning_rate_init,
16                          random_state=2022,
17                      )
18
19      score = -cross_val_score(
20                          model, meta_X, meta_y, cv=5, scoring="neg_mean_absolute_error"
21                              ).mean()
22      if score < best_score:
23          best_score = score
24          best_model = model
25  best_model.fit(meta_X, meta_y)
```

- **라인 1:** best_score를 무한대로 초기화합니다. 메타 모델은 메타 특징과 하이퍼파라미터가 입력이고 그때의 성능 지표인 F1 점수가 출력인 회귀 모델이므로 성능 지표가 작을수록 좋습니다.
- **라인 19~21:** cross_val_score 함수를 이용해 5-겹 교차 검증을 수행합니다. scoring 인자에 "neg_mean_absolute_error"를 입력했으므로 전체 결과에 마이너스 부호를 붙여 다시 양수로 바꿨습니다.
- **라인 25:** 성능이 가장 좋았던 모델을 전체 데이터로 재학습합니다.

```
MLPRegressor(hidden_layer_sizes=(18, 5, 2, 1),
        learning_rate_init=0.0007337677668986755, max_iter=250,
        random_state=2022)
```

메타 모델 저장

학습한 메타 모델을 필요할 때마다 사용할 수 있도록 pickle 모듈을 이용해 저장하겠습니다.

메타 모델 저장

```
with open("MyAutoML3_meta_model.pckl", "wb") as f:
    pickle.dump(best_model, f)
```

3. MyAutoML3

3 시스템 구현

생성자 정의
(1) 인자 정의

클래스 생성자의 인자를 정의합니다.

클래스 생성자 입력 정의

```
class MyAutoML3:
    ## 생성자
    def __init__(
        self,
        seed=None,
        cv=5,
        scoring="f1",
        summarize_scoring="mean",
        num_iter=1000,
        num_candidate=100,
        num_init=10,
        num_sample=1,
    ):

```

생성자 정의
(2) 인자값 검사

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

인자값 검사: seed

```
15      # self.seed 정의
16      if (type(seed) != int) and (seed is not None):
17          raise ValueError("seed는 int 형 혹은 None이어야 합니다.")
18      self.seed = seed
19
```

인자값 검사: cv

```
20      # self.cv 정의
21      if type(cv) != int:
22          raise ValueError("cv는 int 형이어야 합니다.")
23      if cv < 2:
24          raise ValueError("cv는 2보다는 커야 합니다.")
25      self.cv = cv
26
```

생성자 정의
(2) 인자값 검사 (계속)

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

인자값 검사: scoring

```
27    # self.scoring 정의
28    scoring_dict = {
29        "accuracy": accuracy_score,
30        "precision": precision_score,
31        "recall": recall_score,
32        "f1": f1_score,
33    }
34
35    if scoring not in scoring_dict.keys():
36        msg = "scoring은 {}중 하나여야 합니다.".format(scoring_dict.keys())
37        raise ValueError(msg)
38    self.scoring = scoring_dict[scoring]
39
```

- **라인 28~33:** 이진 분류만 대상으로 하는 시스템이므로 평가 함수로 accuracy_score, precision_score, recall_score, f1_score만 사용합니다.

생성자 정의
(2) 인자값 검사 (계속)

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

인자값 검사: summarize_scoring

```
40      # self.summarize_scoring 정의
41      summarize_scoring_dict = {"mean": np.mean, "max": np.max, "min": np.min}
42
43      if summarize_scoring not in ["mean", "max", "min"]:
44          msg = "summarize_scoring는 {'mean', 'max', 'min'}중 하나여야 합니다."
45          raise ValueError(msg)
46      self.summarize_scoring = summarize_scoring_dict[summarize_scoring]
47
```

인자값 검사: num_iter

```
48      # self.num_iter 정의
49      if type(num_iter) != int:
50          raise ValueError("num_iter는 int 자료형이어야 합니다.")
51      elif num_iter <= 0:
52          raise ValueError("num_iter는 0보다 커야 합니다.")
53      self.num_iter = num_iter
54
```

생성자 정의
(2) 인자값 검사 (계속)

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

인자값 검사: num_candidate

```
55      # self.num_candidate 정의
56      if type(num_candidate) != int:
57          raise ValueError("num_candidate는 int 자료형이어야 합니다.")
58      elif num_candidate <= 0:
59          raise ValueError("num_candidate는 0보다 커야 합니다.")
60      self.num_candidate = num_candidate
61
```

인자값 검사: num_init

```
62      # self.num_init 정의
63      if type(num_init) != int:
64          raise ValueError("num_init은 int 자료형이어야 합니다.")
65      elif num_init <= 0:
66          raise ValueError("num_init은 0보다 커야 합니다.")
67      self.num_init = num_init
68
```

생성자 정의
(2) 인자값 검사 (계속)

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

인자값 검사: num_sample

```
69      # self.num_sample 정의
70      if type(num_sample) != int:
71          raise ValueError("num_sample은 int 자료형이어야 합니다.")
72      elif num_sample <= 0:
73          raise ValueError("num_sample은 0보다 커야 합니다.")
74      elif num_sample > num_candidate:
75          raise ValueError("num_sample은 num_candidate보다 작아야 합니다.")
76      self.num_sample = num_sample
77
```

생성자 정의
(3) 메타 불러오기

생성자를 호출할 때 이전에 학습해서 저장했던 메타 모델도 불러오겠습니다.

메타 불러오기

```
78      # self.meta_model 정의
79      with open("MyAutoML3_meta_model.pckl", "rb") as f:
80          self.meta_model = pickle.load(f)
81
```

- **라인 79~80:** "MyAutoML3_meta_model.pckl"에 저장된 메타 모델을 피클 모듈을 이용해 불러와 self.meta_model에 저장합니다.

_extract_meta_features 메서드 정의

_extract_meta_features 메서드는 다음과 같이 정의합니다. 메타 모델을 학습할 때 사용했던 코드와 거의 같습니다.

_extract_meta_features 메서드 정의

```
82  ## _extract_meta_features 메서드
83  def _extract_meta_features(self, X, y):
84      m1, m2 = X.shape  # 샘플 개수, 특징 개수
85      y_vc = y.value_counts()  # 라벨 분포
86      m3 = y_vc.iloc[0] / y_vc.iloc[-1]  # 클래스 불균형 비율
87      m4 = m2 / m1  # 샘플 대비 특징 비율
88      m5 = sum(X.dtypes == float) / m2  # 정수형 특징 비율
89      m6 = sum(X.dtypes == int) / m2  # 실수형 특징 비율
90      m7 = (X.max() - X.min()).max()  # 특징별 범위의 최댓값
91      m8 = (X.max() - X.min()).min()  # 특징별 범위의 최솟값
92      m9 = sum(X.min() > 0) / m2  # 모든 값이 양수인 비율
93
94      return [m1, m2, m3, m4, m5, m6, m7, m8, m9]
95
```

_sampling 메서드 정의

_sampling 메서드를 다음과 같이 정의합니다. 메타 모델을 학습할 때 사용했던 코드와 거의 같습니다.

_extract_meta_features 메서드 정의

```
96  ## _sampling 메서드
97  def _sampling(self):
98      h1 = poisson(15).rvs()
99      h2 = poisson(3).rvs()
100     h3 = poisson(2).rvs() if h2 > 0 else 0
101     h4 = poisson(1).rvs() if h3 > 0 else 0
102     max_iter = int(loguniform(100, 10000).rvs())
103     learning_rate_init = loguniform(0.0001, 0.1).rvs()
104     s1 = np.random.choice([0, 1])
105     s2 = np.random.choice([0, 1])
106     s3 = np.random.choice([0, 1]) if s2 == 0 else 0
107
108     return [h1, h2, h3, h4, max_iter, learning_rate_init, s1, s2, s3]
109
```

_solution_evaluate 메서드 정의

_solution_evaluate 메서드는 입력받은 해를 바탕으로 모델을 생성하는 단계와 모델을 평가하는 단계로 구분됩니다.

_solution_evaluate 메서드: 모델 생성

```
110  ## _solution_evaluate 메서드
111  def _solution_evaluate(self, solution, X, y):
112          h1, h2, h3, h4, max_iter, learning_rate_init, s1, s2, s3 = solution
113          h1, h2, h3, h4, max_iter = tuple(map(int, (h1, h2, h3, h4, max_iter)))
114          if h2 == 0:
115              hidden_layer_sizes = (h1,)
116          elif h3 == 0:
117              hidden_layer_sizes = (h1, h2)
118          elif h4 == 0:
119              hidden_layer_sizes = (h1, h2, h3)
120          else:
121              hidden_layer_sizes = (h1, h2, h3, h4)
122          model = MLPC(
123                          hidden_layer_sizes=hidden_layer_sizes,
124                          max_iter=max_iter,
125                          learning_rate_init=learning_rate_init,
126                          random_state=2022
127                          )
128
```

- **라인 113:** map 함수를 사용해 h1, h2, h3, h4, max_iter를 전부 int 형으로 변환했습니다. map(function, iterable) 함수는 순회 가능한 객체 iterable의 각 요소에 function을 일괄적으로 적용합니다. map 함수는 이터레이터(iterator)를 반환하므로 다시 튜플로 형 변환을 했습니다. 이처럼 int 형으로 바꾼 이유는 뒤에서 해를 선택하는 과정에서 solution이 ndarray로 변하기 때문입니다. 즉, ndarray로 바뀌면서 solution의 모든 요소가 최상위 자료형인 float으로 바뀝니다.

_solution_evaluate 메서드 정의 (계속)

_solution_evaluate 메서드는 입력받은 해를 바탕으로 모델을 생성하는 단계와 모델을 평가하는 단계로 구분됩니다.

_solution_evaluate 메서드: 모델 평가

```
129     fold_score_list = []
130     kf = KFold(n_splits=5, shuffle=True, random_state=2022)
131     for train_index, test_index in kf.split(X):
132         X_train, X_test = X[train_index], X[test_index]
133         y_train, y_test = y[train_index], y[test_index]
134
135       if s1 == 1:
136         scaler = MinMaxScaler().fit(X_train)
137         X_train = scaler.transform(X_train)
138         X_test = scaler.transform(X_test)
139       if s2 == 1:
140         X_train, y_train =SMOTE(k_neighbors=3, random_state=2022).fit_resample(X_train, y_train)
143       elif s3 == 1:
144         X_train, y_train = NearMiss().fit_resample(X_train, y_train)
145       model.fit(X_train, y_train)
146       y_pred = model.predict(X_test)
147       fold_score = self.scoring(y_test, y_pred)
148       fold_score_list.append(fold_score)
149     score = self.summarize_scoring(fold_score_list)
150     return score
```

_EI 메서드

가우시안 프로세스 대체 모델인 surrogate_model로 한 이터레이션에서 생성한 X_new의 평균과 표준편차를 바탕으로 X_new의 개선 기댓값을 계산합니다.

_EI 메서드

```
152  ## _EI 메서드
153  def _EI(self, X_new, surrogate_model, best_mu, e=0.01):
154          mu, sigma = surrogate_model.predict(X_new, return_std=True)
155          z = np.zeros(len(X_new))
156          z[sigma > 0] = ((mu - best_mu - e) / sigma)[sigma > 0]
157          return (mu - best_mu - e) * norm.cdf(z) + sigma * norm.pdf(z)
158
```

fit 메서드
(1) 자료형 변환

```
159  ## fit 메서드
160  def fit(self, X, y):
161      # X, y 포맷 변경
162      if isinstance(X, pd.DataFrame):
163          X = X.values
164      elif isinstance(X, list) or isinstance(X, tuple):
165          X = np.array(X)
166      if isinstance(y, pd.Series):
167          y = y.values
168      elif isinstance(y, list) or isinstance(y, tuple):
169          y = np.array(y)
```

fit 메서드
(2) 베이지안 최적화

```
170    # 베이지안 최적화 시작
171    meta_features = self._extract_meta_features(pd.DataFrame(X), pd.Series(y))
172    candidate_list = []
173    for _ in range(self.num_candidate):
174        candidate = meta_features + self._sampling()
175        candidate_list.append(candidate)
176    candidate_score_list = self.meta_model.predict(candidate_list)
177    top_num_init_idx_list = (-candidate_score_list).argsort()[:self.num_init]
178
```

- **라인 171:** _extract_meta_features를 이용하여 X와 y로부터 추출한 메타 특징을 meta_features에 저장합니다. 단, 이 메서드는 데이터프레임과 시리즈를 입력으로 받으므로 X와 y를 각각 데이터프레임과 시리즈로 변환해서 입력했습니다.
- **라인 172~175:** num_candidate만큼 반복해서 meta_features와 하이퍼파라미터로 구성된 리스트인 candidate를 생성하고 candidate_list에 추가합니다.
- **라인 176:** candidate_list를 meta_model에 투입해 나온 출력을 candidate_score_list에 저장합니다.
- **라인 177:** candidate_score_list에서 값이 큰 self.num_init개 요소의 인덱스를 top_num_init_idx_list에 저장합니다. 이 인덱스를 바탕으로 해를 선택할 것입니다.

fit 메서드
(2) 베이지안 최적화

```
179     GP_X = []
180     GP_y = []
181     for idx in top_num_init_idx_list:
182         gp_x = candidate_list[idx][len(meta_features) :]
183         gp_y = self._solution_evaluate(gp_x, X, y)
184
185         GP_X.append(gp_x)
186         GP_y.append(gp_y)
187     surrogate_model = GPR(kernel=Matern(), random_state=2022).fit(GP_X, GP_y)
188
189
```

- **라인 179~180:** 대체 모델을 학습하는 데 사용할 특징 집합과 라벨 집합인 GP_X와 GP_y를 빈 리스트로 초기화합니다.
- **라인 181~183:** top_num_init_idx_list를 idx로 순회하며 gp_x와 gp_y를 정의합니다. gp_x는 candidate_list에서 인덱스가 idx인 요소인데, 여기에는 메타 특징도 섞여 있으므로 메타 특징을 제거합니다. gp_y는 _solution_evaluate 메서드로 gp_x를 평가한 결과입니다.
- **라인 187:** GP_X와 GP_y를 사용해 커널이 Matern인 가우시안 프로세스 회귀를 학습합니다.

fit 메서드
(2) 베이지안 최적화 (계속)

```
189     best_mu = max(surrogate_model.predict(GP_X))
190     for _ in range(self.num_iter - 1):
191       candidate_list = np.array(
192                                   [self._sampling() for _ in range(self.num_candidate)]
193                                   )
194       candidate_score_list = self._EI(candidate_list, surrogate_model, best_mu)
195
196       new_GP_X = list(
197         candidate_list[(-candidate_score_list).argsort()[: self.num_sample]]
198       )
199       new_GP_y = [
200         self._solution_evaluate(new_gp_x, X, y) for new_gp_x in new_GP_X
201       ]
202
203       GP_X += new_GP_X
204       GP_y += new_GP_y
205
206       current_best_mu = max(surrogate_model.predict(new_GP_X))
207       if current_best_mu > best_mu:
208         best_mu = current_best_mu
209       surrogate_model = GPR(kernel=Matern(), random_state=2022).fit(GP_X, GP_y)
```

- **라인 189:** 초기 해만 포함된 GP_X를 사용해 best_mu를 계산합니다.
- **라인 190:** self.num_iter - 1회만큼의 이터레이션을 수행합니다. 1을 뺀 이유는 이미 메타 모델을 이용해 초기 해를 탐색했기 때문입니다.
- **라인 191~193:** _sampling 메서드를 이용해 num_candidate만큼의 해를 생성해 candidate_list에 저장합니다.
- **라인 194:** _EI 메서드를 이용해 candidate_list에 있는 해를 평가합니다. 여기서 _solution_evaluate 메서드를 사용하지 않도록 주의합니다 .
- **라인 196~198:** candidate_list에서 candidate_score_list가 큰 self.num_sample 개를 선택해 new_GP_X에 저장합니다.
- **라인 199~201:** new_GP_X에 있는 해를 실제로 평가한 결과를 new_GP_y에 저장합니다.
- **라인 203~204:** new_GP_X와 new_GP_y를 각각 GP_X와 GP_y에 추가합니다.
- **라인 206~208:** new_GP_X만 사용하여 새로운 해의 예측 평균 가운데 최댓값을 계산하고 그 값이 best_mu보다 크다면 best_mu를 업데이트합니다.
- **라인 209:** GP_X와 GP_y를 사용해 surrogate_model을 재학습합니다.

fit 메서드
(3) self.leaderboard 정의

GP_X와 GP_y가 탐색 결과이므로 이를 활용하여 self.leaderboard를 정의합니다.

```
211     self.leaderboard = pd.DataFrame(
212                                             GP_X,
213                                             columns=[
214                                                         "h1",
215                                                         "h2",
216                                                         "h3",
217                                                         "h4",
218                                                         "max_iter",
219                                                         "learning_rate_init",
220                                                         "s1",
221                                                         "s2",
222                                                         "s3",
223         ],
224     )
225     self.leaderboard["점수"] = GP_y
226
```

fit 메서드
(4) 모델 재학습

성능이 가장 우수한 모델로 전체 데이터에 대해 재학습하겠습니다.

```
227     # 최종 모델 선정 및 학습
228     h1,h2,h3,h4,max_iter,learning_rate_init,s1,s2,s3=GP_X[np.array(GP_y).argmax()]
229     h1, h2, h3, h4, max_iter = tuple(map(int, (h1, h2, h3, h4, max_iter)))
230     if h2 == 0:
231         hidden_layer_sizes = (h1,)
232     elif h3 == 0:
233         hidden_layer_sizes = (h1, h2)
234     elif h4 == 0:
235         hidden_layer_sizes = (h1, h2, h3)
236     else:
237         hidden_layer_sizes = (h1, h2, h3, h4)
238     best_model = MLPC(
239                             hidden_layer_sizes=hidden_layer_sizes,
240                             max_iter=max_iter,
241                             learning_rate_init=learning_rate_init,
242                             random_state=2022,
243     )
244
245     if s1 == 1:
246         scaler = MinMaxScaler().fit(X)
247         X = scaler.transform(X)
248     if s2 == 1:
249         X, y = SMOTE(k_neighbors=3, random_state=2022).fit_resample(X, y)
250     elif s3 == 1:
251         X, y = NearMiss().fit_resample(X, y)
252
253     self.model = best_model.fit(X, y)
```

- **라인 228~243:** GP_y가 가장 큰 GP_X의 하이퍼파라미터를 바탕으로 최종 모델 인스턴스 best_model을 생성합니다.
- **라인 245~251:** 전처리 하이퍼파라미터를 바탕으로 X와 y를 전처리합니다.
- **라인 253:** 최종 모델을 학습하여 self.model에 저장합니다.

predict와 show_leaderboard 메서드

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

predict 메서드

```
255   ## predict 메서드
256   def predict(self, X):
257         return self.model.predict(X)
258
```

show_leaderboard 메서드

```
259   ## show_leaderboard 메서드
260   def show_leaderboard(self):
261         return self.leaderboard
262
```

3. MyAutoML3

4 시스템 활용

활용 예제 1

데이터 불러오기

```
1  # 데이터 불러오기
2  df = pd.read_csv("../../data/classification/glass4.csv")
3  X = df.drop('y', axis = 1)
4  y = df['y']
```

머신러닝 자동화

```
1  aml = MyAutoML3(scoring = "accuracy")
2  aml.fit(X, y)
3  result = aml.show_leaderboard()
4  display(result.sort_values(by = "점수", ascending = False))
```

	h1	h2	h3	h4	max_iter	learning_rate_init	s1	s2	s3	점수
830	11.0	2.0	0.0	0.0	1422.0	0.005899	1.0	1.0	0.0	0.976523
852	12.0	8.0	3.0	0.0	5107.0	0.001567	0.0	1.0	0.0	0.976523
802	18.0	3.0	7.0	1.0	8217.0	0.000789	1.0	1.0	0.0	0.971982
558	18.0	2.0	4.0	0.0	4396.0	0.001914	1.0	1.0	0.0	0.971872
862	12.0	6.0	0.0	0.0	681.0	0.041453	1.0	1.0	0.0	0.971872
...	...	...	...	...	...	...	...	...	...	...
885	7.0	5.0	1.0	1.0	7877.0	0.049869	0.0	1.0	0.0	0.060797
404	20.0	6.0	2.0	1.0	5194.0	0.000389	0.0	1.0	0.0	0.060797
401	29.0	3.0	3.0	1.0	1614.0	0.013089	1.0	1.0	0.0	0.060797
19	13.0	1.0	0.0	0.0	8088.0	0.052823	0.0	0.0	1.0	0.060797
448	18.0	3.0	1.0	4.0	3573.0	0.001046	0.0	0.0	1.0	0.056146

1009 rows × 10 columns

- 최고 0.976523에서 최저 0.056146까지 성능이 나옴을 알 수 있음
- 하이퍼파라미터에 따라 성능이 굉장히 크게 차이 나는 문제에서 좋은 하이퍼파라미터를 잘 찾는다고 할 수 있음

활용 예제 2

데이터 불러오기

```
1  # 데이터 불러오기
2  df = pd.read_csv("../../data/classification/vehicle1.csv")
3  X = df.drop('y', axis = 1)
4  y = df['y']
```

머신러닝 자동화

```
1  aml = MyAutoML3(scoring = "f1", num_iter = 100)
2  aml.fit(X, y)
3  result = aml.show_leaderboard()
4  display(result.sort_values(by = "점수", ascending = False))
```

	h1	h2	h3	h4	max_iter	learning_rate_init	s1	s2	s3	점수
24	14.0	3.0	2.0	0.0	3667.0	0.006209	1.0	1.0	0.0	0.705368
12	13.0	0.0	0.0	0.0	5240.0	0.005206	1.0	1.0	0.0	0.701981
76	8.0	4.0	1.0	0.0	3604.0	0.007573	1.0	1.0	0.0	0.682786
80	19.0	4.0	3.0	0.0	862.0	0.000868	1.0	1.0	0.0	0.679591
39	12.0	4.0	4.0	0.0	9268.0	0.007311	1.0	0.0	0.0	0.675799
...	...	...	...	...	...	...	...	...	...	...
57	18.0	4.0	2.0	2.0	1253.0	0.004606	1.0	1.0	0.0	0.000000
84	14.0	4.0	5.0	0.0	4209.0	0.001301	0.0	0.0	0.0	0.000000
16	16.0	5.0	1.0	1.0	4474.0	0.004674	0.0	0.0	0.0	0.000000
42	18.0	5.0	3.0	1.0	7592.0	0.000338	0.0	0.0	0.0	0.000000
43	14.0	4.0	0.0	0.0	2515.0	0.061859	0.0	0.0	0.0	0.000000

109 rows × 10 columns