들어가기 전에

이번 파트에서는 지금까지 학습한 내용을 바탕으로 머신러닝 자동화 시스템을 구축해보겠습니다.

시스템명	과제 유형	범위	활용 방법론
MyAutoML1	분류	• 모델 선택과 하이퍼파라미터 튜닝	• 그리드 서치
MyAutoML2	회귀	• 모델 선택과 하이퍼파라미터 튜닝	• 실험 • 랜덤 서치
MyAutoML3	이진 분류	• 하이퍼파라미터 튜닝 • 스케일링 • 재샘플링 • 스태킹 앙상블	• 메타 모델 • 베이지안 최적화

1. MyAutoML1

1 문제 정의 및 클래스 설계

탐색 공간

MyAutoml_1은 탐색 공간 내 모든 분류 모델과 하이퍼파라미터를 비교함으로써 최적의 분류 모델과 하이퍼파라미터를 탐색합니다. 여기서 사용자가 탐색하지 않을 모델을 고를 수 있습니다.

모델	하이퍼 파라미터	탐색 공간
k-최근접 이웃 (KNeighborsClassifier)	n_neighbors	{3, 5, 7, 9, 11}
	metric	{"euclidean", "manhattan"}
결정 나무 (DecisionTreeClassifier)	max_depth	{3, 5, 7, 9}
	min_samples_split	{2, 5, 10}
랜덤 포레스트 (RandomForestClassifier)	n_estimators	{50, 100, 200}
	max_depth	{2, 3, 4}
	max_features	{0.2, 0.4, 0.6, 0.8, 1.0}
신경망 (MLPClassifier)	hidden_layer_sizes	{(10,), (10, 10), (20, 20), (15, 15, 15), (20, 20, 20), (15, 15, 15, 15), (30, 30, 30, 30)}
	max_iter	{2000}

클래스 설계
:인자 정의

MyAutoml_1은 다음과 같은 인자를 갖는 파이썬 클래스로 개발합니다.

인자	설명	기본값
exclude_models	탐색에서 제외할 모델 목록 ["KNN", "DT", "RF", "MLP"]의 부분 집합	[ ]
seed	k-최근접 이웃을 제외한 각 모델의 시드	2022
cv	폴드 개수	5
scoring	분류 평가 척도	“accuracy”
summarize_score	평가 결과 요약 방법	“mean”
early_stopping	교차 검증에서의 조기 종료 여부	False
early_stopping_criteria	조기 종료 기준으로 다음을 만족하면 조기 종료를 수행 1 - (현재 폴드에서의 성능/현재까지 가장 좋은 모델의 성능) > early_stopping_criteria	0.1

클래스 설계
:메서드 정의

MyAutoml_1은 다음과 같은 메서드를 갖습니다.

메서드	설명
fit	특징 X와 라벨 y를 입력받아 그리드 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.

1. MyAutoML1

2 시스템 구현

모듈 불러오기

시스템 구현에 필요한 모듈과 함수를 다음과 같이 불러옵니다.

모듈 불러오기

```
1  import pandas as pd
2  from sklearn.model_selection import ParameterGrid, KFold
3  from sklearn.neighbors import KNeighborsClassifier
4  from sklearn.tree import DecisionTreeClassifier
5  from sklearn.ensemble import RandomForestClassifier
6  from sklearn.neural_network import MLPClassifier
7  from sklearn.metrics import *
8  from functools import partial
9  import numpy as np
```

- **라인 8**: functools.partial은 인자가 기본값이 아닌 값으로 설정된 함수를 변수에 저장하는 데 사용합니다. 여기서는 average 인자가 "macro"와 "weighted"로 설정해야 하는 다중 클래스 분류 평가 지표를 정의하는 데 사용합니다.

생성자 정의
(1) 인자 정의

클래스 생성자의 인자를 정의합니다.

클래스 생성자 입력 정의

```
class MyAutoML1:
    ## 생성자
    def __init__(
        self,
        exclude_models=[],
        seed=None,
        cv=5,
        scoring="accuracy",
        summarize_scoring="mean",
        early_stopping=False,
        early_stopping_criteria=0.1,
):
```

생성자 정의
(2) 인자값 검사

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

인자값 검사: exclude_models

```
13    # self.exclude_models 정의
14    model_set = {"KNN", "DT", "RF", "MLP"}
15    if set(exclude_models) == model_set:
16        raise ValueError("모든 모델을 제외할 수 없습니다.")
17    improper_models = set(exclude_models) - model_set
18    if len(improper_models) >= 1:
19        raise ValueError(
20            "{}는 exclude_models에 포함할 수 없습니다".format(improper_models)
21      )
22    self.exclude_models = exclude_models
23
```

- **라인 14:** 입력할 수 있는 전체 모델 목록인 model_set를 집합 자료형으로 정의합니다.
- **라인 15~16:** exclude_models와 model_set이 같으면 모든 모델이 탐색 대상에서 제외되므로 ValueError를 발생시킵니다. raise는 의도적으로 오류를 발생시키는 데 사용합니다.
- **라인 17:** exclude_models에는 포함됐으나, model_set에는 포함되지 않은 집합(차집합)을 improper_models라고 정의합니다. 즉, "KNN", "DT", "RF", "MLP" 외의 값은 improper_models에 포함됩니다.
- **라인 18~21:** improper_models에 한 개의 요소라도 포함되면 오류를 발생시킵니다.
- **라인 22:** exclude_models를 self.exclude_models에 저장합니다.

생성자 정의
(2) 인자값 검사 (계속)

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

인자값 검사: seed

```
24      # self.seed 정의
25      if (type(seed) != int) and (seed is not None):
26          raise ValueError("seed는 int 형 혹은 None이어야 합니다.")
27      self.seed = seed
28
```

- **라인 25~26:** seed는 int 자료형 혹은 None이어야 하므로 seed가 int 자료형이나 None이 아니라면 오류를 발생시킵니다.

인자값 검사: cv

```
29      # self.cv 정의
30      if type(cv) != int:
31          raise ValueError("cv는 int 형이어야 합니다.")
32      if cv < 2:
33          raise ValueError("cv는 2보다는 커야 합니다.")
34      self.cv = cv
35
```

- **라인 30~33:** k-겹 교차 검증에서 폴드의 개수를 나타내는 cv는 2 이상의 자연수여야 합니다. 따라서 int 자료형이 아니거나 2보다 작은 값이라면 오류를 발생시킵니다.

생성자 정의
(2) 인자값 검사 (계속)

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

인자값 검사: scoring

```
36    # self.scoring 정의
37    scoring_dict = {
38      "accuracy": accuracy_score,
39      "precision": precision_score,
40      "weighted-precision": partial(precision_score, average="weighted"),
41      "macro-precision": partial(precision_score, average="macro"),
42      "recall": recall_score,
43      "weighted-recall": partial(recall_score, average="weighted"),
44      "macro-recall": partial(recall_score, average="macro"),
45      "f1": f1_score,
46      "weighted-f1": partial(f1_score, average="weighted"),
47      "macro-f1": partial(f1_score, average="macro"),
48    }
49
50    if scoring not in scoring_dict.keys():
51          msg = "scoring은 {}중 하나여야 합니다.".format(scoring_dict.keys())
52          raise ValueError(msg)
53    self.scoring = scoring_dict[scoring]
54
```

- **라인 37~48:** 사용자가 입력한 문자열을 대응되는 함수로 바꾸기 위한 사전 자료형인 scoring_dict를 정의합니다. 예를 들어, "accuracy"가 입력되면 accuracy_score 함수를 사용합니다(라인 38). 라인 40과 같이 "weighted-precision"이 입력되면 precision_score 함수에서 average 인자를 "weighted"로 설정한 함수를 전달해야 하므로 partial 함수를 사용했습니다. partial 함수는 함수를 호출하지 않고 일부 입력을 미리 설정해놓은 함수를 반환합니다.
- **라인 50~52:** 입력한 scoring이 scoring_dict의 키가 아니면 오류를 발생시킵니다.
- **라인 53:** scoring_dict를 scoring으로 인덱싱한 함수를 self.scoring에 저장합니다.

생성자 정의
(2) 인자값 검사 (계속)

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

인자값 검사: summarize_scoring

```
55      # self.summarize_scoring 정의
56      summarize_scoring_dict = {"mean": np.mean, "max": np.max, "min": np.min}
57
58      if summarize_scoring not in ["mean", "max", "min"]:
59          msg = "summarize_scoring는 {'mean', 'max', 'min'}중 하나여야 합니다."
60          raise ValueError(msg)
61      self.summarize_scoring = summarize_scoring_dict[summarize_scoring]
62
```

- **라인 56:** 사용자가 입력한 문자열을 대응되는 함수로 바꾸기 위한 사전 자료형인 summarize_scoring_dict를 정의합니다. 여기서 볼 수 있듯이, 최소, 평균, 최대 중 하나로 k-겹 교차 검증 결과를 요약합니다.

생성자 정의
(2) 인자값 검사 (계속)

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

인자값 검사: early_stopping

```
63    # self.early_stopping 정의
64    if type(early_stopping) is not bool:
65        raise ValueError("early_stopping은 True 혹은 False여야 합니다.")
66    self.early_stopping = early_stopping
67
```

- **라인 64~65:** early_stopping은 부울 자료형입니다. 부울 자료형이 아닌 다른 자료형이 입력되면 오류를 발생시킵니다.

인자값 검사: early_stopping_critera

```
68    # early_stopping_criteria 정의
69    if type(early_stopping_criteria) is not float:
70        raise ValueError("early_stopping_criteria 자료형은 float이어야 합니다.")
71    if early_stopping_criteria <= 0 or early_stopping_criteria >= 1:
72        raise ValueError("early_stopping_criteria는 0과 1 사이의 값이어야 합니다.")
73    self.early_stopping_criteria = early_stopping_criteria
74
```

- **라인 69~72:** early_stopping_criteria는 0과 1 사이의 float 자료형입니다. 그 외의 값이 들어오면 오류를 발생시킵니다.

fit 메서드
(1) 입력값 변환

특징 벡터 X와 라벨 y가 입력됐을 때 그리드 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

```
75  ## fit 메서드
76  def fit(self, X, y):
77      # X, y 포맷 변경
78      if isinstance(X, pd.DataFrame):
79          X = X.values
80      elif isinstance(X, list) or isinstance(X, tuple):
81          X = np.array(X)
82      if isinstance(y, pd.Series):
83          y = y.values
84      elif isinstance(y, list) or isinstance(y, tuple):
85          y = np.array(y)
```

- **라인 78~79:** X가 데이터프레임이라면 values 속성을 사용해 X를 ndarray로 변환합니다. 여기서 isinstance(instance, class)는 instance가 class의 객체이면 True를, 그렇지 않으면 False를 반환합니다.
- **라인 80~81:** X가 리스트 혹은 튜플이라면 np.array 함수를 사용해 ndarray로 변환합니다.

fit 메서드
(2) 그리드 설계

모델별 하이퍼 파라미터 범위를 기반으로 모델의 그리드를 설계합니다.

fit 메서드: 그리드 설계

```
# K최근접 이웃 그리드 정의
kNN_grid = ParameterGrid(
        {"n_neighbors": [3, 5, 7, 9, 11], "metric": ["euclidean", "manhattan"]}
)
# 결정 나무 그리드 정의
DT_grid = ParameterGrid(
        {"max_depth": [3, 5, 7, 9], "min_samples_split": [2, 5, 10]}
)
# 랜덤 포레스트 그리드 정의
RFR_grid = ParameterGrid(
    {
        "n_estimators": [50, 100, 200],
        "max_depth": [2, 3, 4],
        "max_features": [0.2, 0.4, 0.6, 0.8, 1.0],
    }
)
```

```
# 신경망 그리드 정의
MLP_grid = ParameterGrid(
    {
        "hidden_layer_sizes": [
            (10,),
            (10, 10),
            (20, 20),
            (15, 15, 15),
            (20, 20, 20),
            (15, 15, 15, 15),
            (30, 30, 30, 30),
        ],
        "max_iter": [2000],
    }
)

```

fit 메서드
(2) 그리드 설계

각 모델의 그리드를 병합하는 방식으로 전체 그리드인 grid를 정의합니다. grid는 키가 분류 모델 클래스이고 값이 대응되는 분류 모델의 그리드인 사전 자료형입니다.

fit 메서드: 그리드 설계

```
118    # 전체 그리드 정의
119    grid = {
120        KNeighborsClassifier: kNN_grid,
121        DecisionTreeClassifier: DT_grid,
122        RandomForestClassifier: RFR_grid,
123        MLPClassifier: MLP_grid
124    }
125
```

fit 메서드
(3) 그리드 서치

grid에 정의된 모든 모델과 하이퍼파라미터를 탐색하겠습니다. 먼저, 모델과 하이퍼 파라미터를 순회합니다.

fit 메서드: 그리드 서치 (모델 및 하이퍼 파라미터 순회)

```
126     # 그리드 서치 시작
127     best_score = 0
128     self.leaderboard = []
129     for model_func in grid.keys():
130         if model_func in self.exclude_models:
131             continue
132         for params in grid[model_func]:
133             if model_func != KNeighborsClassifier:
134                 params["random_state"] = self.seed
135             kf = KFold(n_splits=self.cv, shuffle=True, random_state=self.seed)
136             fold_score_list = []
```

- **라인 127:** 현재까지 찾은 최고의 점수를 0으로 초기화합니다.
- **라인 128:** 리더보드 속성을 빈 리스트로 초기화합니다. 그리드 서치를 수행하면서 이 리스트에 탐색 결과를 추가하고 마지막에는 데이터프레임으로 변환하겠습니다.
- **라인 129:** 분류 모델 클래스를 model_func으로 순회합니다.
- **라인 130~131:** model_func이 exclude_models에 속하면 탐색하지 않습니다.
- **라인 132:** model_func의 그리드를 params로 순회합니다.
- **라인 133~134:** model_func이 KNeighborsClassifier가 아니라면 사전인 params에 {"random_state":self.seed}를 추가합니다. k-최근접 이웃은 random_state 인자가 없으므로 추가하지 않았습니다.
- **라인 135:** self.cv와 self.seed를 이용해 KFold 인스턴스를 정의합니다.
- **라인 136:** 각 모델과 하이퍼파라미터를 교차 검증한 결과를 담을 fold_score_list를 빈 리스트로 초기화합니다. fold_score_list[i]는 i번째 폴드로 평가한 결과가 저장됩니다.

fit 메서드
(3) 그리드 서치 (계속)

모델과 하이퍼파라미터를 k-겹 교차 검증으로 평가합니다. 조기 종료 여부에 따라 다르게 평가합니다.

fit 메서드: 그리드 서치 (모델과 하이퍼 파라미터 평가)

```
137         # 조기 종료를 하는 경우
138         if self.early_stopping:
139             for train_index, test_index in kf.split(X):
140                 X_train, X_test = X[train_index], X[test_index]
141                 y_train, y_test = y[train_index], y[test_index]
142                 model = model_func(**params).fit(X_train, y_train)
143                 y_pred = model.predict(X_test)
144                 fold_score = self.scoring(y_test, y_pred)
145                 fold_score_list.append(fold_score)
146                 if fold_score < best_score * (1 - self.early_stopping_criteria):
147                     break
```

- **라인 139~143:** kf로 구분한 train_index와 test_index를 사용해 학습 데이터와 평가 데이터를 정의한 뒤 모델을 학습하고 예측까지 수행합니다. 여기서 X와 y는 모두 ndarray이므로 loc와 같은 인덱서를 사용하지 않았습니다.
- **라인 144~145:** self.scoring을 사용해 y_test와 y_pred를 비교한 결과를 fold_score에 저장하고 이를 fold_score_list에 추가합니다.
- **라인 146~147:** fold_score가 현재까지 최고 점수보다 self.early_stopping_criteria * 100%이상 낮다면 조기 종료합니다.

fit 메서드
(3) 그리드 서치 (계속)

모델과 하이퍼파라미터를 k-겹 교차 검증으로 평가합니다. 조기 종료 여부에 따라 다르게 평가합니다.

fit 메서드: 그리드 서치 (모델과 하이퍼 파라미터 평가)

```
148         # 조기 종료를 하지 않는 경우
149         else:
150             for train_index, test_index in kf.split(X):
151                 X_train, X_test = X[train_index], X[test_index]
152                 y_train, y_test = y[train_index], y[test_index]
153                 model = model_func(**params).fit(X_train, y_train)
154                 y_pred = model.predict(X_test)
155                 fold_score = self.scoring(y_test, y_pred)
156                 fold_score_list.append(fold_score)
```

- **라인 149~156:** self.early_stopping이 False라면 조기 종료하지 않고 모델과 하이퍼파라미터를 평가합니다.

fit 메서드
(3) 그리드 서치 (계속)

현재 탐색한 모델과 하이퍼파라미터의 평가 점수를 계산하고 최종 모델과 리더보드를 업데이트하겠습니다.

fit 메서드: 그리드 서치 (평가 점수 계산 및 최종 모델과 리더보드 업데이트)

```
157         # 현재까지 찾은 최고의 해 및 리더보드 업데이트
158         score = self.summarize_scoring(fold_score_list)
159         if score > best_score:
160             best_score = score
161             best_model_func = model_func
162             best_params = params
163         self.leaderboard.append([model_func, params, score])
164     self.model = best_model_func(**best_params).fit(X, y)
165     self.leaderboard = pd.DataFrame(self.leaderboard,
166                                     columns=["모델", "파라미터", "점수"])
167
```

- **라인 158:** self.summarizing_scoring을 이용해 fold_score_list를 대푯값으로 요약합니다.
- **라인 159~162:** 현재까지 찾은 해보다 현재 찾은 해가 더 좋은 해라면 best_score, best_model_func, best_params를 업데이트합니다.
- **라인 163:** 현재 해 정보를 self.leaderboard에 추가합니다.
- **라인 164:** 최종 모델과 하이퍼파라미터를 갖는 모델을 전체 데이터에 대해 재학습합니다.
- **라인 165~166:** self.leaderboard를 모델, 파라미터, 점수라는 칼럼을 갖는 데이터프레임으로 변환합니다.

predict와 show_leaderboard 메서드

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

predict 메서드

```
168  ## predict 메서드
169  def predict(self, X):
170          return self.model.predict(X)
171
```

show_leaderboard 메서드

```
172  ## show_leaderboard 메서드
173  def show_leaderboard(self):
174          return self.leaderboard
175
```

1. MyAutoML1

3 시스템 활용

활용 예제 1

데이터 불러오기

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

머신러닝 자동화

```
1  aml = MyAutoML1()
2  aml.fit(X, y)
3  result = aml.show_leaderboard()
4  display(result.sort_values(by = "점수", ascending = False))
```

	모델	파라미터	점수
57	<class 'sklearn.ensemble._forest.RandomForestC...	{'max_depth': 4, 'max_features': 0.4, 'n_estim...	0.605390
54	<class 'sklearn.ensemble._forest.RandomForestC...	{'max_depth': 4, 'max_features': 0.2, 'n_estim...	0.604771
62	<class 'sklearn.ensemble._forest.RandomForestC...	{'max_depth': 4, 'max_features': 0.8, 'n_estim...	0.604136
65	<class 'sklearn.ensemble._forest.RandomForestC...	{'max_depth': 4, 'max_features': 1.0, 'n_estim...	0.603501
66	<class 'sklearn.ensemble._forest.RandomForestC...	{'max_depth': 4, 'max_features': 1.0, 'n_estim...	0.602878
...	...	...	...
2	<class 'sklearn.neighbors._classification.KNei...	{'metric': 'euclidean', 'n_neighbors': 7}	0.507819
4	<class 'sklearn.neighbors._classification.KNei...	{'metric': 'euclidean', 'n_neighbors': 11}	0.507185
3	<class 'sklearn.neighbors._classification.KNei...	{'metric': 'euclidean', 'n_neighbors': 9}	0.496526
1	<class 'sklearn.neighbors._classification.KNei...	{'metric': 'euclidean', 'n_neighbors': 5}	0.494075
0	<class 'sklearn.neighbors._classification.KNei...	{'metric': 'euclidean', 'n_neighbors': 3}	0.489673

74 rows × 3 columns

활용 예제 2

데이터 불러오기

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

머신러닝 자동화

```
aml = MyAutoML1(scoring = "f1", early_stopping = True, early_stopping_criteria = 0.05)
aml.fit(X, y)
result = aml.show_leaderboard()
display(result.sort_values(by = "점수", ascending = False))
display(aml.predict(X)[:5])
```

	모델	파라미터	점수
61	<class 'sklearn.ensemble._forest.RandomForestC...	{'max_depth': 4, 'max_features': 0.8, 'n_estim...	0.803052
66	<class 'sklearn.ensemble._forest.RandomForestC...	{'max_depth': 4, 'max_features': 1.0, 'n_estim...	0.801491
40	<class 'sklearn.ensemble._forest.RandomForestC...	{'max_depth': 3, 'max_features': 0.4, 'n_estim...	0.799017
41	<class 'sklearn.ensemble._forest.RandomForestC...	{'max_depth': 3, 'max_features': 0.4, 'n_estim...	0.792135
44	<class 'sklearn.ensemble._forest.RandomForestC...	{'max_depth': 3, 'max_features': 0.6, 'n_estim...	0.788412
...	...	...	...
5	<class 'sklearn.neighbors._classification.KNei...	{'metric': 'manhattan', 'n_neighbors': 3}	0.631579
13	<class 'sklearn.tree._classes.DecisionTreeClas...	{'max_depth': 5, 'min_samples_split': 2, 'rand...	0.621622
2	<class 'sklearn.neighbors._classification.KNei...	{'metric': 'euclidean', 'n_neighbors': 7}	0.615385
21	<class 'sklearn.tree._classes.DecisionTreeClas...	{'max_depth': 9, 'min_samples_split': 10, 'ran...	0.613333
18	<class 'sklearn.tree._classes.DecisionTreeClas...	{'max_depth': 7, 'min_samples_split': 10, 'ran...	0.609756

74 rows × 3 columns

```
array([1, 1, 1, 1, 1], dtype=int64)
```