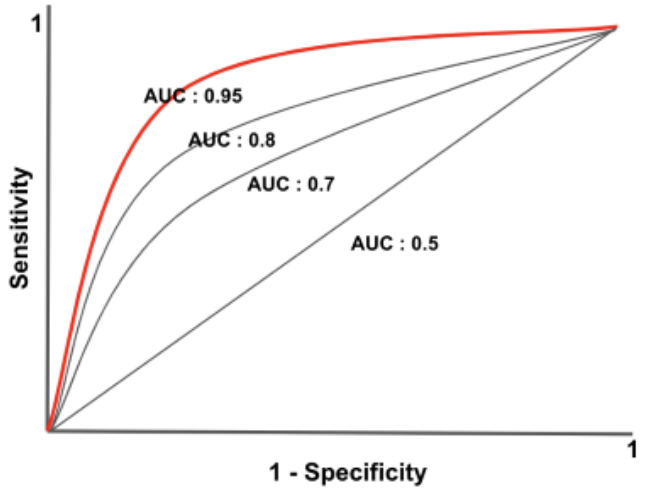
목차

table of contents

1. 프로젝트 소개

★ Genetic Algorithm(GA)을 이용한 머신러닝 모델의 하이퍼파라미터 최적화(HPO)

- 데이콘 유저들의 행동 데이터를 활용
- 모델: RandomForest Classifier
- GA를 이용해 구한 하이퍼파라미터를 제출하여 스코어 비교
- Grid Search, Random Search, Optuna 등 다른 HPO Method와 비교



2. 전체적인 알고리즘 구현

(0) 사용할 라이브러리 소개

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, accuracy_score, f1_score
import time
import warnings
warnings.filterwarnings('ignore')
pd.options.mode.chained_assignment = None
```

★ 라이브러리

- numpy : 데이터의 구조 변경 및 연산
- pandas : 데이터프레임 사용
- matplotlib : 시각화
- sklearn.ensemble의 RandomForestClassifier
  : 랜덤포레스트 분류 모델 사용
- sklearn.metrics의 roc_auc_score, accuracy_score, f1_score
  : 평가 지표 계산
- time : 실행 시간 계산

2. 전체적인 알고리즘 구현

(1) 초기해 생성

```python
def parameter_setting(params={}):
    try:
        rf_parameters = []

        if 'n_estimators' in params.keys():
            rf_parameters.append(params['n_estimators'])
        else:
            rf_parameters.append(np.arange(1, 1000))

        if 'criterion' in params.keys():
            rf_parameters.append(params['criterion'])
        else:
            rf_parameters.append(['gini', 'entropy'])

        if 'max_depth' in params.keys():
            rf_parameters.append(params['max_depth'])
        else:
            rf_parameters.append(np.arange(2, 20))

        if 'min_samples_split' in params.keys():
            rf_parameters.append(params['min_samples_split'])
        else:
            rf_parameters.append(2)

        if 'min_samples_leaf' in params.keys():
            rf_parameters.append(params['min_samples_leaf'])
        else:
            rf_parameters.append(np.arange(1, 20))

        if 'min_weight_fraction_leaf' in params.keys():
            rf_parameters.append(params['min_weight_fraction_leaf'])
        else:
            rf_parameters.append(np.arange(0.0, 0.5, 0.02))

        if 'max_features' in params.keys():
            rf_parameters.append(params['max_features'])
        else:
            rf_parameters.append(['sqrt', 'auto', 'log2', None])

        if 'max_leaf_nodes' in params.keys():
            rf_parameters.append(params['max_leaf_nodes'])
        else:
            rf_parameters.append(None)

        if 'min_impurity_decrease' in params.keys():
            rf_parameters.append(params['min_impurity_decrease'])
        else:
            rf_parameters.append(np.arange(0.0, 1.0, 0.05))

        if 'bootstrap' in params.keys():
            rf_parameters.append(params['bootstrap'])
        else:
            rf_parameters.append([True, False])

        return rf_parameters
    except Exception as e:
        print("Exception parameter error:",e)
```

```python
params = {
    'n_estimators': [100, 300, 400, 600],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split' : [2, 4, 10],
    'min_samples_leaf': [5,10, 15,30],
    'min_weight_fraction_leaf': [0.0, 0.2],
    'max_features':['auto','sqrt','log2',None],
    'max_leaf_nodes':[None, 10, 20, 60],
    'min_impurity_decrease':[0.0, 0.3],
    'bootstrap': [True,False]
}
```

★ 초기해 생성에 필요한 함수

- **parameter_setting(params={})**
  : 입력 받는 각 하이퍼파라미터의 값(범위)를 딕셔너리 형태에서 리스트로 변경

★ 하이퍼파라미터(10개)

- n_estimators
- criterion
- max_depth
- min_samples_split
- min_samples_leaf
- min_weight_fraction_leaf
- max_features
- max_leaf_nodes
- min_impurity_decrease
- bootstrap

2. 전체적인 알고리즘 구현

(1) 초기해 생성

```python
def get_model(row, random_state):
    '''
    row : population의 row = 하이퍼파라미터 세트
    random_state : 랜덤 고정 상수

    ** 각 row(하이퍼파라미터 세트)에 대한 모델을 return **
    '''

    return RandomForestClassifier(
                n_estimators = int(row[0]),
                criterion = row[1],
                max_depth = row[2],
                min_samples_split = int(row[3]),
                min_samples_leaf = int(row[4]),
                min_weight_fraction_leaf = float(row[5]),
                max_features = row[6],
                max_leaf_nodes = row[7],
                min_impurity_decrease = float(row[8]),
                bootstrap = row[9],
                random_state = random_state,
                n_jobs = -1)
```

```python
def evaluation(model, error_metric, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]

    if error_metric == 'accuracy_score':
        eval = accuracy_score(y_test, predictions)
    elif error_metric == 'f1_score':
        eval = f1_score(y_test, y_pred_proba, average='binary')   # average = [None, 'micro', 'macro', 'samples', 'weighted', 'binary']
    elif error_metric == 'roc_auc_score':
        eval = roc_auc_score(y_test, y_pred_proba, average='macro')    # average = [None, 'micro', 'macro', 'weighted']

    return eval
```

★ 초기해 생성에 필요한 함수

- **get_model(row, random_state)**

  : 각 row(하이퍼파라미터 세트)에 대한 모델을 return

- **evaluation(model, error_metric, X_train, X_test, y_train, y_test)**

  : 모델을 훈련시키고 예측을 수행한 후, 선택된 평가 지표를 사용하여 모델 성능을 평가

✓ model: 평가할 머신러닝 모델

✓ error_metric: 평가 지표

('accuracy_score', 'f1_score', '**roc_auc_score**' 중 하나)

✓ X_train, X_test, y_train, y_test: 훈련 및 테스트 데이터

2. 전체적인 알고리즘 구현

(1) 초기해 생성

★ 해표현: DataFrame의 열(column)을 하이퍼파라미터로 생성했을 때 <u>각 행 값을 갖는 모델</u>

	n_estimators	criterion	max_depth	min_samples_split	min_samples_leaf	min_weight_fraction_leaf	max_features	max_leaf_nodes	min_impurity_decrease	bootstrap	score
0	400	gini	40	2	10	0.0	sqrt	60	0.3	False	0.500000

```python
params = {
    'n_estimators': [100, 300, 400, 600],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split' : [2, 4, 10],
    'min_samples_leaf': [5,10, 15,30],
    'min_weight_fraction_leaf': [0.0, 0.2],
    'max_features':['auto','sqrt','log2',None],
    'max_leaf_nodes':[None, 10, 20, 60],
    'min_impurity_decrease':[0.0, 0.3],
    'bootstrap': [True,False]
}
rf_parameters = parameter_setting(params)
random_state = 42
error_metric = 'roc_auc_score'
① X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2, shuffle=True, stratify=y_train, random_state=42)

② objective_function = lambda i: objective(error_metric, i, X_train, X_test, y_train, y_test, random_state=random_state)

③ model = RandomForestClassifier(random_state=42, **params)
④ population_size = 10
population = pd.DataFrame()   # parameter들로만 DataFrame 생성
np.random.seed(random_state)
⑤ population['n_estimators'] = np.random.choice(rf_parameters[0], size=population_size)
population['criterion'] = np.random.choice(rf_parameters[1], size=population_size)
population['max_depth'] = np.random.choice(rf_parameters[2], size=population_size)
population['min_samples_split'] = np.random.choice(rf_parameters[3], size=population_size)
population['min_samples_leaf'] = np.random.choice(rf_parameters[4], size=population_size)
population['min_weight_fraction_leaf'] = np.random.choice(rf_parameters[5], size=population_size)
population['max_features'] = np.random.choice(rf_parameters[6], size=population_size)
population['max_leaf_nodes'] = np.random.choice(rf_parameters[7], size=population_size)
population['min_impurity_decrease'] = np.random.choice(rf_parameters[8], size=population_size)
population['bootstrap'] = np.random.choice(rf_parameters[9], size=population_size)

⑥ population['score'] = population.apply(objective_function, axis=1)
```

- 'params' 딕셔너리에 하이퍼파라미터의 searching 범위 정의

```python
def objective(error_metric, row, X_train, X_test, y_train, y_test, random_state):
    '''
    목적함수를 위한 모델의 평가값을 계산하는 함수
    '''

    if 'score' in row and row['score'] == row['score']:  # row[error_metric] == row[error_metric]
        return row['score']  # checks that row[error_metric] is not NaN.

    return evaluation(get_model(row, random_state=random_state), error_metric,
                      X_train, X_test, y_train, y_test)
```

1. Train, Test data 분리
2. 목적함수(objective function) 정의
3. 모델 생성
4. 초기해 size 설정(population_size = 10)
5. 초기해 랜덤 생성
6. score 열을 추가하여 각 행의 score 값(평가값)으로 판단

2. 전체적인 알고리즘 구현

(1) 초기해 생성

- 랜덤으로 생성된 10개의 초기해와 각 해의 평가값

population

	n_estimators	criterion	max_depth	min_samples_split	min_samples_leaf	min_weight_fraction_leaf	max_features	max_leaf_nodes	min_impurity_decrease	bootstrap	score
0	400	gini	40	2	10	0.0	sqrt	60	0.3	False	0.500000
1	600	gini	10	2	15	0.0	auto	None	0.3	False	0.500000
2	100	gini	30	10	30	0.0	sqrt	60	0.3	False	0.500000
3	400	gini	10	10	15	0.0	auto	20	0.3	False	0.500000
4	400	entropy	30	10	30	0.2	sqrt	20	0.3	False	0.500000
5	600	gini	40	4	30	0.2	None	10	0.3	False	0.500000
6	100	entropy	None	10	5	0.0	None	None	0.3	False	0.500000
7	100	entropy	30	4	15	0.2	log2	60	0.3	False	0.500000
8	400	entropy	10	4	5	0.2	None	10	0.0	True	0.829825
9	300	gini	40	10	15	0.2	log2	60	0.0	False	0.658333

2. 전체적인 알고리즘 구현

(2) 선택 후 교배

★ 선택 함수(룰렛 휠 선택)

- **selection(population, random_state)**
  : 높은 score를 가지는 parameter set가 높은 확률로 선택될 수 있도록 Roulette Wheel Selection

```python
def selection(population, random_state = 42):
    '''
    population : 전체 population
    random_state : 랜덤 고정 상수

    ** 높은 score를 가지는 parameter set가 높은 확률로 선택될 수 있도록 Roulette Wheel Selection **
    '''
    # 랜덤 시드 고정
    np.random.seed(random_state)
    # population 개수 저장
    length = population.shape[0]
    # 각 행의 score의 확률 계산
    prob = population['score'] / population['score'].sum()
    # 각 행의 score 확률에 기반해 랜덤 index 선택(size 개수만큼 선택)
    indices = np.random.choice(np.arange(length), p=prob, size=length)

    return population.iloc[indices]
```

- 교배 시킬 Population쌍 DataFrame 생성

```python
pairs = selection(population, random_state = 42)
pairs
```

	n_estimators	criterion	max_depth	min_samples_split	min_samples_leaf	min_weight_fraction_leaf	max_features	max_leaf_nodes	min_impurity_decrease	bootstrap	score
4	400	entropy	30	10	30	0.2	sqrt	20	0.3	False	0.500000
9	300	gini	40	10	15	0.2	log2	60	0.0	False	0.658333
8	400	entropy	10	4	5	0.2	None	10	0.0	True	0.829825
6	100	entropy	None	10	5	0.0	None	None	0.3	False	0.500000
1	600	gini	10	2	15	0.0	auto	None	0.3	False	0.500000
1	600	gini	10	2	15	0.0	auto	None	0.3	False	0.500000
0	400	gini	40	2	10	0.0	sqrt	60	0.3	False	0.500000
8	400	entropy	10	4	5	0.2	None	10	0.0	True	0.829825
6	100	entropy	None	10	5	0.0	None	None	0.3	False	0.500000
7	100	entropy	30	4	15	0.2	log2	60	0.3	False	0.500000

2. 전체적인 알고리즘 구현

(2) 선택 후 교배

```python
def crossover(pairs, random_state = 42):
    '''
    pairs : roughlette_wheel_selection으로 선택된 population 쌍
    random_state : 랜덤 고정 상수

    ** 전체 popuation을 기준으로 무작위 교차 **
    '''
    # 랜덤 시드 고정
    np.random.seed(random_state)
    # pairs의 행, 열 개수 저장
    length, width = pairs.shape
    # 열 개수 -1  --> 마지막 열 = 'score'이므로 score는 제외하고 교배
    width -= 1

    # 교배율X --> 0, 1을 각 parameter마다 랜덤 생성하여 새로운 배열의 자손 생성
    ## 0, 1을 전체 parameter 수 만큼 랜덤 생성
    a = np.random.choice((0, 1), size=length * width) ①
    ## 교배(교차)를 시키기 위한 단계(짝수번째에 -1을 곱함)
    a[np.arange(1, length * width, 2)] *= -1 ②
    ## 새로운 배열을 생성할 인덱스 'i' 리스트 생성
    i = np.arange(length * width) + a ③

    ④
    # i 리스트를 이용하여 새로운 자손 생성
    ## score열을 제외하고 DataFrame을 flatten 시켜 1차원 배열로 생성
    gene_list = np.array(pairs.drop('score', axis=1)).reshape(-1, order='F')
    ## 1차원 배열로 생성된 parameter 값들에 i 리스트에 해당하는 값들로 새롭게 자손 parameter 세트 생성
    return pd.DataFrame(gene_list[i].reshape((-1, width), order='F'), columns=pairs.columns[:-1])
```

★ 교배 함수

- **crossover(pairs, random_state)**
  : 선택된 개체 쌍을 기반으로 무작위 교차를 수행하여 새로운 자손을 생성

- score 열을 제외하여 교배

★ 교배 과정:

1. 각 parameter 마다 '0' 또는 '1'을 무작위로 생성
2. 무작위로 생성된 '0' 또는 '1'에 짝수 번째에만 '-1'을 곱함
3. 새로운 자손을 생성할 인덱스 리스트(i) 생성

4. parameter의 DataFrame을 1차원 배열로 변경하여 i 리스트에 맵핑하여 교배된 새로운 자손 DataFrame 생성

2. 전체적인 알고리즘 구현

(2) 선택 후 교배

```
length, width = pairs.shape
width -= 1
a = np.random.choice((0, 1), size=length * width)
a[np.arange(1, length * width, 2)] *= -1
i = np.arange(length * width) + a

print(a)
print(i)
```

```
[ 1  0  1 -1  1 -1  1 -1  1 -1  0  0  1 -1  1  0  1  0  0  0  0  0  1 -1
  1 -1  1  0  1 -1  0 -1  0 -1  0 -1  1  0  0  0  0  0  0  0  0 -1  1  0
  1 -1  1 -1  0 -1  0 -1  1 -1  0 -1  0 -1  0 -1  0  0  1  0  1 -1  1 -1
  1 -1  1 -1  1 -1  1  0  0 -1  1 -1  1 -1  1 -1  1  0  1  0  1 -1  0 -1
  0 -1  1  0]
[ 1  1  3  2  5  4  7  6  9  8 10 11 13 12 15 15 17 17 18 19 20 21 23 22
 25 24 27 27 29 28 30 30 32 32 34 34 37 37 38 39 40 41 42 43 44 44 47 47
 49 48 51 50 52 52 54 54 57 56 58 58 60 60 62 62 64 65 67 67 69 68 71 70
 73 72 75 74 77 76 79 79 80 80 83 82 85 84 87 86 89 89 91 91 93 92 94 94
 96 96 99 99]
```

- length = 10 (행 개수)
- width = 10 (열 개수)
- 생성할 배열의 길이(개수) = length × width = 100

- a : 무작위로 0 또는 1로 채워진 배열을 생성 후, 짝수 번째에 -1을 곱한 배열

- i : 0부터 99까지의 정수 배열을 생성하고, ‘a’를 더한 새로운 배열 생성

2. 전체적인 알고리즘 구현

(2) 선택 후 교배

```
gene_list = np.array(pairs.drop('score', axis=1)).reshape(-1, order='F')
gene_list
```

```
array([400, 300, 400, 100, 600, 600, 400, 400, 100, 100, 'entropy',
       'gini', 'entropy', 'entropy', 'gini', 'gini', 'gini', 'entropy',
       'entropy', 'entropy', 30, 40, 10, None, 10, 10, 40, 10, None, 30,
       10, 10, 4, 10, 2, 2, 2, 4, 10, 4, 30, 15, 5, 5, 15, 15, 10, 5, 5,
       15, 0.2, 0.2, 0.2, 0.0, 0.0, 0.0, 0.0, 0.2, 0.0, 0.2, 'sqrt',
       'log2', None, None, 'auto', 'auto', 'sqrt', None, None, 'log2', 20,
       60, 10, None, None, None, 60, 10, None, 60, 0.3, 0.0, 0.0, 0.3,
       0.3, 0.3, 0.3, 0.0, 0.3, 0.3, False, False, True, False, False,
       False, False, True, False, False], dtype=object)
```

- score 열을 제외한 하이퍼파라미터 값들을 1차원 배열로 저장
- 랜덤하게 선택된 하이퍼파라미터 값 출력

```
pd.DataFrame(gene_list[i].reshape((-1, width), order='F'), columns=pairs.columns[:-1])
```

	n_estimators	criterion	max_depth	min_samples_split	min_samples_leaf	min_weight_fraction_leaf	max_features	max_leaf_nodes	min_impurity_decrease	bootstrap
0	300	entropy	30	10	30	0.2	sqrt	60	0.3	False
1	300	gini	40	10	15	0.2	sqrt	20	0.3	False
2	100	entropy	None	4	5	0.2	None	None	0.3	False
3	400	entropy	10	4	5	0.2	None	10	0.0	True
4	600	gini	10	2	15	0.0	auto	None	0.3	False
5	600	gini	10	2	15	0.0	auto	None	0.3	False
6	400	entropy	10	4	5	0.2	None	10	0.0	False
7	400	entropy	10	4	5	0.0	None	60	0.3	False
8	100	entropy	30	10	15	0.0	log2	60	0.3	False
9	100	entropy	None	4	5	0.0	None	60	0.3	False

- 앞에서 생성한 'i' 배열을 맵핑시켜 교배
- 랜덤하게 교배된 새로운 자손 출력

```
children = crossover(pairs, random_state = 42)
if 'score' in children.columns:
    children.drop('score', axis=1, inplace=True)
children
```

2. 전체적인 알고리즘 구현

(3) 돌연변이

```python
def mutate(population, mutation_rate, rf_parameters, random_state):
    '''
    population : 돌연변이할 파라미터(자손)
    mutation_rate : 돌연변이율
    rf_parameters : hyper-parameter 리스트
    random_state : 랜덤 고정 상수
    '''

    # children의 행, 열 개수 저장
    length, width = population.shape
    # 랜덤 시드 고정
    np.random.seed(random_state)

    # 전체 파라미터를 1차원 배열로 flatten
    pop_array = np.array(population).reshape(-1, order='F')

    # 돌연변이가 발생할 인덱스 리스트 생성
    change_indices = np.random.choice(
        np.arange(length * width),
        size = int(mutation_rate * length * width),
        replace = False
    )

    # 돌연변이가 발생할 새 데이터 리스트 생성
    change_indices.sort()
    new_values = []

    for i in range(width):
        # change_indices에서 i번째 열에 해당하는 인덱스 범위 추출
        indices_in_column = change_indices[(i * length <= change_indices) & (change_indices < (i + 1) * length)]
        # print(f'indices_in_column: {indices_in_column}')
        # 해당 인덱스 수 만큼 랜덤 값을 선택하여 new_values에 추가
        new_values.extend(np.random.choice(rf_parameters[i], size=len(indices_in_column)))

    # 돌연변이가 발생할 랜덤 인덱스에 새 데이터로 업데이트(돌연변이)
    pop_array[change_indices] = new_values
    # mutants(return)는 자손에서 돌연변이 시킨 parameter DataFrame
    mutants = pd.DataFrame(pop_array.reshape((-1, width), order='F'))
    mutants.columns = population.columns

    return mutants
```

★ 돌연변이 함수

- **mutate(population, mutation_rate, rf_parameters, random_state)**

  : 교배로 생성된 자손을 돌연변이율에 따른 개수로 랜덤 돌연변이 수행

2. 전체적인 알고리즘 구현

(3) 돌연변이

```
pop_array = np.array(children).reshape(-1, order='F')
pop_array
```

```
array([400, 400, 400, 100, 600, 600, 400, 400, 100, 100, 'entropy',
       'gini', 'entropy', 'entropy', 'gini', 'gini', 'entropy', 'gini',
       'entropy', 'entropy', 40, 40, None, 10, 10, 10, 10, 40, 30, None,
       10, 10, 10, 4, 2, 2, 4, 4, 10, 4, 30, 15, 5, 5, 15, 15, 5, 5, 15,
       5, 0.2, 0.2, 0.2, 0.2, 0.0, 0.0, 0.2, 0.2, 0.0, 0.2, 'sqrt',
       'log2', None, None, 'auto', 'auto', None, None, 'log2', None, 60,
       20, 10, 10, None, None, 10, 60, None, None, 0.3, 0.3, 0.0, 0.0,
       0.3, 0.3, 0.0, 0.0, 0.3, 0.3, False, False, False, True, False,
       False, True, False, False, False], dtype=object)
```

- ‘children’ 데이터 프레임을 1차원 배열로 변환

```
length, width = children.shape
mutation_rate = 0.2
change_indices = np.random.choice(
        np.arange(length * width),
        size = int(mutation_rate * length * width),
        replace=False
)
change_indices
```

- 0부터 99까지의 숫자 중에서 20%에 해당하는 개수만큼의 숫자를 비복원 추출하여 change_indices에 저장
- 돌연변이율을 20%로 임의 지정

```
array([53, 49, 78, 56, 98, 29,  1, 82,  5, 35, 19, 42, 15, 67, 24, 31, 20,
       51,  3, 16])
```

- 돌연변이 될 값의 인덱스(위치)

2. 전체적인 알고리즘 구현

(3) 돌연변이

```
change_indices.sort()
new_values = []

for i in range(width):
    # change_indices에서 i번째 열에 해당하는 인덱스 범위 추출
    indices_in_column = change_indices[(i * length <= change_indices) & (change_indices < (i + 1) * length)]
    print(f'indices_in_column: {indices_in_column}')
    # 해당 인덱스 수 만큼 랜덤 값을 선택하여 new_values에 추가
    new_values.extend(np.random.choice(rf_parameters[i], size=len(indices_in_column)))
new_values
```

```
indices_in_column: [1 3 5]
indices_in_column: [15 16 19]
indices_in_column: [20 24 29]
indices_in_column: [31 35]
indices_in_column: [42 49]
indices_in_column: [51 53 56]
indices_in_column: [67]
indices_in_column: [78]
indices_in_column: [82]
indices_in_column: [98]
```

- 각 열에서 돌연변이 될 인덱스

```
[300,
 600,
 600,
 'entropy',
 'entropy',
 'entropy',
 20,
 10,
 30,
 2,
 10,
 10,
 5,
 0.0,
 0.0,
 0.0,
 'sqrt',
 None,
 0.3,
 True]
```

- 돌연변이 된 값

- 돌연변이 된 값으로 갱신

```
pop_array[change_indices] = new_values
pop_array
```

```
array([400, 300, 400, 600, 600, 600, 400, 400, 100, 100, 'entropy',
       'gini', 'entropy', 'entropy', 'gini', 'entropy', 'entropy', 'gini',
       'entropy', 'entropy', 20, 40, None, 10, 10, 10, 10, 40, 30, 30, 10,
       2, 10, 4, 2, 10, 4, 4, 10, 4, 30, 15, 10, 5, 15, 15, 5, 5, 15, 5,
       0.2, 0.0, 0.2, 0.0, 0.0, 0.0, 0.0, 0.2, 0.0, 0.2, 'sqrt', 'log2',
       None, None, 'auto', 'auto', None, 'sqrt', 'log2', None, 60, 20, 10,
       10, None, None, 10, 60, None, None, 0.3, 0.3, 0.3, 0.0, 0.3, 0.3,
       0.0, 0.0, 0.3, 0.3, False, False, False, True, False, False, True,
       False, True, False], dtype=object)
```

2. 전체적인 알고리즘 구현

(3) 돌연변이

```
mutants = pd.DataFrame(pop_array.reshape((-1, width), order='F'))
mutants.columns = children.columns
mutants
```

	n_estimators	criterion	max_depth	min_samples_split	min_samples_leaf	min_weight_fraction_leaf	max_features	max_leaf_nodes	min_impurity_decrease	bootstrap
0	400	entropy	20	10	30	0.2	sqrt	60	0.3	False
1	300	gini	40	2	15	0.0	log2	20	0.3	False
2	400	entropy	None	10	10	0.2	None	10	0.3	False
3	600	entropy	10	4	5	0.0	None	10	0.0	True
4	600	gini	10	2	15	0.0	auto	None	0.3	False
5	600	entropy	10	10	15	0.0	auto	None	0.3	False
6	400	entropy	10	4	5	0.0	None	10	0.0	True
7	400	gini	40	4	5	0.2	sqrt	60	0.0	False
8	100	entropy	30	10	15	0.0	log2	None	0.3	True
9	100	entropy	30	4	5	0.2	None	None	0.3	False

- 데이터프레임으로 변환
- ‘mutants’는 ‘children’에서 돌연변이 시킨 DataFrame

```
mutants = mutate(children, mutation_rate = 0.2, rf_parameters = rf_parameters, random_state = 42)
mutants
```

★ 함수 사용

- children(자손)의 DataFrame을 돌연변이 처리

2. 전체적인 알고리즘 구현

(4) Population 갱신

- population에 children과 mutants를 추가하고 children의 열을 기준으로 중복 행 제거
- 새로 추가된 parameter 세트의 score 값은 NaN

```
# population에 children과 mutants를 추가하고 children의 열을 기준으로 중복 행 제거
population = pd.concat([population, children, mutants], ignore_index=True, sort=False).drop_duplicates(subset=children.columns)
population
```

	n_estimators	criterion	max_depth	min_samples_split	min_samples_leaf	min_weight_fraction_leaf	max_features	max_leaf_nodes	min_impurity_decrease	bootstrap	score
0	400	gini	40	2	10	0.0	sqrt	60	0.3	False	0.500000
1	600	gini	10	2	15	0.0	auto	None	0.3	False	0.500000
2	100	gini	30	10	30	0.0	sqrt	60	0.3	False	0.500000
3	400	gini	10	10	15	0.0	auto	20	0.3	False	0.500000
4	400	entropy	30	10	30	0.2	sqrt	20	0.3	False	0.500000
5	600	gini	40	4	30	0.2	None	10	0.3	False	0.500000
6	100	entropy	None	10	5	0.0	None	None	0.3	False	0.500000
7	100	entropy	30	4	15	0.2	log2	60	0.3	False	0.500000
8	400	entropy	10	4	5	0.2	None	10	0.0	True	0.829825
9	300	gini	40	10	15	0.2	log2	60	0.0	False	0.658333
10	400	entropy	40	10	30	0.2	sqrt	60	0.3	False	NaN
11	400	gini	40	10	15	0.2	log2	20	0.3	False	NaN
12	400	entropy	None	10	5	0.2	None	10	0.0	False	NaN
13	100	entropy	10	4	5	0.2	None	10	0.0	True	NaN
17	400	gini	40	4	5	0.2	None	60	0.0	False	NaN
18	100	entropy	30	10	15	0.0	log2	None	0.3	False	NaN
19	100	entropy	None	4	5	0.2	None	None	0.3	False	NaN
20	400	entropy	40	10	30	0.2	sqrt	20	0.3	False	NaN
21	400	gini	40	2	15	0.2	log2	20	0.3	False	NaN
22	400	gini	None	10	5	0.2	None	10	0.0	False	NaN
23	100	entropy	10	4	5	0.2	None	None	0.0	True	NaN
24	100	gini	10	2	10	0.0	auto	None	0.3	False	NaN
25	600	gini	10	2	5	0.0	auto	None	0.3	False	NaN
26	400	entropy	10	4	5	0.2	None	60	0.0	True	NaN
28	100	gini	30	10	15	0.0	log2	None	0.3	False	NaN
29	100	entropy	None	10	5	0.2	None	None	0.3	False	NaN

- 새로 추가된 parameter 세트에 대해서만 score 계산

```
# 새로 추가된 parameter 세트에 대해서만 score 계산
population.loc[population['score'].isna(), 'score'] = population[population['score'].isna()].apply(objective_function, axis=1)
# population.loc[:, 'score'] = population.apply(objective_function, axis=1)
population
```

	n_estimators	criterion	max_depth	min_samples_split	min_samples_leaf	min_weight_fraction_leaf	max_features	max_leaf_nodes	min_impurity_decrease	bootstrap	score
0	400	gini	40	2	10	0.0	sqrt	60	0.3	False	0.500000
1	600	gini	10	2	15	0.0	auto	None	0.3	False	0.500000
2	100	gini	30	10	30	0.0	sqrt	60	0.3	False	0.500000
3	400	gini	10	10	15	0.0	auto	20	0.3	False	0.500000
4	400	entropy	30	10	30	0.2	sqrt	20	0.3	False	0.500000
5	600	gini	40	4	30	0.2	None	10	0.3	False	0.500000
6	100	entropy	None	10	5	0.0	None	None	0.3	False	0.500000
7	100	entropy	30	4	15	0.2	log2	60	0.3	False	0.500000
8	400	entropy	10	4	5	0.2	None	10	0.0	True	0.829825
9	300	gini	40	10	15	0.2	log2	60	0.0	False	0.658333
10	400	entropy	40	10	30	0.2	sqrt	60	0.3	False	0.500000
11	400	gini	40	10	15	0.2	log2	20	0.3	False	0.500000
12	400	entropy	None	10	5	0.2	None	10	0.0	False	0.734211
13	100	entropy	10	4	5	0.2	None	10	0.0	True	0.829825
17	400	gini	40	4	5	0.2	None	60	0.0	False	0.734211
18	100	entropy	30	10	15	0.0	log2	None	0.3	False	0.500000
19	100	entropy	None	4	5	0.2	None	None	0.3	False	0.500000
20	400	entropy	40	10	30	0.2	sqrt	20	0.3	False	0.500000
21	400	gini	40	2	15	0.2	log2	20	0.3	False	0.500000
22	400	gini	None	10	5	0.2	None	10	0.0	False	0.734211
23	100	entropy	10	4	5	0.2	None	None	0.0	True	0.829825
24	100	gini	10	2	10	0.0	auto	None	0.3	False	0.500000
25	600	gini	10	2	5	0.0	auto	None	0.3	False	0.500000
26	400	entropy	10	4	5	0.2	None	60	0.0	True	0.829825
28	100	gini	30	10	15	0.0	log2	None	0.3	False	0.500000
29	100	entropy	None	10	5	0.2	None	None	0.3	False	0.500000

2. 전체적인 알고리즘 구현

(4) Population 갱신

- score 기준으로 내림차순 정렬하여 population_size(=10) 수 만큼 population 갱신

  -> 좋은 population을 다음 세대에 보내줌

```
# score 기준 내림차순 정렬 후 population_size 수 만큼만 새롭게 갱신
population = population.sort_values(by=['score'], ascending=False).iloc[:population_size]
population
```

	n_estimators	criterion	max_depth	min_samples_split	min_samples_leaf	min_weight_fraction_leaf	max_features	max_leaf_nodes	min_impurity_decrease	bootstrap	score
13	100	entropy	10	4	5	0.2	None	10	0.0	True	0.829825
26	400	entropy	10	4	5	0.2	None	60	0.0	True	0.829825
23	100	entropy	10	4	5	0.2	None	None	0.0	True	0.829825
8	400	entropy	10	4	5	0.2	None	10	0.0	True	0.829825
22	400	gini	None	10	5	0.2	None	10	0.0	False	0.734211
12	400	entropy	None	10	5	0.2	None	10	0.0	False	0.734211
17	400	gini	40	4	5	0.2	None	60	0.0	False	0.734211
9	300	gini	40	10	15	0.2	log2	60	0.0	False	0.658333
18	100	entropy	30	10	15	0.0	log2	None	0.3	False	0.500000
28	100	gini	30	10	15	0.0	log2	None	0.3	False	0.500000

2. 전체적인 알고리즘 구현

(5) Best Parameter Set 반환

```
# score가 가장 높은 parameter 세트를 반환
output_score = []
output_params = population.iloc[0]
# output_params[error_metric] = round(output_params['score'], 2)
output_score.append(output_params['score'])
output_params.drop('score', inplace=True)
print(output_score)
output_params
```

```
[0.8298245614035088]
n_estimators                100
criterion                entropy
max_depth                   10
min_samples_split            4
min_samples_leaf             5
min_weight_fraction_leaf   0.2
max_features              None
max_leaf_nodes              10
min_impurity_decrease      0.0
bootstrap                 True
Name: 13, dtype: object
```

→ 반환된 Best Parameter Set

2. 전체적인 알고리즘 구현

(6) GA_HPO Optimizer 클래스 구현

```
class optimizer():
    def __init__(self):
        self.output_score = []
    def best_estimator(self, X_train, X_test, y_train, y_test, rf_parameters, error_metric, population_size,
                       number_of_generation, mutation_rate, random_state=42):

        '''
        X_train, X_test, y_train, y_test : X, y의 train, test data (**DataFrame)
        rf_parameters : 모델에 최적화할 파라미터 (범위) 입력
        rf_parameters = {'n_estimators': [2,3,4,.....1000],
            'max_features': ['sqrt', 'auto', 'log2', None],
            'min_samples_leaf': [2,3,4,5,6,.....16],
            'max_depth': [2,3,4,5,6,.....20],
            'criterion': ['gini', 'entropy'],
            'oob_score': [True, False]
            }
        error_metric : 평가 지표
        population_size :초기해 크기
        number_of_generation : 최대 세대수
        mutation_rate : 돌연변이율
        random_state : 랜덤 고정 상수
        '''

        # ----------------------------------------  Model function  ----------------------------------------
        ## i라는 row에 대한 평가값을 계산하는 목적 함수
        objective_function = lambda i: self.objective(error_metric, i, X_train, X_test, y_train, y_test,
                                                      random_state)
        # ------------------------------------------------------------------------------------------------------
        # ----------------------------------------  초기해 생성  ----------------------------------------
        population = pd.DataFrame()  # parameter들로만 DataFrame 생성
        np.random.seed(random_state)
        population['n_estimators'] = np.random.choice(rf_parameters[0], size=population_size)
        population['criterion'] = np.random.choice(rf_parameters[1], size=population_size)
        population['max_depth'] = np.random.choice(rf_parameters[2], size=population_size)
        population['min_samples_split'] = np.random.choice(rf_parameters[3], size=population_size)
        population['min_samples_leaf'] = np.random.choice(rf_parameters[4], size=population_size)
        population['min_weight_fraction_leaf'] = np.random.choice(rf_parameters[5], size=population_size)
        population['max_features'] = np.random.choice(rf_parameters[6], size=population_size)
        population['max_leaf_nodes'] = np.random.choice(rf_parameters[7], size=population_size)
        population['min_impurity_decrease'] = np.random.choice(rf_parameters[8], size=population_size)
        population['bootstrap'] = np.random.choice(rf_parameters[9], size=population_size)

        population['score'] = population.apply(objective_function, axis=1)
        self.output_score = []
        # ----------------------------------------  iteration  ----------------------------------------
        for g in range(number_of_generation):
            # 부모 쌍 생성
            pairs = self.selection(population, random_state)

            # 교배로 자손 생성
            children = self.crossover(pairs, random_state)
            if 'score' in children.columns:
                children.drop('score', axis=1, inplace=True)

            # 돌연변이 시행
            mutants = self.mutate(children, mutation_rate, rf_parameters, random_state)

            # 자손과 돌연변이를 population에 추가 후 자손의 열을 기준으로 중복되는 값 제거하여 population Update
            population = pd.concat([population, children, mutants], ignore_index=True, sort=False).drop_duplicates(subset=children.columns)

            # 새롭게 추가된 population에 대해서 평가
            # population.loc[:, error_metric] = population.apply(objective_function, axis=1)
            population.loc[population['score'].isna(), 'score'] = population[population['score'].isna()].apply(objective_function, axis=1)

            # population을 score 기준으로 내림차순 정렬
            population = population.sort_values(by=['score'], ascending=False).iloc[:population_size]

            # 가장 좋은 parameter 세트 선택
            output_params = population.iloc[0]
            # output_params[error_metric] = round(output_params['score'], 2)

            self.output_score.append(output_params['score'])
            output_params.drop('score', inplace=True)

        return [self.get_model(population.loc[population['score'].idxmax()]), output_params]
```

```
    # 룰렛 휠 선택
    def selection(self, population, random_state):
        '''
        population : 전체 population
        random_state : 랜덤 고정 상수

        ** 높은 score를 가지는 parameter set가 높은 확률로 선택될 수 있도록 Roulette Wheel Selection **
        '''
        # 랜덤 시드 고정
        np.random.seed(random_state)
        # population 개수 저장
        length = population.shape[0]
        # 각 행의 score의 확률 계산
        prob = population['score'] / population['score'].sum()
        # 각 행의 score 확률에 기반해 랜덤 index 선택(size 개수만큼 선택)
        indices = np.random.choice(np.arange(length), p=prob, size=length)

        return population.iloc[indices]
```

2. 전체적인 알고리즘 구현

(6) GA_HPO Optimizer 클래스 구현

```python
# 교배
def crossover(self, pairs, random_state):
    '''
    pairs : roughlette_wheel_selection으로 선택된 population 쌍
    random_state : 랜덤 고정 상수

    ** 전체 popuation을 기준으로 무작위 교차 **
    '''
    # 랜덤 시드 고정
    np.random.seed(random_state)
    # pairs의 행, 열 개수 저장
    length, width = pairs.shape
    # 열 개수 -1  --> 마지막 열 = 'score'이므로 score는 제외하고 교배
    width -= 1

    # 교배율X --> (0, 1)를 각 parameter마다 랜덤 생성하여 새로운 배열의 자손 생성
    ## 0, 1을 전체 parameter 수 만큼 랜덤 생성
    a = np.random.choice((0, 1), size=length * width)
    ## 교배(교차)를 시키기 위한 단계(짝수번 째에 -1을 곱함)
    a[np.arange(1, length * width, 2)] *= -1
    ## 새로운 배열을 생성할 인덱스 'i' 리스트 생성
    i = np.arange(length * width) + a

    # i 리스트를 이용하여 새로운 자손 생성
    ## score열을 제외하고 DataFrame을 flatten 시켜 1차원 배열로 생성
    gene_list = np.array(pairs.drop('score', axis=1)).reshape(-1, order='F')
    ## 1차원 배열로 생성된 parameter 값들에 i 리스트에 해당하는 값들로 새롭게 자손 parameter 세트 생성
    return pd.DataFrame(gene_list[i].reshape((-1, width)), order='F'), columns=pairs.columns[:-1])
```

```python
# 돌연변이
def mutate(self, population, mutation_rate, rf_parameters, random_state):
    '''
    population : 전체 population
    mutation_rate : 돌연변이율
    rf_parameters : hyper-parameter 리스트
    random_state : 랜덤 고정 상수

    ** 돌연변이율에 따라 population 돌연변이 **
    '''
    # children의 행, 열 개수 저장
    length, width = population.shape
    # 랜덤 시드 고정
    np.random.seed(random_state)

    # 전체 파라미터를 1차원 배열로 flatten
    pop_array = np.array(population).reshape(-1, order='F')

    # 돌연변이가 발생할 인덱스 리스트 생성
    change_indices = np.random.choice(
        np.arange(length * width),
        size = int(mutation_rate * length * width),
        replace = False)

    # 돌연변이가 발생할 새 데이터 리스트 생성
    change_indices.sort()
    new_values = []

    for i in range(width):
        # change_indices에서 i번째 열에 해당하는 인덱스 범위 추출
        indices_in_column = change_indices[(i * length <= change_indices) & (change_indices < (i + 1) * length)]
        # 해당 인덱스 수 만큼 랜덤 값을 선택하여 new_values에 추가
        new_values.extend(np.random.choice(rf_parameters[i], size=len(indices_in_column)))

    # 돌연변이가 발생할 랜덤 인덱스에 새 데이터로 업데이트(돌연변이)
    pop_array[change_indices] = new_values
    # mutants(return)는 자손에서 돌연변이 시킨 parameter DataFrame
    mutants = pd.DataFrame(pop_array.reshape((-1, width), order='F'))
    mutants.columns = population.columns

    return mutants
```

```python
# 모델 return
def get_model(self, row, random_state=42):
    '''
    row : 평가값이 가장 좋은 population의 row
    random_state : 랜덤 고정 상수

    ** best hyper-parameter를 갖는 모델을 return **
    '''

    return RandomForestClassifier(
        n_estimators = int(row[0]),
        criterion = row[1],
        max_depth = row[2],
        min_samples_split = int(row[3]),
        min_samples_leaf = int(row[4]),
        min_weight_fraction_leaf = float(row[5]),
        max_features = row[6],
        max_leaf_nodes = row[7],
        min_impurity_decrease = float(row[8]),
        bootstrap = row[9],
        random_state = random_state,
        n_jobs = -1)

# 평가값 계산
def evaluation(self, model, error_metric, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    # predictions = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]

    if error_metric == 'accuracy_score':
        eval = accuracy_score(y_test, y_pred_proba)
    elif error_metric == 'f1_score':
        eval = f1_score(y_test, y_pred_proba, average='micro')
    elif error_metric == 'roc_auc_score':
        eval = roc_auc_score(y_test, y_pred_proba, average='macro')    # average = [None, 'micro', 'macro', 'weighted']

    return eval
```

```python
# objective function에 적용되는 평가값 계산하여 반환
def objective(self, error_metric, row, X_train, X_test, y_train, y_test, random_state):
    '''
    ** score를 계산 **
    '''

    if 'score' in row and row['score'] == row['score']:  # row[error_metric] == row[error_metric]
        return row['score']

    return self.evaluation(self.get_model(row, random_state=random_state), error_metric,
                           X_train, X_test, y_train, y_test)

def get_output_score(self):
    return self.output_score
```

2. 전체적인 알고리즘 구현

(7) RandomForest Classifier_HPO 함수 구현

```
def optimize_rfc(X_train, X_test, y_train, y_test, params = {}, error_metric = 'roc_auc_score', population_size = 50,
                 number_of_generation = 10, mutation_rate = 0.05, random_state = 42):
    try:
        rf_parameters = []

        if 'n_estimators' in params.keys():
            rf_parameters.append(params['n_estimators'])
        else:
            rf_parameters.append(np.arange(1, 1000))

        if 'criterion' in params.keys():
            rf_parameters.append(params['criterion'])
        else:
            rf_parameters.append(['gini', 'entropy'])

        if 'max_depth' in params.keys():
            rf_parameters.append(params['max_depth'])
        else:
            rf_parameters.append(np.arange(2, 20))

        # min_samples_split 구현
        if 'min_samples_split' in params.keys():
            rf_parameters.append(params['min_samples_split'])
        else:
            rf_parameters.append(2)

        if 'min_samples_leaf' in params.keys():
            rf_parameters.append(params['min_samples_leaf'])
        else:
            rf_parameters.append(np.arange(1, 20))
```

★ RandomForest Classifier_HPO 함수

- **optimize_rfc(X_train, X_test, y_train, y_test, params = {}, error_metric = 'roc_auc_score', population_size = 50, number_of_generation = 10, mutation_rate = 0.05, random_state = 42)**
  : RandomForest Classifier의 하이퍼파라미터를 최적화하여 최적화된 모델과 최적 하이퍼파라미터 출력

2. 전체적인 알고리즘 구현

(7) RandomForest Classifier_HPO 함수 구현

```
# min_weight_fraction_leaf 구현
if 'min_weight_fraction_leaf' in params.keys():
    rf_parameters.append(params['min_weight_fraction_leaf'])
else:
    rf_parameters.append(np.arange(0.0, 0.5, 0.02))

if 'max_features' in params.keys():
    rf_parameters.append(params['max_features'])
else:
    rf_parameters.append(['sqrt', 'auto', 'log2', None])

# max_leaf_nodes 구현
if 'max_leaf_nodes' in params.keys():
    rf_parameters.append(params['max_leaf_nodes'])
else:
    rf_parameters.append(None)

# min_impurity_decrease 구현
if 'min_impurity_decrease' in params.keys():
    rf_parameters.append(params['min_impurity_decrease'])
else:
    rf_parameters.append(np.arange(0.0, 1.0, 0.05))

# bootstrap 구현
if 'bootstrap' in params.keys():
    rf_parameters.append(params['bootstrap'])
else:
    rf_parameters.append([True, False])
```

```
    obj = optimizer()
    model, row = obj.best_estimator(X_train, X_test, y_train, y_test, rf_parameters, error_metric,
                                    population_size, number_of_generation, mutation_rate, random_state)

    output_score = obj.get_output_score()
    plt.plot(range(len(output_score)), output_score)
    plt.xlabel('Generation')
    plt.ylabel('Score')
    plt.title('Score Over Generation')
    plt.grid(True)
    plt.show()

    return model, row
except Exception as e:
    print("Exception parameter error:",e)
    return None, None
```

2. 전체적인 알고리즘 구현

(8) 알고리즘 검증

```
start_time = time.time()

train = pd.read_csv('/content/drive/MyDrive/지능형알고리즘/dataset/train.csv')

X = train.drop(['person_id', 'login'], axis=1)
y = train['login']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    shuffle = True, stratify = y, random_state=42)

# params = {
#     'n_estimators': [100, 300, 600, 800, 850, 900, 950, 1000],
#     'criterion': ['gini', 'entropy'],
#     'max_depth': [1, 10, 20, 35, 50],
#     'min_samples_split' : [2, 10, 20, 30],
#     'min_samples_leaf': [5,30,50],
#     'min_weight_fraction_leaf': [0.0, 0.2,0.35,0.5],
#     'max_features':['auto','sqrt','log2',],
#     'max_leaf_nodes':[10, 60, 100],
#     'min_impurity_decrease':[0.0, 0.3, 0.5],
#     'bootstrap': [True,False]
# }

# param_search_space = {
#     'n_estimators': [10, 50, 100,150,200,250,300,350,400,450,500,550,600,650,700,750,800,850,900,950,1000],
#     'critertion': ['gini','entropy'],
#     'max_depth': [None, 1,5,10,15,20,25,30,35,40,45,50],
#     'min_samples_split' : [2,4,6,8,10,12,14,16,18,20,22,24,26,28,30],
#     'min_samples_leaf': [1, 5,10,15,20,25,30,35,40,45,50],
#     'min_weight_fraction_leaf': [0.0, 0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5],
#     'max_features':['auto','sqrt','log2',None],
#     'max_leaf_nodes':[None,10,20,30,40,50,60,70,80,90,100],
#     'min_impurity_decrease':[0.0, 0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5],
#     'bootstrap': [True,False]
# }
```

- 최적의 랜덤 포레스트 모델 탐색
- optimize_rfc 함수를 호출
  -> 다양한 하이터파라미터 값에 대한 모델 평가, 모델과 하이퍼파라미터 값 반환
- 평가지표: AUC(Area Under the ROC Curve)
- **매 세대마다 가장 좋은 score를 시각화**

실제 성능 비교에 사용할 parameter search 범위

2. 전체적인 알고리즘 구현

(8) 알고리즘 검증

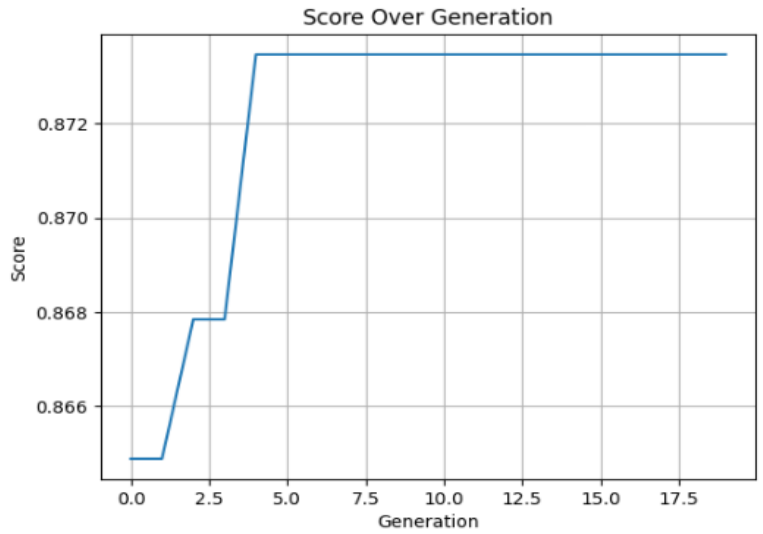
★ 주어진 데이터에 대해 RandomForest Classifier 모델의 최적 하이퍼파라미터 탐색, 실행시간 측정 -> 모델 출력, 평가값 도출

- **최대 세대수 = 20, 초기해 수 = 50, 돌연변이율 = 0.1**

```
params = {
    'n_estimators': [100, 300, 400, 600],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split' : [2, 4, 10],
    'min_samples_leaf': [5,10, 15,30],
    'min_weight_fraction_leaf': [0.0, 0.2],
    'max_features':['auto','sqrt','log2',None],
    'max_leaf_nodes':[None, 10, 20, 60],
    'min_impurity_decrease':[0.0, 0.3],
    'bootstrap': [True,False]
}

model, hyp_param = optimize_rfc(X_train=X_train, y_train=y_train, y_test=y_test, X_test=X_test,
                                params=params, number_of_generation=20, population_size=50,
                                error_metric='roc_auc_score', mutation_rate=0.1, random_state = 42)
end_time = time.time()
print(hyp_param)
print(end_time - start_time)
```

실험 parameter search 범위




```
n_estimators                100
criterion                entropy
max_depth                   30
min_samples_split            4
min_samples_leaf            15
min_weight_fraction_leaf   0.0
max_features              None
max_leaf_nodes              10
min_impurity_decrease      0.0
bootstrap                 True
Name: 0, dtype: object
468.52419686317444
```

```
model
```

```
RandomForestClassifier(criterion='entropy', max_depth=30, max_features=None,
                       max_leaf_nodes=10, min_samples_leaf=15,
                       min_samples_split=4, random_state=42)
```

```
y_pred_proba = model.predict_proba(X_test)[:, 1]
# predictions = model.predict(X_test)
auc_score = roc_auc_score(y_test, y_pred_proba, average='macro')
auc_score
```

```
0.8734645552760102
```

3. 최종 발표 계획

- 다양한 초기해, 최대 세대수를 설정하여 실험 수행
- Grid Search, Random Search, Optuna 등 다른 HPO Method로 도출한 결과와 비교
- GA_HPO로 도출한 최적 하이퍼파라미터를 DACON Competition에 제출하여 private score 비교 및 리더보드 확인

감사합니다.
Q&A