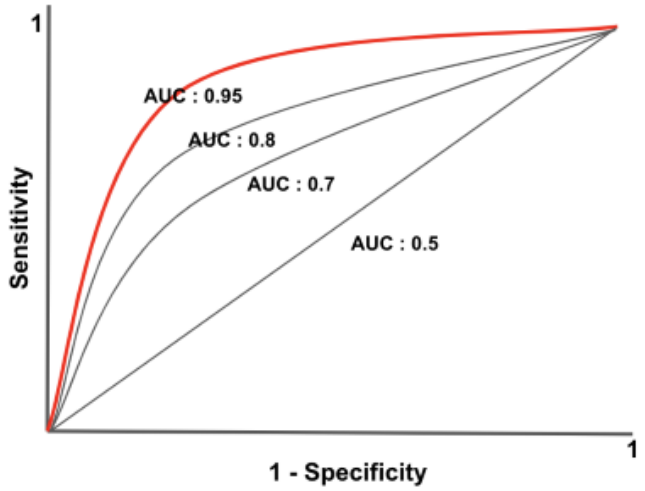
목차 table of contents

1. 프로젝트 소개 및 요약

★ 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

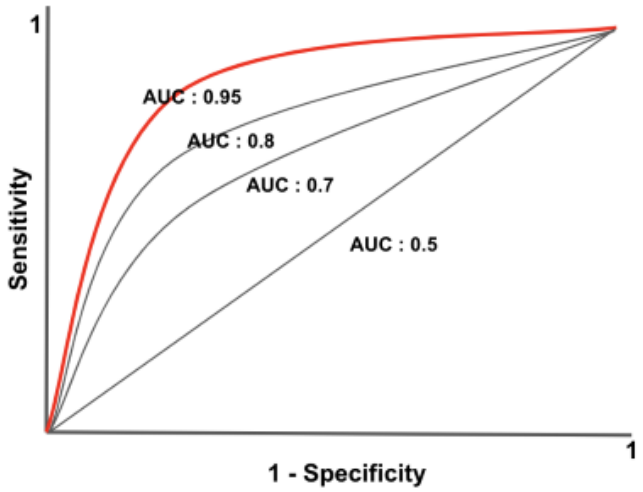


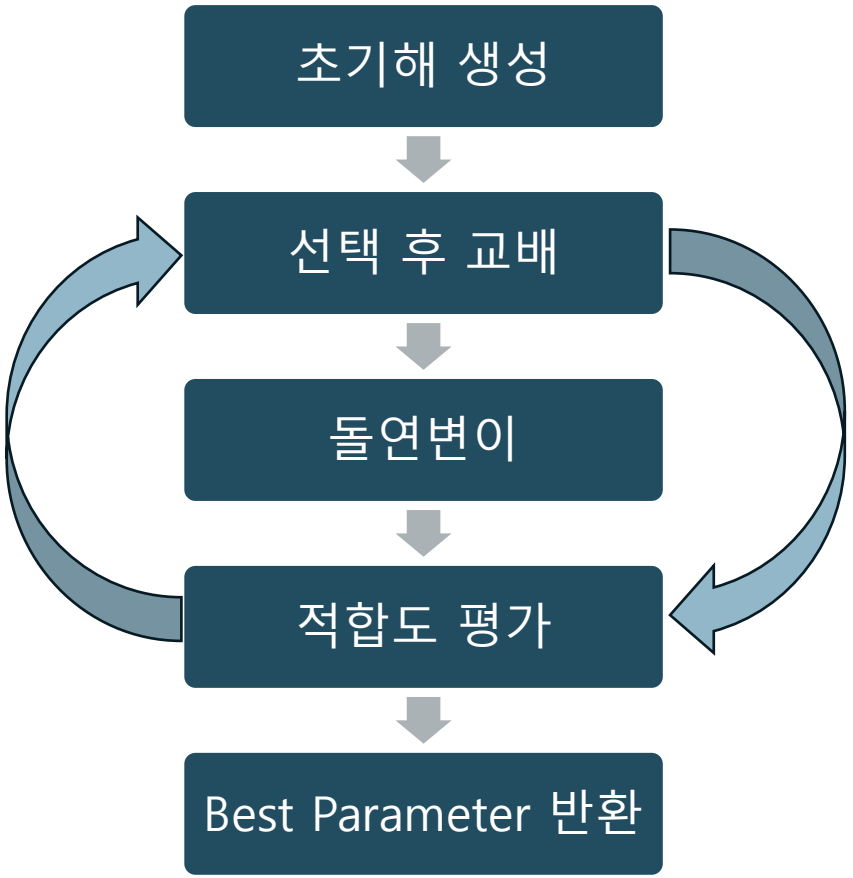
1. 프로젝트 소개 및 요약

★ 해표현: DataFrame의 열(column)을 하이퍼파라미터로 생성했을 때 각 행 값을 갖는 RF(RandomForest)모델

	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

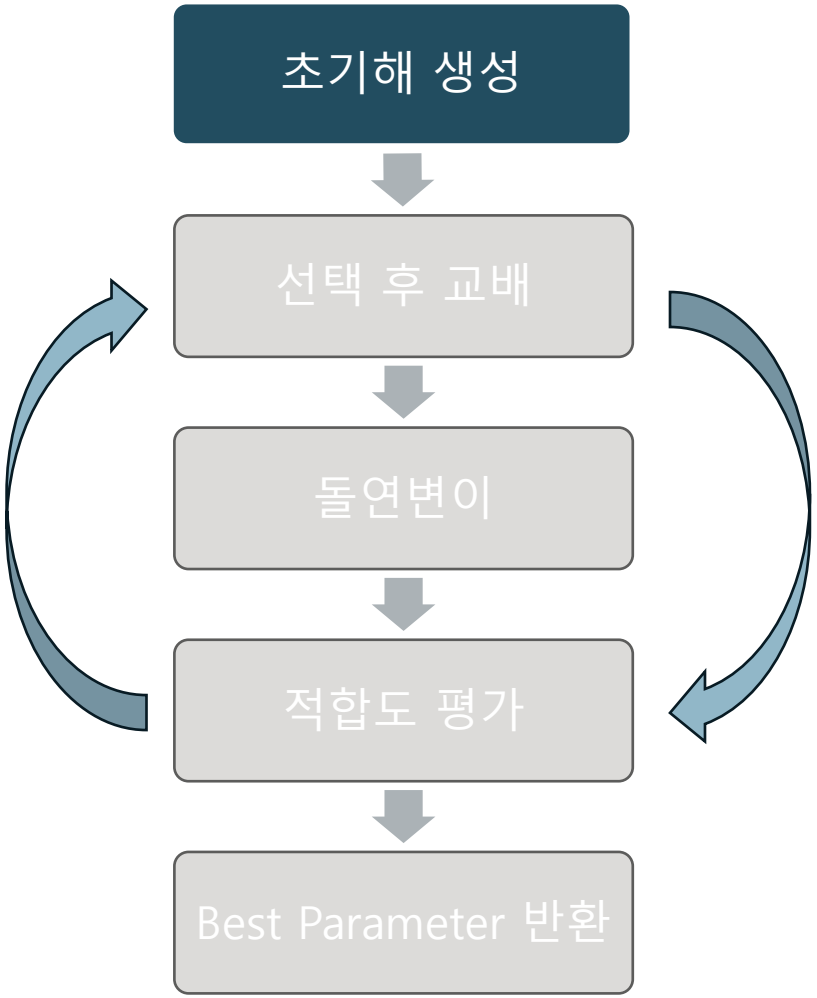
★ 평가 지표: AUC (Area Under the ROC Curve)





1. 프로젝트 소개 및 요약

(1) 초기해 생성




```python
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'] = rf_parameters[0].rvs(size=population_size)
        population['criterion'] = np.random.choice(rf_parameters[1], size=population_size)
        population['max_depth'] = rf_parameters[2].rvs(size=population_size)
        population['min_samples_split'] = rf_parameters[3].rvs(size=population_size)
        population['min_samples_leaf'] = rf_parameters[4].rvs(size=population_size)
        population['min_weight_fraction_leaf'] = rf_parameters[5].rvs(size=population_size)
        population['max_features'] = np.random.choice(rf_parameters[6], size=population_size)
        population['max_leaf_nodes'] = rf_parameters[7].rvs(size=population_size)
        population['min_impurity_decrease'] = rf_parameters[8].rvs(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 = []
```

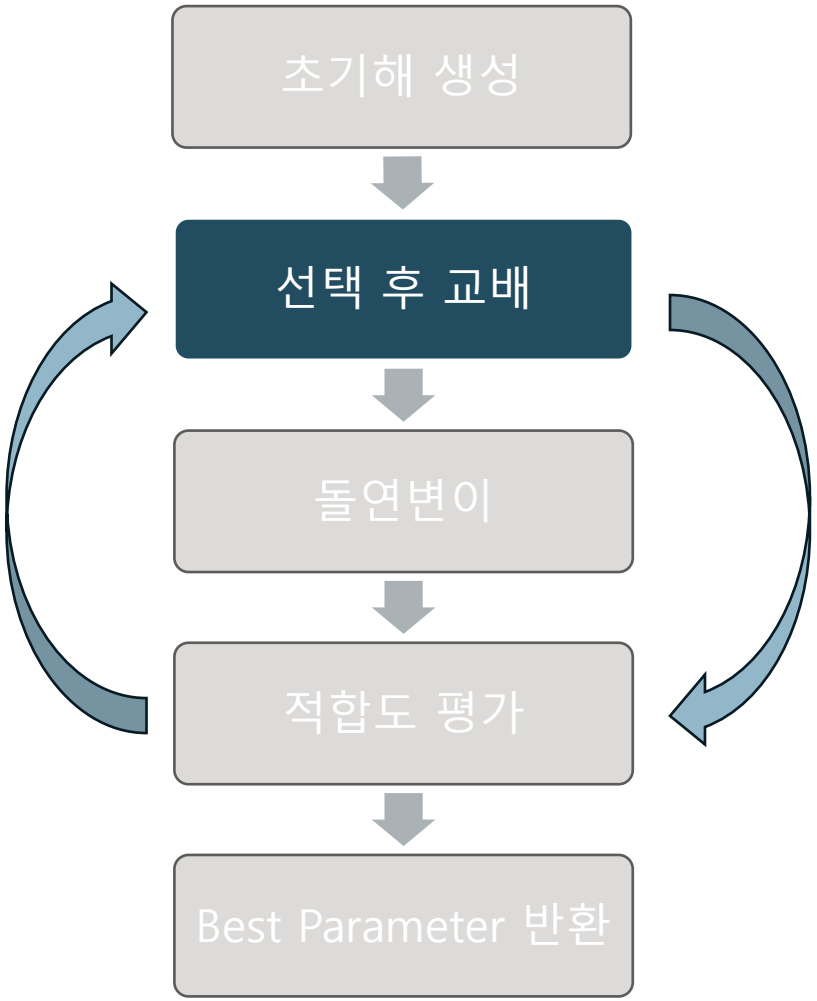
ver.2 : 탐색 범위 내의 값 랜덤 탐색

```python
from scipy.stats import uniform, randint
params= {
    'n_estimators': randint(10, 1000),
    'critertion': ['gini','entropy'],
    'max_depth': randint(1, 100),
    'min_samples_split' : randint(2, 100),
    'min_samples_leaf': randint(1, 100),
    'min_weight_fraction_leaf': uniform(0.0, 0.5),
    'max_features':['sqrt','log2',None],
    'max_leaf_nodes':randint(2, 100),
    'min_impurity_decrease':uniform(0.0, 1.0),
    'bootstrap': [True,False]
}
```

ver.2 : 탐색 범위 내의 값 랜덤 탐색

1. 프로젝트 소개 및 요약

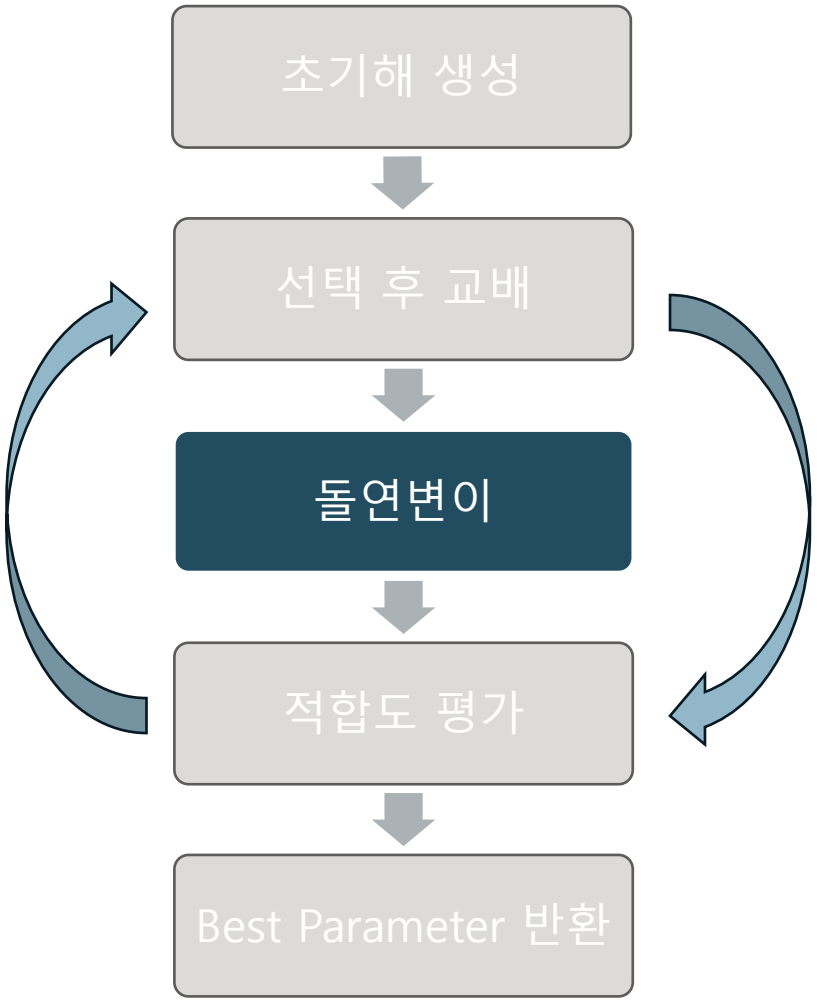
(2) 선택 후 교배




```python
# 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]

# 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)을 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])
```

1. 프로젝트 소개 및 요약

(3) 돌연변이

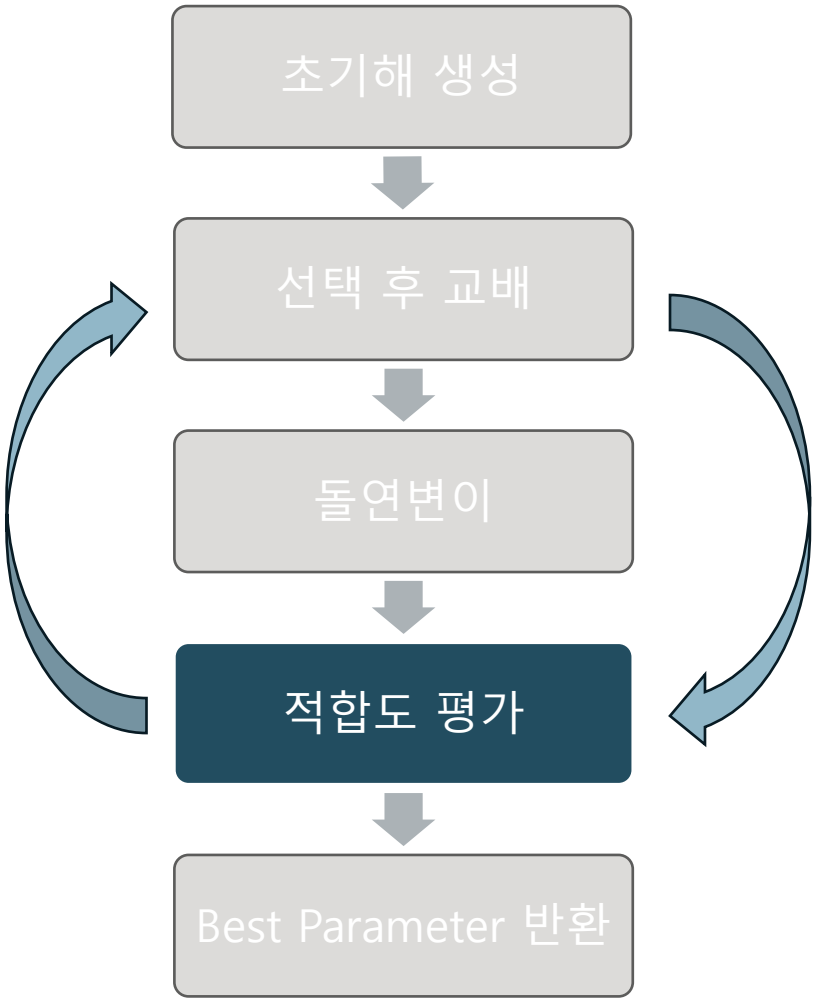



```python
# 돌연변이
def mutate(self, population, mutation_rate, rf_parameters, random_state):
    '''
    population : 전체 population
    mutation_rate : 돌연변이율
    rf_parameters : hyper-parameter 리스트
    random_state : 랜덤 고정 상수

    ** 돌연변이율에 따라 population 돌연변이 **
    Mutate the population with a given mutation rate.
    '''
    # 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
        if i == 1 or i == 6 or i == 9:
            new_values.extend(np.random.choice(rf_parameters[i], size=len(indices_in_column)))
        else:
            new_values.extend(rf_parameters[i].rvs(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
```

ver.2 : 탐색 범위 내의 값 랜덤 탐색

1. 프로젝트 소개 및 요약

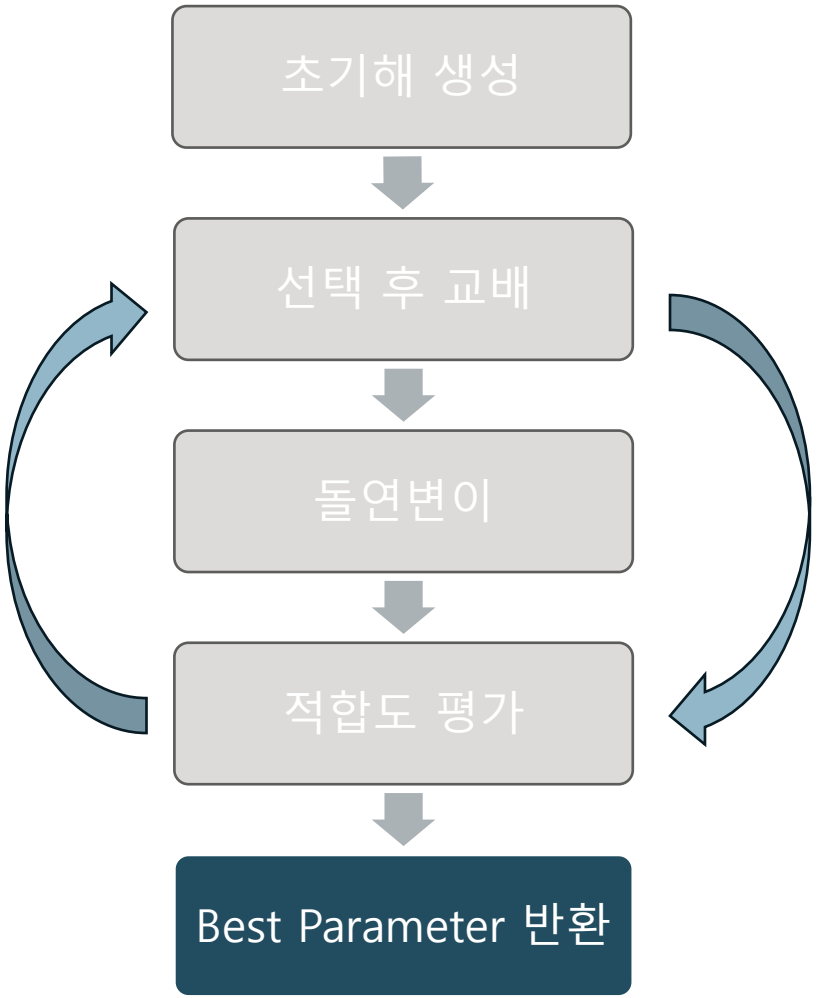
(4) 적합도 평가




```python
# 평가값 계산
def evaluation(self, model, error_metric, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    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')

    return eval

# 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']:
        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
```

1. 프로젝트 소개 및 요약

(5) Best Parameter 반환




```
from scipy.stats import uniform, randint

start_time = time.time()

train = pd.read_csv('../data/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': [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':['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]
# }

params= {
    'n_estimators': randint(10, 1000),
    'critertion': ['gini','entropy'],
    'max_depth': randint(1, 100),
    'min_samples_split' : randint(2, 100),
    'min_samples_leaf': randint(1, 100),
    'min_weight_fraction_leaf': uniform(0.0, 0.5),
    'max_features':['sqrt','log2',None],
    'max_leaf_nodes':randint(2, 100),
    'min_impurity_decrease':uniform(0.0, 1.0),
    '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=200,
                                error_metric='roc_auc_score', mutation_rate=0.2, random_state = 42)
end_time = time.time()
print(hyp_param)
print(end_time - start_time)
```

1. 프로젝트 소개 및 요약

*(6) 2가지 파라미터 탐색 방법

- Ver.1 : 탐색 범위 안의 값을 직접 지정
- Ver.2 : 탐색 범위 지정 후 랜덤으로 탐색 (scipy 라이브러리의 uniform, randint 함수 사용)

Ver.1

```
params = {
    '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]
}
```

Ver.2

```
from scipy.stats import uniform, randint
params= {
    'n_estimators': randint(10, 1000),
    'critertion': ['gini','entropy'],
    'max_depth': randint(1, 100),
    'min_samples_split' : randint(2, 100),
    'min_samples_leaf': randint(1, 100),
    'min_weight_fraction_leaf': uniform(0.0, 0.5),
    'max_features':['sqrt','log2',None],
    'max_leaf_nodes':randint(2, 100),
    'min_impurity_decrease':uniform(0.0, 1.0),
    'bootstrap': [True,False]
}
```

→ Ver.1과 Ver.2로 나누어 구현하였으며 2가지 버전의 방식으로 각각 실험 진행

2. 실험 결과 및 비교 분석

1. GA_HPO를 이용하여 여러 번의 실험을 통해 private score가 제일 좋은 하이퍼파라미터를 도출

2. 최적의 파라미터 값으로 설정하여 Random Search, Grid Search, Optuna의 private score와 valid score를 비교하여 성능 확인

2. 실험 결과 및 비교 분석

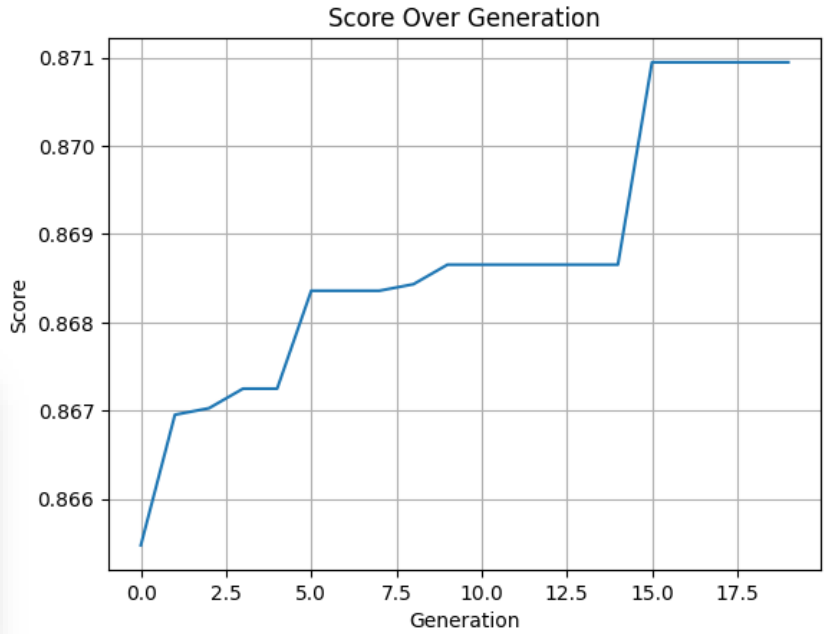
(1) Parameter별 실험 결과

```
params = {
    'n_estimators': [10, 50, 100,150,200,250,300,350,400,450,500,550,600,650,700,750,800,850,900,950,1000],
     'criterion': ['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]
}
```

- 여러 번의 실험 결과, 세대 수가 20 이전에 수렴하기에 최대 세대수를 **20**으로 설정하고 진행
- 돌연변이율은 0.1에서 가장 좋은 성능을 보여 **0.1**로 설정
- 실험결과

Ver.1	Valid Score	Private Score	실행시간
초기해 50	0.87635	0.85726	20분 23초
초기해 100	0.87894	0.7972	31분 40초
초기해 200	0.87095	0.86356 (4등)	60분 11초

n_estimators	100
criterion	entropy
max_depth	10
min_samples_split	20
min_samples_leaf	30
min_weight_fraction_leaf	0.0
max_features	None
max_leaf_nodes	10
min_impurity_decrease	0.0
bootstrap	True



2. 실험 결과 및 비교 분석

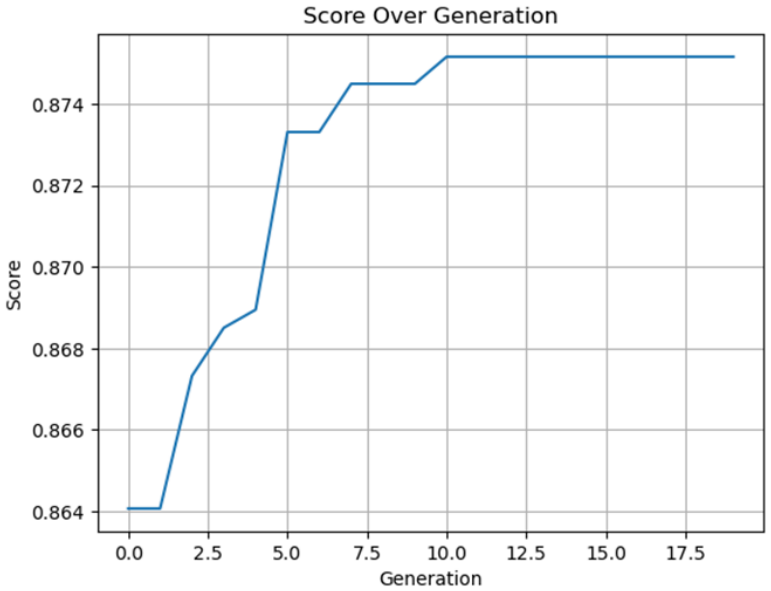
(1) Parameter별 실험 결과

```
params= {
    'n_estimators': randint(10, 1000),
    'criterion': ['gini','entropy'],
    'max_depth': randint(1, 100),
    'min_samples_split' : randint(2, 100),
    'min_samples_leaf': randint(1, 100),
    'min_weight_fraction_leaf': uniform(0.0, 0.5),
    'max_features':['sqrt','log2',None],
    'max_leaf_nodes':randint(2, 100),
    'min_impurity_decrease':uniform(0.0, 1.0),
    'bootstrap': [True,False]
}
```

- 여러 번의 실험 결과, 세대 수가 20 이전에 수렴하기에 최대 세대수를 **20**으로 설정하고 진행
- 돌연변이율은 0.1에서 가장 좋은 성능을 보여 **0.1**로 설정
- 실험결과

Ver.2	Valid Score	Private Score	실행시간
초기해 50	0.87095	0.7997	12분 27초
초기해 100	0.87539	0.79554	35분 52초
초기해 200	0.87516	0.84402 (45등)	52분 41초

n_estimators	227
criterion	entropy
max_depth	93
min_samples_split	96
min_samples_leaf	34
min_weight_fraction_leaf	0.015194
max_features	None
max_leaf_nodes	83
min_impurity_decrease	0.003346
bootstrap	True



2. 실험 결과 및 비교 분석

(2) HPO 방법별 파라미터 설정

★ **Grid Search에서 이전의 파라미터 탐색 범위로 실행 시 메모리 부족 현상 발생**
→ 이전 최적화 결과 값을 포함한 작은 Size의 탐색 범위로 재구성

- **sklearn라이브러리의 GridSearchCV, RandomizedSearchCV 사용**

  - **Grid Search**
  - **Random Search**

```
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':[None,'sqrt','log2',],
    'max_leaf_nodes':[10, 60, 100],
    'min_impurity_decrease':[0.0, 0.3, 0.5],
    'bootstrap': [True,False]
}
```

- **optuna 라이브러리 사용**

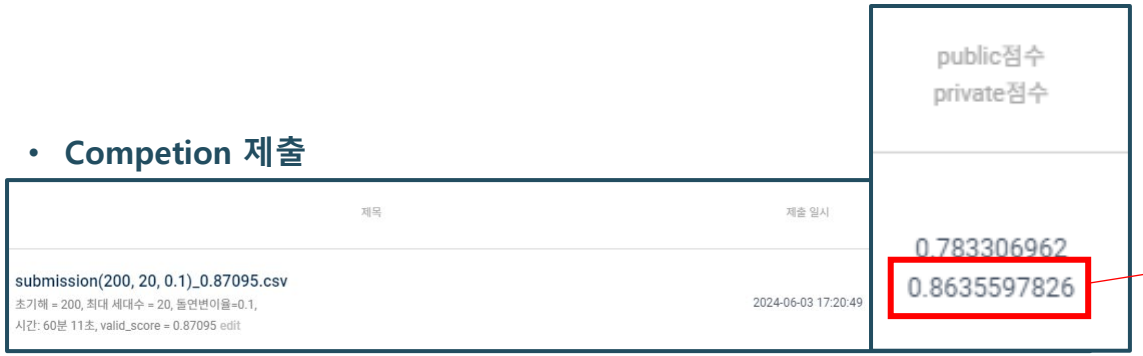
  - **Optuna**

```
params = {
    'n_estimators': trial.suggest_categorical('n_estimators', [100, 300, 600, 800, 850, 900, 950, 1000]),
    'criterion': trial.suggest_categorical('criterion', ['gini', 'entropy']),
    'max_depth': trial.suggest_categorical('max_depth', [1, 10, 20, 35, 50]),
    'min_samples_split': trial.suggest_categorical('min_samples_split', [2, 10, 20, 30]),
    'min_samples_leaf': trial.suggest_categorical('min_samples_leaf', [5, 30, 50]),
    'min_weight_fraction_leaf': trial.suggest_categorical('min_weight_fraction_leaf', [0.0, 0.2, 0.35, 0.5]),
    'max_features': trial.suggest_categorical('max_features', [None, 'sqrt','log2']),
    'max_leaf_nodes': trial.suggest_categorical('max_leaf_nodes', [10, 60, 100]),
    'min_impurity_decrease': trial.suggest_categorical('min_impurity_decrease', [0.0, 0.3, 0.5]),
    'bootstrap': trial.suggest_categorical('bootstrap', [True, False]),
    'n_jobs': -1
}
```

2. 실험 결과 및 비교 분석

(3) 비교 분석

- Competion 제출



- Competition LeaderBoard

	Random	Grid	Optuna	GA
Valid Score	0.90292	0.84771	0.82489	0.87095
Private Score	0.80372	0.85622	0.86339	0.86356
실행시간	2분 20초	8시간 16분 49초	44분 50초	60분 11초
LeaderBoard	125등(상위 25%)	20등(상위 4%)	**4등(수상)**	**4등(수상)**

※ 수상 기준 : LeaderBoard(private score) 1, 2, 3, 4, 5등

→ **GA 방법이 가장 우수하였으며 그 다음으로 Optuna, Grid, Random 순**(private score 기준)

3. 결론

1. Ver.1, Ver.2 실험에서 Valid Score가 높다고 해서 Private Score가 높은 것은 아니었고, 데이터의 수가 약 1300개 정도로 적어 과적합이 일어나기 쉬운 Competition
   (※ 이번 대회 2등 수상자의 public 등수와 private 등수: 347등 -> 2등)

2. Grid Search에서 파라미터 탐색 범위가 많아질수록 **메모리 부족 현상**이 발생!
   But, GA에서는 발생하지 않음
   → 동일한 컴퓨터 조건에서 GA가 더 다양한 파라미터 탐색 가능

3. 성능적인 측면에서 GA_HPO가 대체로 우수한 것을 확인
   But, 초기해 수와 반복 횟수를 높일수록 실행 시간이 기하급수적으로 증가
   (※ 초기해 수=100, 최대 세대수=2000 → 32시간 33분)
   → 각 상황에 맞는 적절한 초기해 수와 최대 세대수를 정하는 것이 필요

감사합니다.
Q&A