2. 실습 – 하이퍼 파라미터 튜닝

1 문제 정의

사용 데이터

데이터 정보

항목	내용
데이터명	vehicle.csv
과제 종류	다중 분류
샘플 수	846개
특징 수	18개
클래스 수	4개

데이터 불러오기

```
import pandas as pd
df = pd.read_csv("../../data/classification/vehicle.csv")
X = df.drop('y', axis = 1)
y = df['y']
```

하이퍼 파라미터 범위 및 평가 방법

고려하는 모델은 LightGBM이며 하이퍼파라미터는 결정 나무 개수(n_estimator), 학습률(learning_rate), 최대 잎 노드 개수(num_leaves)입니다.
평가는 5겹 교차 검증 방식으로 계산한 정확도를 사용합니다.

하이퍼 파라미터 범위

하이퍼 파라미터	범위
결정 나무 개수	10개 이상 100개 이하
학습률	0.001 이상 0.5 이하
최대 잎 노드 개수	20개 이상 120개 이하

2. 실습 – 하이퍼 파라미터 튜닝

2 구성 요소 정의

목적 함수 정의

목적 함수는 하이퍼파라미터에 따른 5-겹 교차 검증 방식으로 계산한 정확도입니다.

목적 함수

```
1  from sklearn.model_selection import cross_val_score
2  from lightgbm import LGBMClassifier as LGBC
3  def obj_func(parameter):
4      model = LGBC(random_state = 2022,
5                   n_estimators = int(parameter[0]),
6                   learning_rate = parameter[1],
7                   num_leaves = int(parameter[2]))
8
9      score = cross_val_score(model, X, y, scoring = "accuracy", cv = 5).mean()
10     return score
```

- **라인 3:** 크기가 3인 배열 parameter를 입력받는 obj_func 함수를 정의합니다. 여기서 0번째 요소는 결정 나무 개수, 1번째 요소는 학습률, 2번째 요소는 잎 노드 개수를 나타냅니다.
- **라인 4~7:** 입력받은 하이퍼파라미터를 갖는 LightGBM 모델 인스턴스인 model을 정의합니다. 이때 공정한 평가를 위해 random_state는 2022로 고정했습니다.
- **라인 9:** 5-겹 교차 검증 방식으로 계산한 model의 평균 정확도를 score에 저장합니다.

샘플러 정의

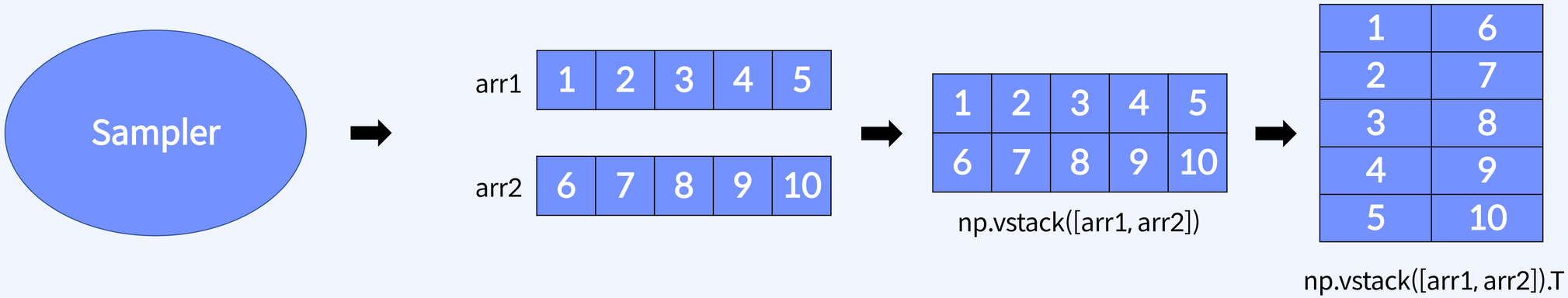
각 하이퍼 파라미터는 유니폼 분포로부터 샘플링하겠습니다. 샘플러는 다음과 같이 정의합니다.

샘플러

```
1  import numpy as np
2  from scipy.stats import uniform
3  def sampler(n):
4      n_estimator = np.random.randint(10, 100, n)
5      learning_rate = uniform(0.001, 0.5).rvs(n)
6      num_leaves = np.random.randint(20, 120, n)
7      parameter_arr = np.vstack([n_estimator, learning_rate,  num_leaves]).T
8      return parameter_arr
```

- **라인 4:** 10부터 100 사이의 n 개의 정수를 n_estimator에 저장합니다.
- **라인 5:** 0.001부터 0.5 사이의 n 개의 실수를 learning_rate에 저장합니다.
- **라인 7:** 크기가 n인 배열 n_estimator, learning_rate, num_leaves를 수직 방향으로 병합한 뒤, T를 사용해 전치시킵니다. 칼럼별로 샘플링한 결과를 합칠 때 사용하는 테크닉입니다 (뒷 페이지 참고)

칼럼별 샘플링 결과 병합 테크닉



획득 함수 정의

획득 함수로 개선 기대 함수를 사용하겠습니다.

획득 함수

```
1  from scipy.stats import norm
2  def EI(X_new, model, best_mu, e = 0.01):
3      mu, sigma = model.predict(X_new, return_std = True)
4      z = np.zeros(len(X_new))
5      z[sigma > 0] = ((mu - best_mu - e) / sigma)[sigma > 0]
6      return (mu - best_mu - e) * norm.cdf(z) + sigma * norm.pdf(z)
```

2. 실습 – 하이퍼 파라미터 튜닝

3 메인 코드

메인 함수

초기에 n1 개의 해를 평가하고, 매 이터레이션마다 n2 개의 후보 해를 샘플링해서 획득 함수를 사용해 k 개의 해를 선택하여 평가합니다.

```
1  from sklearn.gaussian_process import GaussianProcessRegressor as GPR
2  from sklearn.gaussian_process.kernels import RBF, WhiteKernel
3
4  def main(n1, n2, k, num_iter):
5      P = sampler(n1)
6      E = np.apply_along_axis(obj_func, axis = 1, arr = P)
7      for _ in range(num_iter):
8          model = GPR(kernel=RBF() + WhiteKernel(), random_state=2022).fit(P, E)
9          best_mu = max(model.predict(P))
10         P_new = sampler(n2)
11         score_list = EI(P_new, model, best_mu)
12
13         P_new_star = P_new[(-score_list.argsort())[:k]]
14         E_new = np.apply_along_axis(obj_func, axis = 1, arr = P_new_star)
15         P = np.vstack([P, P_new_star])
16         E = np.append(E, E_new)
17
18     return P[E.argmax()], E.max()
```

- **라인 5~6:** n1 개의 해를 임의로 샘플링한 결과를 P에 저장하고 obj_func 함수로 P의 각 요소를 평가한 결과를 E에 저장합니다.
- **라인 10~11:** n2 개의 해를 임의로 샘플링하여 획득 함수인 EI로 각각의 해를 평가합니다.
- **라인 13:** P_new에서 EI 함수의 출력이 가장 큰 상위 k 개의 해를 P_new_star에 저장합니다.
- **라인 14:** P_new_star에 있는 해를 obj_func 함수로 평가하고 그 결과를 E_new에 저장합니다.
- **라인 15~16:** P_new_star와 E_new를 각각 P와 E에 추가합니다.

하이퍼 파라미터 튜닝 수행

하이퍼 파라미터 튜닝 수행

```
1    print(main(100, 100, 10, 50))
```

- **라인 1:** n1 = 100, n2 = 100, k = 10, num_iter = 50으로 설정했습니다.

```
(array([62., 0.17392454, 21.]), 0.7896206056387052)
```