fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

3. 튜닝 범위 설정

1 튜닝 범위 설정의 필요성

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

(Revisit) 하이퍼 파라미터 튜닝 문제의 특징

하이퍼파라미터 튜닝 문제는 분석적으로 풀 수 없고 푸는데 시간이 오래 걸립니다.

분석적으로 해결할 수 없어, 여러 해를 평가하고 비교하여 좋은 해를 찾아야 합니다.

탐색 공간을 모두 탐색하는 것은 비현실적이고 비효율적임
(예: 라쏘의 하이퍼파라미터 α는 0 이상이기만 하면 되므로 선택할 수 있는 경우의 수가 무한대임)

하이퍼파라미터에 대한 탐색 공간은 사용자가 직접 정의하며, 이 과정에서 모델과 하이퍼파라미터에 대한 이해가 큰 역할을 함
(예 1: 라쏘의 하이퍼파라미터 α는 특정 수치 이상 설정할 필요가 없음)
(예 2: 결정 나무에서 불순도 지표는 탐색할 필요가 없음)

하나의 해를 평가하려면 모델을 학습해야 하므로 오랜 시간이 걸림. 따라서 탐색 공간을 잘 정의하는 것과 좋을 것이라 예상되는 해를 먼저 탐색하는 것이 매우 중요함

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

튜닝할 하이퍼 파라미터 선정

모든 하이퍼파라미터를 튜닝하는 것은 현실적으로 불가능하므로 튜닝할 하이퍼파라미터를 선택해야 합니다.

하이퍼파라미터 튜닝 문제를 효율적으로 풀기 어려운 이유는 개별 해를 평가하는 데 오랜 시간이 들기도 하지만 탐색 공간 자체가 매우 넓음

(예) 사이킷런에서 DecisionTreeClassifier의 하이퍼파라미터는 12개나 되며, 몇몇 하이퍼파라미터는 연속형 변수임

따라서 모든 하이퍼파라미터를 튜닝하는 것은 현실적으로 불가능하므로 모델의 성능에 큰 영향을 끼치는 하이퍼파라미터를 선택해야 함

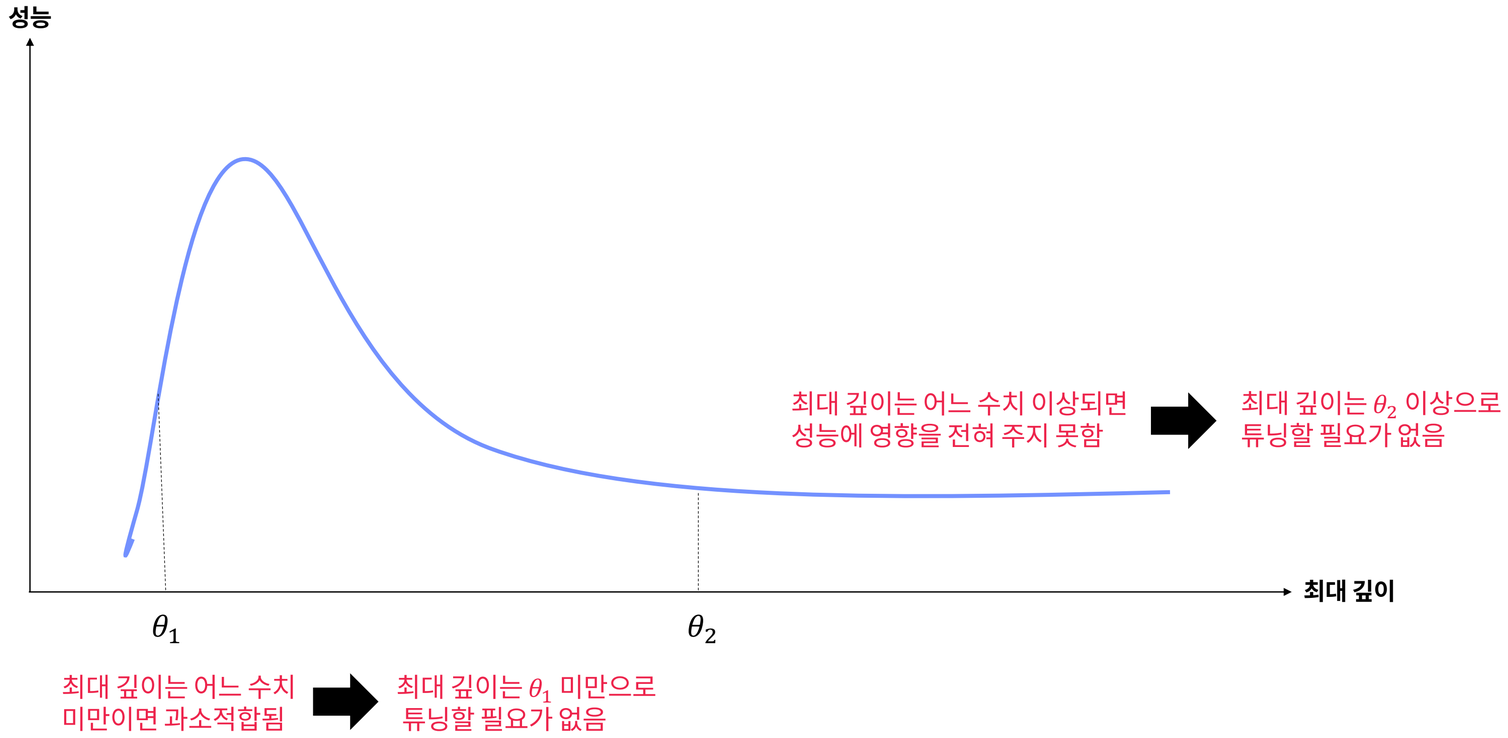
(예) 결정 나무에서 노드를 분지하는 기준을 나타내는 하이퍼파라미터 criterion은 " gini " 와 " entropy " 가운데 어느 값을 설정하더라도 성능 차이가 크지 않다고 알려져 있으므로 튜닝 대상에 포함할 필요가 없음

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

튜닝할 하이퍼 파라미터 범위 선정

어떤 하이퍼파라미터를 튜닝할 것인가도 중요하지만, 어느 범위에서 튜닝할 것인가도 매우 중요합니다.

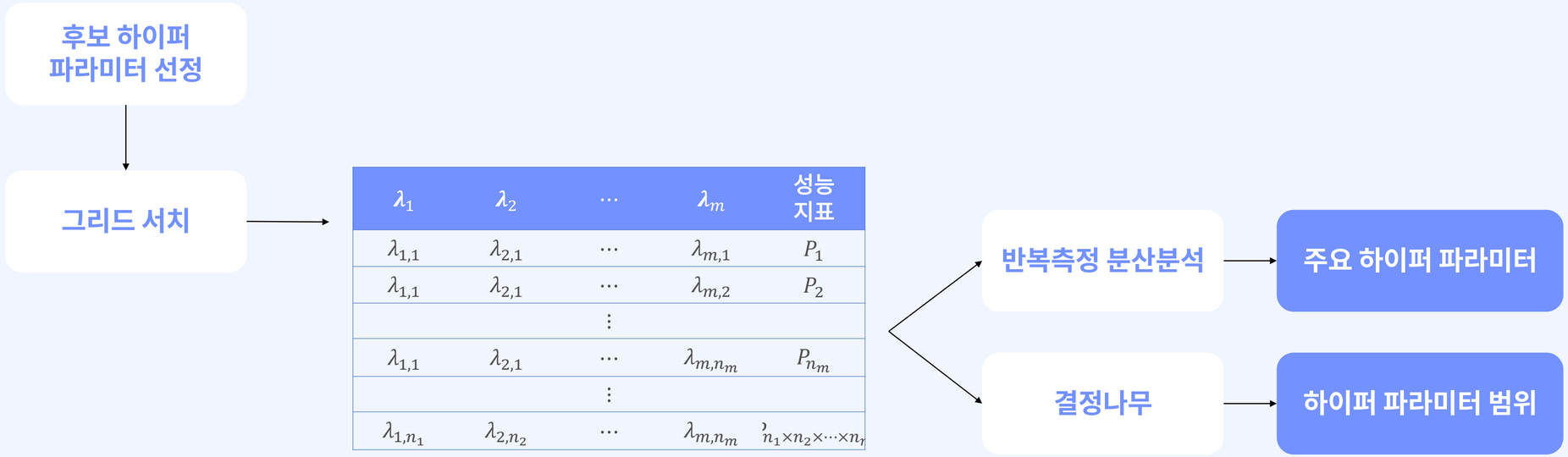
(예시) 결정 나무의 최대 깊이: max_depth



Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

하이퍼 파라미터 및 범위 설정을 위한 실험 프로세스

후보 하이퍼파라미터에 대해 그리드 서치를 적용한 결과를 실험 데이터로 사용합니다. 실험 데이터에 따라 다른 결과가 나올 수 있으므로 실제 시스템에 적용하려면 다양한 데이터를 바탕으로 실험 데이터를 구축해야 합니다. 또한, 반복측정 분산분석을 통해 주요 하이퍼파라미터를 식별하고 결정 나무를 이용해 적절한 하이퍼파라미터의 범위를 찾습니다.



fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

3. 튜닝 범위 설정

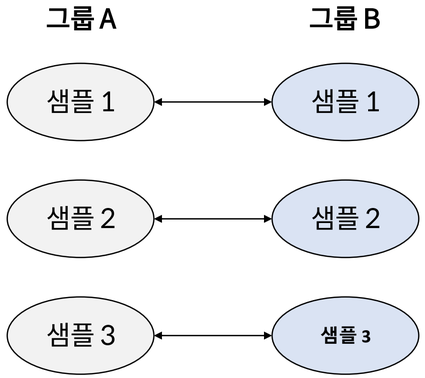
2 주요 하이퍼 파라미터 식별

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

반복측정 분산분석이란?

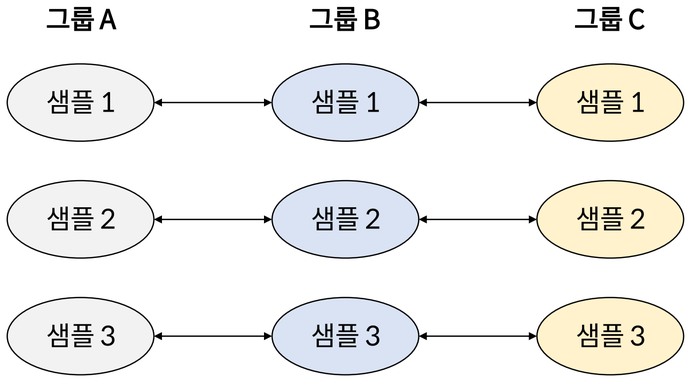
반복측정 분산분석(One-way repeated measures analysis of variance; RM ANOVA)은 반복측정 자료에 대해 그룹별 평균 차이를 검정하기 위한 가설 검정 기법입니다. 대응 표본 t-검정 기법을 확장한 기법이라고도 볼 수 있습니다.

대응 표본 t-검정이 필요한 상황



(예시) 임상 시험에서 환자에게 약을 주사하기 전과 후의 혈당 등을 비교

반복측정 분산분석이 필요한 상황



(예시) 환자에게 약을 주사하기 전, 주사하고 10분 후, 1시간 후의 혈당 등을 비교

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

statsmodels를 이용한 반복측정 분산분석

statsmodels는 가설 검정, 회귀 분석, 시계열 분석 등 다양한 통계 분석 기능을 제공하는 파이썬 패키지입니다. statsmodels의 AnovaRM 클래스를 사용하면 손쉽게 반복측정 분산분석을 구현할 수 있습니다.

statsmodels 설치

```
$ pip install statsmodels
```

AnovaRM 클래스의 주요 인자

인자	설명
data	분석에 사용할 데이터프레임
depvar	종속 변수명 (data의 컬럼명)
subject	실험 대상 ID (data의 컬럼명)
within	요인 (data의 컬럼명 목록)

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

반복측정 분산분석 예제

statsmodels의 AnovaRM 클래스를 사용해서 반복측정 분산분석을 구현해보겠습니다.

예제 데이터 불러오기

```
1  import pandas as pd
2  data = pd.read_csv("AnovaRM_예제.csv")
3  display(data)
```

	sub_id	y	x
0	1	10	a
1	2	20	a
2	3	30	a
3	1	20	b
4	2	40	b
5	3	60	b
6	1	30	c
7	2	60	c
8	3	90	c

- 3개의 샘플에 대해 요인 x를 바꿔가며 세 개의 실험을 수행했을 때의 종속 변수 y의 값으로 구성된 데이터임
- sub_id가 1인 샘플의 y는 x가 a일 때 10, b일 때 20, c일 때 30으로 x가 y에 영향을 끼침을 알 수 있음

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

반복측정 분산분석 예제 (계속)

AnovaRM을 사용해 x가 y에 얼마나 영향을 끼치는지 확인해보겠습니다.

AnovaRM 사용

```
1  from statsmodels.stats.anova import AnovaRM
2  aovrm = AnovaRM(data, 'y', 'sub_id', within=['x'])
3  res = aovrm.fit()
4  print(res)
```

- **라인 1:** statsmodels.stats.anova에 속한 AnovaRM 클래스를 불러옵니다.
- **라인 2:** AnovaRM의 객체 aovrm을 생성합니다.
- **라인 3:** fit 메서드를 적용한 결과를 res에 저장합니다. statsmodels에 속한 대부분의 인스턴스가 fit를 이용해 통계 분석을 수행합니다. 또한, fit 메서드를 사용하면 AnovaResults라는 객체를 반환하므로 res라는 변수에 저장합니다.

```
Anova
==============================
 F Value Num DF Den DF Pr > F
------------------------------
x 12.0000 2.0000 4.0000 0.0204
==============================
```

- 행의 이름이 x라고 돼 있음
- 즉, 인자가 x 하나이므로 x가 y에 얼마나 영향을 끼치는지만 표시됨
- 결과를 해석할 때 보는 값은 일원분산분석에서 계산한 F-통계량인 F Value와 통계적으로 얼마나 유의한 결과인지를 나타내는 Pr > F (p-value)임
- 일반적으로 p-value가 0.05보다 작으면 통계적으로 유의한 영향을 끼친다고 볼 수 있음

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

반복측정 분산분석 예제 (계속)

AnovaRM을 사용해 x가 y에 얼마나 영향을 끼치는지 확인해보겠습니다.

AnovaRM 사용

```
1  display(res.__dict__['anova_table'])
```

- **라인 1:** __dict__ 속성을 사용해 AnovaResult를 사전 자료형으로 변환한 뒤, “anova_table” 키로 값을 가져옵니다. 이 값은 앞서 출력한 분석 결과가 데이터프레임으로 정리된 것입니다.

	F Value	Num DF	Den DF	Pr > F
x	12.0	2.0	4.0	0.020408

- 이전 출력값인 문자열보다 데이터프레임이라서 사용하기 적절함

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

실험 데이터 준비
: 후보 하이퍼 파라미터 선정

LightGBM의 주요 하이퍼파라미터를 색출하고 그 튜닝 범위를 설정하겠습니다.

하이퍼 파라미터 범위

하이퍼 파라미터	설명	튜닝 범위
n_estimators	결정 나무 개수	{50, 100, 150, 200}
learning_rate	학습률	{0.01, 0.05, 0.1, 0.3}
num_leaves	최대 잎 노드 개수	$\{2^3, 2^4, 2^5, 2^6\}$
boosting_type	부스팅 방법	{gbdt, goss}
random_state	시드	{2020, 2021, 2022, 2023}
metric	노드 분할 기준	{mae, mse}
reg_alpha	L1 페널티에 대한 가중치	{0, 0.1, 1.0, 10}
colsample_bytree	결정 나무별 사용하는 특징의 비율	{0.5, 0.7, 0.9}

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

실험 데이터 준비
: 데이터 불러오기

실험에 사용할 데이터 mortgage.csv를 불러오고 특징과 라벨로 분리합니다.

예제 데이터 불러오기

```
1  import pandas as pd
2  df = pd.read_csv("../../data/regression/mortgage.csv")
3  X = df.drop('y', axis = 1)
4  y = df['y']
```

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

실험 데이터 준비
: 그리드 서치

파라미터 그리드인 grid를 생성하고 각 해를 평가합니다.

그리드 서치

```
from lightgbm import LGBMRegressor as LGB
from sklearn.model_selection import ParameterGrid, cross_val_score
grid = ParameterGrid({"n_estimators":[50, 100, 150, 200],
                      "learning_rate":[0.01, 0.05, 0.1, 0.3],
                      "num_leaves":[2**3, 2**4, 2**5, 2**6],
                      "boosting_type":["gbdt", "goss", "rf"],
                      "random_state":[2020, 2021, 2022, 2023],
                      "metric":["mae", "mse"],
                      "reg_alpha":[0, 0.1, 1.0, 10]})

score_list = []
for param in grid:
    score = (-cross_val_score(LGB(**param), X, y,
                              scoring = "neg_mean_absolute_error")).mean()
    score_list.append(score)
```

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

실험 데이터 준비
: 그리드 서치 (계속)

grid를 데이터프레임으로 변환하고 거기에 score_list를 칼럼으로 추가하겠습니다. 또한, 그 데이터를 저장하겠습니다.

그리드 서치

```
1  grid_search_data = pd.DataFrame(grid)
2  grid_search_data['score'] = score_list
3  display(grid_search_data.head())
4  grid_search_data.to_csv("LightGBM_하이퍼파라미터선택_실험데이터.csv", index = False)
```

	boosting_type	colsample_bytree	learning_rate	metric	n_estimators	num_leaves	random_state	reg_alpha	score
0	gbdt	0.5	0.01	mae	50	8	2020	0.0	1.499215
1	gbdt	0.5	0.01	mae	50	8	2020	0.1	1.499427
2	gbdt	0.5	0.01	mae	50	8	2020	1.0	1.502761
3	gbdt	0.5	0.01	mae	50	8	2020	10.0	1.523360
4	gbdt	0.5	0.01	mae	50	8	2021	0.0	1.499254

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

주요 하이퍼 파라미터 식별 :boosting_type 예시

반복측정 분산분석을 이용해 boosting_type의 중요도를 측정해보겠습니다.

주요 하이퍼 파라미터 식별 예시

```
1  hyper_params = grid_search_data.columns[:-1].tolist()
2  hyper_params.remove("boosting_type")
3  hyper_params.insert(0, "boosting_type")
4  grid_search_data.sort_values(by = hyper_params, inplace = True)
5  boosting_type_size = len(grid_search_data['boosting_type'].unique())
6  num_ID = int(len(grid_search_data) / boosting_type_size)
7  grid_search_data['subject_ID'] = list(range(num_ID)) * boosting_type_size
```

- **라인 1:** grid_search_data의 칼럼에서 맨 뒤 요소(score)를 제외한 모든 요소로 구성된 리스트를 hyper_params에 저장합니다.
- **라인 2~3:** hyper_params에서 중요도를 측정하고자 하는 하이퍼파라미터를 맨 앞에 오도록 했습니다. 즉, remove 메서드를 사용해 "boosting_type"을 지운 뒤에 insert 메서드를 사용해 다시 맨 앞으로 추가했습니다.
- **라인 4:** hyper_params를 기준으로 오름차순 정렬합니다. 정렬하지 않으면 ID가 부적절하게 부여될 수 있습니다.
- **라인 5~6:** boosting_type가 갖는 유니크한 값의 개수를 바탕으로 ID 개수를 계산합니다. boosting_type 변수는 3개의 값을 가질 수 있어, 3개의 그룹에 대해 반복측정 분산분석을 적용해야 합니다. 또한, 각 그룹에 들어가는 샘플 수만큼 ID를 부여합니다.
- **라인 7:** [0, 1, 2, ⋯, num_ID – 1]을 boosting_type_size만큼 반복하여 만든 ID를 subject_ID 칼럼으로 추가합니다.

샘플 ID는 각 하이퍼파라미터가 독립 변수이고 나머지 하이퍼파라미터가 통제 변수 가 되도록 부여해야 함
즉, 각 하이퍼파라미터값만 다르고 나머지 하이퍼파라미터값이 같으면 샘플 ID가 같게 부여해야 함

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

주요 하이퍼 파라미터 식별
:boosting_type 예시 (계속)

반복측정 분산분석을 이용해 boosting_type의 중요도를 측정해보겠습니다.

주요 하이퍼 파라미터 식별 예시

```
1  aovrm = AnovaRM(grid_search_data, 'score', 'subject_ID', within=["boosting_type"])
2  res = aovrm.fit()
3  res = res.__dict__['anova_table']
4  display(res)
```

	F Value	Num DF	Den DF	Pr > F
boosting_type	6568.24873	1.0	6143.0	0.0

- p-value가 0.0 으로 boosting_type이 성능에 영향을 준다고 할 수 있음
- 그러나 데이터가 충분히 많으면 P-value가 0.0이 나오는 경우는 매우 흔하므로, P-value만 보고 매우 중요한 하이퍼파라미터라고 결론짓기는 이름
- 따라서 여러 하이퍼파라미터에 대해 반복측정 분산분석을 적용하여 나온 F-통계량을 비교해야 함

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

주요 하이퍼 파라미터 식별
:전체 데이터에 적용

전체 하이퍼파라미터에 대해 반복측정 분산분석을 수행하겠습니다.

주요 하이퍼 파라미터 식별

```
1  AnovaRM_result = pd.DataFrame() # 분석 결과 초기화
2  param_cols = grid_search_data.columns.tolist()
3  param_cols.remove('score')
4  param_cols.remove('subject_ID')
5
6  for param in param_cols:
7      # 데이터 수정
8      hyper_params = grid_search_data.columns[:-1].tolist()
9      hyper_params.remove(param)
10     hyper_params.insert(0, param)
11     grid_search_data.sort_values(by = hyper_params, inplace = True)
12     param_size = len(grid_search_data[param].unique())
13     num_ID = int(len(grid_search_data) / param_size)
14     grid_search_data['subject_ID'] = list(range(num_ID)) * param_size
15
16     # 분석 결과 추가
17     aovrm = AnovaRM(grid_search_data, 'score', 'subject_ID', within=[param])
18     res = aovrm.fit()
19     res = res.__dict__['anova_table']
20     AnovaRM_result = AnovaRM_result.append(res)
```

- **라인 1:** 전체 하이퍼파라미터에 대한 반복측정 분산분석 결과를 저장할 데이터프레임인 AnovaRM_result를 초기화합니다.
- **라인 2~4:** 하이퍼파라미터 목록을 정의합니다. 이전 코드에서 추가한 subject_ID와 score를 제거합니다.
- **라인 6:** param_cols를 순회하면서 요인으로 사용할 param을 설정합니다.
- **라인 7~14:** param에 대해 반복측정 분산분석을 수행할 수 있도록 grid_search_data를 정제합니다.
- **라인 13~16:** 반복측정 분산분석 결과를 AnovaRM_result에 추가합니다. 라인 15에서 res는 데이터프레임이 되며, 데이터프레임에 데이터프레임을 append하는 것은 행 단위로 병합하는 것과 같습니다.

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

주요 하이퍼 파라미터 식별
:결과 해석

AnovaRM_result를 F-통계량을 기준으로 내림차순 정렬하여 출력하겠습니다.

```
1  display(AnovaRM_result.sort_values(by = "F Value", ascending = False))
```

	F Value	Num DF	Den DF	Pr > F
reg_alpha	64666.215872	3.0	9213.0	0.000000e+00
learning_rate	9682.902945	3.0	9213.0	0.000000e+00
boosting_type	6568.248730	1.0	6143.0	0.000000e+00
num_leaves	4943.618015	3.0	9213.0	0.000000e+00
colsample_bytree	2143.730367	2.0	8190.0	0.000000e+00
n_estimators	1477.375094	3.0	9213.0	0.000000e+00
metric	819.485950	1.0	6143.0	2.703479e-169
random_state	346.890102	3.0	9213.0	1.930288e-213

- reg_alpha, learning_rate, boosting_type, num_leaves 순으로 중요한 하이퍼파라미터임을 알 수 있음
- 여기서 중요하다는 것은 설정한 값에 따라 성능이 크게 바뀐다는 뜻에 주의해야 함

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

주요 하이퍼 파라미터 식별
:결과 해석 (계속)

reg_alpha에 따른 MAE의 평균값을 확인해보겠습니다.

```
1  for val in grid_search_data['reg_alpha'].unique():
2      avg_score = grid_search_data.loc[grid_search_data['reg_alpha'] == val,
3                                       'score'].mean()
4      print("{}:{}".format(val, avg_score))
```

- **라인 1:** val로 reg_alpha의 유니크한 값(0, 0.1, 1, 10)을 순회합니다.
- **라인 2~3:** reg_alpha가 val인 행의 score의 평균을 avg_score에 저장합니다.

```
0.0:  0.2895331565457736
0.1:  0.2902266120608595
1.0:  0.2979066980451965
10.0: 0.3392285961577812
```

- reg_alpha가 커지면 커질수록 MAE도 커짐
- 이 결과만 놓고 보면 reg_alpha가 크면 클수록 MAE가 커지므로 F-통계량이 크게 나왔다고 할 수 있음
- 그러나 0으로 설정하면 더 이상 튜닝할 필요가 없으므로 굳이 튜닝 대상에 포함시키지 않아도 됨
- 단, 0과 0.1 사이에서 더 좋은 성능을 갖는 값이 있을 수도 있음을 항상 기억하고 주의해야 함

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

주요 하이퍼 파라미터 식별
:결과 해석 (계속)

learning_rate에 따른 MAE의 평균값을 확인해보겠습니다.

```
1  for val in grid_search_data['learning_rate'].unique():
2      avg_score = grid_search_data.loc[grid_search_data['learning_rate'] == val,
3                                       'score'].mean()
4      print("{}:{}".format(val, avg_score))
```

```
0.01:  0.8465915564287093
0.05:  0.14289935759632696
0.1:  0.1085783767317755
0.3:  0.11882577205279798
```

- 0.1일 때 가장 좋은 성능을 보임을 알 수 있음
- 0.05와 0.3 사이에서 가장 좋은 성능이 나온다고 추측할 수 있음

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

주요 하이퍼 파라미터 식별
:결과 해석 (계속)

가장 중요하지 않다고 판단된 random_state에 따른 MAE의 평균값을 확인해보겠습니다.

```
for val in grid_search_data['random_state'].unique():
    avg_score = grid_search_data.loc[grid_search_data['random_state'] == val,
                                     'score'].mean()
    print("{}:{}".format(val, avg_score))
```

```
2020:  0.3042900489902465
2021:  0.30343115815837834
2022:  0.3042139604741169
2023:  0.30495989518687033
```

- 예상한 대로 설정값에 따른 MAE의 차이가 두드러지지 않음을 알 수 있음
- 따라서 random_state는 굳이 튜닝하지 않아도 되는 하이퍼파라미터라고 할 수 있음

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

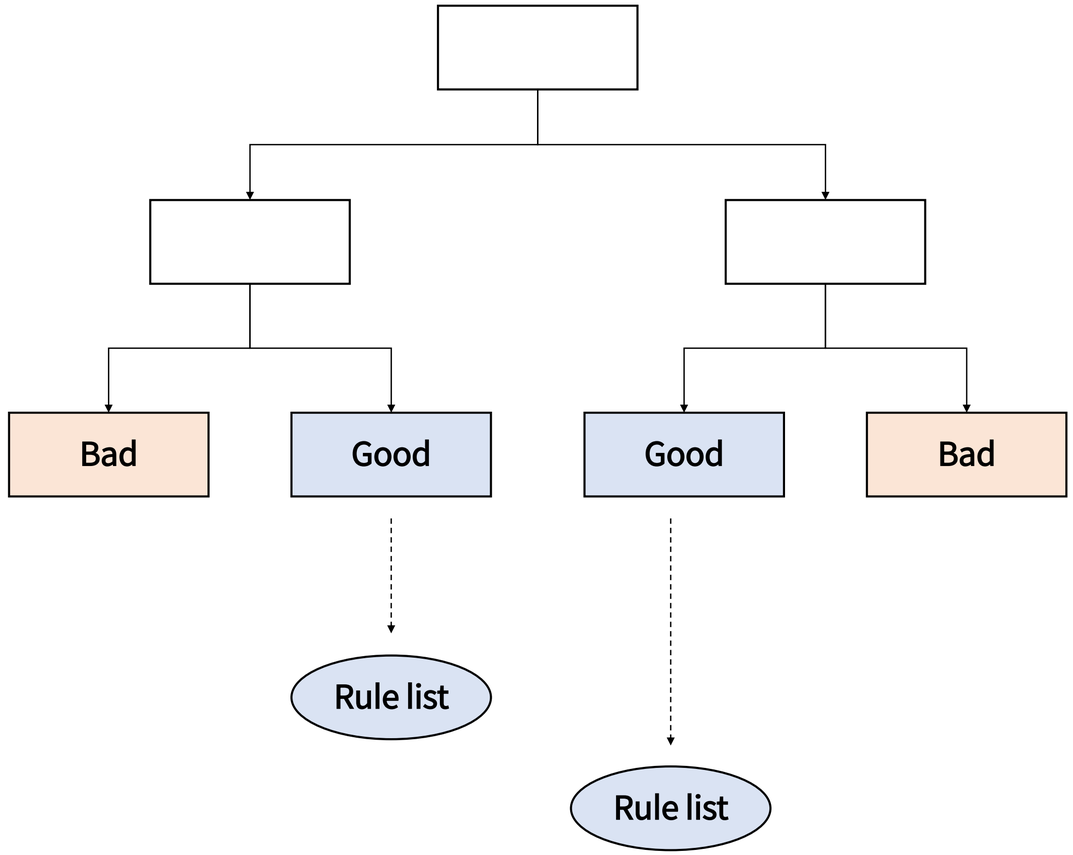
3. 튜닝 범위 설정

3 하이퍼 파라미터 범위 설정

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

결정 나무를 이용한 하이퍼 파라미터 범위 설정

결정 나무를 사용한 방법은 그리드 서치 혹은 랜덤 서치 결과 데이터를 사용해 결정 나무를 학습한 다음, 성능이 좋으리라 예상되는 잎 노드에 대응되는 규칙을 바탕으로 범위를 설정하는 것입니다. 이 방법은 하이퍼파라미터 범위뿐만 아니라 주요 하이퍼파라미터를 선택하는 데도 사용할 수 있습니다. 단, 엄밀한 통계적 분석 기법이 아니므로 결과 해석이 다소 까다롭다는 단점이 있습니다.



Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

데이터 준비

3.
하이퍼 파라미터 범위 설정

결정 나무 학습 결과는 데이터 양에 영향을 많이 받으므로 여러 하이퍼파라미터를 동시에 고려한 그리드 서치 결과보다 소수의 하이퍼파라미터를 다양하게 설정한 그리드 서치 혹은 랜덤 서치 결과를 활용하는 것이 더 적절합니다. 따라서 여건이 허락한다면 반복측정 분산분석으로 선정한 하이퍼파라미터를 대상으로 그리드 서치를 다시 해주는 것이 더 좋습니다.

데이터 준비

```
1  grid =    ParameterGrid({"learning_rate":np.arange(0.05, 0.3, 0.01),
2                            "num_leaves":[2**2, 2**3, 2**4, 2**5, 2**6, 2**7, 2**8],
3                            "boosting_type":["gbdt", "goss"]})
4
5  score_list = []
6  for param in grid:
7      score = (-cross_val_score(LGB(**param), X, y,
8                                scoring = "neg_mean_absolute_error")).mean()
9      score_list.append(score)
10
11 grid_search_data = pd.DataFrame(grid)
12 grid_search_data['score'] = score_list
13 grid_search_data.to_csv("LightGBM_하이퍼파라미터범위설정_실험데이터.csv", index = False)
```

- **라인 1 ~ 3:** 앞서 reg_alpha, learning_rate, boosting_type, num_leaves 순으로 중요한 하이퍼파라미터라는 것과 그 가운데 reg_alpha는 값이 크면 클수록 성능이 저하됐음을 확인했으므로, learning_rate, boosting_type, num_leaves만 튜닝 범위에 포함합니다. 기존 하이퍼파라미터의 탐색 공간보다 더 촘촘하게 정의합니다.
- **라인 5~9**: grid를 param으로 순회하면서 하이퍼파라미터가 param인 LightGBM 모델을 5-겹 교차 검증 방식으로 평가한 결과를 score_list에 저장합니다.
- **라인 11~13**: grid를 grid_search_data라는 데이터프레임으로 변환한 뒤 score_list를 score라는 칼럼으로 추가합니다. 또한, 독자가 직접 코드를 실행하지 않아도 되도록 해당 데이터를 "LightGBM_하이퍼파라미터범위설정_실험데이터.csv"로 저장합니다. 직접 실행하지 않았다면, 해당 데이터를 불러와야 합니다.

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

결정 나무 학습

grid_search_data를 사용해 결정 나무를 학습하겠습니다.

결정나 무 학습

```
1  from sklearn.tree import DecisionTreeRegressor as DTR
2  grid_X = grid_search_data.drop('score', axis = 1)
3  grid_X = pd.get_dummies(grid_X, drop_first = True)
4  grid_y = grid_search_data['score']
5  model = DTR(max_depth = 5).fit(grid_X, grid_y)
```

- **라인 2:** score 칼럼을 제외한 데이터프레임을 grid_X에 저장합니다.
- **라인 3:** get_dummies를 이용해 grid_X에 포함된 범주형 변수(boosting_type)를 더미화합니다. 실전에서 잘 사용하지 않지만, 여기서는 일반적인 지도 학습 모델을 만드는 것이 아니므로 편의를 위해 사용했습니다.
- **라인 5:** 학습 데이터와 평가 데이터로 분할하지 않고 전체 데이터를 사용해 결정 나무를 학습합니다. 해석의 편의를 위해 최대 깊이는 5로 작게 설정했습니다.

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

규칙으로 변환

학습한 결정 나무를 규칙 집합으로 변환하겠습니다.

규칙 나무 변환

```
1  from sklearn.tree import export_text
2  def text_to_rule_list(r):
3      node_list = []
4      leaf_node_list = []
5
6      for i, node in enumerate(r.split("\n")[:-1]):
7          rule = node.split('- ')[1]
8          indent = node.count('  ' * 3)
9          if 'value' in rule:
10             leaf_node_list.append([i, rule, indent])
11         node_list.append([i, rule, indent])
12
13     prediction_rule_list = []
14     for leaf_node in leaf_node_list:
15         prediction_rule = []
16         idx, decision, indent = leaf_node
17         for indent_level in range(indent-1, -1, -1):
18             for node_idx in range(idx, -1, -1):
19                 node = node_list[node_idx]
20                 rule = node[1]
21                 if node[2] == indent_level and "value" not in node[1]:
22                     prediction_rule.append(rule)
23                     break
24         prediction_rule_list.append([prediction_rule, decision])
25
26 return prediction_rule_list
```

- 라인 9, 21: "class"를 "value"로 바꿨다는 것을 제외하면 이전 코드와 완전히 동일합니다. 이렇게 바꾼 이유는 DecisionTreeClassifier는 export_text로 변환했을 때 잎 노드에 "class"라는 문자열을 사용하지만, DecisionTreeRegressor는 "value"라는 문자열을 사용하기 때문입니다.

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

규칙으로 변환 (계속)

export_text와 text_to_rule_list를 이용해 결정 나무를 데이터프레임 형태의 규칙 집합으로 변환하겠습니다.

규칙 나무 변환

```
1  r = export_text(model)
2  result = pd.DataFrame(text_to_rule_list(r),
3                        columns = ["condition", "output"])
4  display(result.head())
```

	condition	output
0	[learning_rate <= 0.05, learning_rate <= 0.16,...	value: [0.13]
1	[learning_rate <= 0.13, learning_rate > 0.05,...	value: [0.12]
2	[learning_rate > 0.13, learning_rate > 0.05,...	value: [0.12]
3	[learning_rate <= 0.18, learning_rate <= 0.27,...	value: [0.13]
4	[learning_rate > 0.18, learning_rate <= 0.27,...	value: [0.13]

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

규칙으로 변환 (계속)

condition 칼럼에 있는 리스트는 각 요소를 &로 연결하고 output에 있는 값은 숫자만 추출하는 방식으로 수정하겠습니다.

규칙 나무 변환

```
1  def extract_float(output):
2      output = output.split('[')[1]
3      output = float(output[:-1])
4      return output
5
6  result['condition'] = result['condition'].apply(' & '.join)
7  result['output'] = result['output'].apply(extract_float)
8  result.sort_values(by = "output", inplace = True)
```

- **라인 1~4**: "value: [0.13]"과 같은 문자열을 0.13이라는 float로 변환하는 함수를 작성합니다. 구체적으로 라인 2에서 [를 기준으로 나눴을 때 1번째 있는 값 0.13]에서]를 제거하고 라인 3에서 float로 변환합니다.
- **라인 6**: apply 메서드를 사용해 condition 칼럼의 모든 요소에 join 함수를 적용합니다. join 함수는 모든 요소가 문자열인 리스트의 각 요소를 구분자로 연결해 문자열로 변환합니다.

Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

하이퍼 파라미터 범위 설정

성능이 높다고 판단한 네 개의 조건을 출력해서 하이퍼파라미터의 범위를 확인해보겠습니다.

```
1 print(result.iloc[0, 0])
2 print(result.iloc[1, 0])
3 print(result.iloc[2, 0])
4 print(result.iloc[3, 0])
```

```
learning_rate <= 0.05 & learning_rate <= 0.06 & num_leaves > 12.00 & boosting_type_goss <= 0.50 & num_leaves > 6.00
learning_rate > 0.05 & learning_rate <= 0.06 & num_leaves > 12.00 & boosting_type_goss <= 0.50 & num_leaves > 6.00
num_leaves <= 24.00 & learning_rate > 0.06 & num_leaves > 12.00 & boosting_type_goss <= 0.50 & num_leaves > 6.00
num_leaves > 24.00 & learning_rate > 0.06 & num_leaves > 12.00 & boosting_type_goss <= 0.50 & num_leaves > 6.00
```

0번째 조건부터 3번째 조건까지 공통되는 내용을 바탕으로 적절한 튜닝 범위를 설정

- 12 < num_leaves <= 24
- boosting_type_goss <= 0.50 (boosting_type == "gbdt")
- learning_rate는 공통되는 조건이 없으므로 특별히 범위를 설정하기 어렵지만, learning_rate가 모든 잎 노드에 등장했으므로 현재 설정한 구간이 적절하다고 할 수 있음