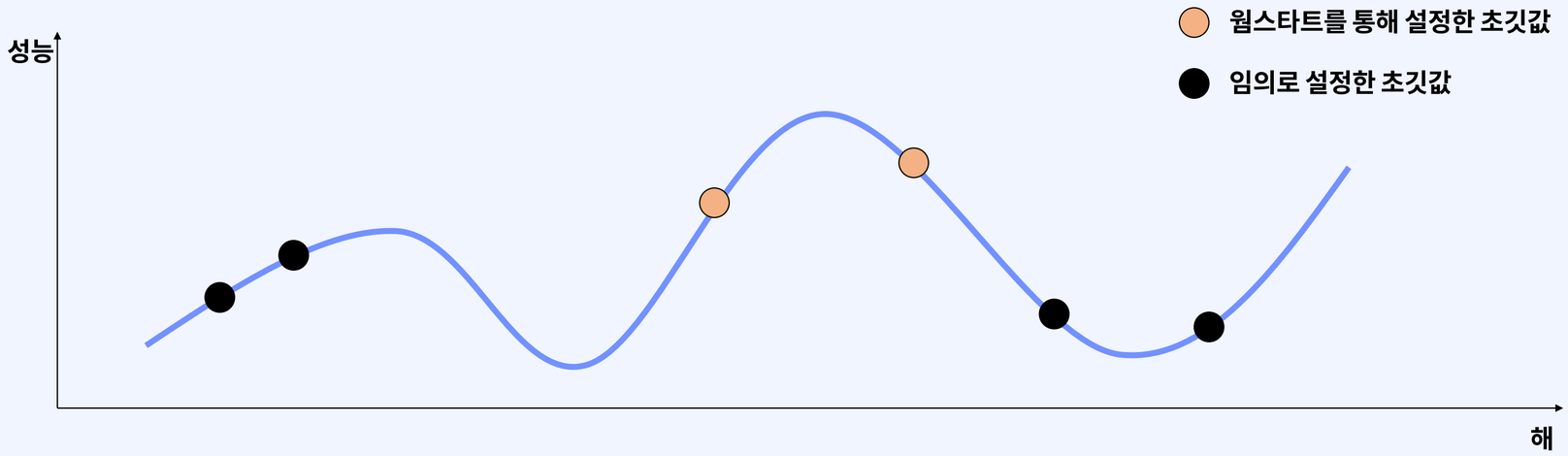
2. 웜 스타트와 메타 학습

1 메타 학습

웜 스타트란?

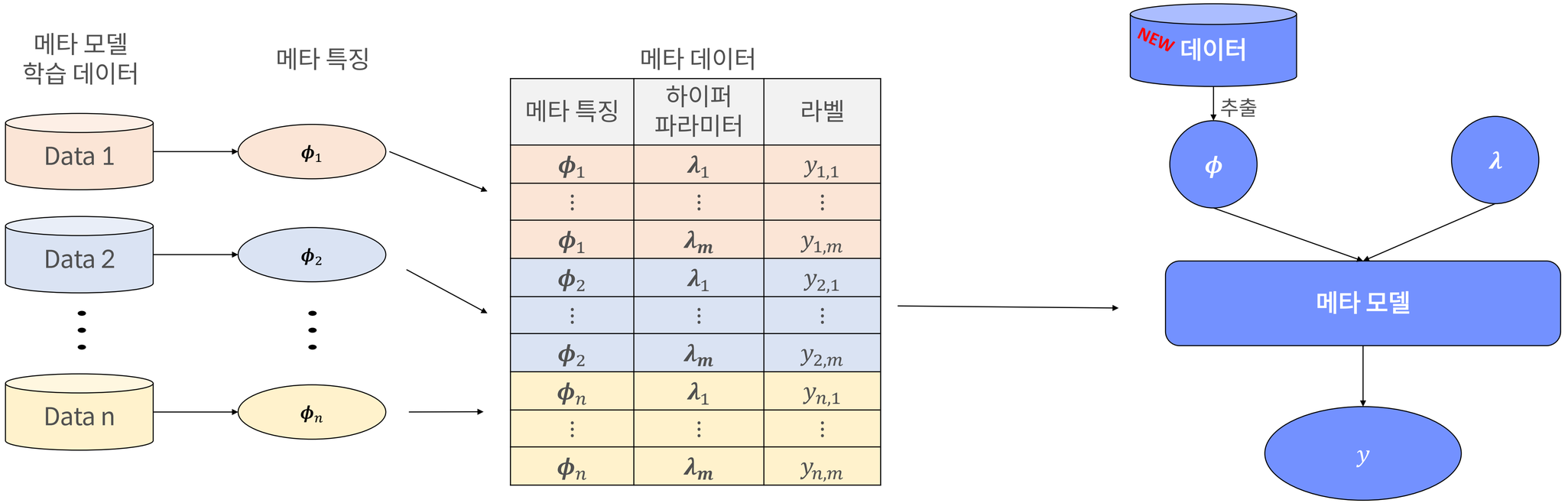
탐색 알고리즘 대부분은 임의로 설정한 초깃값부터 탐색을 시작하며, 초깃값에 따라 탐색 시간과 성능이 좌우됨

웜 스타트는 적절한 초깃값을 설정하여 탐색을 시작하는 것으로, 탐색 시간을 줄이고 좋은 성능의 해를 찾는 것을 도와줌



메타 모델을 활용한 웜스타트

메타 학습이란 데이터에 관한 데이터인 메타 데이터(meta data)로 모델을 학습하는 것을 의미하며, 이렇게 학습한 모델을 메타 모델이라고 합니다.
메타 모델은 데이터와 모델 정보를 입력하면 예상되는 성능을 반환합니다.



자주 사용하는 메타 특징

머신러닝 자동화 시스템에서 주로 사용하는 메타 특징은 다음과 같습니다.

자주 사용하는 메타 특징

특징	설명	특징	설명
샘플 개수	데이터에 포함된 샘플 개수	수치형 특징 개수	데이터에 포함된 수치형 특징 개수
특징 개수	데이터에 포함된 특징 개수	범주형 특징 개수	데이터에 포함된 범주형 특징 개수
클래스 대비 특징 비율	특징 개수 / 클래스 개수	수치형 특징 비율	수치형 특징 개수 / 특징 개수
클래스 비율 분포	(예) 각 클래스 비율의 최댓값, 각 클래스 비율의 최솟값	범주형 특징 비율	범주형 특징 개수 / 특징 개수
연속형 라벨의 분포	(예) 라벨의 범위, 라벨의 평균, 라벨의 표준편차	특징 간 상관관계 분포	특징 간 상관관계의 최댓값, 최솟값 등
샘플 대비 특징 비율	특징 개수 / 샘플 개수		

- 클래스 비율 분포와 특징 간 상관관계 분포를 통해 분포를 요약하는 통계량(예: 평균, 최댓값 등)을 메타 특징으로 사용함을 알 수 있음
- 예를 들어, 두 특징 X1과 X2 간 상관관계가 아니라 가능한 두 특징 간 상관관계의 통계량을 사용함
- 그 이유는 데이터마다 특징 개수, 클래스 개수 등이 다르므로, 각 특징 및 클래스에 대한 값을 메타 특징으로 사용하면 데이터마다 메타 특징 개수가 달라지기 때문

어느 데이터에 대해서나 동일하게 추출할 수 있고 모델 성능에 영향을 끼치는 메타 특징을 정의하는 것이 중요함

메타 모델의 종류

어떠한 지도 학습 모델도 메타 모델로 사용할 수 있지만, 적은 데이터로도 학습이 잘되고 특징의 중요도를 반영할 수 있는 모델이 좋습니다.

메타 모델 요구사항

일반적으로 메타 모델은 데이터 확보가 어려우므로 적은 데이터로 학습하더라도 잘 작동해야 함

→ k-최근접 이웃

메타 특징, 모델, 하이퍼파라미터의 중요도를 조절할 수 있어야 함

2. 웜 스타트와 메타 학습

2 실습 (1) 데이터 준비

메타 데이터 준비

autoMPG8.csv 데이터로 신경망의 하이퍼파라미터를 튜닝할 때 초깃값을 설정하기 위한 메타 학습을 해보겠습니다.

메타 데이터 준비

```
1  import pandas as pd
2  data_path = "../../data/regression"
3  meta_file_list = ["abalone.csv",
4                    "autoMPG6.csv",
5                    "baseball.csv",
6                    "friedman.csv",
7                    "stock.csv",
8                    "wankara.csv"]
9
10 meta_data_list = []
11 for file in meta_file_list:
12     df = pd.read_csv(data_path + "/" + file)
13     X = df.drop('y', axis = 1)
14     y = df['y']
15     meta_data_list.append((X, y))
```

- **라인 11~15:** meta_file_list에 정의된 파일 이름을 순회하면서 데이터를 불러오고 특징과 라벨로 분리한 뒤 튜플 형태로 meta_data_list에 추가합니다.
  즉, meta_data_list[i][0]은 i번째 데이터의 특징 벡터를, meta_data_list[i][1]은 i번째 데이터의 라벨을 나타냅니다.

실제로는 머신러닝 자동화 시스템이 구축돼 새로운 데이터가 끊임없이 들어올 때만 혹은 들어올 것이라 예상될 때만 메타 학습을 사용하며, 한 데이터에 대해서만 메타 모델을 적용하는 것은 매우 비효율적임

메타 특징 추출

메타 특징을 추출하는 함수를 작성하겠습니다.

메타 특징 추출 함수

```
1   def extract_meta_features(X, y):
2       num_samples, num_features = X.shape
3       label_max = y.max()
4       label_min = y.min()
5       label_mean = y.mean()
6       label_std = y.std()
7       corr_mean = X.corr().abs().values.mean()
8       corr_max = X.corr().abs().values.max()
9       corr_min = X.corr().abs().values.min()
10
11      meta_features = [num_samples,
12                       num_features,
13                       label_max,
14                       label_min,
15                       label_mean,
16                       label_std,
17                       corr_mean,
18                       corr_max,
19                       corr_min]
20
21      return meta_features
```

- **라인 2:** shape 메서드를 이용해 X의 행 개수와 열 개수를 각각 num_samples와 num_features에 저장합니다.
- **라인 3~6:** y의 최댓값, 최솟값, 평균, 표준편차를 각각 label_max, label_min, label_mean, label_std에 저장합니다.
- **라인 7~9:** 특징 간 상관관계의 절댓값의 평균, 최댓값, 최솟값을 각각 corr_mean, corr_max, corr_min에 저장합니다. 이때 데이터프레임의 집계 메서드를 사용하면 열별 통계량을 반환하므로 ndarray로 변환 후에 집계 메서드를 사용했습니다.

메타 특징 추출 (계속)

각 데이터로부터 메타 특징을 추출하겠습니다.

메타 특징 추출

```
1  meta_data = []
2  for X, y in meta_data_list:
3      meta_features = extract_meta_features(X, y)
4      meta_data.append(meta_features)
5
6  meta_col_names = ["num_samples",
7                    "num_features",
8                    "label_max",
9                    "label_min",
10                   "label_mean",
11                   "label_std",
12                   "corr_mean",
13                   "corr_max",
14                   "corr_min"]
15
16 meta_data = pd.DataFrame(meta_data, columns = meta_col_names)
17  meta_data['data_name'] = meta_file_list
```

- **라인 2~4:** meta_data_list에 속한 X와 y로부터 메타 특징을 추출하고 그 값을 meta_data에 추가합니다.
- **라인 17:** 각 데이터 이름을 나타내는 data_name 칼럼을 추가합니다. 이 칼럼은 하이퍼파라미터 튜닝 결과 데이터와 병합하겠습니다.

메타 데이터 생성

하이퍼파라미터를 샘플링하는 함수를 다음과 같이 작성하겠습니다.

하이퍼 파라미터 샘플러

```
1  import numpy as np
2  def hyperparameter_sampling():
3      h1 = np.random.randint(5, 15)
4      h2, h3, h4 = np.random.randint(0, 10, 3)
5      if h2 == 0:
6          h3, h4 = 0, 0
7      elif h3 == 0:
8          h4 = 0
9      max_iter = np.random.choice([100, 200, 1000, 2000])
10     random_state = np.random.choice([2020, 2021, 2022])
11     return h1, h2, h3, h4, max_iter, random_state
```

- **라인 3:** 첫 번째 은닉층에 포함될 노드 수 h1을 5와 15 사이에서 임의로 샘플링합니다. 첫 번째 은닉층에 포함될 노드는 0보다 커야 하므로 따로 샘플링했습니다.
- **라인 4:** 두 번째, 세 번째, 네 번째 은닉층에 포함될 노드 수 h2, h3, h4를 0부터 10 사이에서 임의로 샘플링합니다.
- **라인 5~6:** 두 번째 은닉층에 포함될 노드 수가 0이면 세 번째와 네 번째에 포함될 노드 수도 0으로 설정합니다.
- **라인 7~8:** 세 번째 은닉층에 포함될 노드 수가 0이면 네 번째에 포함될 노드 수도 0으로 설정합니다.
- **라인 9:** 최대 이터레이션 횟수 max_iter를 100, 200, 1000, 2000 중 하나로 설정합니다.

메타 데이터 생성 (계속)

데이터마다 1,000번씩 하이퍼파라미터를 샘플링하고 그때의 성능을 experiment_data에 저장하겠습니다.

메타 데이터 생성

```
1  from sklearn.metrics import mean_absolute_error as MAE
2  from sklearn.model_selection import cross_val_score
3  from sklearn.neural_network import MLPRegressor as MLP
4  import warnings
5  from tqdm import tqdm
6  warnings.filterwarnings("ignore")
7  experiment_data = []
```

- **라인 5:** 코드 실행 과정을 출력하기 위해 tqdm 함수를 불러왔습니다.
- **라인 6:** 불필요한 경고가 뜨는 것을 방지했습니다.

메타 데이터 생성 (계속)

데이터마다 1,000번씩 하이퍼파라미터를 샘플링하고 그때의 성능을 experiment_data에 저장하겠습니다.

```
8   for i in range(len(meta_data_list)):
9       data_name = meta_file_list[i]
10      X, y = meta_data_list[i]
11      print(data_name)
12      for j in tqdm(range(1000)):
13          h1, h2, h3, h4, max_iter, random_state = hyperparameter_sampling()
14          if h2 == 0:
15              layers = (h1, )
16          elif h3 == 0:
17              layers = (h1, h2)
18          elif h4 == 0:
19              layers = (h1, h2, h3)
20          else:
21              layers = (h1, h2, h3, h4)
22
23          model = MLP(hidden_layer_sizes = layers,
24                      max_iter = max_iter,
25                      random_state = random_state)
26          score_list = -cross_val_score(model, X, y, cv = 5,
27                                        scoring = "neg_mean_absolute_error")
28          score = score_list.mean()
29          record = [data_name, h1, h2, h3, h4, max_iter, random_state, score]
30          experiment_data.append(record)
31
32  hyper_param_cols = ["h1", "h2", "h3", "h4", "max_iter", "random_state"]
33  experiment_data = pd.DataFrame(experiment_data,
34                                 columns = ["data_name"] + hyper_param_cols + ["score"])
```

- **라인 8~10:** range(meta_data_list)를 순회하는 방식으로 meta_file_list와 meta_data_list를 순회합니다.
- **라인 11 ~ 12:** 코드 실행 시간이 기므로, data_name을 출력하고 tqdm을 사용해 진행률을 출력합니다.
- **라인 13:** hyperparameter_sampling 함수를 사용해 h1, h2, h3, h4, max_iter, random_state를 샘플링합니다.
- **라인 14~21:** h2, h3, h4의 값에 따라 layers를 설정합니다. 예를 들어, h2가 0이라면 layers를 (h1,)으로, h3가 0이라면 layers를 (h1, h2)로 설정합니다.
- **라인 23~30:** 샘플링한 하이퍼파라미터를 갖는 신경망을 k-겹 교차 검증을 통해 평가합니다. 또, 그 결과를 record에 추가하고 record를 experiment_data에 추가합니다.
- **라인 32~34:** experiment_data를 데이터프레임으로 변환합니다.

메타 데이터 생성 (계속)

meta_data에 experiment_data를 부착하여 meta_data를 완성하겠습니다.

```
1  meta_data = pd.merge(meta_data,
2                       experiment_data,
3                       on = "data_name")
4  display(meta_data.head())
```

	num_samples	num_features	label_max	label_min	label_mean	label_std	corr_mean	corr_max	corr_min	data_name	h1	h2	h3	h4	max_iter	random_
0	4176	8	29.0	1.0	9.931034	3.220003	0.807719	1.0	0.418048	abalone.csv	11	4	9	2	200	
1	4176	8	29.0	1.0	9.931034	3.220003	0.807719	1.0	0.418048	abalone.csv	9	8	3	8	2000	
2	4176	8	29.0	1.0	9.931034	3.220003	0.807719	1.0	0.418048	abalone.csv	13	5	3	1	100	
3	4176	8	29.0	1.0	9.931034	3.220003	0.807719	1.0	0.418048	abalone.csv	12	3	2	9	2000	
4	4176	8	29.0	1.0	9.931034	3.220003	0.807719	1.0	0.418048	abalone.csv	7	2	4	2	100	

2. 웜 스타트와 메타 학습

3 실습 2 – 메타 모델 학습 및 활용

메타 모델 학습

메타 모델로 k-최근접 이웃 모델을 사용하겠습니다. 하이퍼 파라미터 튜닝은 따로 하지 않았습니다.

메타 모델 학습

```
1  from sklearn.neighbors import KNeighborsRegressor as KNN
2  from sklearn.preprocessing import MinMaxScaler
3  meta_X = meta_data.drop(['data_name', 'score'], axis = 1)
4  meta_X['random_state_2020'] = (meta_X['random_state'] == 2020).astype(int)
5  meta_X['random_state_2021'] = (meta_X['random_state'] == 2021).astype(int)
6  meta_X = meta_X.drop(['random_state'], axis = 1)
7  meta_y = meta_data['score']
8  scaler = MinMaxScaler().fit(meta_X)
9  meta_X = scaler.transform(meta_X)
10 meta_model = KNN().fit(meta_X, meta_y)
```

- **라인 3:** meta_data에서 특징으로 활용할 수 없는 data_name과 score를 제거합니다.
- **라인 4~6:** random_state는 범주형 변수이므로 범주화합니다. 이 변수만 범주형 변수이므로 직접 더미화를 했습니다.
- **라인 8~9:** meta_X를 0과 1 사이로 스케일링합니다.

메타 모델 활용

새로 입력된 데이터 autoMPG8.csv에 대해 메타 모델을 사용해서 초깃값을 설정하겠습니다.

메타 모델 활용

```
df = pd.read_csv(data_path + "/autoMPG8.csv")
X = df.drop('y', axis = 1)
y = df['y']
meta_features = extract_meta_features(X, y)
sample_list = []
for _ in range(100):
    h1, h2, h3, h4, max_iter, random_state = hyperparameter_sampling()
    if h2 == 0:
        layers = (h1, )
    elif h3 == 0:
        layers = (h1, h2)
    elif h4 == 0:
        layers = (h1, h2, h3)
    else:
        layers = (h1, h2, h3, h4)
    sample_list.append(meta_features + [h1, h2, h3, h4, max_iter, random_state])

sample_list = pd.DataFrame(sample_list,
                           columns = meta_col_names + hyper_param_cols)

sample_list['random_state_2020'] = (sample_list['random_state'] == 2020).astype(int)
sample_list['random_state_2021'] = (sample_list['random_state'] == 2021).astype(int)
sample_list = sample_list.drop(['random_state'], axis = 1)
y_pred = meta_model.predict(scaler.transform(sample_list))
```

메타 모델 활용 (계속)

새로 입력된 데이터 autoMPG8.csv에 대해 메타 모델을 사용해서 초깃값을 설정하겠습니다.

메타 모델 활용

```
1  init_sample = sample_list.loc[np.argsort(y_pred)[:3]]
2  display(init_sample)
```

	num_samples	num_features	label_max	label_min	label_mean	label_std	corr_mean	corr_max	corr_min	h1	h2	h3	h4	max_iter	random_state
52	392	7	46.6	9.0	23.445918	7.805007	0.628158	1.0	0.181528	9	8	7	9	1000	2021
84	392	7	46.6	9.0	23.445918	7.805007	0.628158	1.0	0.181528	9	9	5	8	1000	2021
21	392	7	46.6	9.0	23.445918	7.805007	0.628158	1.0	0.181528	6	6	2	7	2000	2021

메타 모델 검증

sample_list에 속한 값의 라벨인 y_actual을 계산하여 비교해보겠습니다

메타 모델 검증

```
1   y_actual = []
2   for sample in sample_list:
3       h1, h2, h3, h4, max_iter, random_state = sample[-6:]
4       if h2 == 0:
5           layers = (h1, )
6       elif h3 == 0:
7           layers = (h1, h2)
8       elif h4 == 0:
9           layers = (h1, h2, h3)
10      else:
11          layers = (h1, h2, h3, h4)
12
13      model = MLP(hidden_layer_sizes = layers,
14                  max_iter = max_iter,
15                  random_state = random_state)
16
17      score_list = -cross_val_score(model, X, y, cv = 5,
18                                    scoring = "neg_mean_absolute_error")
19
20      score = score_list.mean()
21      y_actual.append(score)
```

- **라인 2~3:** sample_list에 있는 요소를 sample로 순회하면서 하이퍼파라미터와 관련된 부분만 입력받습니다. 왜냐하면 새로운 데이터의 메타 특징은 하나여서 샘플링할 때마다 바뀌지 않으므로 학습에 전혀 도움이 되지 않기 때문입니다.

메타 모델 검증

y_actual과 y_pred 간의 스피어만 상관계수(Spearman correlation coefficient)를 계산해보겠습니다. 스피어만 상관계수는 두 변수의 순위 간 상관관계를 나타내는 지표로 두 변수의 스케일에 영향을 받지 않습니다.

메타 모델 검증

```
1  from scipy.stats import spearmanr, rankdata
2  print(spearmanr(y_actual, y_pred))
3  print(rankdata(y_actual)[np.argsort(y_pred)[:3]])
```

- **라인 2:** y_actual과 y_pred 간 스피어만 상관계수를 계산합니다.
- **라인 3:** 메타 모델이 MAE가 가장 작으리라 예측한 세 개의 해가 실제 평가 결과에서 몇 위를 차지하는지를 출력합니다.

```
SpearmanrResult(correlation=0.6190514766137369, pvalue=6.652410770978535e-12)
[13. 26.  7.]
```

- 예측값과 실젯값 간 스피어만 상관계수가 0.6191로, 두 변수 간 양의 상관관계가 존재함
- 또한, 예측 결과에서 1등부터 3등까지라고 판단한 세 개의 샘플은 실제로는 13등, 26등, 7등으로, 다소 아쉽기는 하지만 사용하기 어려운 수준은 아님