그리드서치

그리드서치(Grid Search)

- 가장 높은 성능을 갖는 최적의 하이퍼파라미터 조합을 찾는 방법
- 하이퍼파라미터 그리드는 하이퍼파라미터 조합 후보를 의미
- 모든 조합 후보들에 대해 검증 데이터를 활용한 교차검증 시도
- 하이퍼파라미터 간 모든 조합에 대해 검증하기 때문에 복잡도가 매우 높음

p \ K	1	3	5	7	10
1	(1, 1)	(1, 3)	(1, 5)	(1, 7)	(1, 10)
2	(2, 1)	(2, 3)	(2, 5)	(2, 7)	(2, 10)

KNN (n_neighbors, p)

그리드서치 과정

- 학습 데이터와 테스트 데이터 구분
- 탐색할 하이퍼파라미터 조합 지정
- 학습 데이터 기반으로 각각의 하이퍼파라미터 조합에 대한 모델 생성
- 교차검증을 통하여 최고의 평균 점수에 해당하는 하이퍼파라미터 조합 결정
- 최적의 하이퍼파라미터와 학습 데이터 기반으로 최종 모델 생성
- 테스트 데이터 기반으로 최종 모델에 대한 점수 평가

sklearn.model_selection.GridSearchCV

class sklearn.model_selection.GridSearchCV(*estimator*, *param_grid*, ***, *scoring=None*, *n_jobs=None*, *refit=True*, *cv=None*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *error_score=nan*, *return_train_score=False*) [source]

Exhaustive search over specified parameter values for an estimator.

estimator : *estimator object*

This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a `score` function, or `scoring` must be passed.

param_grid : *dict or list of dictionaries*

Dictionary with parameters names (`str`) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.

cv : *int, cross-validation generator or an iterable, default=None*

Determines the cross-validation splitting strategy. Possible inputs for cv are:

- None, to use the default 5-fold cross validation,
- integer, to specify the number of folds in a `(Stratified)KFold`,

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV

그리드서치 기반 KNN

```
import seaborn as sns
titanic = sns.load_dataset("titanic")
data = titanic[["sex", "age", "sibsp", "adult_male", "parch"]].copy()
t = titanic["survived"]
data["age"].fillna(30, inplace=True)
data["sex"].replace("male", 1, inplace=True)
data["sex"].replace("female", 0, inplace=True)

from sklearn.model_selection import train_test_split
train_data, test_data, train_target, test_target = train_test_split(
    data, t, test_size=0.3, random_state=42, stratify=t)
```

```
from sklearn.neighbors import KNeighborsClassifier
params = {"n_neighbors": [1, 3, 5, 7, 10], "p": [1, 2]}
from sklearn.model_selection import GridSearchCV
gs = GridSearchCV(KNeighborsClassifier(), params, cv=10)
```

```
gs.fit(train_data, train_target)
kn = gs.best_estimator_
```

```
print(kn.score(train_data, train_target))
print(kn.score(test_data, test_target))
```

```
0.826645264847512
0.7798507462686567
```

best_estimator_ : *estimator*

Estimator that was chosen by the search, i.e. estimator which gave highest score (or smallest loss if specified) on the left out data. Not available if `refit=False`.

K-폴드(K-Fold) 교차검증

```
from sklearn.model_selection import train_test_split
train_data, test_data, train_target, test_target = train_test_split(
    data, t, test_size = 0.3, random_state = 42, stratify = t)

from sklearn.model_selection import cross_validate
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
best_score = best_n = best_m = 0
```

구현
(cv=10)

10-폴드
교차검증

n_neighbors

p

```
[0.6965949820788532, 0.7480286738351254, 0.7831285202252944, 0.8121607782898106, 0.778469022017409]
[0.6901689708141321, 0.74152585765489, 0.7734254992319509, 0.7975934459805426, 0.7672555043522785]
```

평균검증값

cross_validate(), GridSearchCV 비교

```
scores = cross_validate(model, train_data, train_target, cv=10, return_train_score=True)
```

n_neighbors

p

```
[0.6965949820788532, 0.7480286738351254, 0.7831285202252944, 0.8121607782898106, 0.778469022017409]
[0.6901689708141321, 0.74152585765489, 0.7734254992319509, 0.7975934459805426, 0.7672555043522785]
```

cv_results_ : *dict of numpy (masked) ndarrays*

A dict with keys as column headers and values as columns, that can be imported into a pandas `DataFrame`.

```
{'n_neighbors': 7, 'p': 1}
[0.69659498 0.69016897 0.74802867 0.74152586 0.78312852 0.7734255
 0.81216078 0.79759345 0.77846902 0.7672555 ]
[{'n_neighbors': 1, 'p': 1}, {'n_neighbors': 1, 'p': 2}, {'n_neighbors': 3, 'p': 1}, {'n_neighbors': 3, 'p': 2},
{'n_neighbors': 5, 'p': 1}, {'n_neighbors': 5, 'p': 2}, {'n_neighbors': 7, 'p': 1}, {'n_neighbors': 7, 'p': 2},
{'n_neighbors': 10, 'p': 1}, {'n_neighbors': 10, 'p': 2}]
```

그리드서치 결과

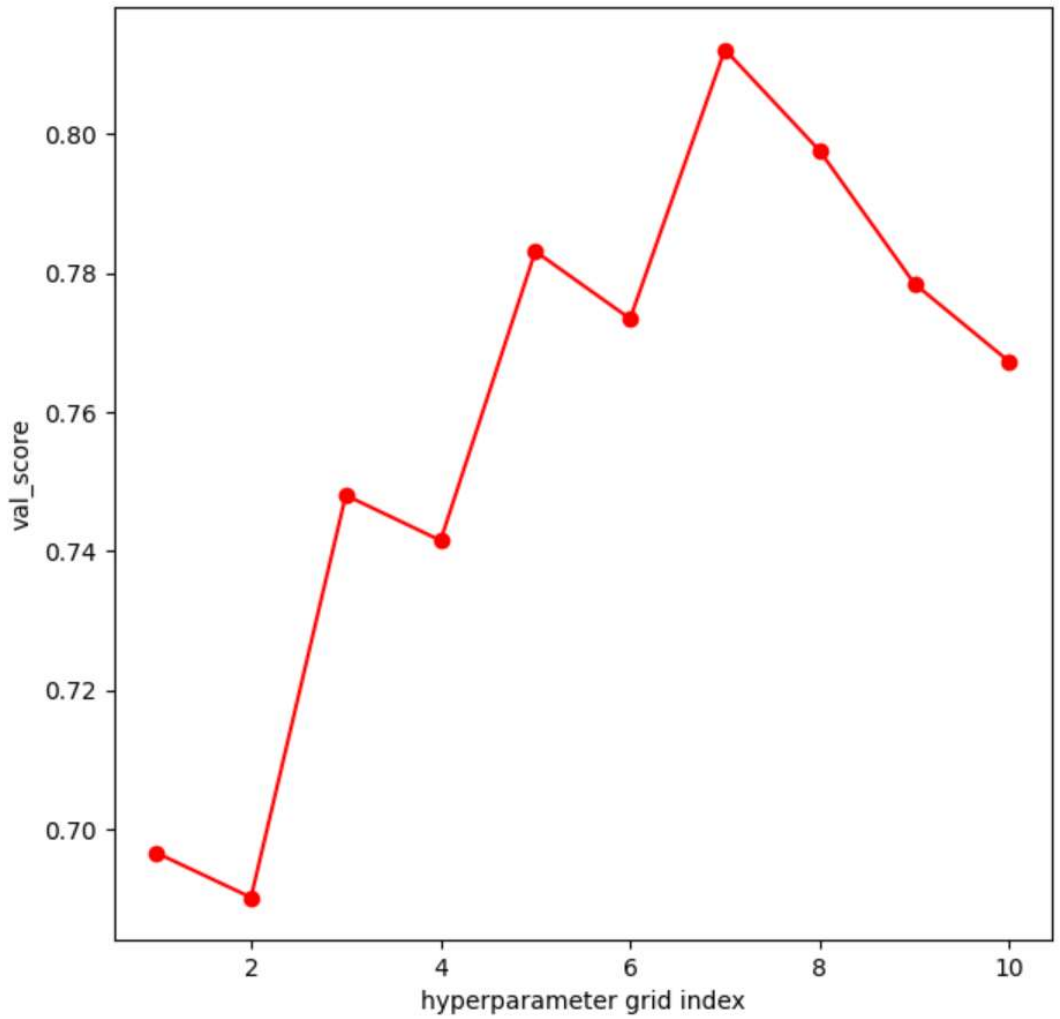
```
import matplotlib.pyplot as plt

구현

plt.show()
```

p \ K	1	3	5	7	10
1	1	3	5	7	9
2	2	4	6	8	10

그리드 인덱스



랜덤서치(Random Search)

- 그리드서치와 동일하게 작동
- 모든 하이퍼파라미터 조합을 평가하지 않고, 지정된 횟수만큼의 조합만 평가
- 각 반복에서 평가되는 조합은 무작위로 선정

p \ K	1	3	5	7	10
1	(1, 1)			(1, 7)	
2		(2, 3)		(2, 7)	(2, 10)

KNN (n_neighbors, p), 5개 조합 평가

sklearn.model_selection.RandomizedSearchCV

class sklearn.model_selection.RandomizedSearchCV(*estimator*, *param_distributions*, ***, *n_iter=10*, *scoring=None*, *n_jobs=None*, *refit=True*, *cv=None*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *random_state=None*, *error_score=nan*, *return_train_score=False*) [source]

Randomized search on hyper parameters.

estimator : *estimator object*

A object of that type is instantiated for each grid point. This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a `score` function, or `scoring` must be passed.

param_distributions : *dict or list of dicts*

Dictionary with parameters names (`str`) as keys and distributions or lists of parameters to try. Distributions must provide a `rvs` method for sampling (such as those from scipy.stats.distributions). If a list is given, it is sampled uniformly. If a list of dicts is given, first a dict is sampled uniformly, and then a parameter is sampled using that dict as above.

n_iter : *int, default=10*

Number of parameter settings that are sampled. n_iter trades off runtime vs quality of the solution.

cv : *int, cross-validation generator or an iterable, default=None*

Determines the cross-validation splitting strategy. Possible inputs for cv are:

- None, to use the default 5-fold cross validation,
- integer, to specify the number of folds in a `(Stratified)KFold`,

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html#sklearn.model_selection.RandomizedSearchCV

랜덤서치 기반 KNN

```
import seaborn as sns
titanic = sns.load_dataset("titanic")
data = titanic[["sex", "age", "sibsp", "adult_male", "parch"]].copy()
t = titanic["survived"]
data["age"].fillna(30, inplace=True)
data["sex"].replace("male", 1, inplace=True)
data["sex"].replace("female", 0, inplace=True)

from sklearn.model_selection import train_test_split
train_data, test_data, train_target, test_target = train_test_split(
    data, t, test_size=0.3, random_state=42, stratify=t)
```

```
from sklearn.neighbors import KNeighborsClassifier
params = {"n_neighbors": [1, 3, 5, 7, 10], "p": [1, 2]}
```

구현 (10-폴드, 5개 조합 탐색)

```
, random_state=42)
```

```
gs.fit(train_data, train_target)
kn = gs.best_estimator_
```

```
print(kn.score(train_data, train_target))
print(kn.score(test_data, test_target))
```

```
0.826645264847512
0.7723880597014925
```

그리드서치, 랜덤서치 비교

```
{'n_neighbors': 7, 'p': 1}
[0.69659498 0.69016897 0.74802867 0.74152586 0.78312852 0.7734255
 0.81216078 0.79759345 0.77846902 0.7672555 ]
[{'n_neighbors': 1, 'p': 1}, {'n_neighbors': 1, 'p': 2}, {'n_neighbors': 3, 'p': 1}, {'n_neighbors': 3, 'p': 2},
{'n_neighbors': 5, 'p': 1}, {'n_neighbors': 5, 'p': 2}, {'n_neighbors': 7, 'p': 1}, {'n_neighbors': 7, 'p': 2},
{'n_neighbors': 10, 'p': 1}, {'n_neighbors': 10, 'p': 2}]
```

그리드서치

```
print(kn.score(train_data, train_target))
print(kn.score(test_data, test_target))
```

```
0.826645264847512
0.7798507462686567
```

```
{'p': 2, 'n_neighbors': 7}
[0.78008193 0.7063236  0.78156682 0.71274962 0.78315412]
[{'p': 1, 'n_neighbors': 10}, {'p': 2, 'n_neighbors': 1}, {'p': 2, 'n_neighbors': 5}, {'p': 1, 'n_neighbors': 1}, {'p': 2, 'n_neighbors': 7}]
```

랜덤서치

```
print(kn.score(train_data, train_target))
print(kn.score(test_data, test_target))
```

```
0.826645264847512
0.7723880597014925
```

참고자료

- 지능기전공학부 최유경 교수님 자료, https://github.com/sejongresearch/2021.MachineLearning
- 코랩(Colab), https://colab.research.google.com/
- 파이썬(Python), https://www.python.org/doc/
- 사이킷런(sckit-learn), https://scikit-learn.org/stable/index.html
- 판다스(pandas), https://pandas.pydata.org/
- 맷플롯립(matplotlib), https://matplotlib.org/
- 씨본(seaborn), https://seaborn.pydata.org/
- 캐글(Kaggle), https://www.kaggle.com/
- 넘파이(numpy), https://numpy.org/doc/stable/
- 스택오퍼플러우(stackoverflow), https://stackoverflow.com/