Hyperparameters

실습 코드 : https://bit.ly/3zVFQPS

2022.08.05.

한국에너지기술연구원 계산과학연구실

이제현

머신 러닝 강좌

- base : Scikit-learn MOOC @inria
  - 머신 러닝 위주. 딥 러닝은 skip (기회가 되면 한번쯤은 다룰 수도..?)
  - 소스 코드 포함 강의 자료 : 원내 게시판 공개
  - 강의 영상 : KIER-Tube & Youtube 공개

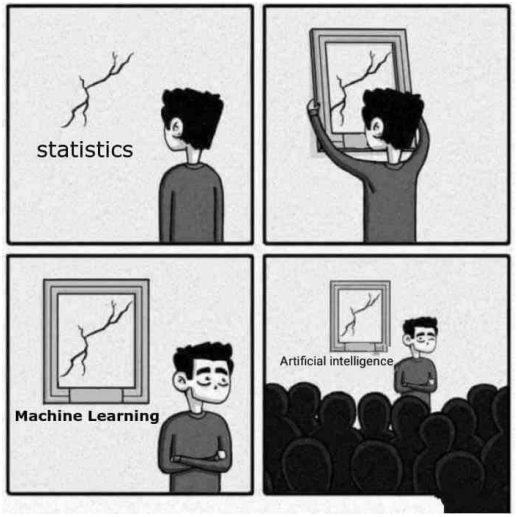
1차 모임 (4월) 머신러닝 기본 개념			
2차 모임 (5월) Modeling pipeline	3차 모임 (6월) Best Model?	4차 모임 (7월) Tree Models	5차 모임 (8월) Hyperparameter
6차 모임 (8월) Nonlinear Models	7차 모임 (9월) Ensemble Models	8차 모임 (10월) Neural Network (?)	9차 모임 (11월) 마무리

짧은 복습

모델보다 데이터가 우선입니다. | **데이터 변환도 학습 후 적용합니다.** | **train set ~ valid set ~ test set 만들기** | **트리 모델 좋습니다. 묶어서도 씁니다.**

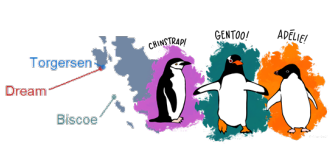
1. 머신 러닝 기본 개념

- 머신 러닝 = 통계학
- 제대로 된 데이터가 없으면?
- 데이터를 제대로 안 나누면?



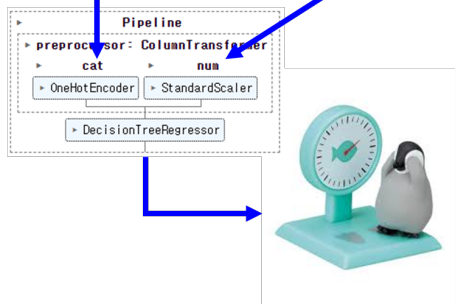
2. 머신 러닝 Pipeline

- 수치형 → 표준화
- 범주형 → 인코딩
- 일관성을 시스템으로 확보



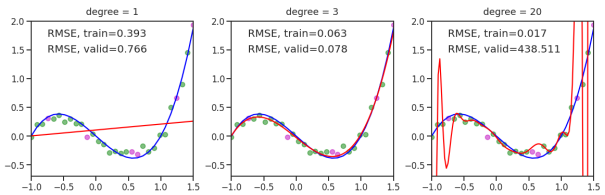
펭귄 서식지 3곳, 종 3가지, 성별

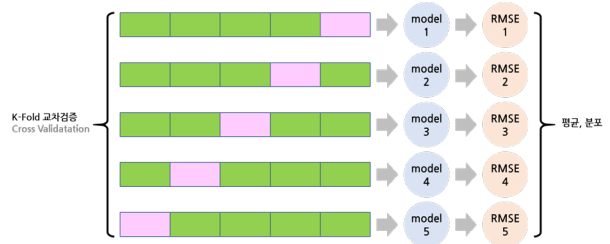
신체치수 3가지



3. Best Model?

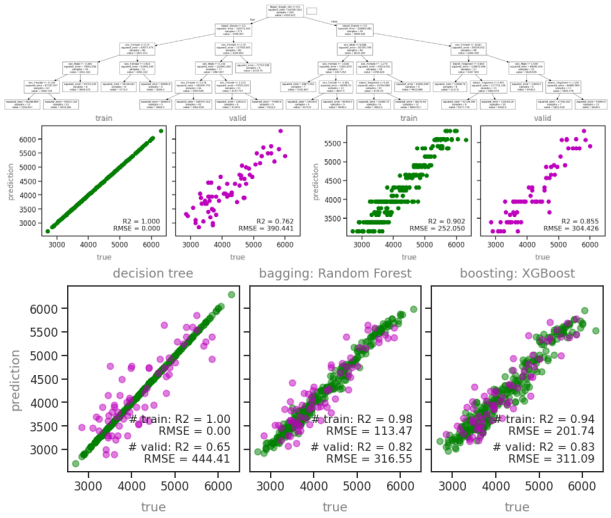
- Fitting vs Machine Learning
- 데이터셋 분할: 층화 추출
- 검증 & 교차 검증





4. Tree Model + AutoML

- Decision Tree + HP tuning
- 앙상블: Bagging & Boosting
- 트리 모델 사용시 주의점



머신 러닝 모델링이 대리석 조각이라면

Hyperparameter tuning

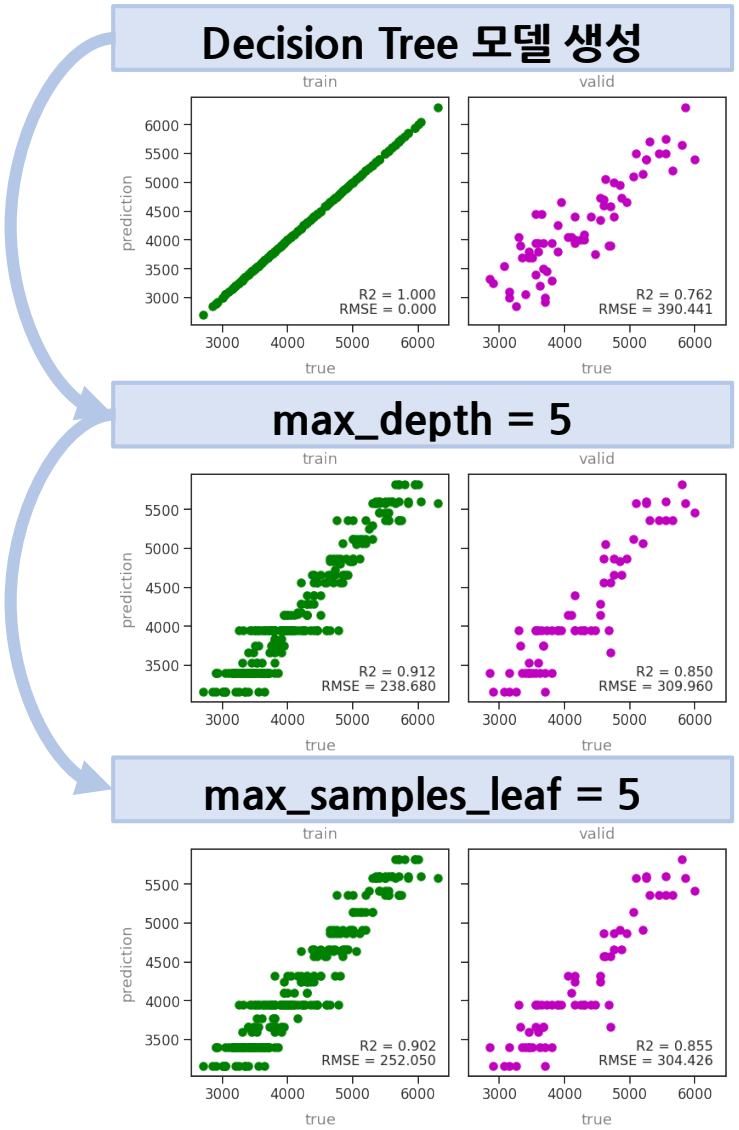
Antonio Corradini, “La Pudicizia” (1752), Napoli, Italia
https://www.flickr.com/photos/catchlightsa/9064677741

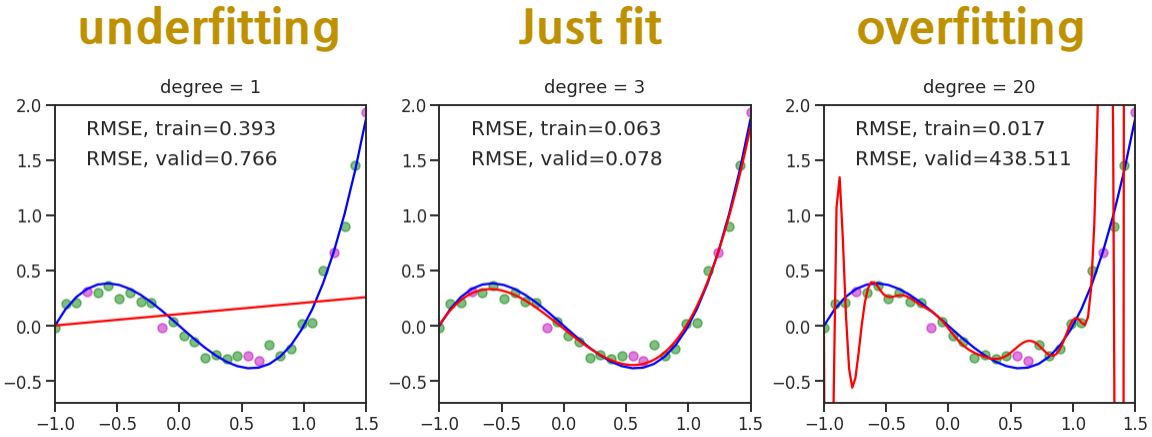
애초에 잘못 다루면 다듬을 것도 없습니다.

- 주제를 잘못 잡았거나
- 데이터를 잘못 수집했거나
- 잘 수집한 데이터 퀄리티가 엉망이거나
- 데이터를 잘못 분석했거나
- 엉뚱한 모델을 썼거나
- 등등등

Banksy, “Fallen Statue” (2011)
https://www.freshnessmag.com/2011/12/21/banksy-fallen-statue/

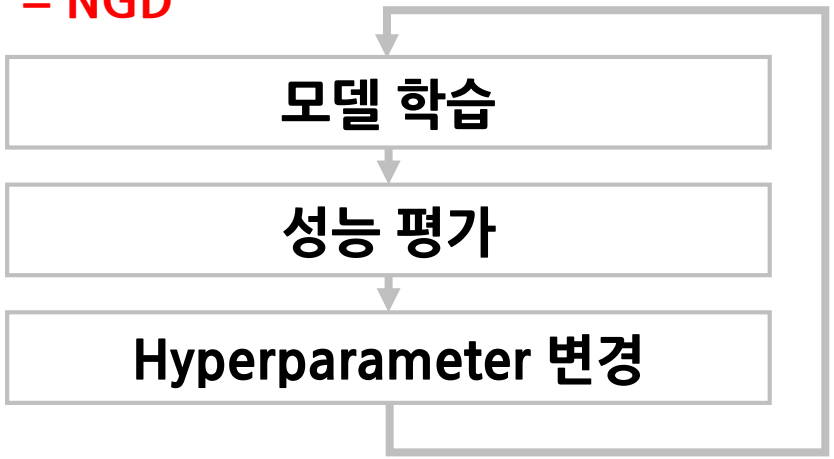
머신 러닝 모델 다듬기



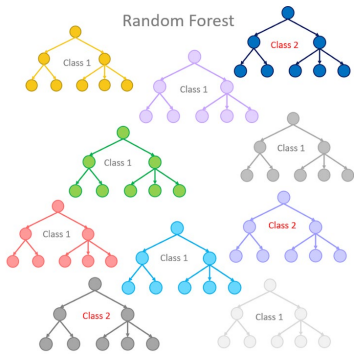


언제까지?
Valid metric이 더 이상 개선되지 않을 때까지
= 여러 번 반복해도 나아지지 않을 때까지

= NGD



Hyperparameter: RandomForestRegressor




```
1 model["ml"].get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

sklearn.ensemble.RandomForestRegressor

n_estimators : *int, default=100*
The number of trees in the forest.

Changed in version 0.22: The default value of `n_estimators` changed from 10 to 100 in 0.22.

criterion : *{"squared_error", "absolute_error", "poisson"}, default="squared_error"*
The function to measure the quality of a split. Supported criteria are "squared_error" for the mean squared error, which is equal to variance reduction as feature selection criterion, "absolute_error" for the mean absolute error, and "poisson" which uses reduction in Poisson deviance to find splits. Training using "absolute_error" is significantly slower than when using "squared_error".

New in version 0.18: Mean Absolute Error (MAE) criterion.

New in version 1.0: Poisson criterion.

Deprecated since version 1.0: Criterion "mse" was deprecated in v1.0 and will be removed in version 1.2. Use `criterion="squared_error"` which is equivalent.

Deprecated since version 1.0: Criterion "mae" was deprecated in v1.0 and will be removed in version 1.2. Use `criterion="absolute_error"` which is equivalent.

max_depth : *int, default=None*
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int or float, default=2*
The minimum number of samples required to split an internal node:

- If int, then consider `min_samples_split` as the minimum number.
- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Changed in version 0.18: Added float values for fractions.

min_samples_leaf : *int or float, default=1*
The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least `min_samples_leaf` training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

- If int, then consider `min_samples_leaf` as the minimum number.
- If float, then `min_samples_leaf` is a fraction and `ceil(min_samples_leaf * n_samples)` are the minimum number of samples for each node.

Changed in version 0.18: Added float values for fractions.

min_weight_fraction_leaf : *float, default=0.0*
The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

max_features : *{"sqrt", "log2", None}, int or float, default=1.0*
The number of features to consider when looking for the best split:

- If int, then consider `max_features` features at each split.
- If float, then `max_features` is a fraction and `round(max_features * n_features)` features are considered at each split.
- If "auto", then `max_features=n_features`.
- If "sqrt", then `max_features=sqrt(n_features)`.
- If "log2", then `max_features=log2(n_features)`.
- If None or 1.0, then `max_features=n_features`.

Note: The default of 1.0 is equivalent to bagged trees and more randomness can be achieved by setting smaller values, e.g. 0.3.

Changed in version 1.1: The default of `max_features` changed from `"auto"` to 1.0.

Deprecated since version 1.1: The `"auto"` option was deprecated in 1.1 and will be removed in 1.3.

Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features.

max_leaf_nodes : *int, default=None*
Grow trees with `max_leaf_nodes` in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

min_impurity_decrease : *float, default=0.0*
A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

The weighted impurity decrease equation is the following:

```
N_t / N * (impurity - N_t_R / N_t * right_impurity
                    - N_t_L / N_t * left_impurity)
```

where `N` is the total number of samples, `N_t` is the number of samples at the current node, `N_t_L` is the number of samples in the left child, and `N_t_R` is the number of samples in the right child.

`N`, `N_t`, `N_t_R` and `N_t_L` all refer to the weighted sum, if `sample_weight` is passed.

New in version 0.19.

bootstrap : *bool, default=True*
Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

oob_score : *bool, default=False*
Whether to use out-of-bag samples to estimate the generalization score. Only available if bootstrap=True.

n_jobs : *int, default=None*
The number of jobs to run in parallel. `fit`, `predict`, `decision_path` and `apply` are all parallelized over the trees. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.

random_state : *int, RandomState instance or None, default=None*
Controls both the randomness of the bootstrapping of the samples used when building trees (if `bootstrap=True`) and the sampling of the features to consider when looking for the best split at each node (if `max_features < n_features`). See Glossary for details.

verbose : *int, default=0*
Controls the verbosity when fitting and predicting.

warm_start : *bool, default=False*
When set to `True`, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest. See the Glossary.

ccp_alpha : *non-negative float, default=0.0*
Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than `ccp_alpha` will be chosen. By default, no pruning is performed. See Minimal Cost-Complexity Pruning for details.

New in version 0.22.

max_samples : *int or float, default=None*
If bootstrap is True, the number of samples to draw from X to train each base estimator.

- If None (default), then draw `X.shape[0]` samples.
- If int, then draw `max_samples` samples.
- If float, then draw `max_samples * X.shape[0]` samples. Thus, `max_samples` should be in the interval `(0.0, 1.0]`.

New in version 0.22.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Hyperparameter: 경우의 수

N_S : number of sample
N_F : number of features

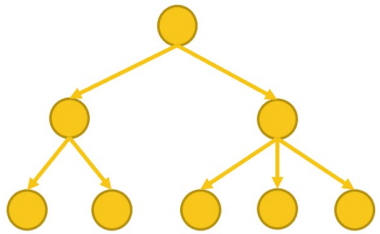
```
1 model["ml"].get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

hyperparameter	경우의 수	설명	입력 가능 범위
bootstrap	2	bootstrap (부분집합) 사용 여부	True, False
ccp_alpha	∞	Cost-Complexity Pruning 상한선	0 ~ ∞
criterion	3	node 분기 손실 함수	“squared_error”, “absolute_error”, “poisson”
max_depth	∞	tree 최대 깊이	0 ~ ∞
max_features	∞	분할에 관여할 최대 feature 수	1 ~ N_F, “auto”, “sqrt”, “log2”
max_leaf_nodes	∞	최대 leaf node 수	2 ~ ∞
max_samples	N_S	bootstratp 적용시 데이터 수	1 ~ N_S
min_inpurity_decrease	∞	node 분할 조건	0 ~ ∞
min_samples_leaf	∞ (N_S)	node내 최소 data 수	1 ~ ∞
min_samples_split	∞ (N_S)	node 분기 가능한 최소 data 수	1 ~ ∞
min_weight_fraction_leaf	∞ (N_S)	~min_samples_leaf. 가중치 포함.	1 ~ ∞
n_estimators	∞	random forest 구성 decision tree 수	1 ~ ∞
random_state	$2^{32} + 1$	bootstrap, feature sampling 적용	1 ~ 2^{32}
warm_start	2	기존 학습 모델 재학습	True, False

Hyperparameter: Decision Tree Regressor

Single Decision Tree



```
1 dts = model["ml"].estimators_
2 dts[0].get_params()
```

```
{'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'presort': 'deprecated',
 'random_state': 600559336,
 'splitter': 'best'}
```

sklearn.tree.DecisionTreeRegressor

ccp_alpha : ***non-negative float, default=0.0***

Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than `ccp_alpha` will be chosen. By default, no pruning is performed. See Minimal Cost-Complexity Pruning for details.

criterion : ***{"squared_error", "absolute_error", "poisson"}, default="squared_error"***

The function to measure the quality of a split. Supported criteria are "squared_error" for the mean squared error, which is equal to variance reduction as feature selection criterion, "absolute_error" for the mean absolute error, and "poisson" which uses reduction in Poisson deviance to find splits. Training using "absolute_error" is significantly slower than when using "squared_error".

max_depth : ***int, default=None***

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

max_features : ***{"sqrt", "log2", None}, int or float, default=1.0***

The number of features to consider when looking for the best split:

- If int, then consider `max_features` features at each split.
- If float, then `max_features` is a fraction and `round(max_features * n_features)` features are considered at each split.
- If "auto", then `max_features=n_features`.
- If "sqrt", then `max_features=sqrt(n_features)`.
- If "log2", then `max_features=log2(n_features)`.
- If None or 1.0, then `max_features=n_features`.

max_leaf_nodes : ***int, default=None***

Grow trees with `max_leaf_nodes` in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

min_impurity_decrease : ***float, default=0.0***

A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

The weighted impurity decrease equation is the following:

```
N_t / N * (impurity - N_t_R / N_t * right_impurity
                    - N_t_L / N_t * left_impurity)
```

where `N` is the total number of samples, `N_t` is the number of samples at the current node, `N_t_L` is the number of samples in the left child, and `N_t_R` is the number of samples in the right child.

`N`, `N_t`, `N_t_R` and `N_t_L` all refer to the weighted sum, if `sample_weight` is passed.

min_samples_leaf : ***int or float, default=1***

The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least `min_samples_leaf` training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

min_impurity_decrease : ***float, default=0.0***

A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

The weighted impurity decrease equation is the following:

```
N_t / N * (impurity - N_t_R / N_t * right_impurity
                    - N_t_L / N_t * left_impurity)
```

where `N` is the total number of samples, `N_t` is the number of samples at the current node, `N_t_L` is the number of samples in the left child, and `N_t_R` is the number of samples in the right child.

`N`, `N_t`, `N_t_R` and `N_t_L` all refer to the weighted sum, if `sample_weight` is passed.

min_samples_split : ***int or float, default=2***

The minimum number of samples required to split an internal node:

- If int, then consider `min_samples_split` as the minimum number.
- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

min_weight_fraction_leaf : ***float, default=0.0***

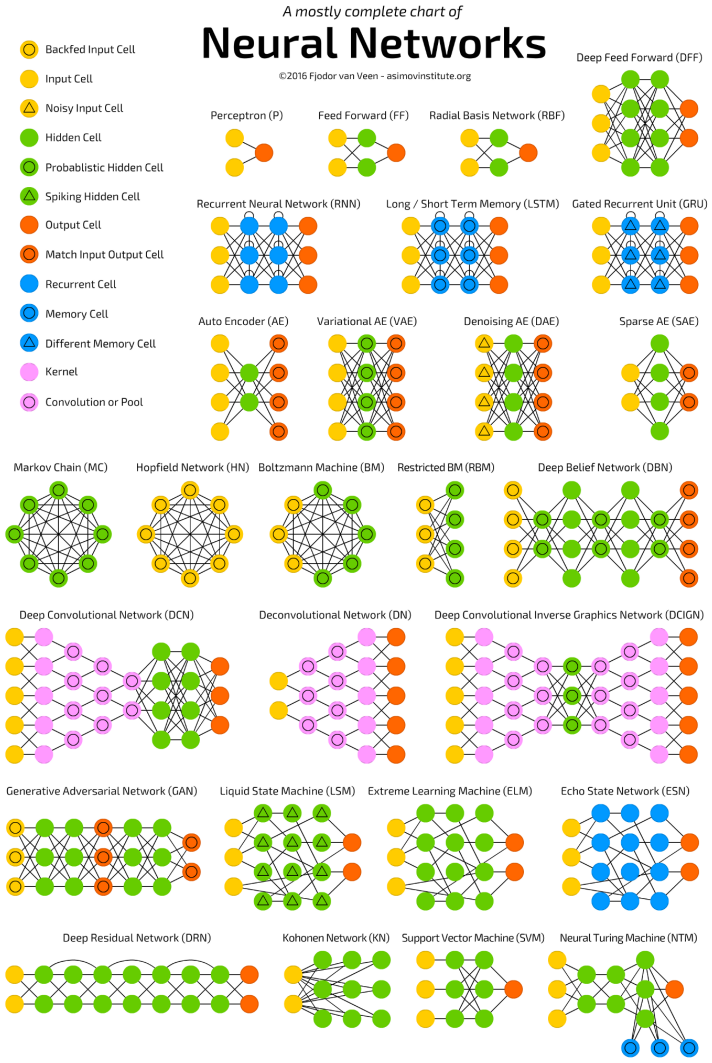
The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

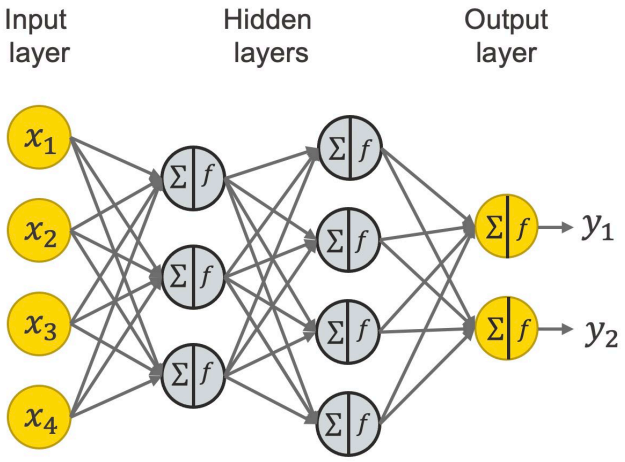
splitter : ***{"best", "random"}, default="best"***

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

Hyperparameter: Deep Neural Network

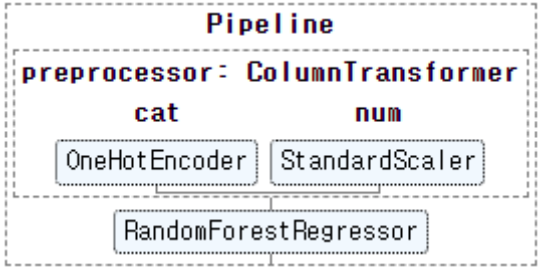




hyperparameter		경우의 수	설명	입력 가능 범위
Network 구조		∞	Layer와 Node의 연결 구조	∞
No. of layers		∞	layer 수	∞
Learning rate		∞	학습 효율	∞
“모든” Hidden Layers	No. of nodes	∞	Node 수	∞
	Activation function	3+α	$W \times X$ 후 activation	sigmoid, tanh, ReLU, Leaky ReLU, PReLU, ELU, Maxout
	Batch Normalization	2	Batch Normalization 적용 여부	True, False
	drop out	∞	학습 불참 node 비율	0 ~ 1

어떤 Hyperparameter를 어떻게 할까요?

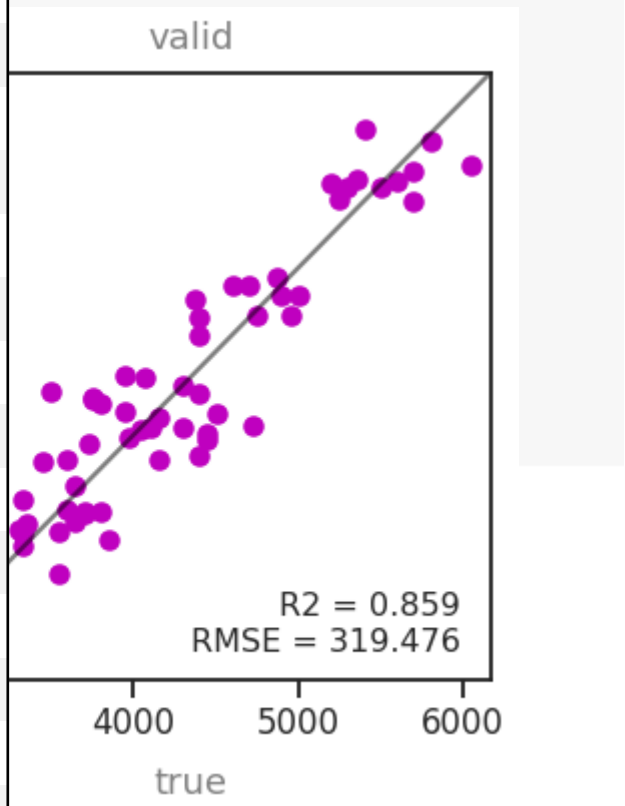
- 기본 Hyperparameter로 학습 Random Forest Pipeline



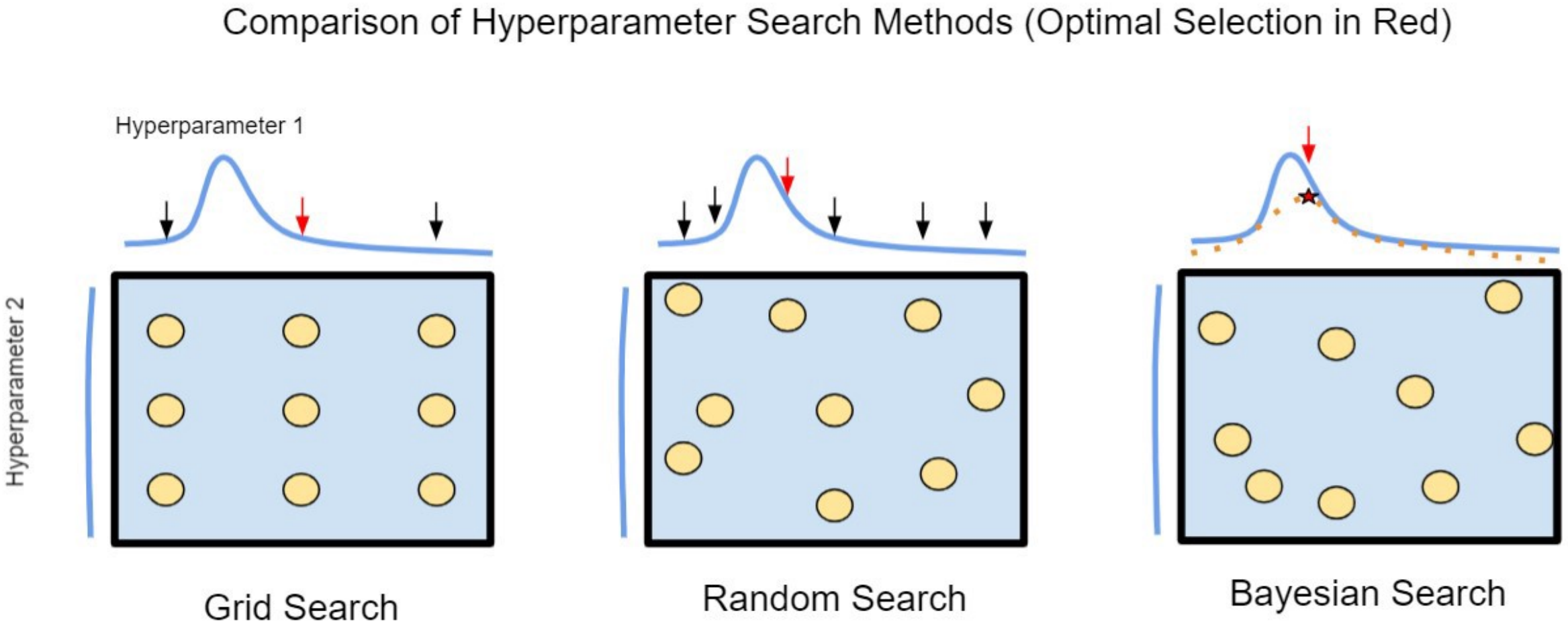

```
model["ml"].get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

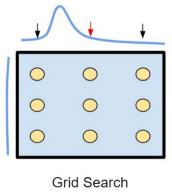
hyperparameter	설명
bootstrap	bootstrap (부분집합) 사용 여부
ccp_alpha	Cost-Complexity Pruning 상한선
criterion	node 분기 손실 함수
max_depth	tree 최대 깊이
max_features	분할에 관여할 최대 feature 수
max_leaf_nodes	최대 leaf node 수
max_samples	bootstratp 적용시 데이터 수
min_inpurity_decrease	node 분할 조건
min_samples_leaf	node내 최소 data 수
min_samples_split	node 분기 가능한 최소 data 수
min_weight_fraction_leaf	~min_samples_leaf. 가중치 포함.
n_estimators	random forest 구성 decision tree 수
random_state	bootstrap, feature sampling 적용
warm_start	기존 학습 모델 재학습



손으로 NGD를 할 바엔 자동으로 시킨다



https://towardsdatascience.com/hyperparameter-tuning-always-tune-your-models-7db7aeaf47e9



Grid Search CV

- 장점 : 관심 hyperparameter 지정 가능
- 단점 : ① 수행 시간
  ② 최적값이 아닐 수 있음.

"ml__n_estimators" :
pipeline 구성요소 중 "ml"의 "n_estimator" 값 범위

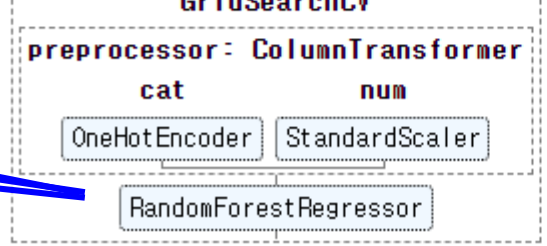
```python
def get_model(method="rf",
              cat_features=["species", "island", "sex"],
              num_features=["bill_length_mm", "bill_depth_mm", "flipper_length_mm"],
              **kwargs):
    # 1-1.categorical feature에 one-hot encoding 적용
    cat_transformer = OneHotEncoder()

    # 1-2.numerical feature는 standard scaler 적용
    num_transformer = StandardScaler()

    # 2. 인자 종류별 전처리 적용
    preprocessor = ColumnTransformer([("cat", cat_transformer, cat_features),
                                      ("num", num_transformer, num_features)])

    # 3. 전처리 후 입력된 방법론 적용
    if method == "rf":
        ml = ("ml", RandomForestRegressor(**kwargs))
    elif method == "lgbm":
        ml = ("ml", LGBMRegressor(**kwargs))
    elif method == "xgb":
        ml = ("ml", XGBRegressor(**kwargs))

    pipeline = Pipeline(steps=[("preprocessor", preprocessor),
                               ml])

    return pipeline
```

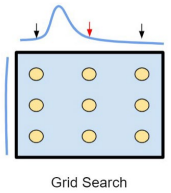
```python
from sklearn.model_selection import GridSearchCV

# parameter grid
params = {"ml__n_estimators": [100, 200, 300],
          "ml__max_depth": [3, 5, 10],
          "ml__min_samples_leaf": [1, 3, 5, 7, 10],
          "ml__max_features": ["auto", "sqrt", "log2"],

# GridSearchCV Pipeline 생성
model = get_model(method="rf", random_state=0)
gscv = GridSearchCV(model, param_grid=params, scoring="r2", refit="r2")

# GridSearchCV 학습 & 학습 시간 측정
time_start = time.time()
gscv.fit(X_train, y_train)
time_end = time.time()
print(f"# running time: {time_end-time_start:.2f} sec.")

gscv
```

경우의 수 : 3 x 3 x 5 x 3 x 5 = 675

5-Fold CV

```
# running time: 213.78 sec.
```





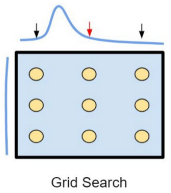
Grid Search CV 실행 결과 : 전체 결과

- 전체 결과 확인 : gscv.cv_results_
- pandas DataFrame 변환 : pd.DataFrame.from_dict(gscv.cv_results_)

```
import pandas as pd

pd.DataFrame.from_dict(gscv.cv_results_)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_ml__max_depth	param_ml__max_features	param_ml__min_samples_leaf	param_ml__n_estimators	param_ml__random_state	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score
0	0.122504	0.001050	0.010759	0.001500	3	auto	1	100	0	{'ml__max_depth': 3, 'ml__max_features': 'auto...	0.830658	0.859541	0.840680	0.902325	0.825754	0.851791	0.027791
1	0.249965	0.008304	0.018684	0.002654	3	auto	1	200	0	{'ml__max_depth': 3, 'ml__max_features': 'auto...	0.830792	0.857789	0.840564	0.900924	0.825572	0.851128	0.027212
2	0.366479	0.010654	0.025440	0.003183	3	auto	1	300	0	{'ml__max_depth': 3, 'ml__max_features': 'auto...	0.830351	0.858731	0.840332	0.902982	0.826142	0.851708	0.027991
3	0.122260	0.001381	0.010921	0.001262	3	auto	3	100	0	{'ml__max_depth': 3, 'ml__max_features': 'auto...	0.830641	0.861055	0.840341	0.899255	0.825404	0.851339	0.026879
4	0.239328	0.005185	0.015683	0.000327	3	auto	3	200	0	{'ml__max_depth': 3, 'ml__max_features': 'auto...	0.830895	0.858942	0.840016	0.898960	0.825328	0.850828	0.026635
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
130	0.236161	0.004565	0.018540	0.001983	10	log2	7	200	0	{'ml__max_depth': 10, 'ml__max_features': 'log...	0.831171	0.857215	0.837155	0.899429	0.825291	0.850052	0.026927
131	0.337821	0.003571	0.022712	0.001172	10	log2	7	300	0	{'ml__max_depth': 10, 'ml__max_features': 'log...	0.830902	0.856753	0.837812	0.901456	0.824952	0.850375	0.027689
132	0.115183	0.001878	0.010150	0.000265	10	log2	10	100	0	{'ml__max_depth': 10, 'ml__max_features': 'log...	0.814565	0.849854	0.833066	0.891985	0.825176	0.842929	0.027098
133	0.224354	0.006636	0.015799	0.000150	10	log2	10	200	0	{'ml__max_depth': 10, 'ml__max_features': 'log...	0.818372	0.850330	0.837381	0.893609	0.824814	0.844901	0.026705
134	0.334805	0.004710	0.028317	0.002455	10	log2	10	300	0	{'ml__max_depth': 10, 'ml__max_features': 'log...	0.815921	0.852626	0.839299	0.893178	0.823931	0.844991	0.027208

135 rows × 18 columns



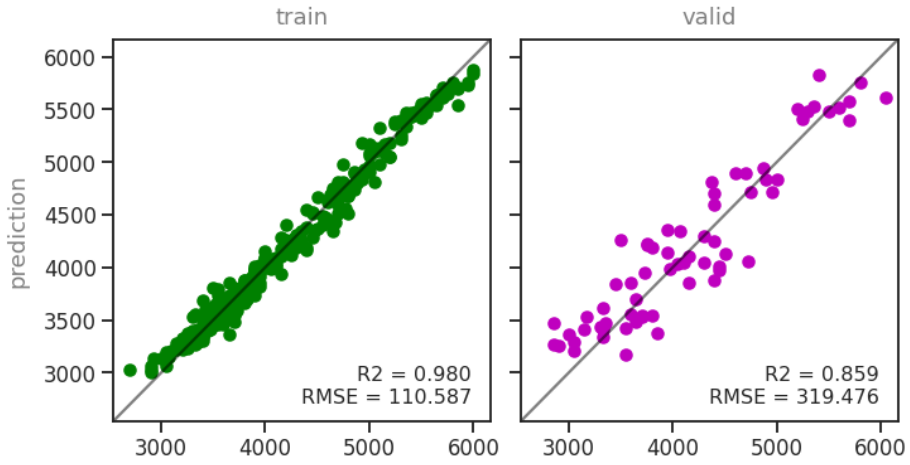
Grid Search CV 실행 결과 : best result

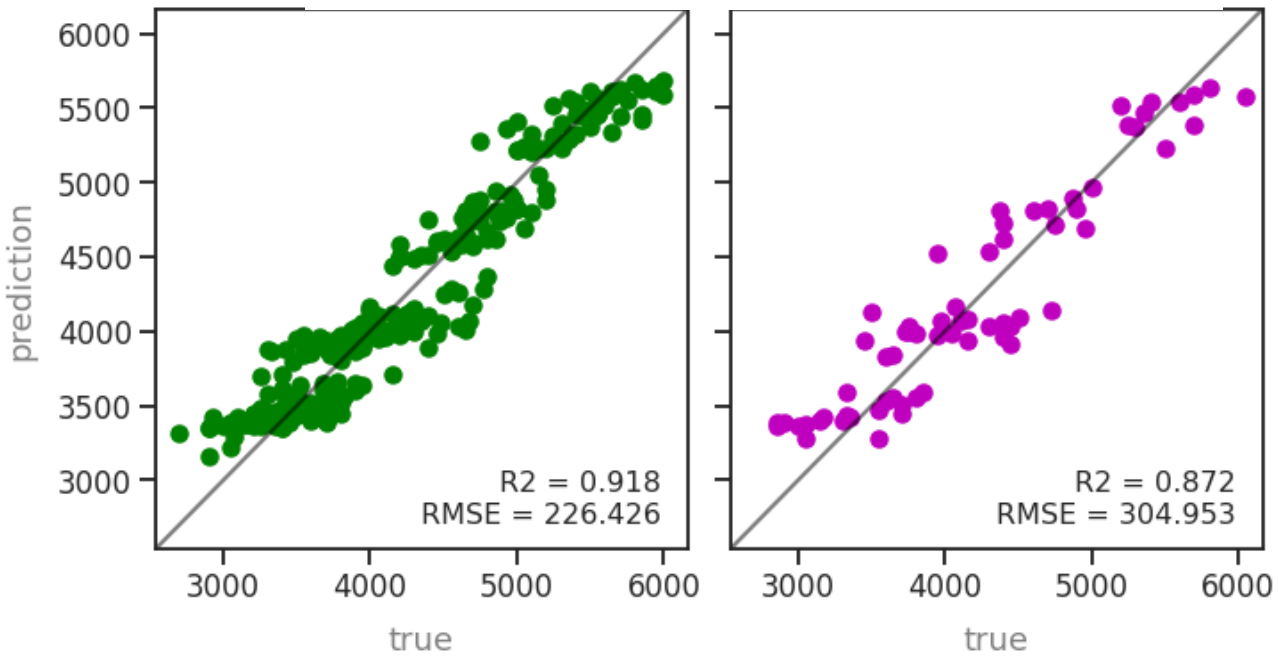
- best hyperparamters : gscv.best_params_
- best score : gscv.best_score_

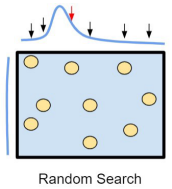
```
1 # best hyperparameters
2 print(gscv.best_params_)
3
4 # best score
5 print(gscv.best_score_)
```

```
{'ml__max_depth': 5, 'ml__max_features': '
0.8599785309332647
```

```
1 y_pred_train = gscv.pr
2 y_pred_val = gscv.pred
3 fig = get_parity(y_tra
```







Random Search CV

- 장점 : 속도 (지정된 수만 실행)
- 단점 : ① 재현성
  ② 최적값이 아닐 수 있음.

max_features : *{"sqrt", "log2", None}, int or float, default=1.0*

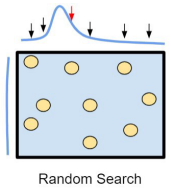
The number of features to consider when looking for the best split:

- If int, then consider `max_features` features at each split.
- If float, then `max_features` is a fraction and `round(max_features * n_features)` features are considered at each split.
- If "auto", then `max_features=n_features`.
- If "sqrt", then `max_features=sqrt(n_features)`.
- If "log2", then `max_features=log2(n_features)`.
- If None or 1.0, then `max_features=n_features`.

```python
from sklearn.model_selection import RandomizedSearchCV
import scipy.stats as stats

# parameter distribution
params = {"ml__n_estimators": stats.randint(100, 300),    # [100, 200, 300]
          "ml__max_depth": stats.randint(3, 10),          # [3, 5, 10],
          "ml__min_samples_leaf": stats.randint(1, 10),   # [1, 3, 5, 7, 10]
          "ml__max_features": stats.uniform(0, 1),        # ["auto", "sqrt", "log2"]
          }

# RandomizedSearchCV Pipeline 생성
model = get_model(method="rf", random_state=0)
rscv = RandomizedSearchCV(model, param_distributions=params, # param_grid
                          n_iter=30,                         # number of HP sampling
                          scoring="r2", refit="r2")

# RandomizedSearchCV 학습 & 학습 시간 측정
time_start = time.time()
rscv.fit(X_train, y_train)
time_end = time.time()
print(f"# running time: {time_end-time_start:.2f} sec.")
```

```
# running time: 37.79 sec.
```

```
# running time: 213.78 sec
```

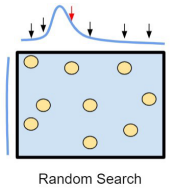


Random Search CV 실행 결과 : 전체 결과

- 전체 결과 확인 : rscv.cv_results_
- pandas DataFrame 변환 : pd.DataFrame.from_dict(rscv.cv_results_)

```
1 pd.DataFrame.from_dict(rscv.cv_results_)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_ml__max_depth	param_ml__max_features	param_ml__min_samples_leaf	param_ml__n_estimators	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score	rank_test_score
0	0.307189	0.063458	0.021505	0.005068	9	0.256242	4	204	{'ml__max_depth': 9, 'ml__max_features': 0.256...	0.833504	0.865727	0.842250	0.910770	0.830120	0.856474	0.029860	8
1	0.320138	0.005921	0.021300	0.002039	5	0.585261	3	258	{'ml__max_depth': 5, 'ml__max_features': 0.585...	0.838652	0.870946	0.838804	0.913568	0.830409	0.858476	0.030849	4
2	0.314544	0.008268	0.021806	0.003025	7	0.42227	7	277	{'ml__max_depth': 7, 'ml__max_features': 0.422...	0.834217	0.863795	0.836356	0.904369	0.828785	0.853504	0.028179	13
3	0.214605	0.002703	0.015335	0.000469	4	0.056072	6	194	{'ml__max_depth': 4, 'ml__max_features': 0.056...	0.807662	0.838728	0.844476	0.893402	0.814411	0.839736	0.030245	28
4	0.333530	0.132293	0.018048	0.002473	3	0.691691	2	224	{'ml__max_depth': 3, 'ml__max_features': 0.691...	0.823169	0.858456	0.848638	0.906500	0.823699	0.852092	0.030514	21
5	0.248621	0.039592	0.017335	0.001446	7	0.69364	7	168	{'ml__max_depth': 7, 'ml__max_features': 0.693...	0.835662	0.868469	0.827964	0.904174	0.830157	0.853285	0.029349	16
6	0.253778	0.038763	0.016980	0.001515	3	0.772953	8	182	{'ml__max_depth': 3, 'ml__max_features': 0.772...	0.833139	0.860696	0.844903	0.901159	0.826060	0.853191	0.026699	17
7	0.230028	0.003359	0.016242	0.000638	9	0.553111	8	196	{'ml__max_depth': 9, 'ml__max_features': 0.553...	0.830836	0.862209	0.832133	0.902147	0.827510	0.850967	0.028482	23
8	0.300948	0.005510	0.019962	0.001196	3	0.403376	8	265	{'ml__max_depth': 3, 'ml__max_features': 0.403...	0.824720	0.859343	0.843059	0.901855	0.821142	0.850024	0.029313	26
9	0.327565	0.006199	0.022951	0.001071	7	0.218073	1	280	{'ml__max_depth': 7, 'ml__max_features': 0.218...	0.850138	0.871245	0.844936	0.909994	0.826240	0.860511	0.028595	1
10	0.303698	0.005819	0.022225	0.002803	6	0.224335	2	264	{'ml__max_depth': 6, 'ml__max_features': 0.224...	0.839171	0.867411	0.842664	0.912205	0.836090	0.859508	0.028584	2
11	0.239474	0.003754	0.017441	0.002846	6	0.739094	6	198	{'ml__max_depth': 6, 'ml__max_features': 0.739...	0.835428	0.868208	0.827199	0.905290	0.829133	0.853051	0.030032	19
12	0.165282	0.001331	0.013521	0.000767	7	0.43554	6	141	{'ml__max_depth': 7, 'ml__max_features': 0.435...	0.840506	0.869734	0.834593	0.901870	0.824405	0.854221	0.028190	11
13	0.208779	0.003235	0.016173	0.002200	5	0.9614	6	169	{'ml__max_depth': 5, 'ml__max_features': 0.961...	0.833167	0.870387	0.828050	0.909768	0.831043	0.854483	0.031660	10
14	0.121128	0.002196	0.010394	0.000603	7	0.472254	6	102	{'ml__max_depth': 7, 'ml__max_features': 0.472...	0.842983	0.862688	0.831374	0.902131	0.827528	0.853341	0.027291	15
15	0.149864	0.005118	0.011858	0.000554	8	0.602771	9	123	{'ml__max_depth': 8, 'ml__max_features': 0.602...	0.833070	0.859720	0.828553	0.902190	0.831513	0.851009	0.027934	22
16	0.292316	0.008175	0.021292	0.001750	7	0.060471	4	254	{'ml__max_depth': 7, 'ml__max_features': 0.060...	0.827549	0.861193	0.841908	0.908057	0.831221	0.853986	0.029457	12
17	0.234585	0.002660	0.018023	0.002756	6	0.688177	4	196	{'ml__max_depth': 6, 'ml__max_features': 0.688...	0.839568	0.870769	0.829077	0.911417	0.833591	0.856884	0.030926	7
18	0.250603	0.003433	0.016534	0.000509	9	0.792562	8	214	{'ml__max_depth': 9, 'ml__max_features': 0.792...	0.828373	0.861466	0.829557	0.902315	0.831047	0.850551	0.028676	25
19	0.125854	0.002308	0.009983	0.000162	8	0.58396	2	101	{'ml__max_depth': 8, 'ml__max_features': 0.583...	0.844125	0.875830	0.837356	0.907488	0.827709	0.858501	0.029333	3
20	0.134393	0.005093	0.011292	0.001151	6	0.486275	2	110	{'ml__max_depth': 6, 'ml__max_features': 0.486...	0.834807	0.872830	0.834261	0.913153	0.833272	0.857665	0.031541	6
21	0.127540	0.002455	0.011255	0.000868	7	0.02223	2	110	{'ml__max_depth': 7, 'ml__max_features': 0.022...	0.835202	0.867628	0.843964	0.914768	0.830168	0.858346	0.031004	5
22	0.309042	0.002828	0.022278	0.001028	5	0.292448	6	269	{'ml__max_depth': 5, 'ml__max_features': 0.292...	0.837287	0.862934	0.840656	0.900959	0.823790	0.853125	0.027023	18
23	0.291293	0.008145	0.019880	0.000437	5	0.096554	9	264	{'ml__max_depth': 5, 'ml__max_features': 0.096...	0.809822	0.830995	0.836614	0.889182	0.808323	0.834987	0.029325	30
24	0.160305	0.004153	0.012726	0.001337	5	0.468903	8	136	{'ml__max_depth': 5, 'ml__max_features': 0.468...	0.837502	0.859606	0.832716	0.894726	0.828407	0.850591	0.024545	24
25	0.188012	0.002035	0.013382	0.000265	9	0.550378	6	159	{'ml__max_depth': 9, 'ml__max_features': 0.550...	0.838512	0.866122	0.827425	0.905199	0.827124	0.852876	0.029767	20
26	0.162337	0.002910	0.012559	0.000516	8	0.298685	5	143	{'ml__max_depth': 8, 'ml__max_features': 0.298...	0.834991	0.865819	0.836991	0.911733	0.827298	0.855366	0.031071	9
27	0.249217	0.004166	0.017396	0.000092	7	0.143846	6	226	{'ml__max_depth': 7, 'ml__max_features': 0.143...	0.811507	0.849755	0.840988	0.898317	0.821234	0.844360	0.030219	27
28	0.240614	0.004104	0.017916	0.002411	8	0.446166	7	210	{'ml__max_depth': 8, 'ml__max_features': 0.446...	0.836769	0.861549	0.838644	0.900751	0.829032	0.853349	0.026064	14
29	0.171626	0.004180	0.013395	0.000645	4	0.169264	7	155	{'ml__max_depth': 4, 'ml__max_features': 0.169...	0.799851	0.843618	0.837393	0.888815	0.813759	0.836687	0.030491	29



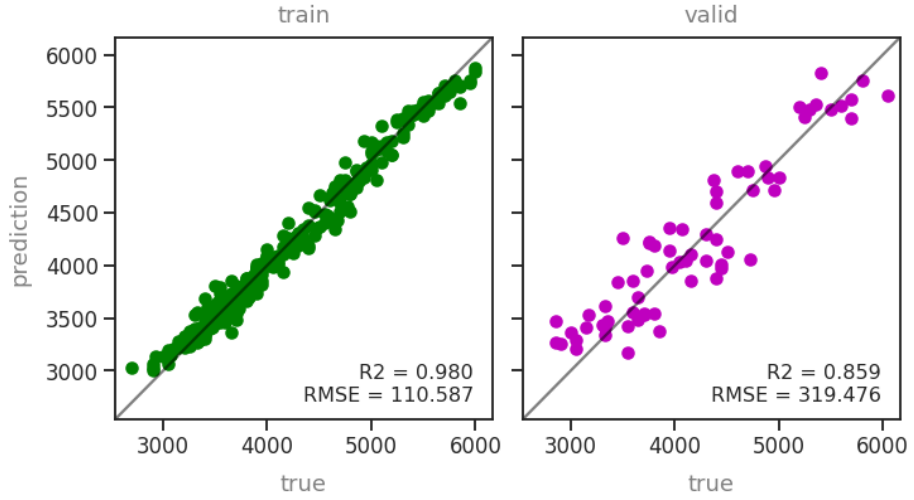
Random Search CV 실행 결과 : best result

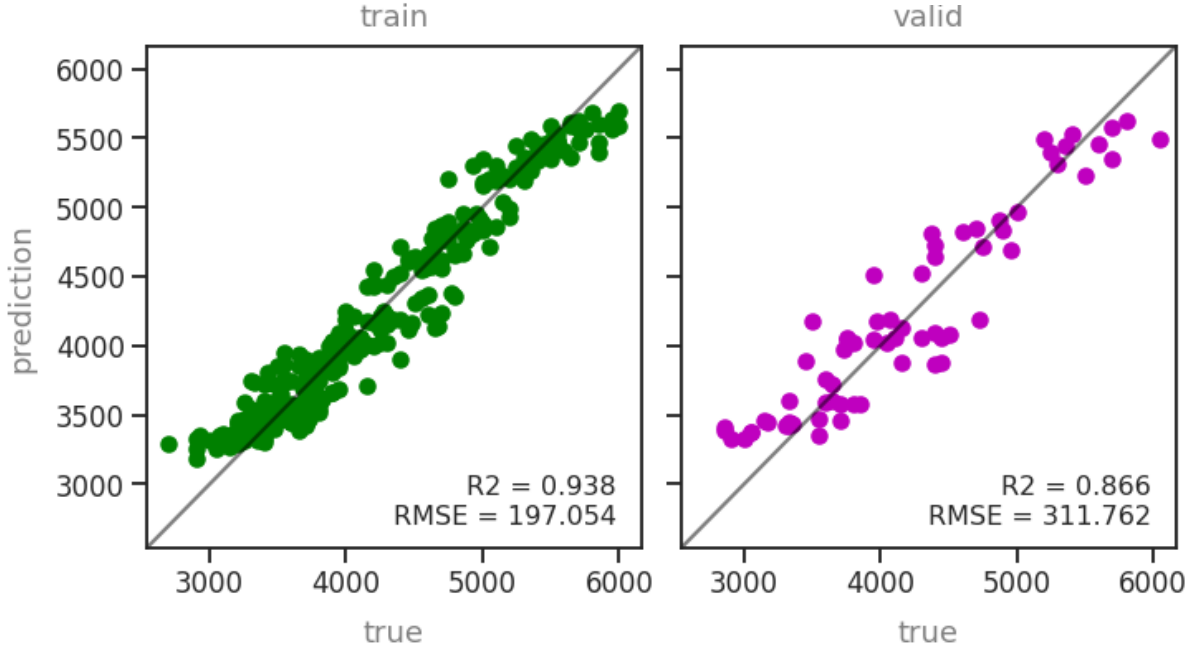
- best hyperparamters : rscv.best_params_
- best score : rscv.best_score_

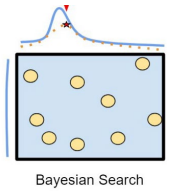
```
1 # best hyperparameters
2 print(rscv.best_params_)
3
4 # best score
5 print(rscv.best_score_)
```

```
{'ml__max_depth': 8, 'ml__max_features': 0.16635539
0.8585201879546958
```

```
1 y_pred_train = rscv.predict(X_train
2 y_pred_val = rscv.predict(X_val)
3 fig = get_parity(y_train, y_val, y_
```

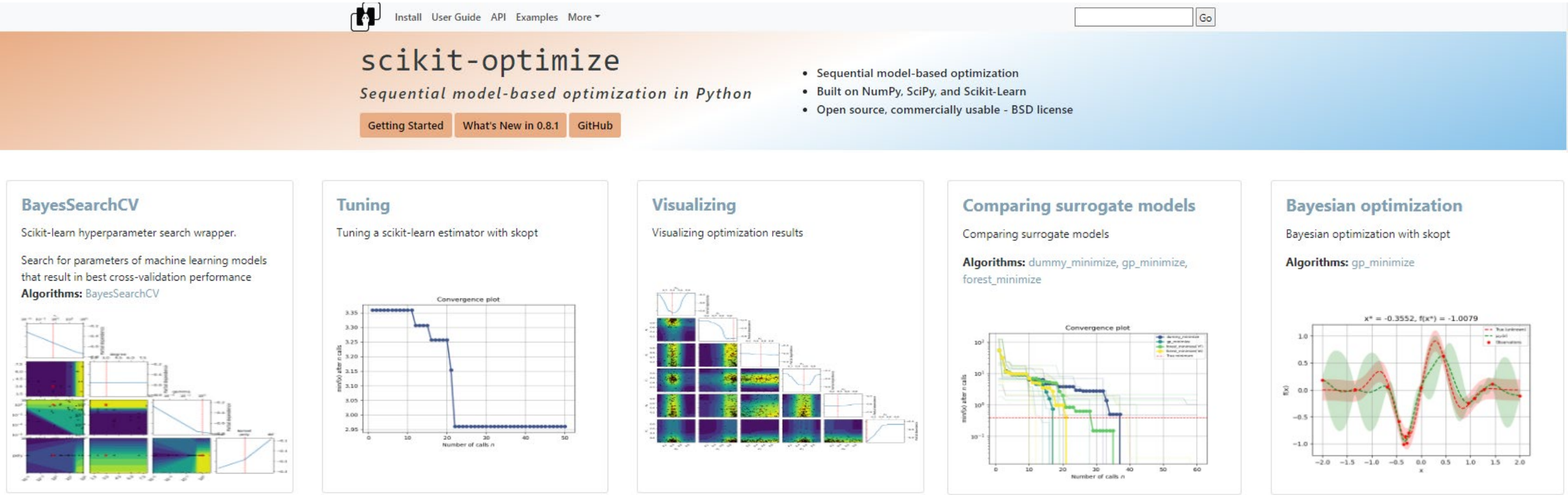




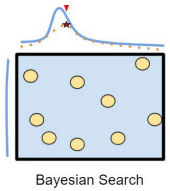


Bayes Search CV

- 장점 : ① 속도 (지정된 수만 실행)
  ② 최적값일 확률이 매우 높음
- 단점 : scikit-learn에 기능이 없음. but, scikit-optimize 등 외부 라이브러리 사용 가능



https://scikit-optimize.github.io/



Bayes Search CV

- scikit-learn 문법 거의 동일

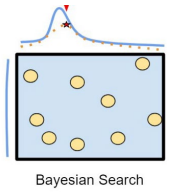
cv : int, cross-validation generator or an iterable, optional

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

- None, to use the default 3-fold cross validation,
- integer, to specify the number of folds in a `(Stratified)KFold`,
- An object to be used as a cross-validation generator.
- An iterable yielding train, test splits.

```python
from skopt import BayesSearchCV
from skopt.space import Categorical, Integer

# parameter distribution
params = {"ml__n_estimators": Integer(100, 300),        # [100, 200, 300]
          "ml__max_depth": Integer(3, 10),              # [3, 5, 10],
          "ml__min_samples_leaf": Integer(1, 10),       # [1, 3, 5, 7, 10]
          "ml__max_features": Categorical(["auto", "sqrt", "log2"]) # ["auto", "sqrt", "log2"]
          }

# BayesSearchCV Pipeline 생성
model = get_model(method="rf", random_state=0)
bscv = BayesSearchCV(model, search_spaces=params,  # param_grid
                     n_iter=30,                    # number of HP sampling
                     cv=5,                         # number of folds in CV
                     scoring="r2", refit="r2")

# BayesSearchCV 학습 & 학습 시간 측정
time_start = time.time()
bscv.fit(X_train, y_train)
time_end = time.time()
print(f"# running time: {time_end-time_start:.2f} sec.")
```

```
# running time: 69.83 sec.
```

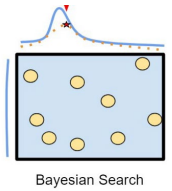


Bayes Search CV 실행 결과 : 전체 결과

- 전체 결과 확인 : bscv.cv_results_
- pandas DataFrame 변환 : pd.DataFrame.from_dict(bscv.cv_results_)

```
1 pd.DataFrame.from_dict(bscv.cv_results_)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_ml__max_depth	param_ml__max_features	param_ml__min_samples_leaf	param_ml__n_estimators	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score	rank_test_score
0	0.147216	0.002332	0.014346	0.002404	10	log2	7	125	{'ml__max_depth': 10, 'ml__max_features': 'log...	0.830924	0.857276	0.834746	0.898549	0.823933	0.849086	0.027136	24
1	0.122353	0.004004	0.010173	0.000358	4	sqrt	6	103	{'ml__max_depth': 4, 'ml__max_features': 'sqrt...	0.830670	0.855565	0.841573	0.902862	0.822192	0.850572	0.028431	20
2	0.193636	0.004219	0.015818	0.002372	7	auto	5	152	{'ml__max_depth': 7, 'ml__max_features': 'auto...	0.834966	0.868277	0.823551	0.907398	0.833871	0.853613	0.030819	14
3	0.175643	0.005661	0.013188	0.000412	8	log2	6	148	{'ml__max_depth': 8, 'ml__max_features': 'log2...	0.836067	0.859880	0.834401	0.900337	0.822948	0.850727	0.027561	19
4	0.277494	0.003321	0.017802	0.000169	4	auto	6	229	{'ml__max_depth': 4, 'ml__max_features': 'auto...	0.829879	0.864682	0.830020	0.902854	0.829473	0.851381	0.029069	17
5	0.335898	0.006525	0.023603	0.003367	5	log2	9	299	{'ml__max_depth': 5, 'ml__max_features': 'log2...	0.822023	0.856553	0.837522	0.896611	0.825073	0.847557	0.027370	28
6	0.155755	0.002557	0.012627	0.000312	7	log2	2	130	{'ml__max_depth': 7, 'ml__max_features': 'log2...	0.842187	0.873136	0.836214	0.910718	0.825387	0.857528	0.030969	8
7	0.207231	0.005103	0.016148	0.002267	4	sqrt	4	184	{'ml__max_depth': 4, 'ml__max_features': 'sqrt...	0.833185	0.858542	0.843357	0.905688	0.823584	0.852871	0.028836	15
8	0.304312	0.003525	0.020650	0.002627	8	auto	9	250	{'ml__max_depth': 8, 'ml__max_features': 'auto...	0.825876	0.856576	0.826197	0.898378	0.833686	0.848142	0.027500	27
9	0.157719	0.017484	0.012125	0.000618	10	log2	7	129	{'ml__max_depth': 10, 'ml__max_features': 'log...	0.830125	0.856950	0.834885	0.897632	0.823903	0.848699	0.026882	26
10	0.342092	0.015894	0.023744	0.004180	3	auto	1	279	{'ml__max_depth': 3, 'ml__max_features': 'auto...	0.830285	0.858213	0.840594	0.903031	0.826242	0.851673	0.027955	16
11	0.154990	0.010198	0.011037	0.000737	9	auto	1	101	{'ml__max_depth': 9, 'ml__max_features': 'auto...	0.839847	0.870949	0.833053	0.898425	0.813819	0.851219	0.029919	18
12	0.391444	0.014485	0.023446	0.001295	9	auto	3	298	{'ml__max_depth': 9, 'ml__max_features': 'auto...	0.839663	0.871473	0.822528	0.909420	0.834564	0.855530	0.031429	11
13	0.349184	0.013742	0.023069	0.001551	4	sqrt	2	300	{'ml__max_depth': 4, 'ml__max_features': 'sqrt...	0.832769	0.864695	0.845209	0.914924	0.832975	0.858115	0.030697	7
14	0.344241	0.016994	0.022909	0.000771	3	sqrt	2	300	{'ml__max_depth': 3, 'ml__max_features': 'sqrt...	0.812450	0.854916	0.846724	0.904409	0.828247	0.849349	0.031229	23
15	0.252148	0.013739	0.019220	0.002338	7	sqrt	7	217	{'ml__max_depth': 7, 'ml__max_features': 'sqrt...	0.831010	0.856560	0.837292	0.900372	0.825748	0.850196	0.027168	21
16	0.124607	0.009696	0.010211	0.000284	4	log2	1	100	{'ml__max_depth': 4, 'ml__max_features': 'log2...	0.828883	0.864200	0.848095	0.912872	0.831448	0.857099	0.030647	10
17	0.122769	0.011219	0.011680	0.002665	9	sqrt	6	101	{'ml__max_depth': 9, 'ml__max_features': 'sqrt...	0.832480	0.860472	0.832480	0.900209	0.821767	0.849482	0.028426	22
18	0.165771	0.005913	0.014900	0.002470	6	log2	1	134	{'ml__max_depth': 6, 'ml__max_features': 'log2...	0.843870	0.870215	0.843720	0.912950	0.830421	0.860235	0.029353	3
19	0.349028	0.015405	0.024424	0.002982	5	log2	1	300	{'ml__max_depth': 5, 'ml__max_features': 'log2...	0.836828	0.871115	0.843785	0.916418	0.831747	0.859979	0.031320	5
20	0.154939	0.003669	0.011544	0.000247	3	auto	10	128	{'ml__max_depth': 3, 'ml__max_features': 'auto...	0.826284	0.855022	0.840602	0.893822	0.828228	0.848792	0.024751	25
21	0.398946	0.019645	0.023677	0.000813	6	auto	3	300	{'ml__max_depth': 6, 'ml__max_features': 'auto...	0.837825	0.872106	0.824946	0.907935	0.833909	0.855344	0.030781	12
22	0.125517	0.012127	0.010489	0.000240	5	sqrt	2	101	{'ml__max_depth': 5, 'ml__max_features': 'sqrt...	0.836817	0.872015	0.838169	0.914712	0.832344	0.858811	0.031336	6
23	0.258239	0.010977	0.017622	0.002108	5	auto	1	196	{'ml__max_depth': 5, 'ml__max_features': 'auto...	0.838560	0.865382	0.839471	0.906310	0.826786	0.855302	0.028452	13
24	0.359444	0.005353	0.023960	0.000946	10	sqrt	1	291	{'ml__max_depth': 10, 'ml__max_features': 'sqr...	0.847707	0.870910	0.840634	0.906669	0.821705	0.857525	0.029186	9
25	0.324147	0.010498	0.021992	0.000908	6	sqrt	1	273	{'ml__max_depth': 6, 'ml__max_features': 'sqrt...	0.843287	0.870216	0.845997	0.914857	0.833441	0.861560	0.029265	2
26	0.358810	0.011419	0.023484	0.000500	7	sqrt	1	300	{'ml__max_depth': 7, 'ml__max_features': 'sqrt...	0.845701	0.871962	0.845949	0.910268	0.827035	0.860183	0.028839	4
27	0.355870	0.012360	0.024082	0.002015	6	sqrt	1	300	{'ml__max_depth': 6, 'ml__max_features': 'sqrt...	0.844546	0.869857	0.846025	0.914646	0.833131	0.861641	0.029072	1
28	0.323234	0.008427	0.023157	0.002539	10	sqrt	10	284	{'ml__max_depth': 10, 'ml__max_features': 'sqr...	0.816903	0.852806	0.839192	0.893159	0.824803	0.845373	0.026872	29
29	0.315435	0.010858	0.022716	0.002718	3	sqrt	10	283	{'ml__max_depth': 3, 'ml__max_features': 'sqrt...	0.811221	0.846796	0.843139	0.890266	0.820585	0.842401	0.027419	30



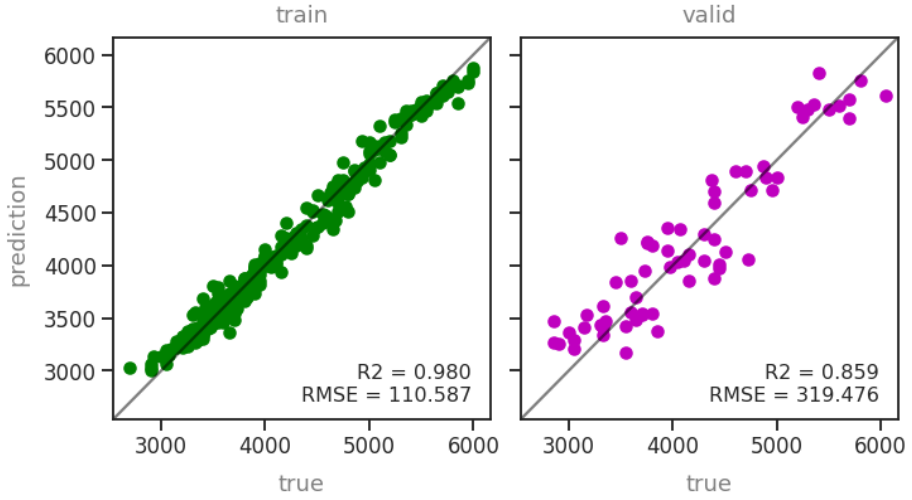
Bayes Search CV 실행 결과 : best result

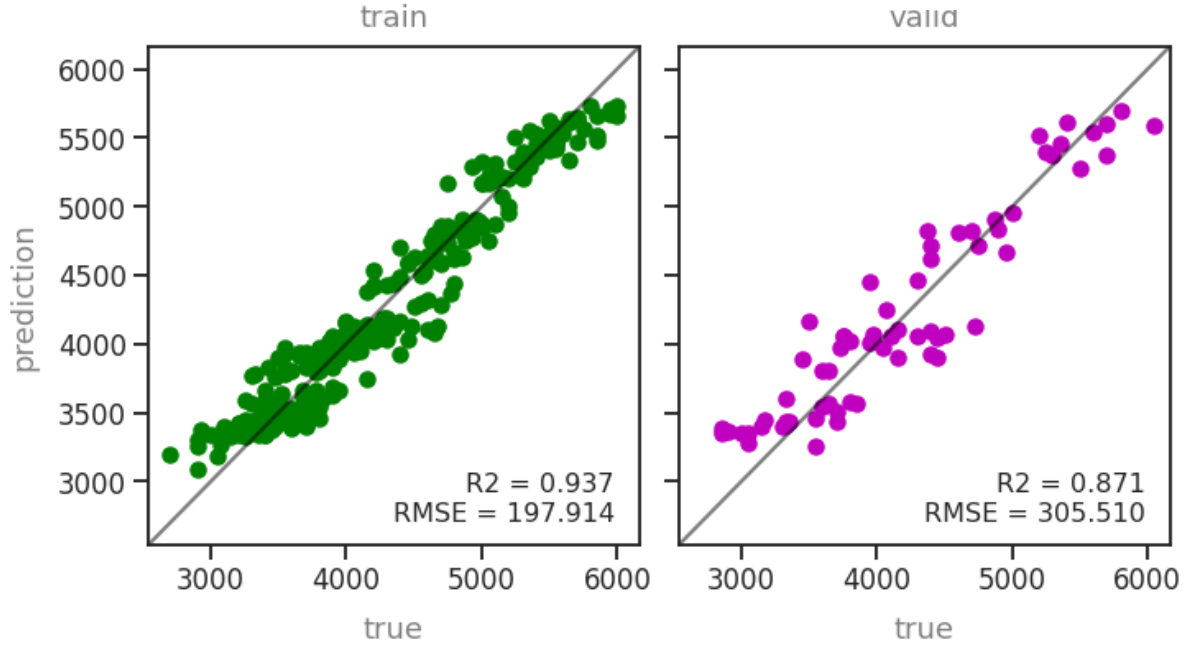
- best hyperparamters : rscv.best_params_
- best score : rscv.best_score_

```
1 # best hyperparameters
2 print(bscv.best_params_)
3
4 # best score
5 print(bscv.best_score_)
```

```
OrderedDict([('ml__max_depth', 6), ('ml__max_feature
0.8616408476192638
```

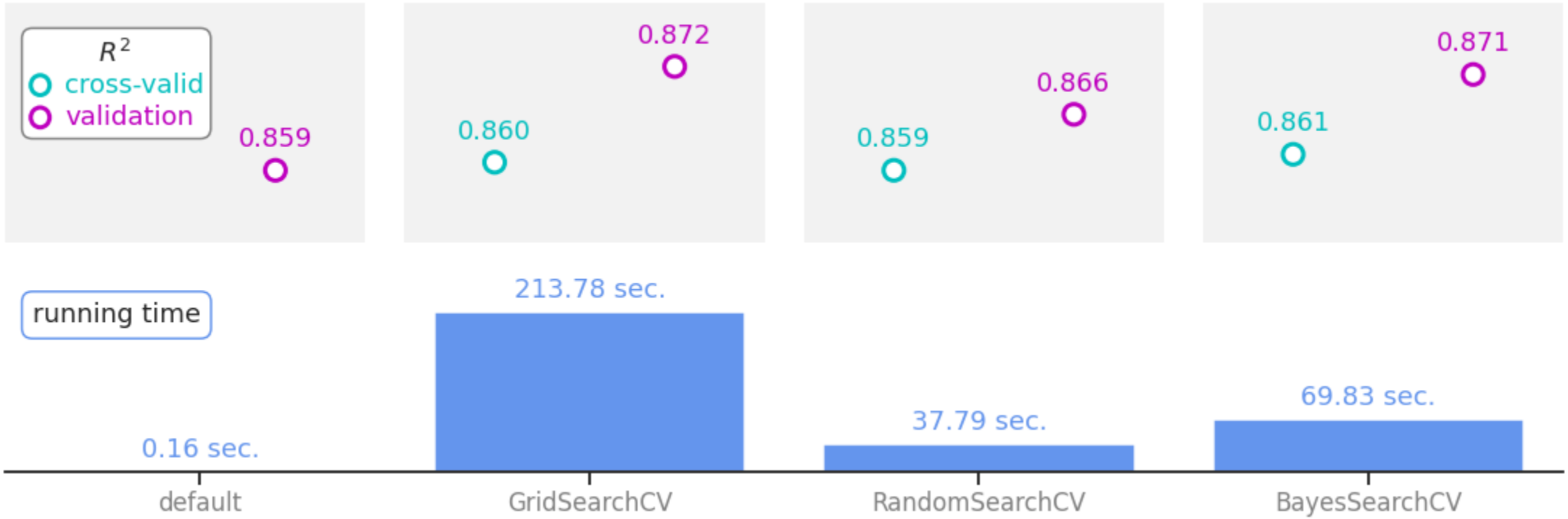
```
1 y_pred_train = bscv.predict(X_train
2 y_pred_val = bscv.predict(X_val)
3 fig = get_parity(y_train, y_val, y_
```





Summary: Grid, Random, Bayes

- 모든 경우에서 hyperparameter tuning 후 성능 향상
- Bayesian이 가장 효과적이면서도 정확함
- default : 문제가 어렵지 않고 데이터가 양호해서 성능이 우수 - “학습용 데이터셋”



Homework

- 다른 데이터셋에 적용 : https://bit.ly/3zYNZmB
- feature space 탐색 시각화