Exercise 6

Task 1

Load 'HalvingRandomSearchCV_random_forest.py' from folder and test it. Improve the score accuracy by changing the parameter ranges in optimization procedure. Include of the score values in your analysis (+zipped code)

In [1]:
```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.experimental import enable_halving_search_cv  # noqa
from sklearn.model_selection import HalvingRandomSearchCV
from scipy.stats import randint
import numpy as np


X, y = load_iris(return_X_y=True)
clf = RandomForestClassifier(random_state=0)
np.random.seed(0)

param_distributions = {
    "max_depth": [1, None],
    "min_samples_split": randint(2, 11),
    "min_samples_leaf": randint(1, 11),
    "bootstrap": [True],
    "criterion": ["gini", "entropy"],
    "max_features": randint(1, 5),
    }


search = HalvingRandomSearchCV(
    clf,
    param_distributions,
    resource='n_estimators',
    max_resources=10,
    random_state=0
    ).fit(X, y)

print("Updated best parameters found:")
print(search.best_params_)

print(search.score(X, y))
```

```
Updated best parameters found:
{'bootstrap': True, 'criterion': 'entropy', 'max_depth': None, 'max_features': 2, 'm
in_samples_leaf': 1, 'min_samples_split': 6, 'n_estimators': 9}
0.98
```

New parameters were added to param_distribution.

Bootstrap is method for sampling data points, default value True. False value would mean that each tree is trained on whole dataset. Using False value, score raises up to 0.993 which

indicates almost perfect score, however the setting may lead to overfitting. Thus, it is not used.

Since original score already is high (0.973), there are little to improve. New set of parameters raises the score to 0.98. The most impact comes from "max_features", since without it the score remains the same. Max_features determines amount of features to evaluate best split. It increases diversity and randomness of the trees. Also it can take values like "auto" or "log" but these do not work for this task due to nan values.

Original results:

{'max_depth': 3, 'min_samples_split': 3, 'n_estimators': 9}

0.9733333333333334

Task 2

Load 'BayesSearchCV_SVC.py' from folder and test it. Change the classifier to AdaBoost classifier:

https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

and include/optimize all the three parameters in Bayes optimization mechanism (exclude the'random_state'). Can you improve the Adaboost result compared to optimized SVC classifier?

In [2]:
```
from skopt import BayesSearchCV
# parameter ranges are specified by one of below
from skopt.space import Real, Categorical, Integer

from sklearn.datasets import load_iris
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

data=load_iris()
X=data.data
y=data.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.75,
                                                    random_state=0)

# log-uniform: understand as search over p = exp(x) by varying x
opt_svc = BayesSearchCV(
    SVC(),
    {
        'C': Real(1e-6, 1e+6, prior='log-uniform'),
        'gamma': Real(1e-6, 1e+1, prior='log-uniform'),
        'degree': Integer(1,8),
        'kernel': Categorical(['linear', 'poly', 'rbf']),
    },
```

```
    n_iter=32,
    random_state=0,
    n_jobs=-1,
)

# executes bayesian optimization
_ = opt_svc.fit(X_train, y_train)

# model can be saved, used for predictions or scoring
print("SVC score: ", opt_svc.score(X_test, y_test))

# parameter setting that gave the best results
print("SVC best params: ", opt_svc.best_params_)
```

```
SVC score:  0.9736842105263158
SVC best params:  OrderedDict({'C': 1.3361910455737007, 'degree': 5, 'gamma': 0.1128
3439533114079, 'kernel': 'linear'})
```

AdaBoostClassifier

In [3]:
```
from sklearn.ensemble import AdaBoostClassifier
# AdamBoost

opt_ada = BayesSearchCV(
    AdaBoostClassifier(),
    {
        'n_estimators': Integer(10, 200),
        'learning_rate': Real(1e-6, 1e+1, prior='log-uniform'),
        'algorithm': Categorical(['SAMME']),
    },
    n_iter=32,
    random_state=0,
    n_jobs=-1
)

# executes bayesian optimization
_ = opt_ada.fit(X_train, y_train)

# model can be saved, used for predictions or scoring
print("AdaBoost score: ", opt_ada.score(X_test, y_test))

# parameter setting that gave the best results
print("AdaBoost best params: ", opt_ada.best_params_)
```

```
AdaBoost score:  0.9736842105263158
AdaBoost best params:  OrderedDict({'algorithm': 'SAMME', 'learning_rate': 0.3039932
179841151, 'n_estimators': 99})
```

AdaBoostClassifier best result is the same as optimized SVC classifier (0.9736). Reducing the gap between parameter values makes the model perform worse and increasing does not have effect. Using preinitialized estimator for AdaBoost does not have effect on result either. Another algorithm choice, 'SAMME.R', does not give better results. It is removed from the params since it will become deprecated and ignore warnings method did not work.

Task 3

Look at the example in:

https://github.com/optuna/optuna-examples/blob/main/pytorch/pytorch_simple.py

Modify the code to test minimum of four different activation functions (for example nn.ReLU(), nn.Tanh(), nn.celu(), nn.Sigmoid()).

The code is in pytorch_simple.py. The activation function is replaced with

```
activation_fn = trial.suggest_categorical("activation", ["ReLU",
"Tanh", "CELU", "Sigmoid", "LeakyReLU"])
if activation_fn == "ReLU":
    layers.append(nn.ReLU())
elif activation_fn == "Tanh":
    layers.append(nn.Tanh())
elif activation_fn == "CELU":
    layers.append(nn.CELU())
elif activation_fn == "Sigmoid":
    layers.append(nn.Sigmoid())
elif activation_fn == "LeakyReLU":
    layers.append(nn.LeakyReLU())
```

which is implemented based on optimizer choice. Trial suggests category and based on that layer is selected. Table shows Best Trial Value of 0.85859375 for ReLU activation function. It is not possible to determine best activation function, since Tanh appears twice on the list with decent results. Additional test runs (results not saved) also showed that activation function for Best Trial Value varies a lot and is depended on many other tunable parameters.

However, optimizer Adam is constantly in Best Trial and it performs better than 'RMSprop' or 'SGD' for this test.

Table of tests results using changes, 5 test runs included.

Run	Number of Finished Trials	Number of Pruned Trials	Number of Complete Trials	Best Trial Value	n_layers	n_units_l0	Activation	dropout_
Run 1	100	68	32	**0.85859375**	1	104	ReLU	0.2986
Run 2	100	75	25	0.8453125	1	82	Tanh	0.4977
Run 3	100	43	57	0.85	1	81	Tanh	0.2110
Run 4	100	55	45	0.84296875	2	68	CELU	0.2749

Run	Number of Finished Trials	Number of Pruned Trials	Number of Complete Trials	Best Trial Value	n_layers	n_units_l0	Activation	dropout_
Run 5	100	67	33	0.84921875	1	125	LeakyReLU	0.3734

Task 4

Look at the example in:

https://github.com/optuna/optuna-examples/blob/main/pytorch/pytorch_simple.py

https://github.com/optuna/optunaexamples/blob/main/pytorch/pytorch_lightning_simple.py

Modify the 'Exercise_train_test.py' (Look at the "Practical example code folder) architecture using optuna mechanism. Include the number of layers, layer sizes and selected activation functions (minimum of 10) in the optimization process. Show the test results (figure) and MSE for the original and optimized codes in your analysis (+zipped code).

Full solution to the task is in pytorch_task4.ipynb and original tutorial is in file pytorch_simple.py

Copy of the analysis:

Original model results, obtained by running train_test.py :

- Correct: 90.51724137931035 %
- MSE: 8006.621066488069

Test is run for the Exercise_test_data and trained with Exercise_Train_data (same as in Parctical example code folder).

Here is results of 4 different runs on trials. For 3/4 runs, the timeout (10min) stopped the run. The results for each run is better than for the original model (90.5%). The Accuracy and MSE is computed in the same way for the original and the test set.

Results indicate best accuracy for the most complex model with 4 layers (86, 329, 377, 665 units). However, the MSE values for more complex models is high, which indicate that results for models with simpler architecture are more reliable. High MSE value may suggest overfitting. Based on these result, the best arhitecture would be from Run 2 or Run 4. Thus, test should be run next with layer variability set to 1-2, this would also increase the Number of Finished Trials since more computing is required for deeper architectures. Also test could be done using MSE report to trial, which would increase focus on MSE (if we want to minimize it).

The dropout and logsoftmax layers are commented out but could be used to optimize model even further. This test only includes the architecture that is shown in the example code and optimization process only does search for number of layers, sizes, activation functions, optimizer and learning rate.

Metric	Run 1	Run 2	Run 3	Run 4
Number of Finished Trials	55	100	63	73
Number of Pruned Trials	36	88	45	55
Number of Complete Trials	19	12	18	18
Best Accuracy (Value)	**92.24%**	91.38%	91.38%	91.81%
Model Structure	4 layers (86, 329, 377, 665 units)	1 layer (432 units)	4 layers (554, 615, 530, 124 units)	2 layers (683, 104 units)
Activation Function	LeakyReLU	Tanh	LeakyReLU	LeakyReLU
Optimizer	Adam	Adam	RMSprop	SGD
Learning Rate	0.001604	0.00199	0.000376	0.02833
Final MSE	4165.37	**361.75**	5985.22	406.46

Second test was made based on previous results. Now using less layers, now 1-2 layers can be selected and max layer size was increased from 700 -> 900. Results are in table below

Metric	Run 1	Run 2	Run 3	Run 4
Number of Finished Trials	43	83	100	67
Number of Pruned Trials	24	65	84	47
Number of Complete Trials	19	18	16	20
Best Accuracy (Value)	90.95%	90.95%	**91.38%**	90.95%
Model Structure	2 layers (347, 120 units)	1 layer (389 units)	2 layers (796, 343 units)	1 layer (507 units)
Activation Function	ReLU	ReLU	Sigmoid	Sigmoid
Optimizer	Adam	Adam	RMSprop	Adam
Learning Rate	0.001237	0.0000722	0.002127	0.001916

Metric	Run 1	Run 2	Run 3	Run 4
Final MSE	2218.88	**258.61**	1868.17	2116.41

Results show best accuracy trend for 90.95 %, which was baseline for 3 runs. Run 3 had best accuracy 91.38 % using 2 layers and Sigmoid activation function. However, the lowest MSE was for Run 2, which had 1 layer. The same trend continues as more complex architecture leads to increased MSE. One more thing to consider is that activation functions are completely different when comparing to first results.