Comparing Randomized Search and Grid Search for Hyperparameter Estimation in Scikit Learn

Hyperparameters are the parameters that determine the behavior and performance of a machine-learning model. These parameters are not learned during training but are instead set prior to training. The process of finding the optimal values for these hyperparameters is known as hyperparameter optimization, and it is an important step in the development of any Machine Learning model.

There are many different methods for performing hyperparameter optimization, but two of the most commonly used methods are grid search and randomized search. In this blog post, we will compare these two methods and provide examples of how to implement them using the Scikit Learn library in Python.

Grid Search Hyperparameter Estimation

Grid search is a method for hyperparameter optimization that involves specifying a list of values for each hyperparameter that you want to optimize, and then training a model for each combination of these values. For example, if you want to optimize two hyperparameters, alpha and beta, with grid search, you would specify a list of values for alpha and a separate list of values for the beta. The grid search algorithm would then train a model using every combination of these values and evaluate the performance of each model. The optimal values for the hyperparameters are then chosen based on the performance of the models.

Additionally, it is recommended to use cross-validation when performing hyperparameter optimization with either grid search or randomized search. Cross-validation is a technique that involves splitting the training data into multiple sets and training the model multiple times, each time using a different subset of the data as the validation set. This can provide a more accurate estimate of the model's performance and help to avoid overfitting.

Here is an example of how to implement grid search in Scikit Learn:

Python3

```
from  sklearn.model_selection  import  GridSearchCV

param_grid  =  {

  'alpha'``: [``0.01``,  0.1``,  1.0``,  10.0``],

  'beta'``: [``0.01``,  0.1``,  1.0``,  10.0``]
```

```
}

model = SomeModel()

grid_search = GridSearchCV(model, param_grid)

grid_search.fit(X, y)

print``(grid_search.best_params_)
```

In this example, we define a dictionary called param_grid that specifies the possible values for the hyperparameters alpha and beta. We then use the GridSearchCV class from sklearn.model_selection module to perform grid search using these values. The fit method is used to train the model with the different combinations of hyperparameters, and the best_params_ attribute is used to access the optimal values for the hyperparameters.

One disadvantage of grid search is that it can be computationally expensive, especially when optimizing many hyperparameters or when the list of possible values for each hyperparameter is large. This is because grid search trains a separate model for every combination of hyperparameter values, which can quickly become infeasible as the number of combinations grows.

Randomized Search for Hyperparameter Estimation

Randomized search is another method for hyperparameter optimization that can be more efficient than grid search in some cases. With randomized search, instead of specifying a list of values for each hyperparameter, you specify a distribution for each hyperparameter. The randomized search algorithm will then sample values for each hyperparameter from its corresponding distribution and train a model using the sampled values. This process is repeated a specified number of times, and the optimal values for the hyperparameters are chosen based on the performance of the models.

Here is an example of how to implement randomized search in Scikit Learn:

Python3

```
from sklearn.model_selection import RandomizedSearchCV

from scipy.stats import uniform

param_distributions = {

  'alpha'``: uniform(``0.01``, 10.0``),

  'beta'``: uniform(``0.01``, 10.0``)
```

```
}

model = SomeModel()

random_search = RandomizedSearchCV(model,

                    param_distributions)

random_search.fit(X, y)

print``(random_search.best_params_)
```

In this example, we define a dictionary called param_distributions that specifies the distributions for the hyperparameters alpha and beta. We use the uniform distribution from the scipy.stats module, which specifies a range of values for each hyperparameter. We then use the RandomizedSearchCV class from the sklearn.model_selection module to perform a randomized search using these distributions. The fit method is used to train the model with the different combinations of hyperparameters, and the best_params_ attribute is used to access the optimal values for the hyperparameters.

One advantage of randomized search is that it can be more efficient than grid search in some cases since it does not train a separate model for every combination of hyperparameter values. Instead, it trains a specified number of models using random samples from the hyperparameter distributions. This can be more efficient when optimizing many hyperparameters or when the distributions for the hyperparameters have a large range of values.

It is important to note that both grid search and randomized search are only effective if the hyperparameter space is properly defined. In other words, it is crucial to specify a reasonable range of values for the hyperparameters that you want to optimize. If the range of values is too small, the optimal values may not be found, while if the range of values is too large, the search process may be inefficient and take a long time to complete.

Comparing Grid Search and Randomized Search

In this example, we are using a dataset consisting of 200 samples with 10 features each, and a binary target variable. We are fitting a Random Forest classifier with a variety of hyperparameters: the number of trees in the forest (n_estimators), the maximum depth of each tree (max_depth), the minimum number of samples required to split an internal node (min_samples_split), and whether or not to use bootstrapped samples when building the trees (bootstrap).

To compare RandomizedSearchCV and GridSearchCV, we define a search space for the hyperparameters using a dictionary, and then pass this dictionary to both RandomizedSearchCV and GridSearchCV along with the model and the number of cross-validation folds (cv). For

RandomizedSearchCV, we also specify the number of iterations (n_iter) to sample from the search space.

Python3

```
import  numpy as np

from  sklearn.ensemble  import  RandomForestClassifier

from  sklearn.model_selection  import  RandomizedSearchCV,\

GridSearchCV

from  scipy.stats  import  uniform

X  =  np.random.rand(``200``,  10``)

y  =  np.random.randint(``2``, size``=``200``)
```

RandomizedSearchCV works by sampling random combinations of hyperparameters from the search space and evaluating them using cross-validation. It continues sampling and evaluating until it has tried the specified number of iterations. This means that it does not explore the entire search space, but rather a randomly selected subset of it. This can be more efficient than GridSearchCV, which explores the entire search space, but it may also be less likely to find the true optimal set of hyperparameters.

Python3

```
model  =  RandomForestClassifier()

param_grid  =  {

  'n_estimators'``: [``10``,  50``,  100``,  200``],

  'max_depth'``: [``None``,  5``,  10``,  15``],

  'min_samples_split'``: np.linspace(``0.1``,  1``,  11``),

  'bootstrap'``: [``True``,  False``]

}

random_search  =  RandomizedSearchCV(

 model,
```

```
  param_grid,

  cv``=``5``,

  n_iter``=``10``,

  random_state``=``42``)

random_search.fit(X, y)
```

On the other hand, GridSearchCV exhaustively searches the entire search space by trying every possible combination of hyperparameters. This can be very computationally expensive, especially if the search space is large or if the model takes a long time to fit. However, it is guaranteed to find the optimal set of hyperparameters if it is able to complete the search.

Python3

```
grid_search  =  GridSearchCV(model, param_grid, cv``=``5``)

grid_search.fit(X, y)

print``(f``"Best hyperparameters found by RandomizedSearchCV: {random_search.best_params_}"``)

print``(f``"Best hyperparameters found by GridSearchCV: {grid_search.best_params_}"``)
```

Output:

```
Best hyperparameters found by RandomizedSearchCV:
 {'n_estimators': 100,
  'min_samples_split': 0.5499999999999999,
  'max_depth': 10,
  'bootstrap': True}

Best hyperparameters found by GridSearchCV:
 {'bootstrap': False,
  'max_depth': None,
  'min_samples_split': 1.0,
  'n_estimators': 10}
```

Why RandomizedSearchCV is better than GridSearchCV?

One advantage of RandomizedSearchCV over GridSearchCV is that RandomizedSearchCV can be more efficient if the search space is large since it only samples a subset of the possible combinations rather than evaluating them all. This can be especially useful if the model is computationally

expensive to fit, or if the hyperparameters have continuous values rather than discrete ones. In these cases, it may not be feasible to explore the entire search space using GridSearchCV.

Another advantage of RandomizedSearchCV is that it can be more robust to the risk of overfitting since it does not exhaustively search the entire search space. If the hyperparameter search space is very large and the model is relatively simple, it is possible that GridSearchCV could overfit to the training data by finding a set of hyperparameters that works well on the training set but not as well on unseen data. RandomizedSearchCV can help mitigate this risk by sampling randomly from the search space rather than evaluating every combination.

It is worth noting that both RandomizedSearchCV and GridSearchCV can be computationally expensive, especially if the model is complex and the search space is large. In these cases, it may be necessary to use techniques such as parallelization or early stopping to speed up the search process.

Conclusion

In conclusion, grid search and randomized search are two commonly used methods for hyperparameter optimization in machine learning. Both methods have their strengths and weaknesses, and which one is more suitable for a given problem will depend on the specific circumstances. It is important to properly define the hyperparameter space and to use cross-validation when performing hyperparameter optimization to obtain the best results.