Machine Learning

| Randomized Search

Prof. Dr. Mostafa Elhosseini

https://youtube.com/@ProfElhosseiniTechVerse

Agenda

- Fine-Tune Your Model
- Hyperparameters
- Randomized Search
  - How it works?
  - Advantages compared to Grid search
  - Use cases
- HalvingRandomSearchCV
  - Merits

Hyperparameters

- Hyperparameters are the configuration settings used to structure the machine learning model, such as learning rate, regularization strength, number of layers or units in a neural network, kernel parameters for support vector machines, and so on.
- Examples include the These settings are not learned from the data but are set prior to the training process.

Hyperparameters

- Hyperparameter Optimization (HPO) involves finding the set of hyperparameters for a learning algorithm that yields the best performance, as measured on a validation set or via cross-validation.

Randomized search

Randomized search for hyperparameter optimization works as follows:

- **Define the hyperparameter search space**: Specify the ranges or discrete values for each hyperparameter to be tuned.
- **Choosing the number of iterations**: Decide how many different parameter settings to try.
- **Sample hyperparameter configurations randomly**: Draw random samples from the hyperparameter search space, where each sample represents a unique combination of hyperparameter values.

Randomized search

Randomized search for hyperparameter optimization works as follows:

- **Evaluate the model**: For each sampled hyperparameter configuration, train and evaluate the machine learning model using a performance metric (e.g., accuracy, F1-score, or any other relevant metric).
- **Keep track of the best configuration**: Store the hyperparameter configuration that yields the best performance so far.
- **Repeat steps 3-5** for a predetermined number of iterations or until a computational budget (e.g., time or number of evaluations) is exhausted.

Randomized search

Advantages:

- The randomized search approach is particularly useful when the hyperparameter space is **high-dimensional, complex**, or when there is little prior knowledge about **which regions might contain good** configurations.
- **Efficiency**: By not searching all possible combinations but instead randomly sampling the space, it significantly **reduces the computational** burden.

Randomized search

Advantages:

- **Scalability**: It's more **scalable to high-dimensional** spaces or when the relationship between the hyperparameters and the model performance is unknown.
- **Surprisingly Effective**: For many problems, randomized search has been found to be as effective, if not more so, than grid search for a fraction of the **computational cost**.

Randomized search

Use cases:

- **High-Dimensional Spaces**: For models with a large number of hyperparameters, such as **deep neural networks**.
- **Unknown Optimal Hyperparameter Regions**: When there's limited prior knowledge about which hyperparameters are most influential or what their optimal values might be.
- **Quick Prototyping**: In early stages of model development, when the goal is to **quickly identify** a reasonably good hyperparameter set for more **refined tuning later**, randomized search provides a balance between exploration and computational cost.

Randomized search

Use cases:

- **Budget-Constrained Optimization**: Limited Computational Resources
- **Imbalanced Hyperparameter Influence**: Dominant Parameters - In scenarios where a few hyperparameters significantly impact model performance while others have minimal effect, randomized search can be more effective than grid search in identifying beneficial combinations, as it does not allocate equal resources to explore all parameters.

Randomized search

Use cases:

- **Complementary to Other Techniques:** Initial Phase in a Two-Stage Process: Randomized search can be used as an initial exploration phase to identify promising regions of the hyperparameter space, which can then be further refined using more precise but computationally expensive methods like **Bayesian optimization**.
- **Large Datasets**: Scalability Concerns: For very large datasets, training even a single model can be time-consuming. Randomized search allows for a more manageable number of model trainings compared to the potentially vast number required by grid search.

RandomizedSearchCV

- It evaluates a fixed number of combinations, selecting a random value for each hyperparameter at every iteration.
- For each hyperparameter, you must provide either a list of possible values, or a probability distribution.

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {'preprocessing__geo__n_clusters': randint(low=3, high=50),
                  'random_forest__max_features': randint(low=2, high=20)}

rnd_search = RandomizedSearchCV(
    full_pipeline, param_distributions=param_distribs, n_iter=10, cv=3,
    scoring='neg_root_mean_squared_error', random_state=42)

rnd_search.fit(housing, housing_labels)
```

HalvingRandomSearchCV

- HalvingRandomSearchCV and HalvingGridSearchCV are parts of the broader family of successive **halving algorithms** and represents an efficient approach to finding optimal hyperparameters for a model.
- It combines the **principles of random search** with the efficiency of **iterative resource allocation** and early-stopping to quickly converge to a high-performing model with significantly less computational cost compared to traditional exhaustive methods like grid search.

HalvingRandomSearchCV

- The algorithm initially allocates a small amount of resources (e.g., a limited number of iterations, data samples, or epochs) to a relatively large number of randomly chosen hyperparameter configurations.
- After the initial evaluation, only the most promising configurations (typically the top half) are selected to proceed to the next round, where they are allocated more resources.
- This process of halving the number of configurations and doubling their resources continues iteratively until a single best configuration remains or until a specified stopping criterion is met.

HalvingRandomSearchCV

- The algorithm initially allocates a small amount of resources (e.g., a limited number of iterations, data samples, or epochs) to a relatively large number of randomly chosen hyperparameter configurations.
- After the initial evaluation, only the most promising configurations (typically the top half) are selected to proceed to the next round, where they are allocated more resources.
- This process of halving the number of configurations and doubling their resources continues iteratively until a single best configuration remains or until a specified stopping criterion is met.

HalvingRandomSearchCV

- Useful when you need to optimize hyperparameters for models with **expensive training processes** or when dealing with **large datasets**.
- Its efficient exploration strategy makes it an attractive choice for initial hyperparameter tuning phases, especially when the optimal hyperparameter space is unknown or when computational resources are limited.

HalvingRandomSearchCV - Summary

1. Starts with a pool of candidate hyperparameter combinations: This pool is defined by you.
2. Trains models: It trains multiple models on a small subset of the data using different combinations from the pool.
    3. Evaluates performance: It evaluates the performance of each model.
    4. Eliminates poorly performing models: The bottom half of the models (based on performance) are discarded.
    5. Allocates more resources: The remaining models are trained on a larger subset of the data.
6. Repeats: Steps 3-5 are repeated until a single model or a specific number of models remain.

HalvingRandomSearchCV

```
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV
```

- Import and enable the experimental functionality for the Halving Random Search.
- The **experimental module** contains features that are not yet part of the stable API but are available for testing and experimentation.
- **HalvingRandomSearchCV**: This class implements the Halving Random Search Cross-Validation algorithm

HalvingRandomSearchCV

```
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV
```

- The **enable_halving_search_cv** line is necessary because the Halving Random Search CV functionality is still experimental in scikit-learn and may be subject to changes or improvements in future releases.

HalvingRandomSearchCV

```
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV
from sklearn.tree import DecisionTreeClassifier

# Setup the model
model = DecisionTreeClassifier()

# Setup Halving Random Search
hrscv = HalvingRandomSearchCV(estimator=model, param_distributions=param_space, n_candidates=40,
factor=2, resource='n_samples', scoring='accuracy', cv=5)
```