GridSearchCV: Hyperparameter Tuning in Machine Learning

Estimated Time: 15 minutes

Objectives

At the end of this reading, you will be able to:

- Explain the concept of GridSearchCV
- Identify scenarios where GridSearchCV can be useful
- Use GridSearchCV to perform hyperparameter tuning for machine learning models

Introduction to GridSearchCV

GridSearchCV in Scikit-Learn is a vital tool for hyperparameter tuning, performing an exhaustive search over specified parameter values for an estimator. It systematically evaluates each combination using cross-validation to identify the optimal settings that maximize model performance based on a scoring metric like accuracy or F1-score. Hyperparameter tuning is crucial as it significantly impacts model performance, preventing underfitting or overfitting. GridSearchCV automates this process, ensuring robust generalization on unseen data. It helps data scientists efficiently find the best hyperparameters, saving time and resources while optimizing model performance, making it an essential tool in the machine learning pipeline.

Parameters of GridSearchCV

GridSearchCV has several important parameters:

Estimator: The model or pipeline to be optimized. This can be any Scikit-Learn estimator like `LogisticRegression()`,`SVC()`, `RandomForestClassifier()`, etc.

param_grid: A dictionary or list of dictionaries with parameter names (as strings) as keys and lists of parameter settings to try as values. Using `param_grid`, you can specify the hyperparameters for various models to find the optimal combination.

Examples of various models hyperparameters for the `param_grid` parameter.

- **Logistic Regression:** When tuning a logistic regression model, GridSearchCV can search through different values of `C`, `penalty`, and `solver` to find the best parameters.

```
1. 1
2. 2
3. 3
```

```
1. parameters = {'C': [0.01, 0.1, 1],
2.               'penalty': ['l2'],
3.               'solver': ['lbfgs']
```

Copied!

`C`: Inverse of regularization strength; smaller values specify stronger regularization.
`penalty`: Specifies the norm of the penalty; `'l2'` is ridge regression.
`solver`: Algorithm to use in the optimization problem.

- **Support Vector Machine:** For SVM, GridSearchCV can explore different `kernels`, `C values`, and `gamma` settings to optimize the model.

```
1. 1
2. 2
3. 3
```

```
1. parameters = {'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
2.               'C': np.logspace(-3, 3, 5),
3.               'gamma': np.logspace(-3, 3, 5)}
```

Copied!

`kernel`: Specifies the kernel type to be used in the algorithm.
`C`: Regularization parameter.
`gamma`: Kernel coefficient.

- **Decision Tree Classifier:** In the case of a decision tree, GridSearchCV can test various criteria, splitters, depths, and other parameters to find the best configuration.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
```

```
1. parameters = {'criterion': ['gini', 'entropy'],
2.               'splitter': ['best', 'random'],
3.               'max_depth': [2*n for n in range(1, 10)],
4.               'max_features': ['auto', 'sqrt'],
5.               'min_samples_leaf': [1, 2, 4],
6.               'min_samples_split': [2, 5, 10]}
```

Copied!

`criterion`: The function to measure the quality of a split.
`splitter`: The strategy used to choose the split at each node.
`max_depth`: The maximum depth of the tree.
`max_features`: The number of features to consider when looking for the best split.
`min_samples_leaf`: The minimum number of samples required to be at a leaf node.
`min_samples_split`: The minimum number of samples required to split an internal node.

- **K-Nearest Neighbors:** For KNN, GridSearchCV can try different numbers of neighbors, algorithms, and power parameters to determine the best model.

```
1
2
3
```

```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1, 2]}
```

Copied!

`n_neighbors`: Number of neighbors to use.
`algorithm`: Algorithm used to compute the nearest neighbors.
`p`: Power parameter for the Minkowski metric.

scoring: A single string or callable to evaluate the predictions on the test set. Common options include `accuracy`, `f1`, `roc_auc`, etc. If none, the estimator's default scorer is used.

n_jobs: The number of jobs to run in parallel. `-1` means using all processors.

pre_dispatch: Controls the number of jobs that get dispatched during parallel execution. It can be an integer or expressions like `2n_jobs`, `3n_jobs`, etc., to limit the number of jobs dispatched at once.

refit: If `True`, refits the best estimator with the entire dataset. The best estimator is stored in the best_estimator_ attribute. Default is `True`.

cv: Determines the cross-validation splitting strategy. It can be an integer to specify the number of folds, a cross-validation generator, or an iterable. Default is `5-fold cross-validation`.

verbose: Controls the verbosity level. Higher values indicate more messages. `verbose=0` is silent, `verbose=1` shows some messages, and `verbose=2` shows more messages.

return_train_score: If `False`, the `cv_results_ attribute` will not include training scores. Default is `False`.

error_score: Value to assign to the score if an error occurs in estimator fitting. `np.nan` is the default, but it can be set to a specific value.

Applications and Advantages of GridSearchCV

- **Model Selection:** GridSearchCV enables the comparison of multiple models and facilitates the selection of the best-performing one for a given data set.
- **Hyperparameter Tuning:** It automates the process of finding the optimal hyperparameters, which can significantly improve the performance of machine learning models.
- **Pipeline Optimization:** GridSearchCV can be applied to complex pipelines involving multiple preprocessing steps and models to optimize the entire workflow.
- **Cross-Validation:** It incorporates cross-validation in the parameter search process, ensuring that the model's performance is robust and not overfitted to a particular train-test split.
- **Exhaustive Search:** GridSearchCV performs an exhaustive search over the specified parameter grid, ensuring that the best combination of parameters is found.
- **Parallel Execution:** With the n_jobs parameter, it can leverage multiple processors to speed up the search process.
- **Automatic Refit:** By setting `refit=True`, GridSearchCV automatically refits the model with the best parameters on the entire data set, making it ready for use.
- **Detailed Output:** The cv_results_ attribute provides detailed information about the performance of each parameter combination, including training and validation scores, which helps in understanding the model's behavior.

Practical Example

Let us demonstrate the use of `GridSearchCV` with a practical example using the Iris data set. We will perform a grid search to find the optimal hyperparameters for a support vector classifier (SVC).

Import necessary libraries: First, import the essential libraries required for loading the data set, splitting the data, performing GridSearchCV, and evaluating the model.

```
1
2
3
4
5
6
7
8
9
```

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report
import warnings
# Ignore warnings
warnings.filterwarnings('ignore')
```

Copied!

Load the Iris data set: The Iris data set is a classic data set in machine learning. Load it using the `load_iris` function from Scikit-Learn.

```
1
2
3
```

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

Copied!

- X: Features of the Iris dataset (sepal length, sepal width, petal length, petal width).
- y: Target labels representing the three species of Iris (setosa, versicolor, virginica).

Splitting the data into training and test set: Divide data set into training and test sets to evaluate how well the model performs on data it has not been trained on.

```
1
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Copied!

- test_size=0.2: 20% of the data is used for testing.
- random_state=42: Ensures reproducibility of the random split.

Define the parameter grid: Specify a grid of hyperparameters for the SVM model to search over. The grid includes different values for `C`, `gamma`, and `kernel`.

```
1
2
3
4
5
```

```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['linear', 'rbf', 'poly']
}
```

Copied!

`C`: Regularization parameter.
`gamma`: Kernel coefficient.
`kernel`: Specifies the type of kernel to be used in the algorithm.

Initialize the SVC model: Create an instance of the support vector classifier (SVC).

```
1
```

```
svc = SVC()
```

Copied!

Initialize GridSearchCV: Set up the GridSearchCV with the SVC model, the parameter grid, and the desired configuration.

```
1
```

```
grid_search = GridSearchCV(estimator=svc, param_grid=param_grid, scoring='accuracy', cv=5, n_jobs=-1, verbose=2)
```

Copied!

`estimator`: The model to optimize (SVC).
`param_grid`: The grid of hyperparameters.
`scoring='accuracy'`: The metric used to evaluate the model's performance.
`cv=5`: 5-fold cross-validation.
`n_jobs=-1`: Use all available processors.
`verbose=2`: Show detailed output during the search.

Fit GridSearchCV to the training data: Perform the grid search on the training data.

```
1
```

```
grid_search.fit(X_train, y_train)
```

Copied!

Check the best parameters and estimator: After fitting, print the best parameters and the best estimator found during the grid search.

```
1
2
```

```
print("Best parameters found: ", grid_search.best_params_)
print("Best estimator: ", grid_search.best_estimator_)
```

Copied!

Make predictions with the best estimator: Use the best estimator to make predictions on the test set.

```
1
```

```
y_pred = grid_search.best_estimator_.predict(X_test)
```

Copied!

Evaluate the performance: Evaluate the model's performance on the test set using the classification_report function, which provides precision, recall, F1-score, and support for each class.

```
1
```

```
print(classification_report(y_test, y_pred))
```

Copied!

Key Points

- GridSearchCV conducts a thorough exploration across a defined parameter grid.
- Parameters include the estimator to optimize, parameter grid, scoring method, number of jobs for parallel execution, cross-validation strategy, and verbosity.
- Practical example demonstrated using GridSearchCV to find the optimal parameters for an SVC model on the Iris data set.
- GridSearchCV helps in selecting the best model by evaluating multiple combinations of hyperparameters.

Summary

In this reading, you learned about GridSearchCV, a powerful tool for hyperparameter tuning in Scikit-Learn. You explored its parameters and saw a practical example using the Iris data set. By leveraging GridSearchCV, you can systematically and efficiently find the best hyperparameters for your machine learning models, leading to improved performance.

Author(s)

Pratiksha Verma

Other Contributors

Malika Singla, Lakshmi Holla