GridSearchCV : Réglage des hyperparamètres dans l'apprentissage automatique

Temps estimé : 15 minutes

Objectifs

À la fin de cette lecture, vous serez en mesure de :

- Expliquer le concept de GridSearchCV
- Identifiez les scénarios où GridSearchCV peut être utile
- Utiliser GridSearchCV pour effectuer le réglage des hyperparamètres pour les modèles d'apprentissage automatique

Introduction à GridSearchCV

GridSearchCV dans Scikit-Learn est un outil essentiel pour le réglage des hyperparamètres, effectuant une recherche exhaustive sur des valeurs de paramètres spécifiées pour un estimateur. Il évalue systématiquement chaque combinaison à l'aide d'une validation croisée afin d'identifier les paramètres optimaux qui optimisent les performances du modèle en fonction d'une métrique de notation telle que la précision ou le score F1. Le réglage des hyperparamètres est crucial car il a un impact significatif sur les performances du modèle, empêchant le sous-ajustement ou le surapprentissage. GridSearchCV automatise ce processus, assurant une généralisation robuste sur des données invisibles. Il aide les data scientists à trouver efficacement les meilleurs hyperparamètres, ce qui permet d'économiser du temps et des ressources tout en optimisant les performances du modèle, ce qui en fait un outil essentiel dans le pipeline d'apprentissage automatique.

Paramètres de GridSearchCV

GridSearchCV a plusieurs paramètres importants :

Estimateur: Le modèle ou le pipeline à optimiser. Il peut s'agir de n'importe quel estimateur Scikit-Learn comme ,, , etc.`LogisticRegression()` `SVC()` `RandomForestClassifier()`

param_grid : Dictionnaire ou liste de dictionnaires avec des noms de paramètres (sous forme de chaînes) comme clés et des listes de paramètres à essayer comme valeurs. À l'aide de , vous pouvez spécifier les hyperparamètres de différents modèles afin de trouver la combinaison optimale.`param_grid`

Exemples de différents modèles d'hyperparamètres pour le paramètre.`param_grid`

- **Régression logistique :** Lors de l'ajustement d'un modèle de régression logistique, GridSearchCV peut rechercher à travers différentes valeurs de , et trouver les meilleurs paramètres.`C` `penalty` `solver`

```
1
2
3
```

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

Copié!

`C`: Inverse de l'intensité de régularisation ; Des valeurs plus petites indiquent une régularisation plus forte.
: Précise la norme de la peine ; est la régression de crête.
: Algorithme à utiliser dans le problème d'optimisation.`penalty` `'l2'` `solver`

- **Machine à vecteurs de support :** Pour SVM, GridSearchCV peut explorer différents paramètres , , et pour optimiser le modèle.`kernels` `C` `values` `gamma`

```
1
2
3
```

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

Copié!

`kernel`: Spécifie le type de noyau à utiliser dans l'algorithme.
: Paramètre de régularisation.
: Coefficient du noyau.`Cgamma`

- **Classificateur d'arbre de décision :** Dans le cas d'un arbre de décision, GridSearchCV peut tester divers critères, séparateurs, profondeurs et autres paramètres pour trouver la meilleure configuration.

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

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

Copié!

`criterion`: La fonction permettant de mesurer la qualité d'un fractionnement.
: La stratégie utilisée pour choisir le fractionnement à chaque nœud.
: La profondeur maximale de l'arbre.
: Le nombre de fonctionnalités à prendre en compte lors de la recherche de la meilleure répartition.
: Le nombre minimum d'échantillons requis pour se trouver à un nœud foliaire.
: Le nombre minimum d'échantillons requis pour diviser un nœud interne.`splittermax_depthmax_featuresmin_samples_leafmin_samples_split`

- **k-Voisins les plus proches :** Pour KNN, GridSearchCV peut essayer différents nombres de voisins, algorithmes et paramètres de puissance pour déterminer le meilleur modèle.

```
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]}
```

Copié!

`n_neighbors`: Nombre de voisins à utiliser.
: Algorithme utilisé pour calculer les voisins les plus proches.
: Paramètre de puissance pour la métrique de Minkowski.`algorithmp`

Marquer: Une seule chaîne ou un appelable pour évaluer les prédictions sur l'ensemble de test. Les options courantes incluent , , , etc. S'il n'y en a pas, c'est le marqueur par défaut de l'estimateur qui est utilisé.`accuracyf1roc_auc`

n_jobs : Nombre de tâches à exécuter en parallèle. signifie l'utilisation de tous les processeurs.`-1`

pre_dispatch : Contrôle le nombre de tâches distribuées lors de l'exécution parallèle. Il peut s'agir d'un entier ou d'expressions telles que , , etc., pour limiter le nombre de tâches distribuées à la fois.`2n_jobs3n_jobs`

radouber: Si , réajuste le meilleur estimateur avec l'ensemble des données. Le meilleur estimateur est stocké dans l'attribut best_estimator_. La valeur par défaut est `.TrueTrue`

CV: Détermine la stratégie de fractionnement de validation croisée. Il peut s'agir d'un entier pour spécifier le nombre de plis, d'un générateur de validation croisée ou d'un itérable. La valeur par défaut est `.5-fold cross-validation`

verbeux: Contrôle le niveau de verbosité. Des valeurs plus élevées indiquent un plus grand nombre de messages. est silencieux, affiche certains messages et en affiche d'autres.`verbose=0verbose=1verbose=2`

return_train_score : Si , le n'inclura pas les scores d'entraînement. La valeur par défaut est `.Falsecv_results_ attributeFalse`

error_score : Valeur à attribuer au score si une erreur se produit dans l'ajustement de l'estimateur. est la valeur par défaut, mais elle peut être définie sur une valeur spécifique.`np.nan`

Applications et avantages de GridSearchCV

- **Sélection du modèle :** GridSearchCV permet de comparer plusieurs modèles et facilite la sélection du plus performant pour un ensemble de données donné.

- **Réglage des hyperparamètres :** Il automatise le processus de recherche des hyperparamètres optimaux, ce qui peut améliorer considérablement les performances des modèles d'apprentissage automatique.
- **Optimisation du pipeline :** GridSearchCV peut être appliqué à des pipelines complexes impliquant plusieurs étapes de prétraitement et des modèles pour optimiser l'ensemble du flux de travail.
- **Validation croisée :** Il intègre la validation croisée dans le processus de recherche de paramètres, garantissant que les performances du modèle sont robustes et qu'elles ne sont pas surajustées à une répartition train-test particulière.
- **Recherche exhaustive :** GridSearchCV effectue une recherche exhaustive sur la grille de paramètres spécifiée, en s'assurant que la meilleure combinaison de paramètres est trouvée.
- **Exécution parallèle :** Avec le paramètre n_jobs, il peut exploiter plusieurs processeurs pour accélérer le processus de recherche.
- **Refit automatique :** En définissant , GridSearchCV réajuste automatiquement le modèle avec les meilleurs paramètres sur l'ensemble de l'ensemble des données, le rendant prêt à l'emploi.`refit=True`
- **Sortie détaillée :** L'attribut cv_results_ fournit des informations détaillées sur les performances de chaque combinaison de paramètres, y compris les scores d'entraînement et de validation, ce qui permet de comprendre le comportement du modèle.

Exemple pratique

Illustrons l'utilisation de à l'aide d'un exemple pratique à l'aide de l'ensemble de données Iris. Nous allons effectuer une recherche par grille pour trouver les hyperparamètres optimaux pour un classificateur de vecteurs de support (SVC).`GridSearchCV`

Importez les bibliothèques nécessaires : Tout d'abord, importez les bibliothèques essentielles nécessaires au chargement de l'ensemble de données, à la division des données, à l'exécution de GridSearchCV et à l'évaluation du modèle.

```
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 function from Scikit-Learn.`load_iris`

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

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

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. 1
```

```
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. 1
```

```
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