Optimization of hyperparameters

Paulo S. A. Sousa

2023-04-18

Abstract

We discuss the need of hyperparameter optimization and introduce grid search with cross-validation.

We present a Python implementation of grid search.

1 Introduction

In machine learning, a hyperparameter is a parameter whose value is set before the training process begins, as opposed to the model parameters that are learned during the training process. Hyperparameters are external to the model and cannot be learned from the data, but they have a significant influence on the performance of the model.

Hyperparameters are used to configure the model architecture, learning process, and optimization algorithm. They can include learning rate, number of layers in a neural network, batch size, activation function, regularization techniques, and many others.

Tuning hyperparameters is an important step in the model development process, as it helps to find the optimal configuration for a given problem. This process often involves techniques such as grid search, random search, or more advanced methods like Bayesian optimization to systematically explore different combinations of hyperparameters and identify the best-performing set. In our course, we will focus only on grid search.

2 Grid search

Grid search is a popular and straightforward technique used for hyperparameter tuning in machine learning. It works by exhaustively evaluating all possible combinations of hyperparameter values in a predefined search space to find the best configuration for a given model. Grid search can be applied to various machine learning algorithms, such as Lasso regression and K-Nearest Neighbors (KNN).

2.1 Python implementation

To illustrate the use of grid search in Python, we will use Lasso regression on the car radios dataset, as performed in the respective lecture:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import Lasso
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import r2_score, mean_absolute_error,
 ↪  mean_squared_error
import datetime

df =
 ↪  pd.read_excel('/home/x/programas/r/pythonL/datasets/data_carradios.xlsx')


def get_ages(col):
  result = (datetime.datetime.now()-col).astype('<m8[Y]')
  result = pd.DataFrame(result)
  return result

ager = Pipeline([
    ('ages', FunctionTransformer(get_ages,
 ↪  feature_names_out='one-to-one')),
    ('scale', StandardScaler())
])

def get_weekdays(col):
  result = col.iloc[:,0].dt.weekday
  result = pd.DataFrame(result)
  return result

weeker = Pipeline([
    ('weekd', FunctionTransformer(get_weekdays,
 ↪  feature_names_out='one-to-one')),
    ('oneh', OneHotEncoder(drop='first'))
])

preprocessor = ColumnTransformer([
    ('ages_tr', ager, ['bdate']),
    ('weekd_tr', weeker, ['datep']),
```

```
    ('team_tr', OneHotEncoder(drop='first'), ['team']),
    ('scaler', StandardScaler(), ['prized', 'prizeq'])],
    remainder='passthrough')

X = df.drop('perc_defec', axis=1)
y = df['perc_defec']

X_train, X_test, y_train, y_test = train_test_split(X, y,
 ↪  test_size=0.2, random_state=45)

pipe = Pipeline([
    ('pre', preprocessor),
    ('lasso', Lasso(alpha=0.1))])
```

The parameter `alpha` is a hyperparameter. It controls the magnitude of penalty Lasso regression introduces. A larger value of `alpha` will increase the penalty, which in turn will shrink the coefficients toward zero. Consequently, with a larger `alpha`, more coefficients are likely to be exactly zero, effectively reducing the number of features used in the model. On the other hand, a smaller value of `alpha` will result in a weaker penalty, leading to fewer coefficients being forced to zero.

Typically, it is used `alpha=0.1`, but we do not know whether there is a better value for `alpha`. Then, we can create a grid of values for `alpha` and search for the best value. To accomplish this in Python, we first need to define a dictionary with the values for `alpha` we want to test (and we want, for instance, to test 0.01, 0.02, 0.03, 0.04, 0.05, 0.5):

```
hyper = {
    'alpha': [0.01, 0.02, 0.03, 0.04, 0.05, 0.5]
}
```

To load the library with the `GridSearchCV` function:

```
from sklearn.model_selection import GridSearchCV
```

To apply the grid search, we need to redefine the final pipeline:

```
pipe = Pipeline([
    ('pre', preprocessor),
    ('grid', GridSearchCV(Lasso(),hyper, cv=5))])
```

The parameter `cv` defines the number of folds for cross-validation. Below, we will discuss what cross-validation is.

After fitting the new pipeline, we will get the best value for `alpha` (among the ones tried out):

```
pipe.fit(X_train, y_train)
pipe.named_steps['grid'].best_params_
```

In our case, the best value for `alpha` that was found is: `{'alpha': 0.03}`.

To make predictions using the optimized value for `alpha`, we just use the usual command, as `pipe` learns and uses the best value for `alpha`:

```
pipe.predict(X_train)
```

2.2 Cross-validation

Cross-validation is used to measure the predictive performance corresponding to each value of the hyperparameter considered.

It starts by dividing the training set into k equally-sized subsets, or folds. In each iteration of the cross-validation process, one of these folds is used as the test fold, and the remaining k-1 folds are used, together, as the training set (do not confuse this training set with the original training set – this training set is a set resulting from the union of all mentioned k-1 folds).

For each value of the hyperparameter in the grid, the model is trained using the training set (formed by the k-1 folds) and evaluated on the test fold. This process is repeated k times, with each fold serving as the test fold once. A predictive performance metric, such as accuracy or mean squared error, is averaged across the k iterations to produce a single performance score for the given hyperparameter value.

After evaluating all possible values of the hyperparameter, the value that yields the best performance score is selected as the optimal value of the hyperparameter for the model.

The parameter `cv=5` used in the `GridSearchCV` function instructs the function to use 5 folds.