15-Hyperparameter_Tuning

October 20, 2024

1 Hyperparameter Tuning

Hyperparameter tuning is the process of finding the optimal values for the hyperparameters of a machine learning model. Unlike model parameters, which are learned from the training data (e.g., weights in a neural network, coefficients in a regression model), hyperparameters are set before the learning process and control how the model is trained or the structure of the model.

Examples of hyperparameters include:

- Learning rate in gradient-based algorithms.
- Number of trees in a Random Forest.
- Depth of a decision tree.
- C (regularization) and kernel in Support Vector Machines.
- k in k-nearest neighbors (KNN).
- Batch size or number of layers in neural networks.

Tuning these hyperparameters is crucial because they can have a significant impact on the model's performance. The goal is to find the best combination of hyperparameter values that yields the highest-performing model.

When to Use Hyperparameter Tuning? Hyperparameter tuning is used:

- **When building any machine learning model**: Most machine learning algorithms have hyperparameters that need to be fine-tuned.
- **When optimizing model performance**: After selecting a base model, hyperparameter tuning can help improve its accuracy, precision, recall, or other performance metrics.
- **To avoid overfitting/underfitting**: The right hyperparameters can prevent a model from overfitting (too complex) or underfitting (too simple) the data.
- **During model validation**: When performing cross-validation, tuning helps adjust the model for better generalization to unseen data.

Hyperparameter tuning is commonly used when:

- Building models for production systems where performance is critical.
- Participating in machine learning competitions.

- Deploying models with specific performance constraints, such as low latency in real-time systems.

1.0.1 How Does Hyperparameter Tuning Work?

The tuning process involves searching through a defined set of hyperparameter values and evaluating the model's performance for each combination of values. There are several methods to tune hyperparameters:

1. Grid Search:

A brute-force approach that evaluates all possible combinations of hyperparameters from a specified grid. You define a grid (list of values) for each hyperparameter, and the algorithm tries all possible combinations. Example: If you have two hyperparameters with 3 values each, Grid Search will evaluate all $3\times3=9$ combinations.

Pros:

- Exhaustive: Tests every combination.
- Simple to implement and understand.

Cons:

- Computationally expensive for large datasets or a large number of hyperparameters.
- May test irrelevant combinations of hyperparameters, leading to inefficiency..

2. Random Search:

- Instead of evaluating all combinations, Random Search randomly samples combinations of hyperparameters.
- You define a range of values for each hyperparameter, and the algorithm randomly selects combinations for evaluation.

Pros:

- More efficient than Grid Search as it focuses on randomly chosen combinations.
- Can still find good hyperparameter values without exhaustive search.

Cons:

- May miss the best combination if it isn't sampled.
- Still computationally expensive if many iterations are performed.

3. Bayesian Optimization:

- Bayesian optimization models the hyperparameter search as a probability problem and selects hyperparameter values based on the likelihood of improving the model's performance.
- This method focuses on exploring the hyperparameter space intelligently rather than randomly, making it more efficient than Grid or Random Search.

Pros:

- More efficient, as it focuses on the most promising hyperparameter areas.

- Can achieve high performance with fewer evaluations.

Cons:

- More complex to implement and requires specialized libraries (e.g., Hyperopt, Optuna).

4. Gradient-Based Optimization:

Similar to gradient descent used in model training, gradient-based methods adjust hyperparameters based on how changes impact performance.

Pros:

- Can quickly converge to optimal hyperparameters.

Cons:

- Limited to continuous hyperparameters (can't handle categorical ones like tree depth or number of estimators).

5. Manual Search:

A trial-and-error approach where hyperparameters are manually tuned based on intuition and experience.

Pros:

- Simple and effective when you have experience with the model or the dataset.

Cons:

- May miss optimal settings due to lack of systematic exploration.
- Time-consuming if tried exhaustively.

1.0.2 Who Should Use Hyperparameter Tuning?

Hyperparameter tuning is essential for:

- Machine learning practitioners and data scientists who want to maximize model performance.
- Researchers testing different hypotheses or models and trying to extract the best possible results from their experiments.
- Beginners who are learning machine learning and want to experience the difference in model behavior with tuned hyperparameters.
- Competitors in machine learning contests, such as Kaggle, where the smallest improvement can make a difference in ranking.

Advantages of Hyperparameter Tuning:

- **Improved performance**: Well-tuned hyperparameters can significantly improve model accuracy, precision, recall, and other metrics.
- **Prevention of overfitting or underfitting**: Tuning can help find the sweet spot between a model that's too complex and one that's too simple.

- **Customization**: Allows for deep customization of machine learning models to suit the specific characteristics of your dataset.

Disadvantages of Hyperparameter Tuning:

- **Computational expense**: For large datasets or complex models, tuning (especially Grid Search) can be time-consuming and resource-intensive.
- **Complexity**: Some tuning methods, such as Bayesian optimization, can be hard to implement and require specialized knowledge.
- **Local minima**: Certain tuning strategies, like gradient-based methods, may get stuck in local minima and miss the global optimum.

1.0.3 Real-World Applications of Hyperparameter Tuning:

- **Deep Learning**: Tuning hyperparameters like learning rate, batch size, and the number of layers to improve model performance on image, text, or speech data.
- **Kaggle Competitions**: Competitors frequently use hyperparameter tuning to optimize their models for the best performance on the leaderboard.
- **Predictive Modeling**: In fields like finance and healthcare, tuning hyperparameters to build accurate predictive models for stock prices, patient outcomes, etc.
- **Natural Language Processing (NLP)**: Tuning hyperparameters for algorithms like transformers or recurrent neural networks for better text classification, translation, or sentiment analysis.

[30]:
```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split, GridSearchCV,
↪RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
```

[31]:
```
mean1 = 55
std_dev1 = 10
num_samples = 500

column1_numbers = np.random.normal(mean1, std_dev1, num_samples)
column1_numbers = np.clip(column1_numbers, 30, 120)
column1_numbers = np.round(column1_numbers).astype(int)

mean2 =18
std_dev2 = 3

column2_numbers = np.random.normal(mean2, std_dev2, num_samples)
column2_numbers = np.clip(column2_numbers, 30, 120)
column2_numbers = np.round(column2_numbers).astype(int)
```

```
column3_numbers = np.random.randint(2, size=num_samples)
column3_numbers[column1_numbers > mean1] = 1

data = {'Miles_Per_Week': column1_numbers, 'Farthest_run': column2_numbers,␣
 ↪'Qualified_Boston_Marathon': column3_numbers}
df = pd.DataFrame(data)
df
```

[31]:

```
     Miles_Per_Week  Farthest_run  Qualified_Boston_Marathon
0                55            30                          0
1                59            30                          1
2                72            30                          1
3                72            30                          1
4                30            30                          1
..              …             …                          …
495              51            30                          0
496              51            30                          1
497              76            30                          1
498              47            30                          1
499              61            30                          1

[500 rows x 3 columns]
```

[32]:

```
X = df.iloc[:, 0:2]
y = df.iloc[:, 2]

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,␣
 ↪random_state=26)

rf = RandomForestClassifier()
param_grid = [{
    'n_estimators': [500, 1000, 1500],
    'criterion': ['entropy', 'gini'],
    'min_samples_split': [5,10,15],
    'min_samples_leaf': [1,2,4],
    'max_depth': [10,20,30]
}]

grid_search = GridSearchCV(rf, param_grid, cv=2, scoring='accuracy', n_jobs=-1)

grid_search.fit(X_train, y_train)
```

[32]:

```
GridSearchCV(cv=2, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid=[{'criterion': ['entropy', 'gini'],
                          'max_depth': [10, 20, 30],
                          'min_samples_leaf': [1, 2, 4],
                          'min_samples_split': [5, 10, 15],
```

```
                            'n_estimators': [500, 1000, 1500]}],
             scoring='accuracy')
```

[33]:
```
grid_search.best_score_
```

[33]:
```
0.7885714285714286
```

[34]:
```
grid_search.best_params_
```

[34]:
```
{'criterion': 'gini',
 'max_depth': 20,
 'min_samples_leaf': 4,
 'min_samples_split': 15,
 'n_estimators': 1500}
```

[35]:
```
random_param_grid = [{
    'n_estimators': [500, 1000, 1500],
    'criterion': ['entropy', 'gini'],
    'min_samples_split': [5,10,15],
    'min_samples_leaf': [1,2,4],
    'max_depth': [10,20,30]
}]

random_grid_search = RandomizedSearchCV(rf, random_param_grid, cv=5,␣
 ↪scoring='accuracy', n_jobs=-1, random_state=26)
random_grid_search.fit(X_train, y_train)
```

[35]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
                   param_distributions=[{'criterion': ['entropy', 'gini'],
                                         'max_depth': [10, 20, 30],
                                         'min_samples_leaf': [1, 2, 4],
                                         'min_samples_split': [5, 10, 15],
                                         'n_estimators': [500, 1000, 1500]}],
                   random_state=26, scoring='accuracy')
```

[36]:
```
random_grid_search.best_score_
```

[36]:
```
0.7457142857142858
```

[37]:
```
random_grid_search.best_params_
```

[37]:
```
{'n_estimators': 1000,
 'min_samples_split': 10,
 'min_samples_leaf': 1,
 'max_depth': 20,
 'criterion': 'entropy'}
```