Implementing Cross-Validation and Hyperparameter Tuning for a Machine Learning Model

Table of Contents

Description

This lab guide provides a step-by-step approach to implementing cross-validation and hyperparameter tuning using a Random Forest classifier on the Iris dataset. You'll learn how to enhance your model's performance by selecting the best parameters and validating the results.

Problem Statement

In machine learning, building a model with high accuracy is often challenging. This lab aims to demonstrate how cross-validation and hyperparameter tuning can significantly improve model performance and reliability.

Prerequisites

Completion of all previous lab guides (up to Lab Guide-06) is required before proceeding with Lab Guide-07.

Software Requirements

- **Python**: Python 3.11.9
- **Visual Studio Code (VSCode)**: A lightweight code editor that provides powerful features for Python development, including extensions for linting, debugging, and version control.
- **Libraries**: `NumPy`, `Pandas`, `Matplotlib`, `Seaborn`, `Scikit-learn`

Hardware Requirements

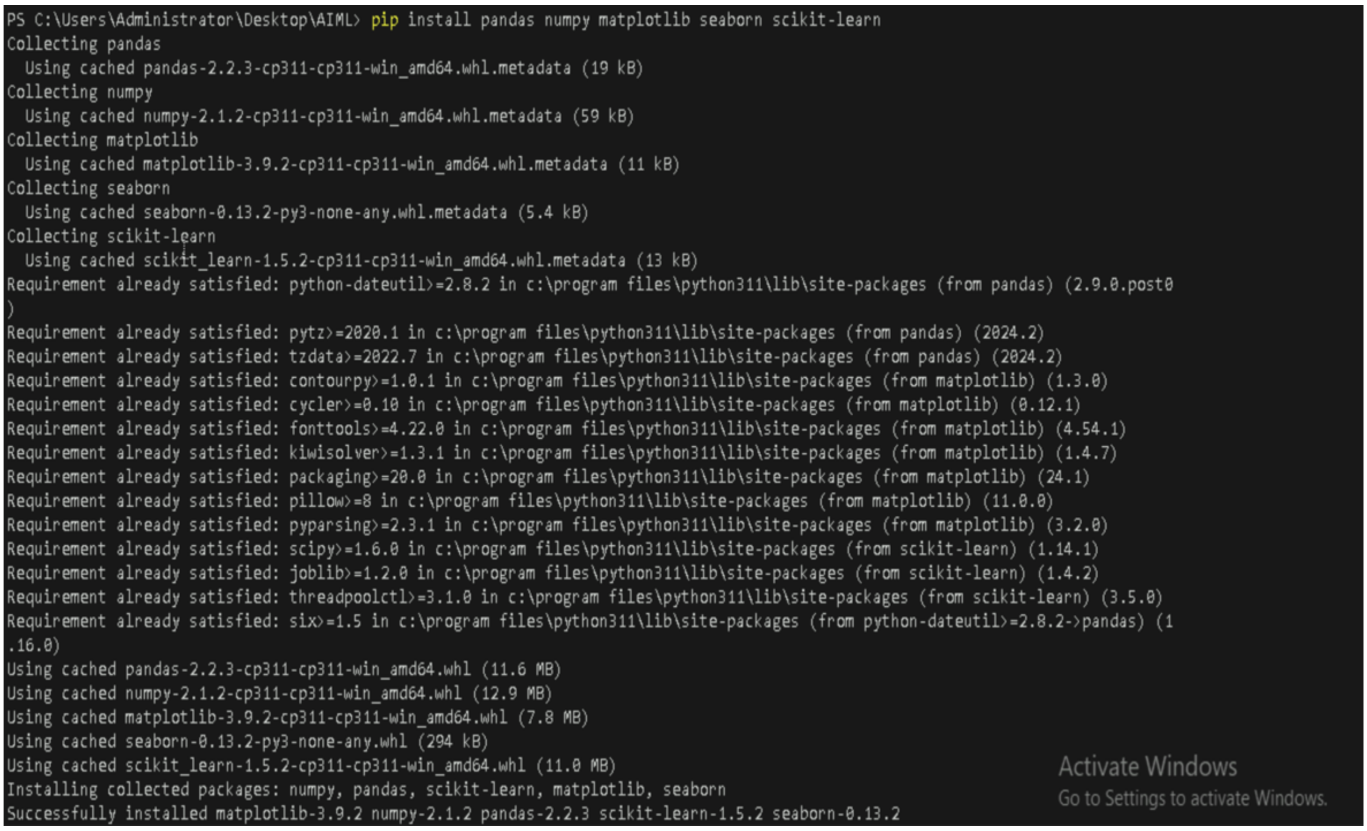
- Minimum 4GB RAM.
- At least 1GB of free disk space.
- A GPU (optional, but recommended for faster training).

Setup Instructions

- **Downloading the Dataset**:For this lab, we will use the Iris dataset, which is included in the Scikit-learn library. If you want to use another dataset, ensure it's in CSV format and accessible in your working directory.

- **Installing Required Libraries**: Make sure you have the necessary libraries installed. You can install them using pip:

```
pip install numpy pandas matplotlib seaborn scikit-learn
```



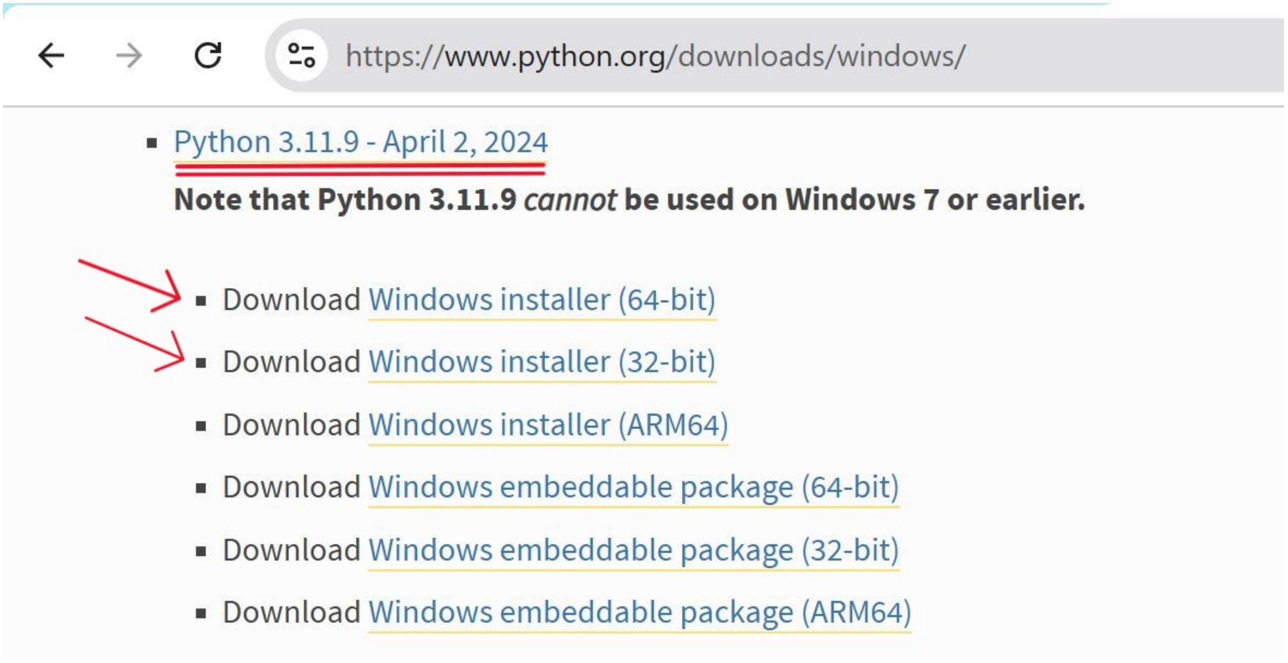
```
PS C:\Users\Administrator\Desktop\AIML> pip install pandas numpy matplotlib seaborn scikit-learn
Collecting pandas
  Using cached pandas-2.2.3-cp311-cp311-win_amd64.whl.metadata (19 kB)
Collecting numpy
  Using cached numpy-2.1.2-cp311-cp311-win_amd64.whl.metadata (59 kB)
Collecting matplotlib
  Using cached matplotlib-3.9.2-cp311-cp311-win_amd64.whl.metadata (11 kB)
Collecting seaborn
  Using cached seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Collecting scikit-learn
  Using cached scikit_learn-1.5.2-cp311-cp311-win_amd64.whl.metadata (13 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\program files\python311\lib\site-packages (from pandas) (2.9.0.post0
)
Requirement already satisfied: pytz>=2020.1 in c:\program files\python311\lib\site-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\program files\python311\lib\site-packages (from pandas) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\program files\python311\lib\site-packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in c:\program files\python311\lib\site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\program files\python311\lib\site-packages (from matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\program files\python311\lib\site-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\program files\python311\lib\site-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in c:\program files\python311\lib\site-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\program files\python311\lib\site-packages (from matplotlib) (3.2.0)
Requirement already satisfied: scipy>=1.6.0 in c:\program files\python311\lib\site-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in c:\program files\python311\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\program files\python311\lib\site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in c:\program files\python311\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1
.16.0)
Using cached pandas-2.2.3-cp311-cp311-win_amd64.whl (11.6 MB)
Using cached numpy-2.1.2-cp311-cp311-win_amd64.whl (12.9 MB)
Using cached matplotlib-3.9.2-cp311-cp311-win_amd64.whl (7.8 MB)
Using cached seaborn-0.13.2-py3-none-any.whl (294 kB)
Using cached scikit_learn-1.5.2-cp311-cp311-win_amd64.whl (11.0 MB)
Installing collected packages: numpy, pandas, scikit-learn, matplotlib, seaborn
Successfully installed matplotlib-3.9.2 numpy-2.1.2 pandas-2.2.3 scikit-learn-1.5.2 seaborn-0.13.2
```

Setting Up the Environment

1. Install Python:

You can download and install Python 3.11.9 from the official Python website:

- Visit the **official Python website**.
- Locate a reliable version of Python 3, **"Download Python 3.11.9"**.
- Choose the correct link for your device from the options provided: either Windows installer (64-bit) or Windows installer (32-bit) and proceed to download the executable file.



https://www.python.org/downloads/windows/

- Python 3.11.9 - April 2, 2024

  Note that Python 3.11.9 *cannot* be used on Windows 7 or earlier.

  - Download Windows installer (64-bit)
  - Download Windows installer (32-bit)
  - Download Windows installer (ARM64)
  - Download Windows embeddable package (64-bit)
  - Download Windows embeddable package (32-bit)
  - Download Windows embeddable package (ARM64)

2. Install Visual Studio Code (VSCode):

Download and install VSCode from the official Visual Studio Code website:

Download Visual Studio Code

Implement Cross-Validation and Hyperparameter Tuning for a Machine Learning Model

Understanding Cross-Validation

Cross-validation is a technique used to evaluate the performance of a machine learning model by splitting the data into training and testing sets multiple times. It helps in assessing how the results of a statistical analysis will generalize to an independent data set.

Understanding Hyperparameter Tuning

Hyperparameters are the parameters that are set before the learning process begins. Tuning these parameters can lead to better model performance. The two main methods for tuning are Grid Search and Random Search.

Model Selection

Choosing the right model is critical. For this lab, we will use the Random Forest classifier due to its robustness and versatility in handling classification tasks.

Implementing Cross-Validation

Create a new python file

- Create a Python file named `cross_validation.py`.

Importing Libraries

Use the following code to implement cross-validation with the Iris dataset:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
```

Load Dataset Splitting the data into training and testing sets

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) #splitting the dataset into train and test
```

Implementing Cross-Validation

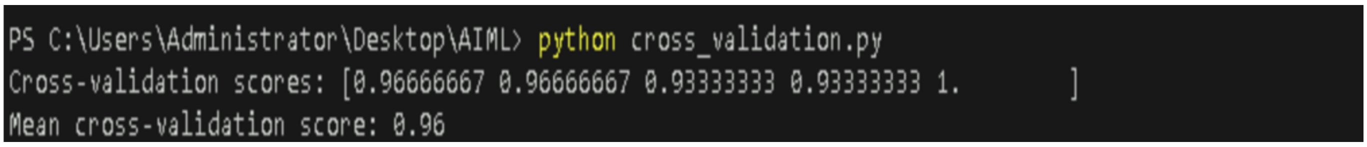
```
model = RandomForestClassifier()
scores = cross_val_score(model, X, y, cv=5)
print("Cross-validation scores:", scores)
print("Mean cross-validation score:", np.mean(scores))
```

Run the Python file

- Use the command below in your terminal to run the Python file:

```
python cross_validation.py
```

Output



```
PS C:\Users\Administrator\Desktop\AIML> python cross_validation.py
Cross-validation scores: [0.96666667 0.96666667 0.93333333 0.93333333 1.        ]
Mean cross-validation score: 0.96
```

Hyperparameter Tuning

- **Grid Search**
- **Random Search**

Create a new python file

- Create a Python file named `parameter_tuning.py` and write following code in it.

Grid Search

To find the best hyperparameters using Grid Search:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) #splitting the dataset
model = RandomForestClassifier()
scores = cross_val_score(model, X, y, cv=5)

# Hyperparameter tuning using Grid Search
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}
```

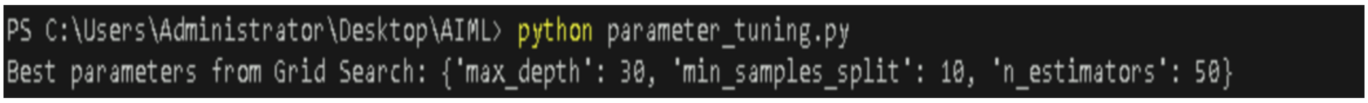
```
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,
scoring='accuracy')
grid_search.fit(X_train, y_train)

# Best parameters from Grid Search
print("Best parameters from Grid Search:", grid_search.best_params_)
```

Run the Python file

- Use the command below in your terminal to run the Python file:

```
python parameter_tuning.py
```

Output



```
PS C:\Users\Administrator\Desktop\AIML> python parameter_tuning.py
Best parameters from Grid Search: {'max_depth': 30, 'min_samples_split': 10, 'n_estimators': 50}
```

Random Search

You can also perform hyperparameter tuning using Random Search:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
data = load_iris()
X = data.data
y = data.target
#splitting the dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Hyperparameter tuning using Random Search
random_param_dist = {
    'n_estimators': np.arange(50, 200, 10),
    'max_depth': [None] + list(np.arange(10, 30, 5)),
    'min_samples_split': np.arange(2, 20, 2)
}

random_search = RandomizedSearchCV(estimator=model,
param_distributions=random_param_dist,n_iter=100, cv=5, scoring='accuracy',
random_state=42)
random_search.fit(X_train, y_train)
```

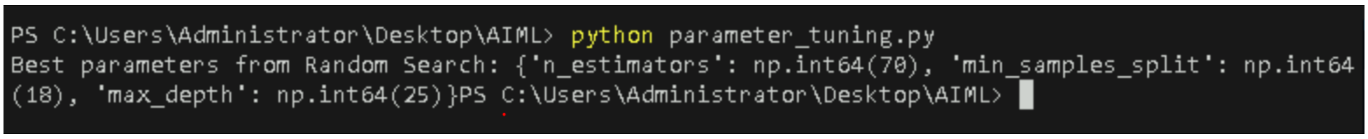
```
# Best parameters from Random Search
print("Best parameters from Random Search:", random_search.best_params_)
```

Run the Python file

- Use the command below in your terminal to run the Python file:

```
python parameter_tuning.py
```

Output



```
PS C:\Users\Administrator\Desktop\AIML> python parameter_tuning.py
Best parameters from Random Search: {'n_estimators': np.int64(70), 'min_samples_split': np.int64
(18), 'max_depth': np.int64(25)}PS C:\Users\Administrator\Desktop\AIML>
```

Model Evaluation

After hyperparameter tuning, evaluate your model using accuracy scores and confusion matrices:

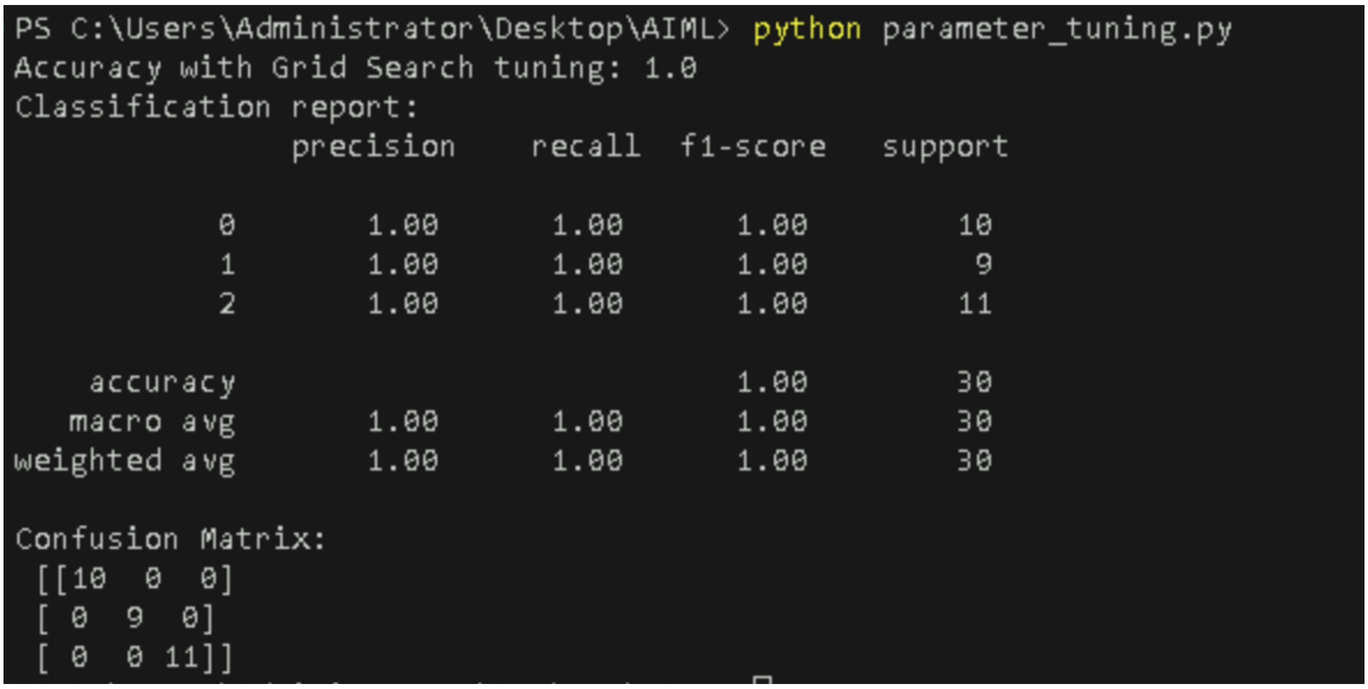
```
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# Evaluate the model with best parameters
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

print("Accuracy with Grid Search tuning:", accuracy_score(y_test, y_pred))
print("Classification report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

Run the Python file

- Use the command below in your terminal to run the Python file:

```
python parameter_tuning.py
```

Output



```
PS C:\Users\Administrator\Desktop\AIML> python parameter_tuning.py
Accuracy with Grid Search tuning: 1.0
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

Visualizing Results

Visualize the confusion matrix to better understand your model's performance:

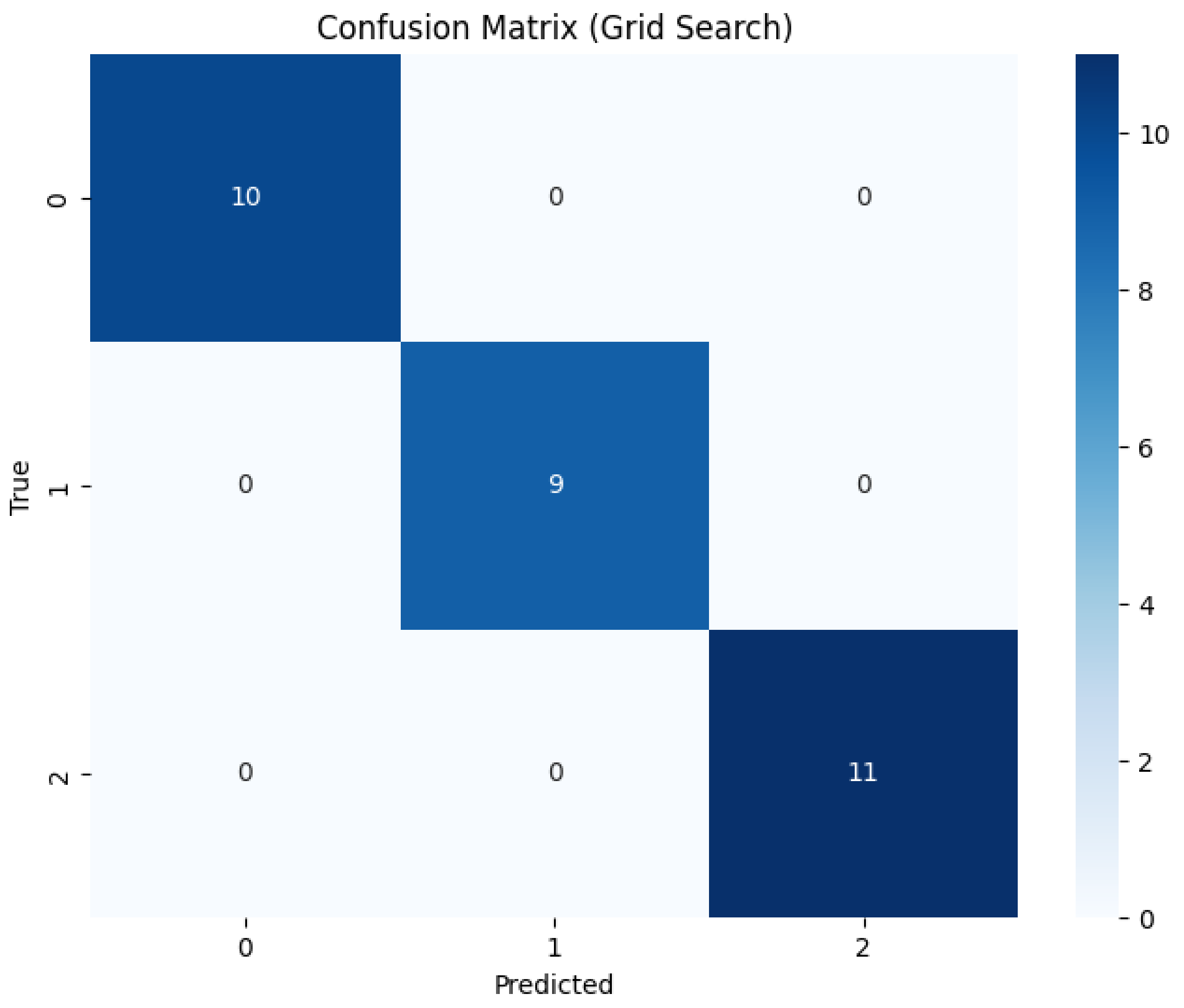
```python
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix (Grid Search)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Run the Python file

- Use the command below in your terminal to run the Python file:

```
python parameter_tuning.py
```

Output



Visualizing the Confusion Matrix for the best model from Random Search

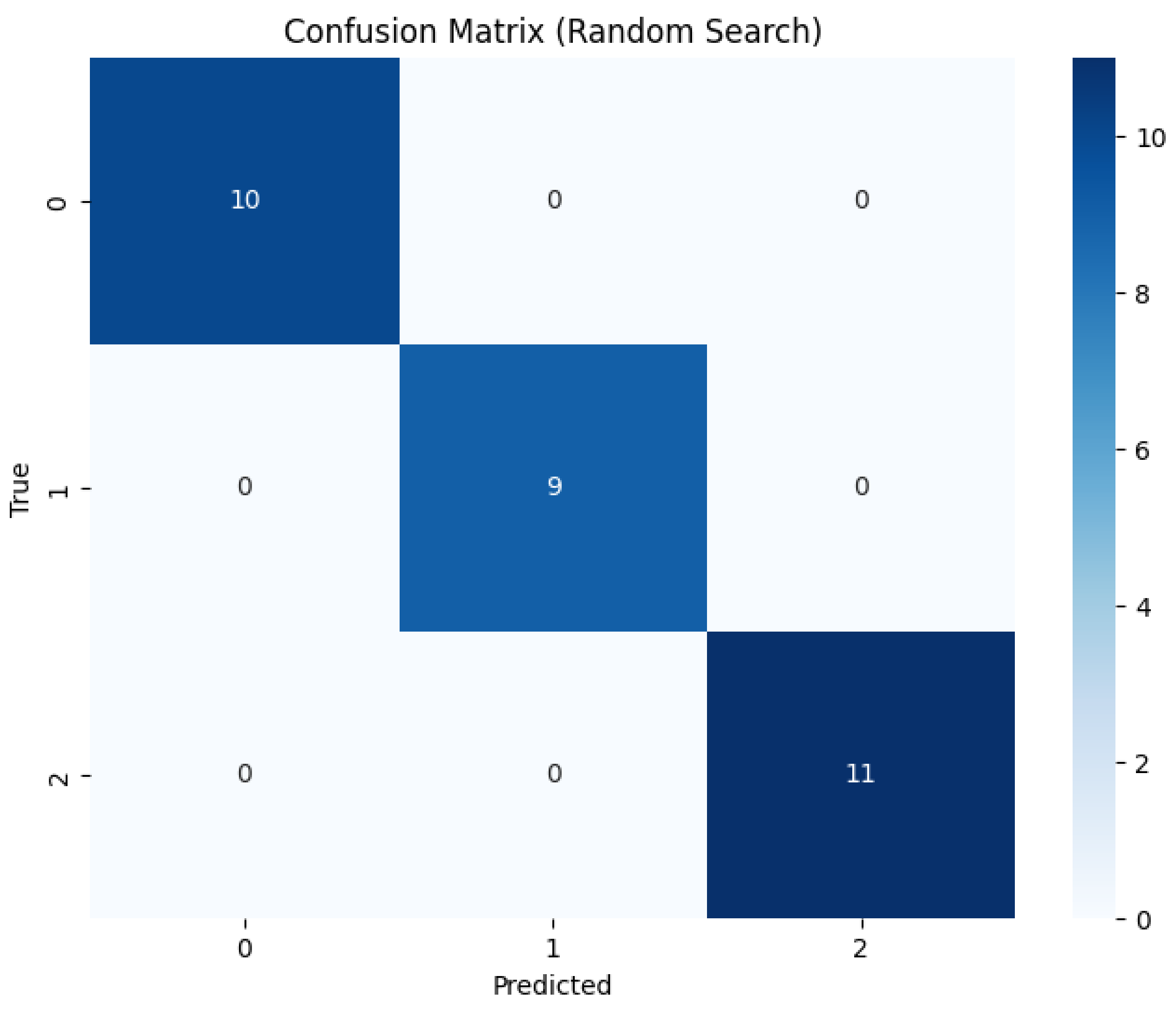
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix (Random Search)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Run the Python file

- Use the command below in your terminal to run the Python file:

```
python parameter_tuning.py
```

Output



References

- Scikit-learn documentation
- Cross Validation and Hyperparameter Tuning
- Grid Search
- Randomized Search