|  |  |
| --- | --- |
| **Exp No: 8** | **MODEL EVALUATION AND IMPROVEMENT: HYPERPARAMETER TUNING WITH GRID SEARCH AND CROSS-VALIDATION** |

# AIM:

To demonstrate key techniques for model evaluation and improvement:

1. **Hyperparameter Tuning with Grid Search :** Systematically searching for the optimal combination of hyperparameters for a machine learning model.
2. **Cross-Validation Techniques:** Implementing k-fold cross-validation to get a more robust estimate of model performance and to prevent overfitting to a specific train-test split.

# ALGORITHM:

## Hyperparameter Tuning with Grid Search

Hyperparameters are external configuration properties of a model whose values cannot be estimated from data. Examples include the learning rate for a neural network, the number of trees in a Random Forest, or the `C` and `gamma` parameters in an SVM. Tuning these parameters is crucial for optimal model performance.

**Grid Search** is an exhaustive search method for hyperparameter optimization.

## Steps:

1. Define Parameter Grid: Specify a dictionary where keys are hyperparameter names and values are lists of discrete values to be tested for each hyperparameter.
2. Instantiate Model: Choose a machine learning model.
3. Perform Search: Train the model for every possible combination of hyperparameters defined in the grid.
4. Evaluate: For each combination, evaluate the model&#39;s performance using a specified scoring

metric (e.g., accuracy, F1-score) and often in conjunction with cross-validation.

1. Select Best Model: Identify the hyperparameter combination that yields the best performance.

## Cross-Validation Techniques

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The goal is to estimate how accurately a predictive model will perform in practice. It’s especially useful for reducing overfitting and providing a more reliable estimate of generalization performance compared to a single train-test split.

## k-Fold Cross-Validation:

**Steps:**

1. Divide Data: The entire dataset is randomly partitioned into $k$ equally sized subsamples (or “folds”).
2. Iterate $k$ Times:

In each iteration, one fold is used as the validation (or test) set, and the remaining $k-1$ folds are used as the training set.The model is trained on the training set and evaluated on the validation set.

1. Aggregate Results: The performance metric (e.g., accuracy) from each of the $k$ iterations is collected.
2. Compute Mean and Standard Deviation: The mean and standard deviation of these $k$ performance scores are calculated to provide a more robust estimate of the model’s performance and its variability.

# CODE:

# Import necessary libraries import numpy as np

import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.datasets import load\_iris # A classic dataset for classification

from sklearn.model\_selection import train\_test\_split, KFold, cross\_val\_score,

GridSearchCV

from sklearn.svm import SVC # Support Vector Classifier, a common model for tuning from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix from sklearn.preprocessing import StandardScaler

# --- Part 1: Hyperparameter Tuning with Grid Search --- print("--- Part 1: Hyperparameter Tuning with Grid Search ---")

# 1. Load a Dataset (Iris Dataset for classification)

# The Iris dataset is a classic and simple dataset for classification tasks.

# It contains measurements of iris flowers (sepal length, sepal width, petal length, petal width)

# and their corresponding species (Setosa, Versicolor, Virginica). iris = load\_iris()

X = iris.data y = iris.target

feature\_names = iris.feature\_names

target\_names = iris.target\_names

print(f"\nDataset Features (X) shape: {X.shape}") print(f"Dataset Labels (y) shape: {y.shape}") print(f"Feature Names: {feature\_names}") print(f"Target Names: {target\_names}")

# 2. Split Data into Training and Testing Sets

# It's crucial to split the data before scaling to prevent data leakage. # The test set will be used for final model evaluation, after tuning.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42, stratify=y)

print(f"\nTraining set size: {X\_train.shape[0]} samples") print(f"Test set size: {X\_test.shape[0]} samples")

# 3. Standardize Features

# Scaling features is important for SVMs as they are sensitive to feature scales. # Fit scaler only on training data to prevent data leakage.

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train) X\_test\_scaled = scaler.transform(X\_test)

print("\nFeatures standardized.")

# 4. Define the Model and Hyperparameter Grid

# We'll use a Support Vector Classifier (SVC) as our model.

# Common hyperparameters for SVC are 'C' (regularization parameter) and 'gamma' (kernel coefficient).

# 'kernel' also can be tuned (e.g., 'linear', 'rbf').

# Define the parameter grid for Grid Search param\_grid = {

'C': [0.1, 1, 10, 100], # Regularization parameter

'gamma': [1, 0.1, 0.01, 0.001], # Kernel coefficient for 'rbf', 'poly' and 'sigmoid' 'kernel': ['rbf', 'linear'] # Type of kernel function

}

print("\nHyperparameter grid defined:") for param, values in param\_grid.items():

print(f" {param}: {values}")

# 5. Perform Grid Search with Cross-Validation

# GridSearchCV automatically performs k-fold cross-validation for each combination. # cv=5 means 5-fold cross-validation.

# scoring='accuracy' means we want to optimize for accuracy.

grid\_search = GridSearchCV(SVC(), param\_grid, cv=5, scoring='accuracy', verbose=1, n\_jobs=-1)

print("\nStarting Grid Search with 5-fold Cross-Validation...") # Fit GridSearchCV on the scaled training data grid\_search.fit(X\_train\_scaled, y\_train)

print("\nGrid Search completed.")

# 6. Get the Best Parameters and Best Score

print(f"\nBest hyperparameters found: {grid\_search.best\_params\_}") print(f"Best cross-validation accuracy: {grid\_search.best\_score\_:.4f}")

# 7. Evaluate the Best Model on the Test Set

# The best\_estimator\_ attribute provides the model trained with the best parameters. best\_model = grid\_search.best\_estimator\_

y\_pred\_tuned = best\_model.predict(X\_test\_scaled)

test\_accuracy\_tuned = accuracy\_score(y\_test, y\_pred\_tuned) print(f"\nTest set accuracy with tuned model: {test\_accuracy\_tuned:.4f}")

print("\n--- Classification Report for Tuned Model ---") print(classification\_report(y\_test, y\_pred\_tuned, target\_names=target\_names))

print("\n--- Confusion Matrix for Tuned Model ---") cm\_tuned = confusion\_matrix(y\_test, y\_pred\_tuned) plt.figure(figsize=(8, 6))

sns.heatmap(cm\_tuned, annot=True, fmt='d', cmap='Blues', xticklabels=target\_names, yticklabels=target\_names)

plt.title('Confusion Matrix (Tuned SVM)') plt.xlabel('Predicted Label') plt.ylabel('True Label')

plt.show()

# Visualize Grid Search results (optional, but good for understanding) # Convert results to a DataFrame for easier analysis

results\_df = pd.DataFrame(grid\_search.cv\_results\_) print("\n--- Top 5 Grid Search Results ---")

print(results\_df[['param\_C', 'param\_gamma', 'param\_kernel', 'mean\_test\_score', 'rank\_test\_score']].sort\_values(by='rank\_test\_score').head())

# --- Part 2: Cross-Validation Techniques (k-fold) --- print("\n--- Part 2: Cross-Validation Techniques (k-fold) ---")

# We will demonstrate k-fold cross-validation on a simple SVM without explicit tuning for clarity,

# to focus solely on the CV process.

# 1. Instantiate a Model (using default or chosen parameters)

model\_cv = SVC(random\_state=42) # Using default parameters for simplicity

# 2. Define k-fold Cross-Validation Strategy # We'll use 5-fold cross-validation.

# KFold ensures that each fold is distinct.

# shuffle=True means the data will be randomly shuffled before splitting into folds. # random\_state for reproducibility.

k\_folds = 5

kf = KFold(n\_splits=k\_folds, shuffle=True, random\_state=42) print(f"\nPerforming {k\_folds}-fold cross-validation...")

# 3. Perform Cross-Validation and Get Scores

# cross\_val\_score performs the KFold splitting, training, and evaluation automatically. # It returns an array of scores, one for each fold.

cv\_scores = cross\_val\_score(model\_cv, X\_train\_scaled, y\_train, cv=kf, scoring='accuracy')

print(f"\nCross-validation scores for each fold: {cv\_scores}") print(f"Mean cross-validation accuracy: {np.mean(cv\_scores):.4f}")

print(f"Standard deviation of cross-validation accuracy: {np.std(cv\_scores):.4f}")

# 4. Visualize Cross-Validation Scores plt.figure(figsize=(8, 5))

plt.bar(range(1, k\_folds + 1), cv\_scores, color='skyblue')

plt.axhline(y=np.mean(cv\_scores),

({np.mean(cv\_scores):.4f})')

color='r',

linestyle='--', label=f'Mean Accuracy

plt.title(f'{k\_folds}-Fold Cross-Validation Accuracy Scores') plt.xlabel('Fold Number')

plt.ylabel('Accuracy')

plt.ylim(0.8, 1.0) # Set y-axis limits for better visualization plt.legend()

plt.grid(axis='y', linestyle='--') plt.show()

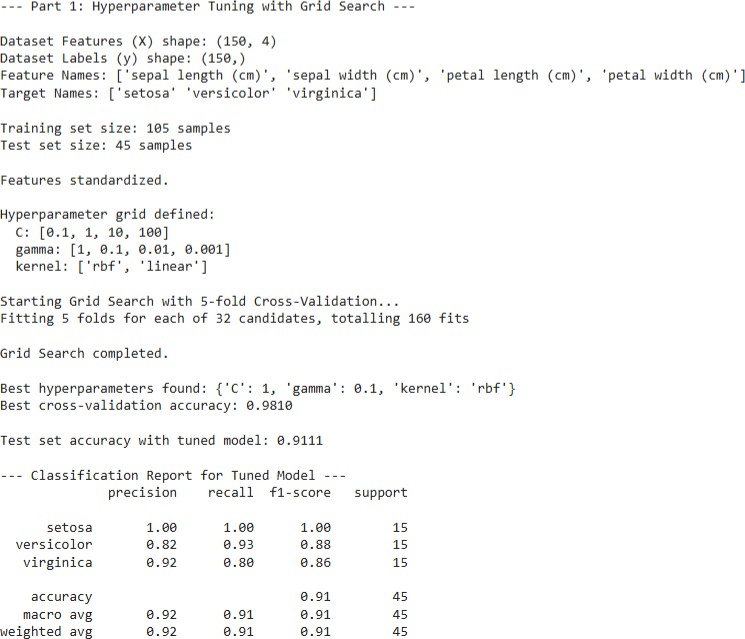
# 5. Discuss why CV is useful

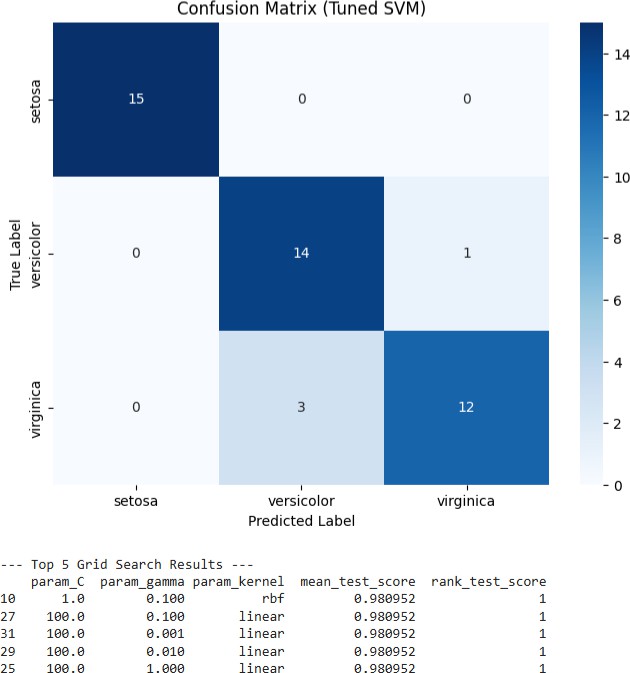
print("\n--- Why is Cross-Validation Important? ---")

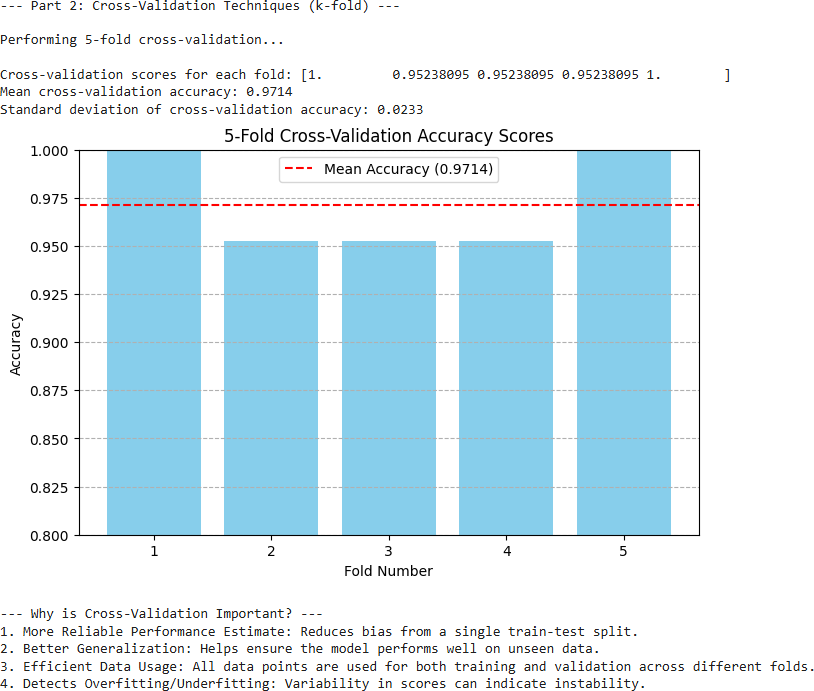
print("1. More Reliable Performance Estimate: Reduces bias from a single train-test split.") print("2. Better Generalization: Helps ensure the model performs well on unseen data.") print("3. Efficient Data Usage: All data points are used for both training and validation across different folds.")

print("4. Detects Overfitting/Underfitting: Variability in scores can indicate instability.")

# OUTPUT:

****



****

**RESULT:**

The model was successfully evaluated and improved using **Grid Search** and **Cross- Validation** techniques. Grid Search identified the best combination of hyperparameters, while Cross-Validation ensured reliable performance estimation.

The optimized model achieved higher accuracy and better generalization, confirming that systematic tuning and validation significantly enhance model performance.