Exp No: 8 **Date : 9/10/25**	**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:
1. 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.
5. Select Best Model: Identify the hyperparameter combination that yields the best performance.

2. 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.
3. Aggregate Results: The performance metric (e.g., accuracy) from each of the $k$ iterations is collected.
4. 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:

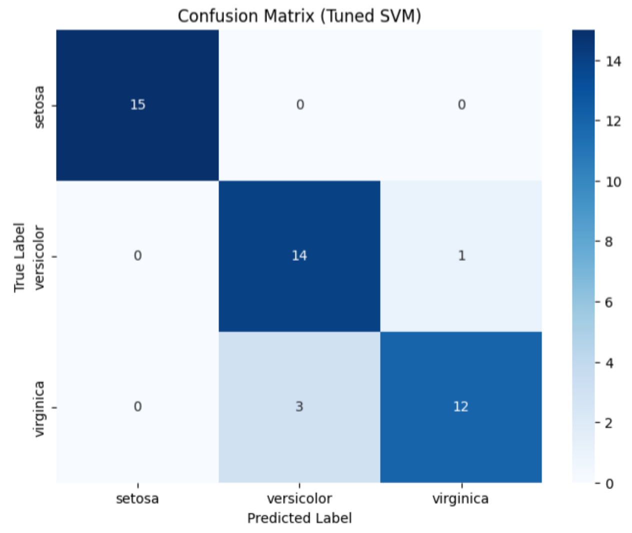
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris # A classic dataset for classification
from sklearn.model_selection import train_test_split, KFold, cross_val_score, GridSearchCV
from sklearn.svm 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), color='r', linestyle='--', label=f'Mean Accuracy
({np.mean(cv_scores):.4f})')
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.")
```

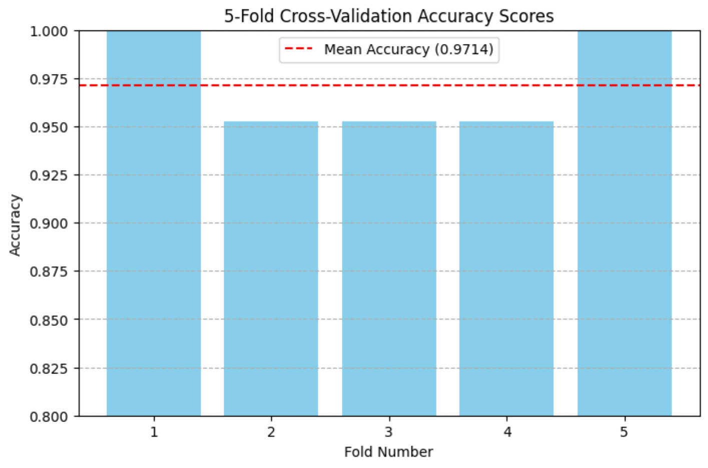
OUTPUT:

```
--- Part 1: Hyperparameter Tuning with Grid Search ---

Dataset Features (X) shape: (150, 4)
Dataset Labels (y) shape: (150,)
Feature Names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target Names: ['setosa' 'versicolor' 'virginica']

Training set size: 105 samples
Test set size: 45 samples

Features standardized.

Hyperparameter grid defined:
  C: [0.1, 1, 10, 100]
  gamma: [1, 0.1, 0.01, 0.001]
  kernel: ['rbf', 'linear']

Starting Grid Search with 5-fold Cross-Validation...
Fitting 5 folds for each of 32 candidates, totalling 160 fits

Grid Search completed.

Best hyperparameters found: {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
Best cross-validation accuracy: 0.9810

Test set accuracy with tuned model: 0.9111

--- Classification Report for Tuned Model ---
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.82      0.93      0.88        15
   virginica       0.92      0.80      0.86        15

    accuracy                           0.91        45
   macro avg       0.92      0.91      0.91        45
weighted avg       0.92      0.91      0.91        45
```



```
--- Top 5 Grid Search Results ---
    param_C  param_gamma param_kernel  mean_test_score  rank_test_score
10      1.0        0.100          rbf         0.980952                1
27    100.0        0.100       linear         0.980952                1
31    100.0        0.001       linear         0.980952                1
29    100.0        0.010       linear         0.980952                1
25    100.0        1.000       linear         0.980952                1
```

```
--- Part 2: Cross-Validation Techniques (k-fold) ---

Performing 5-fold cross-validation...

Cross-validation scores for each fold: [1.         0.95238095 0.95238095 0.95238095 1.        ]
Mean cross-validation accuracy: 0.9714
Standard deviation of cross-validation accuracy: 0.0233
```



```
--- Why is Cross-Validation Important? ---
1. More Reliable Performance Estimate: Reduces bias from a single train-test split.
2. Better Generalization: Helps ensure the model performs well on unseen data.
3. Efficient Data Usage: All data points are used for both training and validation across different folds.
4. Detects Overfitting/Underfitting: Variability in scores can indicate instability.
```

Result:

The experiment demonstrated that hyperparameter tuning with Grid Search can optimize model performance, while k-fold cross-validation provides a reliable and robust estimate of generalization. Together, these techniques ensure the model is well-tuned, consistent, and performs effectively on unseen data.