**PRACTICAL NO : 7**

**Model Evaluation and Hyperparameter Tuning**

**Aim:[A] Implement cross-validation techniques (k-fold, stratified, etc.) for robust model evaluation.**

**CODE:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

# ✅ Load the Iris dataset

from sklearn.datasets import load\_iris

iris = load\_iris()

X, y = iris.data, iris.target

# ✅ Split dataset into training & testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

# ✅ Initialize the model

model = RandomForestClassifier(random\_state=42)

# ✅ K-Fold Cross-Validation

kf = KFold(n\_splits=5, shuffle=True, random\_state=42)

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

print(f"K-Fold Cross-Validation Accuracy: {kf\_scores.mean():.4f} ± {kf\_scores.std():.4f}")

# ✅ Stratified K-Fold (ensures class balance in each fold)

skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

skf\_scores = cross\_val\_score(model, X, y, cv=skf, scoring='accuracy')

print(f"Stratified K-Fold Accuracy: {skf\_scores.mean():.4f} ± {skf\_scores.std():.4f}")

# ✅ Define hyperparameter grid

param\_grid = {

'n\_estimators': [50, 100, 150, 200],

'max\_depth': [3, 5, 7, None],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

# ✅ Grid Search CV

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

grid\_search.fit(X\_train, y\_train)

print(f"Best Parameters (Grid Search): {grid\_search.best\_params\_}")

print(f"Best Score (Grid Search): {grid\_search.best\_score\_:.4f}")

# ✅ Randomized Search CV (more efficient for large grids)

random\_search = RandomizedSearchCV(model, param\_grid, n\_iter=10, cv=5, scoring='accuracy', n\_jobs=-1, random\_state=42)

random\_search.fit(X\_train, y\_train)

print(f"Best Parameters (Randomized Search): {random\_search.best\_params\_}")

print(f"Best Score (Randomized Search): {random\_search.best\_score\_:.4f}")

# ✅ Train the final model using the best parameters

best\_model = grid\_search.best\_estimator\_

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

# ✅ Evaluate the final model

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Test Accuracy: {accuracy:.4f}")

print(classification\_report(y\_test, y\_pred, target\_names=iris.target\_names))

# ✅ Convert Grid Search results to DataFrame

results = pd.DataFrame(grid\_search.cv\_results\_)

# ✅ Extract relevant columns and make a copy

heatmap\_data = results[['param\_n\_estimators', 'param\_max\_depth', 'mean\_test\_score']].copy()

# ✅ Convert parameter values safely

heatmap\_data.loc[:, 'param\_n\_estimators'] = heatmap\_data['param\_n\_estimators'].astype(int)

heatmap\_data.loc[:, 'param\_max\_depth'] = heatmap\_data['param\_max\_depth'].astype(str) # Keep as category

# ✅ Group by to handle duplicates (take mean if needed)

heatmap\_pivot = heatmap\_data.groupby(['param\_n\_estimators', 'param\_max\_depth']).mean().reset\_index()

# ✅ Pivot the table for heatmap

heatmap\_pivot = heatmap\_pivot.pivot(index='param\_n\_estimators', columns='param\_max\_depth', values='mean\_test\_score')

# ✅ Plot the heatmap

plt.figure(figsize=(8, 5))

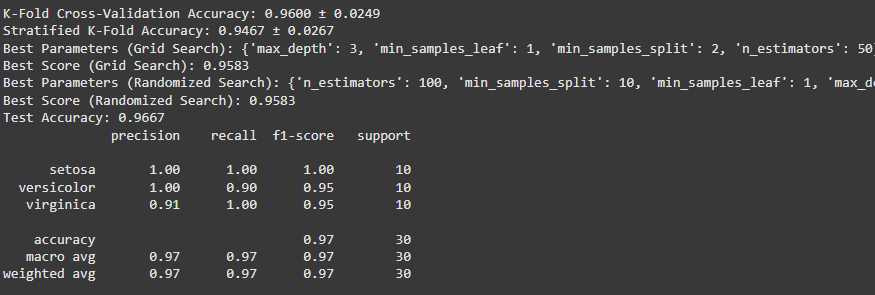
sns.heatmap(heatmap\_pivot, annot=True, cmap="coolwarm")

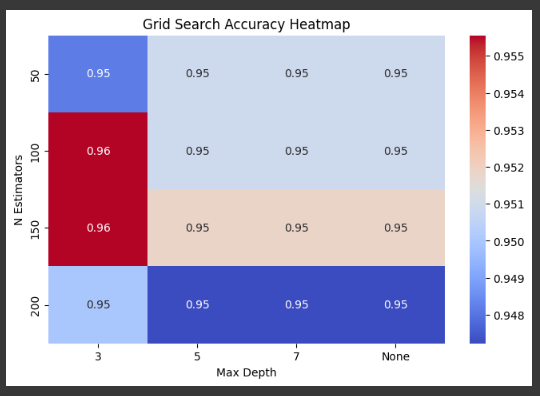
plt.title("Grid Search Accuracy Heatmap")

plt.xlabel("Max Depth")

plt.ylabel("N Estimators")

plt.show()





**Aim:[B] Systematically explore combinations of hyper parameters to optimize model performance.(use grid and randomized search)**

**CODE:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split, GridSearchCV, RandomizedSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import load\_iris

from sklearn.metrics import accuracy\_score, classification\_report

# ✅ Load dataset

iris = load\_iris()

X, y = iris.data, iris.target

# ✅ Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

# ✅ Define hyperparameter grid

param\_grid = {

'n\_estimators': [50, 100, 150, 200],

'max\_depth': [3, 5, 7, None],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

grid\_search = GridSearchCV(RandomForestClassifier(random\_state=42), param\_grid, cv=5, scoring='accuracy', n\_jobs=-1)

grid\_search.fit(X\_train, y\_train)

print(f"🔹 Best Parameters (Grid Search): {grid\_search.best\_params\_}")

print(f"🔹 Best Score (Grid Search): {grid\_search.best\_score\_:.4f}")

random\_search = RandomizedSearchCV(RandomForestClassifier(random\_state=42), param\_grid, n\_iter=10, cv=5, scoring='accuracy', n\_jobs=-1, random\_state=42)

random\_search.fit(X\_train, y\_train)

print(f"🔹 Best Parameters (Randomized Search): {random\_search.best\_params\_}")

print(f"🔹 Best Score (Randomized Search): {random\_search.best\_score\_:.4f}")

best\_model = grid\_search.best\_estimator\_

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

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"✅ Test Accuracy: {accuracy:.4f}")

print(classification\_report(y\_test, y\_pred, target\_names=iris.target\_names))

results = pd.DataFrame(grid\_search.cv\_results\_)

# ✅ Extract relevant columns

heatmap\_data = results[['param\_n\_estimators', 'param\_max\_depth', 'mean\_test\_score']].copy()

# ✅ Convert parameter values safely

heatmap\_data.loc[:, 'param\_n\_estimators'] = heatmap\_data['param\_n\_estimators'].astype(int)

heatmap\_data.loc[:, 'param\_max\_depth'] = heatmap\_data['param\_max\_depth'].astype(str)

# ✅ Handle duplicates by averaging

heatmap\_pivot = heatmap\_data.groupby(['param\_n\_estimators', 'param\_max\_depth']).mean().reset\_index()

heatmap\_pivot = heatmap\_pivot.pivot(index='param\_n\_estimators', columns='param\_max\_depth', values='mean\_test\_score')

# ✅ Plot the heatmap

plt.figure(figsize=(8, 5))

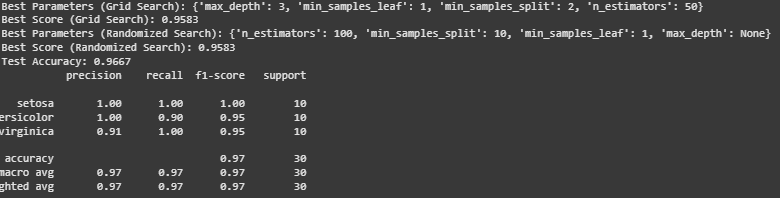
sns.heatmap(heatmap\_pivot, annot=True, cmap="coolwarm")

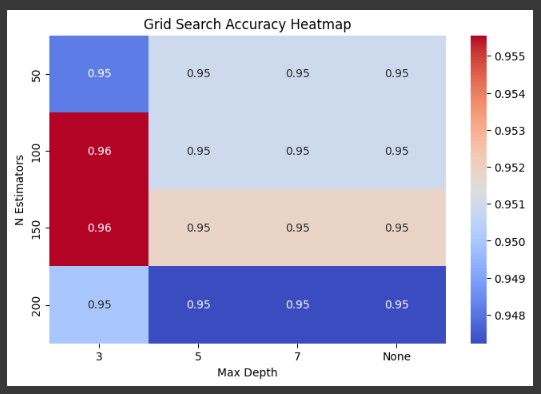
plt.title("Grid Search Accuracy Heatmap")

plt.xlabel("Max Depth")

plt.ylabel("N Estimators")

plt.show()





**PRACTICAL NO : 8**

**Bayesian Learning**

**Aim:[A] Implement Bayesian Learning using inferences.**

**CODE:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.datasets import load\_iris

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# ✅ Load dataset

iris = load\_iris()

X, y = iris.data, iris.target

# ✅ Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

# ✅ Train Naïve Bayes model (Bayesian Inference)

model = GaussianNB()

model.fit(X\_train, y\_train)

# ✅ Make Predictions

y\_pred = model.predict(X\_test)

# ✅ Model Evaluation

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"✅ Accuracy: {accuracy:.4f}")

print("\n🔹 Classification Report:")

print(classification\_report(y\_test, y\_pred, target\_names=iris.target\_names))

# ✅ Confusion Matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_matrix, annot=True, cmap="Blues", fmt="d", xticklabels=iris.target\_names, yticklabels=iris.target\_names)

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.title("Confusion Matrix for Naïve Bayes")

plt.show()

# ✅ Posterior Probabilities for Inference

sample = X\_test[0].reshape(1, -1) # Taking one sample for inference

probs = model.predict\_proba(sample)

print(f"🔹 Posterior Probabilities for Sample {sample}:")

for class\_label, prob in zip(iris.target\_names, probs[0]):

print(f" {class\_label}: {prob:.4f}")

