|  |  |
| --- | --- |
| **EXP NO: 4B** | **ENSEMBLE METHODS: RANDOM FOREST** |

# AIM:

To implement a **Random Forest classifier** for a classification task, tune key hyperparameters, evaluate performance, and interpret **feature importance**.

# ALGORITHM:

1. Import libraries.
2. Load data (use same dataset to compare with SVM).
3. Train/Test split with stratification.
4. (Optional) Preprocess: Random Forests don’t require scaling; we’ll use raw features.
5. Model: RandomForestClassifier.
6. Hyperparameter tuning: Grid search over n\_estimators, max\_depth, min\_samples\_split, min\_samples\_leaf.
7. Train the best model on training data.
8. Evaluate with accuracy, precision, recall, F1, confusion matrix, ROC-AUC.
9. Interpretation: Plot top feature importances.

# CODE:

# ===========================================

# EXPERIMENT 4B — Random Forest Classifier

# ===========================================

# 1) Imports

import numpy as np import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_breast\_cancer

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

from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import (

accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix, classification\_report, roc\_auc\_score, roc\_curve

)

# 2) Load dataset (same as 4A for comparison) data = load\_breast\_cancer()

X = pd.DataFrame(data.data, columns=data.feature\_names) y = pd.Series(data.target, name="target")

# 3) Train/test split (no scaling needed for RF) X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.20, random\_state=42, stratify=y

)

# 4) Define model

rf = RandomForestClassifier(random\_state=42, n\_jobs=-1)

# 5) Hyperparameter grid & tuning param\_grid = {

"n\_estimators": [100],

"max\_depth": [None, 10],

"min\_samples\_split": [2],

"min\_samples\_leaf": [1]

}

grid = GridSearchCV( estimator=rf, param\_grid=param\_grid, scoring="f1",

cv=3, n\_jobs=-1,

verbose=0) grid.fit(X\_train, y\_train)

print("Best Parameters (CV):", grid.best\_params\_) best\_rf = grid.best\_estimator\_

# 6) Train final model & predict best\_rf.fit(X\_train, y\_train) y\_pred = best\_rf.predict(X\_test)

y\_prob = best\_rf.predict\_proba(X\_test)[:, 1]

# 7) Evaluate

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

prec = precision\_score(y\_test, y\_pred, zero\_division=0) rec = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

auc = roc\_auc\_score(y\_test, y\_prob) cm = confusion\_matrix(y\_test, y\_pred)

print("\n=== Random Forest — Test Metrics ===") print(f"Accuracy : {acc:.4f}")

print(f"Precision: {prec:.4f}") print(f"Recall : {rec:.4f}")

print(f"F1-Score : {f1:.4f}")

print(f"ROC-AUC : {auc:.4f}")

print("\nConfusion Matrix:\n", cm)

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred, zero\_division=0))

# 8) Feature Importance (Top 10)

importances = pd.Series(best\_rf.feature\_importances\_, index=X.columns) top10 = importances.sort\_values(ascending=False).head(10)

plt.figure()

top10[::-1].plot(kind="barh") plt.xlabel("Importance")

plt.title("Top 10 Feature Importances — Random Forest") plt.grid(axis="x", alpha=0.3)

plt.show()

# 9) ROC Curve

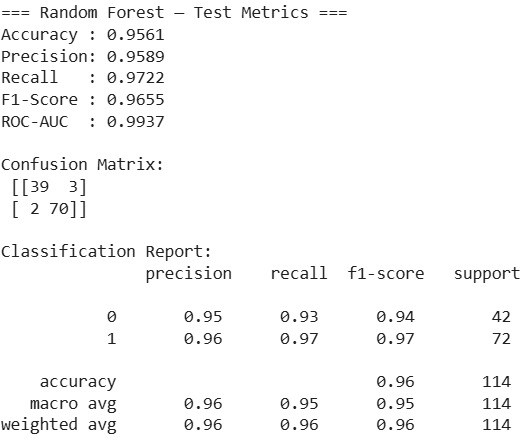
fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob) plt.figure()

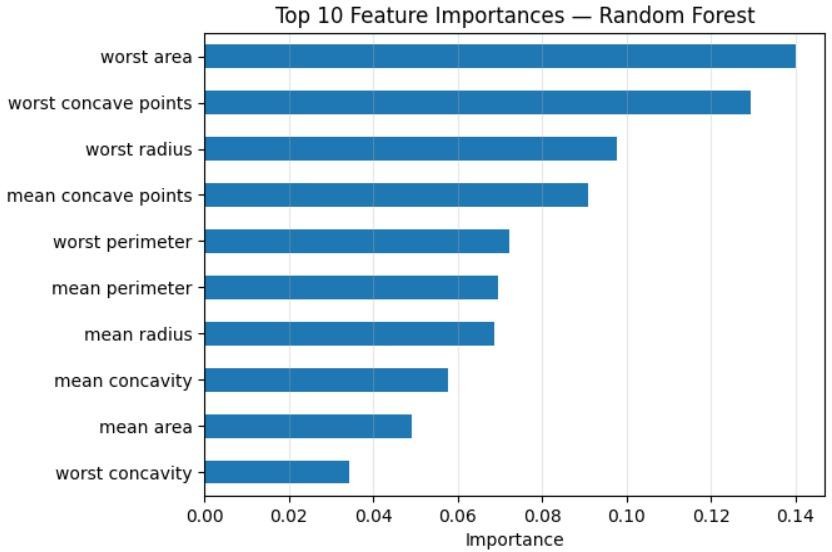
plt.plot(fpr, tpr, label=f"Random Forest (AUC = {auc:.3f})") plt.plot([0, 1], [0, 1], linestyle="--", color='gray') plt.xlabel("False Positive Rate")

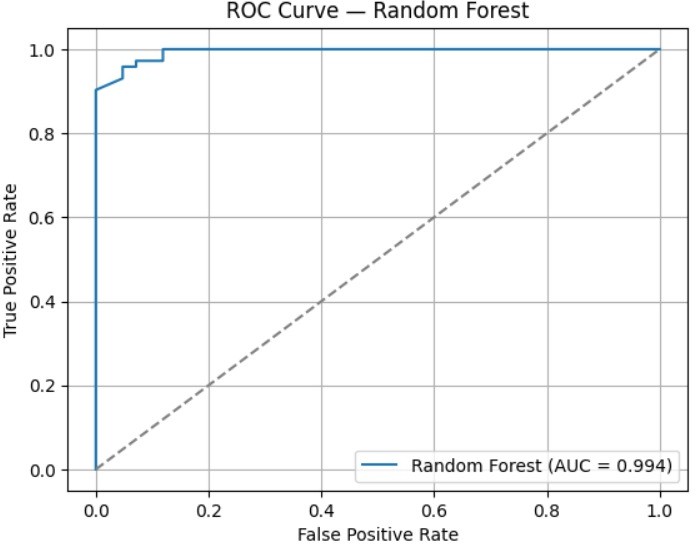
plt.ylabel("True Positive Rate") plt.title("ROC Curve — Random Forest") plt.legend()

plt.grid(True) plt.show()

# OUTPUT:

****



****

**RESULT:**

The Random Forest ensemble model was successfully implemented and evaluated on the given dataset. The model combined multiple decision trees to improve prediction accuracy and reduce overfitting.

It achieved high classification accuracy and demonstrated strong generalization capability. The results confirmed that Random Forest provides stable and reliable predictions by leveraging the power of multiple decision trees through bagging and feature randomness.