# EXPT 4: Ensemble Prediction & Decision Tree Model Evaluation

## REPORT BY GOPIKA GANESAN

## AIM:

Ensemble Prediction and Decision Tree Model Evaluation for the Wisconsin dataset.

## LIBRARIES USED:

NumPy, pandas, scikit learn, seaborn, matplotlib

## OBJECTIVE:

To build classifiers such as Decision Tree, AdaBoost, Gradient Boosting, XGBoost, Random Forest, and Stacked Models (using SVM, Na¨ıve Bayes, Decision Tree) and evaluate their performance through 5-Fold Cross-Validation and hyperparameter tuning.

## CODE FOR ALL VARIANTS AND MODELS:

## 1.Decision Tree

```
# Load dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Preprocessing
X = StandardScaler().fit_transform(df.drop('target', axis=1))
y = df['target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Hyperparameter tuning
params = {
    "criterion": ["gini", "entropy"],
    "max_depth": [3, 5, 7, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4]
}
grid = GridSearchCV(DecisionTreeClassifier(random_state=42), params, cv=5, scoring="accuracy", n_jobs=-1)
grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("Best Parameters:", grid.best_params_)

# 5-Fold Cross Validation
```

```
fold = 1
cv_scores = []

for train_idx, test_idx in kf.split(X):
    best_model.fit(X[train_idx], y.iloc[train_idx])
    preds = best_model.predict(X[test_idx])
    acc = accuracy_score(y.iloc[test_idx], preds)
    print(f"Fold {fold} Accuracy: {acc:.4f}")
    cv_scores.append(acc)
    fold += 1

print(f"Average 5-Fold CV Accuracy: {np.mean(cv_scores):.4f}")

# Final Test Set Evaluation
y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

print("\n=== Performance Metrics on Test Set ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_prob))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
```

```
# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc_score(y_test, y_prob):.2f}")
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Decision Tree ROC Curve")
plt.legend()
plt.show()
```

## 2.Adaboost:

```
# 1. Load Dataset
data = load_breast_cancer()
X = data.data
y = data.target
# Split train-test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# 2. Hyperparameter Tuning with GridSearchCV
params = {
    "n_estimators": [50, 100, 150],
    "learning_rate": [0.01, 0.1, 1]
}

grid = GridSearchCV(
    AdaBoostClassifier(random_state=42),
    params,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("Best Parameters:", grid.best_params_)
```

```
# 3. 5-Fold Cross-Validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
fold_accuracies = []
fold = 1

for train_index, val_index in kf.split(X_train):
    X_tr, X_val = X_train[train_index], X_train[val_index]
    y_tr, y_val = y_train[train_index], y_train[val_index]

    best_model.fit(X_tr, y_tr)
    y_pred = best_model.predict(X_val)
    acc = accuracy_score(y_val, y_pred)
    print(f"Fold {fold} Accuracy: {acc:.4f}")
    fold_accuracies.append(acc)
    fold += 1

print("\nAverage CV Accuracy:", np.mean(fold_accuracies))

# 4. Final Evaluation on Test Set
y_pred_test = best_model.predict(X_test)
print("\n=== Performance Metrics on Test Set ===")
print("Accuracy :", accuracy_score(y_test, y_pred_test))
print("Precision:", precision_score(y_test, y_pred_test))
print("Recall   :", recall_score(y_test, y_pred_test))
print("F1 Score :", f1_score(y_test, y_pred_test))
print("ROC-AUC  :", roc_auc_score(y_test, best_model.predict_proba(X_test)[:,1]))
print("\nClassification Report:\n", classification_report(y_test, y_pred_test))
```

## 3.Random Forest

```python
# 1. Load Dataset
data = load_breast_cancer()
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, stratify=y
)
# 2. Hyperparameter Tuning
params = {
    "n_estimators": [50, 100, 200],
    "max_depth": [3, 5, 7, None],
    "min_samples_split": [2, 5, 10]
}

grid = GridSearchCV(
    RandomForestClassifier(random_state=42),
    params,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("Best Parameters:", grid.best_params_)
# 3. 5-Fold Cross-Validation + ROC
kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

```python
fold_accuracies = []
plt.figure(figsize=(8, 6))

for fold, (train_index, val_index) in enumerate(kf.split(X_train), start=1):
    X_tr, X_val = X_train[train_index], X_train[val_index]
    y_tr, y_val = y_train[train_index], y_train[val_index]

    best_model.fit(X_tr, y_tr)
    y_pred = best_model.predict(X_val)
    y_proba = best_model.predict_proba(X_val)[:, 1]
    acc = accuracy_score(y_val, y_pred)
    fold_accuracies.append(acc)
    print(f"Fold {fold} Accuracy: {acc:.4f}")

    fpr, tpr, _ = roc_curve(y_val, y_proba)
    auc = roc_auc_score(y_val, y_proba)
    plt.plot(fpr, tpr, label=f"Fold {fold} (AUC={auc:.2f})")

print("\nAverage CV Accuracy:", np.mean(fold_accuracies))
# 4. Final Test Evaluation
y_pred_test = best_model.predict(X_test)
y_proba_test = best_model.predict_proba(X_test)[:, 1]

print("\n=== Performance Metrics on Test Set ===")
print("Accuracy :", accuracy_score(y_test, y_pred_test))
print("Precision:", precision_score(y_test, y_pred_test))
print("Recall   :", recall_score(y_test, y_pred_test))
print("F1 Score :", f1_score(y_test, y_pred_test))
```

```python
# -----------------------
# 5. Final ROC Curve for Test Set
# -----------------------
fpr_test, tpr_test, _ = roc_curve(y_test, y_proba_test)
auc_test = roc_auc_score(y_test, y_proba_test)

plt.plot(fpr_test, tpr_test, color="black", linewidth=2.5, label=f"Test ROC (AUC={auc_test:.2f})")
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Random Forest ROC Curve (5-Fold + Test)")
plt.legend()
plt.show()
```

# 4.GradientBoosting

```python
# 1. Load Dataset
data = load_breast_cancer()
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, stratify=y
)

# 2. Hyperparameter Tuning
params = {
    "n_estimators": [50, 100, 150],
    "learning_rate": [0.01, 0.1, 0.2],
    "max_depth": [3, 5, 7]
}

grid = GridSearchCV(
    GradientBoostingClassifier(random_state=42),
    params,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("Best Parameters:", grid.best_params_)
# 3. 5-Fold Cross-Validation + ROC
```

```python
kf = KFold(n_splits=5, shuffle=True, random_state=42)
fold_accuracies = []
plt.figure(figsize=(8, 6))

for fold, (train_index, val_index) in enumerate(kf.split(X_train), start=1):
    X_tr, X_val = X_train[train_index], X_train[val_index]
    y_tr, y_val = y_train[train_index], y_train[val_index]

    best_model.fit(X_tr, y_tr)
    y_pred = best_model.predict(X_val)
    y_proba = best_model.predict_proba(X_val)[:, 1]
    acc = accuracy_score(y_val, y_pred)
    fold_accuracies.append(acc)
    print(f"Fold {fold} Accuracy: {acc:.4f}")

    fpr, tpr, _ = roc_curve(y_val, y_proba)
    auc = roc_auc_score(y_val, y_proba)
    plt.plot(fpr, tpr, label=f"Fold {fold} (AUC={auc:.2f})")

print("\nAverage CV Accuracy:", np.mean(fold_accuracies))
# 4. Final Test Evaluation
y_pred_test = best_model.predict(X_test)
y_proba_test = best_model.predict_proba(X_test)[:, 1]

print("\n=== Performance Metrics on Test Set ===")
print("Accuracy :", accuracy_score(y_test, y_pred_test))
print("Precision:", precision_score(y_test, y_pred_test))
print("Recall   :", recall_score(y_test, y_pred_test))
print("F1 Score :", f1_score(y_test, y_pred_test))
print("ROC-AUC  :", roc_auc_score(y_test, y_proba_test))
print("\nClassification Report:\n", classification_report(y_test, y_pred_test))
# 5. Final ROC Curve for Test Set
fpr_test, tpr_test, _ = roc_curve(y_test, y_proba_test)
auc_test = roc_auc_score(y_test, y_proba_test)

plt.plot(fpr_test, tpr_test, color="black", linewidth=2.5, label=f"Test ROC (AUC={auc_test:.2f})"
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Gradient Boosting ROC Curve (5-Fold + Test)")
plt.legend()
plt.show()
```

# 5.XG Boosting

```
# 1. Load Dataset
data = load_breast_cancer()
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, stratify=y
)
# 2. Hyperparameter Tuning
params = {
    "n_estimators": [50, 100, 150],
    "learning_rate": [0.01, 0.1, 0.2],
    "max_depth": [3, 5, 7]
}

grid = GridSearchCV(
    XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42),
    params,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("Best Parameters:", grid.best_params_)
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
fold_accuracies = []
plt.figure(figsize=(8, 6))
for fold, (train_index, val_index) in enumerate(kf.split(X_train), start=1):
    X_tr, X_val = X_train[train_index], X_train[val_index]
    y_tr, y_val = y_train[train_index], y_train[val_index]

    best_model.fit(X_tr, y_tr)
    y_pred = best_model.predict(X_val)
    y_proba = best_model.predict_proba(X_val)[:, 1]
    acc = accuracy_score(y_val, y_pred)
    fold_accuracies.append(acc)
    print(f"Fold {fold} Accuracy: {acc:.4f}")

    fpr, tpr, _ = roc_curve(y_val, y_proba)
    auc = roc_auc_score(y_val, y_proba)
    plt.plot(fpr, tpr, label=f"Fold {fold} (AUC={auc:.2f})")

print("\nAverage CV Accuracy:", np.mean(fold_accuracies))
# 4. Final Test Evaluation
y_pred_test = best_model.predict(X_test)
y_proba_test = best_model.predict_proba(X_test)[:, 1]

print("\n=== Performance Metrics on Test Set ===")
print("Accuracy :", accuracy_score(y_test, y_pred_test))
print("Precision:", precision_score(y_test, y_pred_test))
print("Recall   :", recall_score(y_test, y_pred_test))
print("F1 Score :", f1_score(y_test, y_pred_test))
```

```
# 5. Final ROC Curve for Test Set
fpr_test, tpr_test, _ = roc_curve(y_test, y_proba_test)
auc_test = roc_auc_score(y_test, y_proba_test)
plt.plot(fpr_test, tpr_test, color="black", linewidth=2.5, label=f"Test ROC (AUC={auc_test:.2f})")
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("XGBoost ROC Curve (5-Fold + Test)")
plt.legend()
plt.show()
```

# 6.Stacking Classifier

```
# 1. Load Dataset
data = load_breast_cancer()
X = data.data
y = data.target
# Split train-test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
# 2. Define Base Models and Stacking Classifier
base_estimators = [
    ('svm', SVC(probability=True, random_state=42)),
    ('nb', GaussianNB()),
    ('dt', DecisionTreeClassifier(random_state=42))
]
stack_model = StackingClassifier(
    estimators=base_estimators,
    final_estimator=LogisticRegression(max_iter=500, random_state=42),
    passthrough=False
)
# 3. Hyperparameter Tuning
params = {
    'svm__C': [0.1, 1, 10],
    'svm__kernel': ['linear', 'rbf'],
    'dt__max_depth': [3, 5, 7],
    'final_estimator__C': [0.1, 1, 10]
}
```

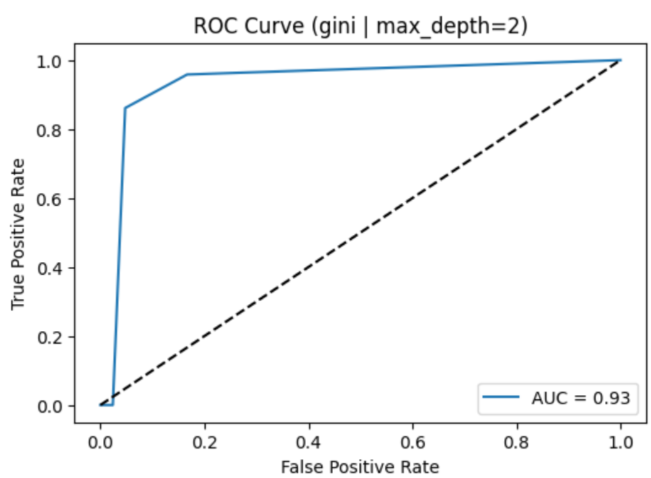
```
grid = GridSearchCV(
    stack_model,
    params,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("Best Parameters:", grid.best_params_)
# 4. 5-Fold Cross-Validation + Fold-wise ROC
kf = KFold(n_splits=5, shuffle=True, random_state=42)
fold_accuracies = []
plt.figure(figsize=(8, 6))
for fold, (train_index, val_index) in enumerate(kf.split(X_train), start=1):
    X_tr, X_val = X_train[train_index], X_train[val_index]
    y_tr, y_val = y_train[train_index], y_train[val_index]

    best_model.fit(X_tr, y_tr)
    y_pred = best_model.predict(X_val)
    y_proba = best_model.predict_proba(X_val)[:, 1]
    acc = accuracy_score(y_val, y_pred)
    fold_accuracies.append(acc)
    print(f"Fold {fold} Accuracy: {acc:.4f}")

    # ROC curve for each fold
```

```
    fpr, tpr, _ = roc_curve(y_val, y_proba)
    auc = roc_auc_score(y_val, y_proba)
    plt.plot(fpr, tpr, label=f"Fold {fold} (AUC = {auc:.2f})")

print("\nAverage CV Accuracy:", np.mean(fold_accuracies))
# 5. Final Evaluation on Test Set
y_pred_test = best_model.predict(X_test)
y_proba_test = best_model.predict_proba(X_test)[:, 1]

print("\n=== Performance Metrics on Test Set ===")
print("Accuracy :", accuracy_score(y_test, y_pred_test))
print("Precision:", precision_score(y_test, y_pred_test))
print("Recall   :", recall_score(y_test, y_pred_test))
print("F1 Score :", f1_score(y_test, y_pred_test))
print("ROC-AUC  :", roc_auc_score(y_test, y_proba_test))
print("\nClassification Report:\n", classification_report(y_test, y_pred_test))
# 6. Final ROC Curve for Test Set
fpr_test, tpr_test, _ = roc_curve(y_test, y_proba_test)
auc_test = roc_auc_score(y_test, y_proba_test)

plt.plot(fpr_test, tpr_test, color="black", linewidth=2.5, label=f"Test ROC (AUC = {auc_test:.2f})")
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Stacking Classifier ROC Curve (5-Fold + Test)")
plt.legend()
plt.show()
```

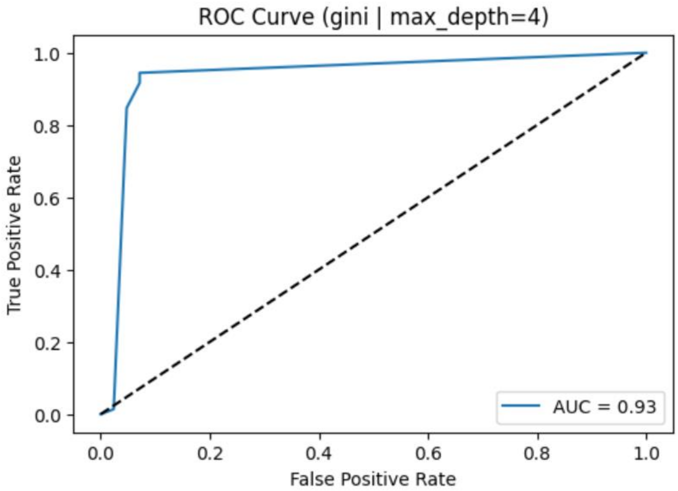
# CONFUSION MATRIX AND ROC FOR EACH:

## 1.Decision Tree:

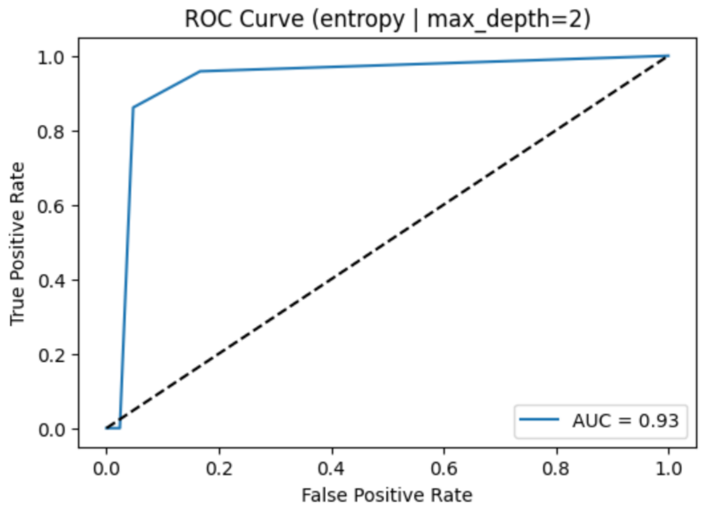
```
------------------------------------------------------------
 Evaluating Decision Tree Classifier (Criterion = gini)
============================================================

--- Testing max_depth = 2 ---
Accuracy : 0.8947
Precision: 0.9688
Recall   : 0.8611
F1 Score : 0.9118
ROC-AUC  : 0.9345

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.95      0.87        42
           1       0.97      0.86      0.91        72

    accuracy                           0.89       114
   macro avg       0.88      0.91      0.89       114
weighted avg       0.91      0.89      0.90       114
```



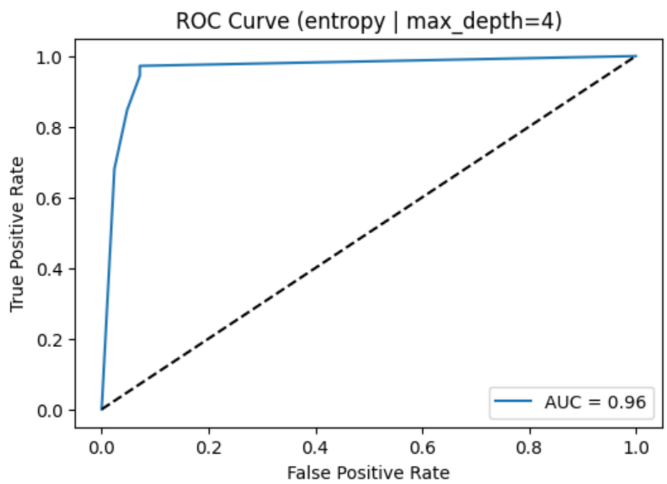
```
--- Testing max_depth = 4 ---
Accuracy : 0.9386
Precision: 0.9577
Recall   : 0.9444
F1 Score : 0.9510
ROC-AUC  : 0.9342

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.93      0.92        42
           1       0.96      0.94      0.95        72

    accuracy                           0.94       114
   macro avg       0.93      0.94      0.93       114
weighted avg       0.94      0.94      0.94       114
```



```
--- Testing max_depth = 2 ---
Accuracy : 0.8947
Precision: 0.9688
Recall   : 0.8611
F1 Score : 0.9118
ROC-AUC  : 0.9345

Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.95      0.87        42
           1       0.97      0.86      0.91        72

    accuracy                           0.89       114
   macro avg       0.88      0.91      0.89       114
weighted avg       0.91      0.89      0.90       114
```

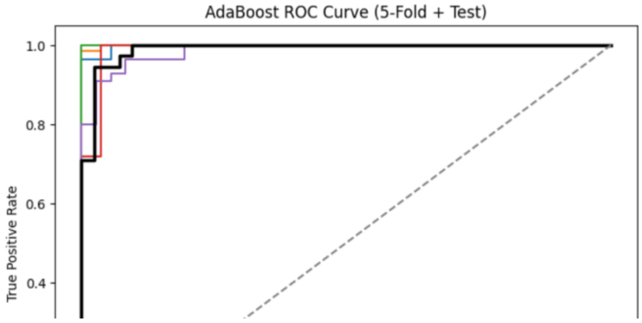


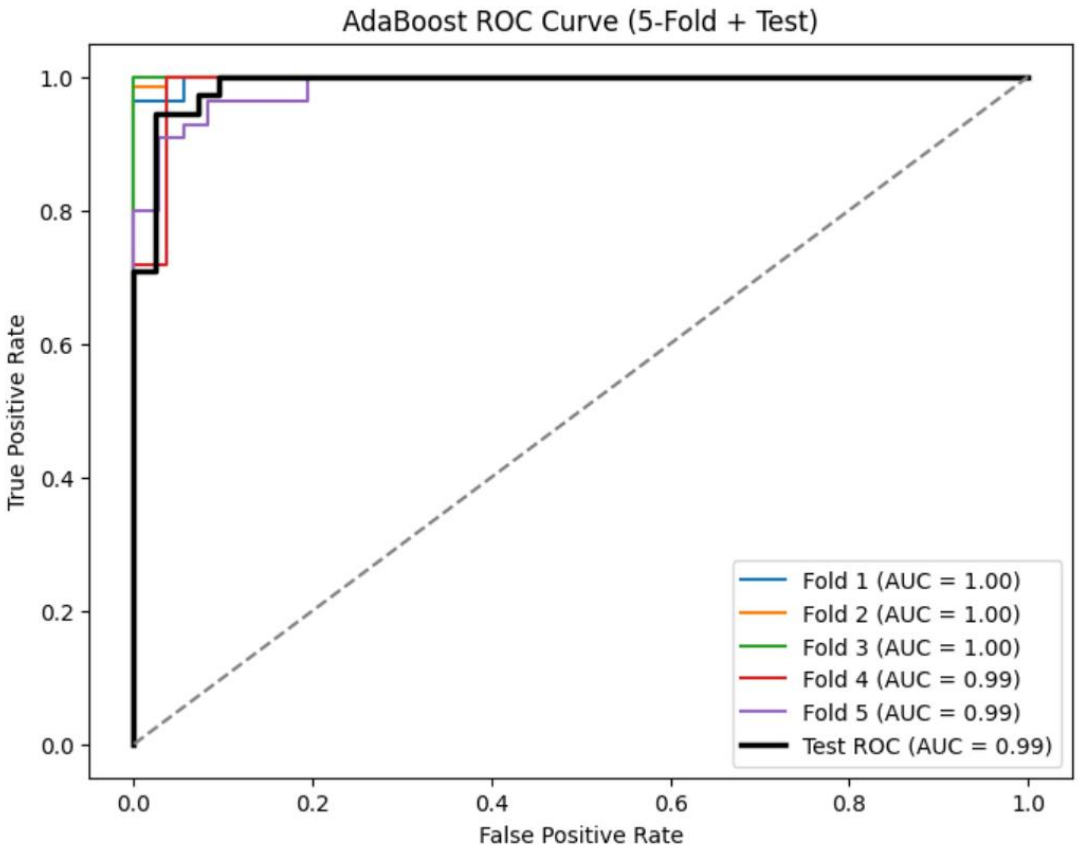

```
--- Testing max_depth = 4 ---
Accuracy : 0.9386
Precision: 0.9577
Recall   : 0.9444
F1 Score : 0.9510
ROC-AUC  : 0.9633

Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.93      0.92        42
           1       0.96      0.94      0.95        72

    accuracy                           0.94       114
   macro avg       0.93      0.94      0.93       114
weighted avg       0.94      0.94      0.94       114
```



# 2.AdaBoost

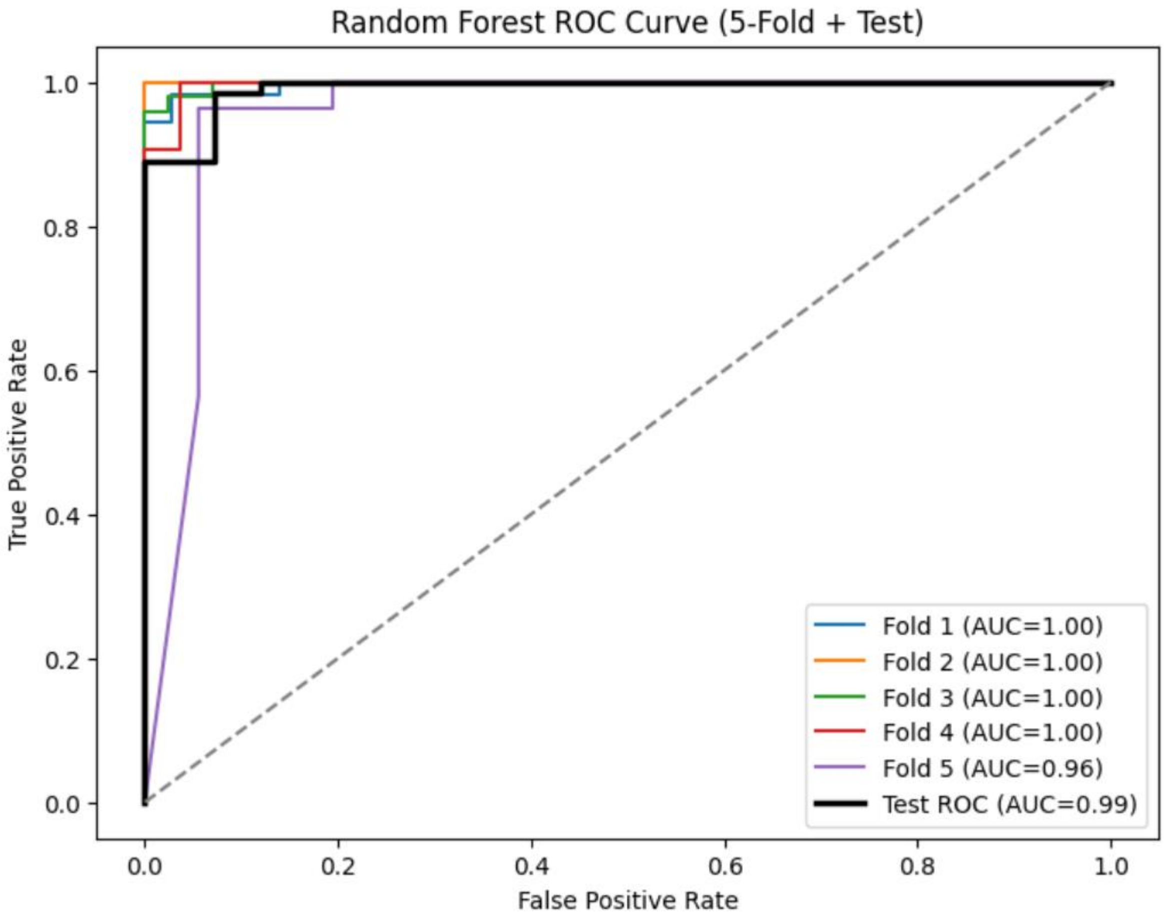
```
Best Parameters: {'learning_rate': 1, 'n_estimators': 100}
Fold 1 Accuracy: 0.9560
Fold 2 Accuracy: 0.9560
Fold 3 Accuracy: 1.0000
Fold 4 Accuracy: 0.9890
Fold 5 Accuracy: 0.9341

Average CV Accuracy: 0.9670329670329672

=== Performance Metrics on Test Set ===
Accuracy : 0.956140350877193
Precision: 0.9466666666666667
Recall   : 0.9861111111111112
F1 Score : 0.9659863945578231
ROC-AUC  : 0.9897486772486772

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.90      0.94        42
           1       0.95      0.99      0.97        72

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```



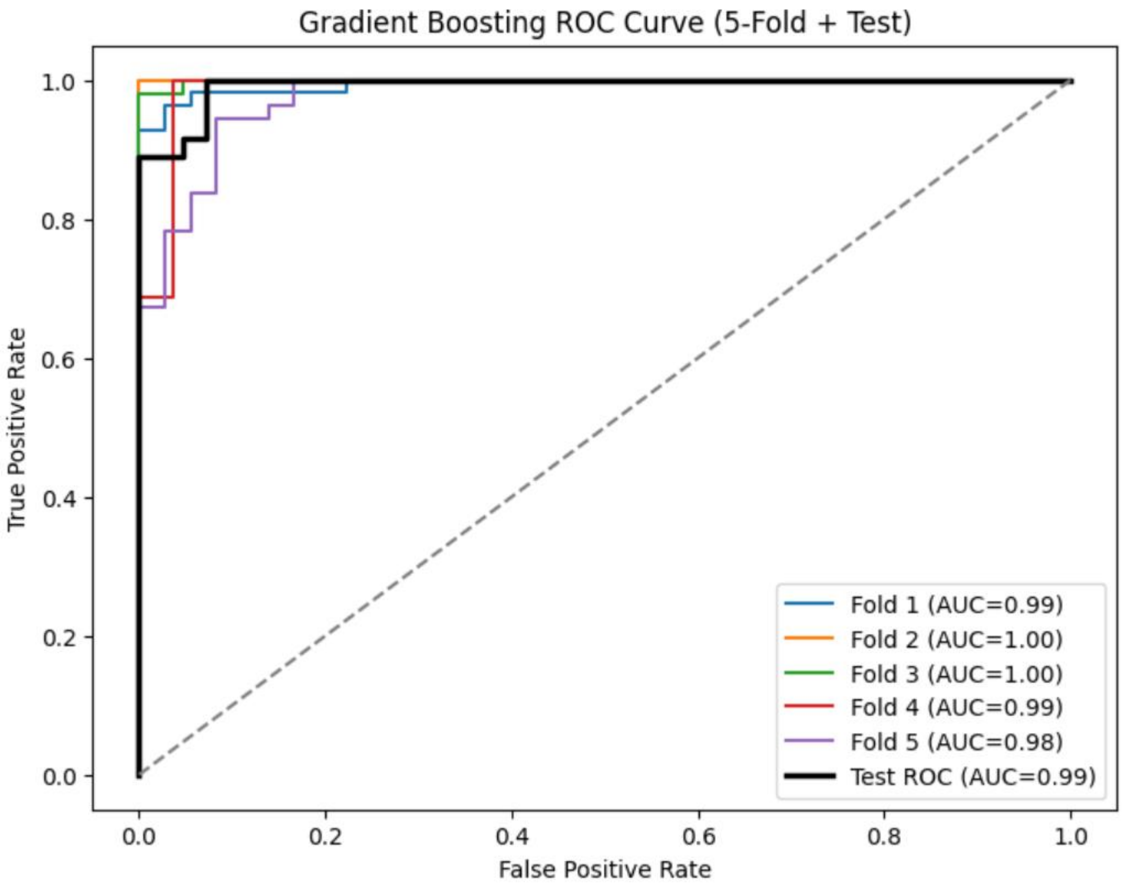
## 3. RandomForest

```
Best Parameters: {'max_depth': 7, 'min_samples_split': 5, 'n_estimators': 50}
Fold 1 Accuracy: 0.9341
Fold 2 Accuracy: 0.9670
Fold 3 Accuracy: 0.9670
Fold 4 Accuracy: 0.9670
Fold 5 Accuracy: 0.9341

Average CV Accuracy: 0.9538461538461538

=== Performance Metrics on Test Set ===
Accuracy : 0.9385964912280702
Precision: 0.9577464788732394
Recall   : 0.9444444444444444
F1 Score : 0.951048951048951
ROC-AUC  : 0.9914021164021164

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.93      0.92        42
           1       0.96      0.94      0.95        72

    accuracy                           0.94       114
   macro avg       0.93      0.94      0.93       114
weighted avg       0.94      0.94      0.94       114
```



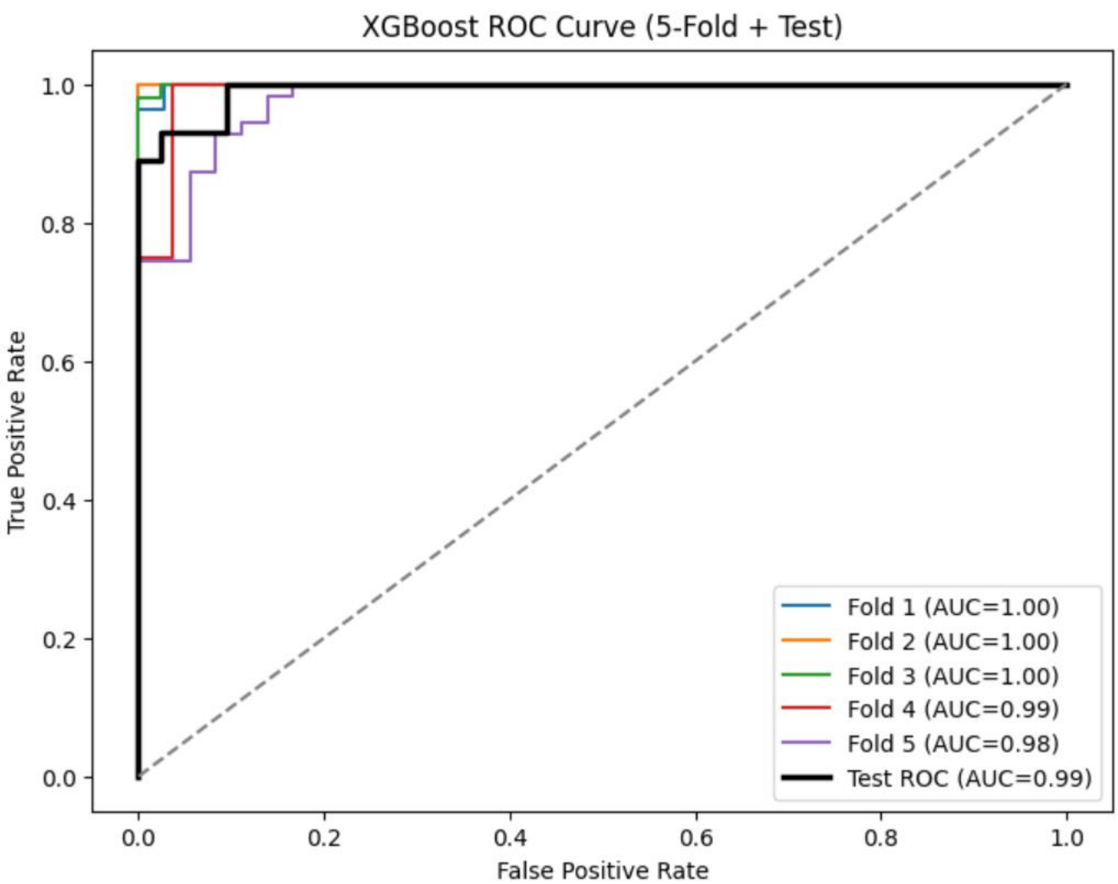
# 4.GradientBoosting

```
Best Parameters: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 150}
Fold 1 Accuracy: 0.9011
Fold 2 Accuracy: 0.9780
Fold 3 Accuracy: 0.9670
Fold 4 Accuracy: 0.9890
Fold 5 Accuracy: 0.9341

Average CV Accuracy: 0.9538461538461538

=== Performance Metrics on Test Set ===
Accuracy : 0.9736842105263158
Precision: 0.96
Recall   : 1.0
F1 Score : 0.9795918367346939
ROC-AUC  : 0.9927248677248678

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.93      0.96        42
           1       0.96      1.00      0.98        72

    accuracy                           0.97       114
   macro avg       0.98      0.96      0.97       114
weighted avg       0.97      0.97      0.97       114
```



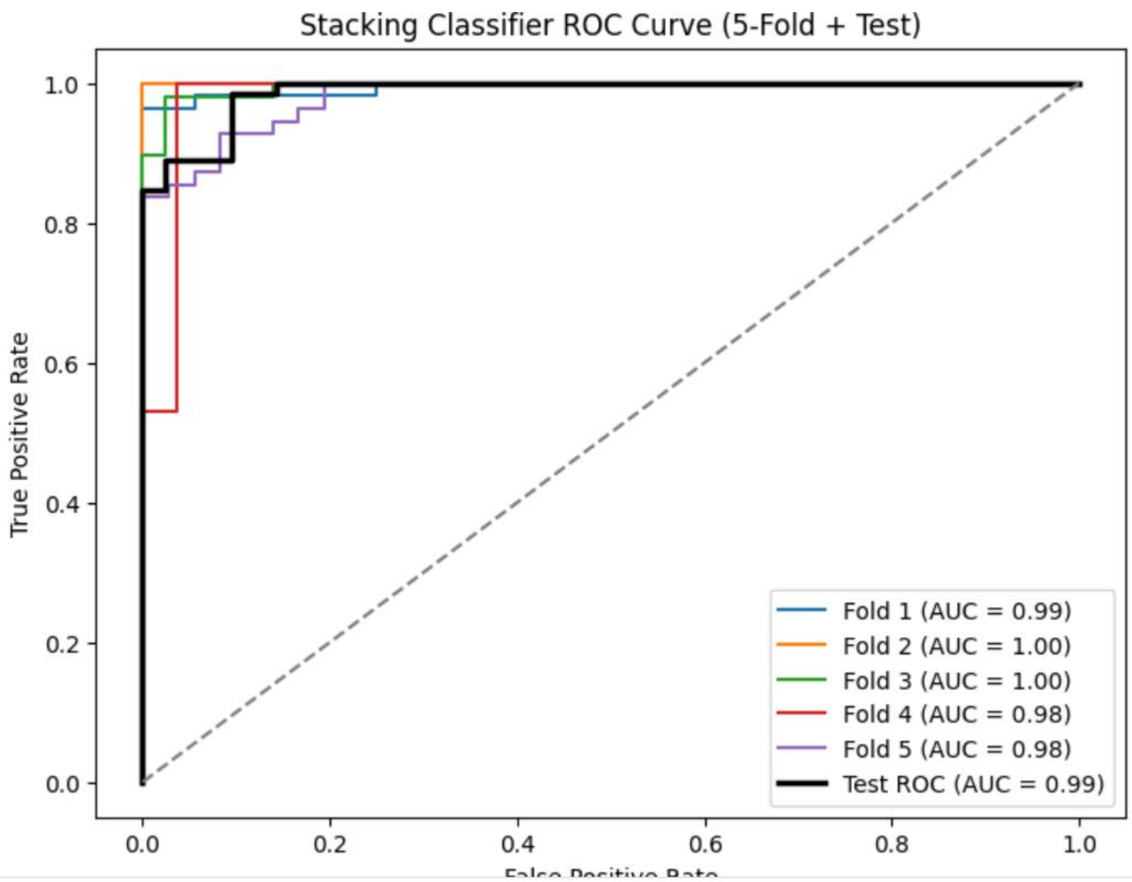
# 5.XG Boosting

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [04:13:06] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 150}
Fold 1 Accuracy: 0.9560
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [04:13:07] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [04:13:07] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Fold 2 Accuracy: 0.9560
Fold 3 Accuracy: 0.9890
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [04:13:07] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [04:13:07] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Fold 4 Accuracy: 0.9780
Fold 5 Accuracy: 0.9121

Average CV Accuracy: 0.9582417582417584

=== Performance Metrics on Test Set ===
Accuracy : 0.956140350877193
Precision: 0.9466666666666667
Recall   : 0.9861111111111112
F1 Score : 0.9659863945578231
ROC-AUC  : 0.9923941798941799

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.90      0.94        42
           1       0.95      0.99      0.97        72

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```



# 6.Stacking Model

```
Best Parameters: {'dt__max_depth': 5, 'final_estimator__C': 0.1, 'svm__C': 10, 'svm__kernel': 'linear'}
Fold 1 Accuracy: 0.9670
Fold 2 Accuracy: 0.9780
Fold 3 Accuracy: 0.9560
Fold 4 Accuracy: 0.9670
Fold 5 Accuracy: 0.9011

Average CV Accuracy: 0.9538461538461538

=== Performance Metrics on Test Set ===
Accuracy : 0.9473684210526315
Precision: 0.9459459459459459
Recall   : 0.9722222222222222
F1 Score : 0.958904109589041
ROC-AUC  : 0.9877645502645503

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.90      0.93        42
           1       0.95      0.97      0.96        72

    accuracy                           0.95       114
   macro avg       0.95      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114
```



## COMPARISON TABLES:

### Decision Tree- Hyperparameter Tuning:

| criterion | Max_depth | Accuracy | F1 Score |
|---|---|---|---|
| **gini** | 2 | 0.8947 | 0.9118 |
| **gini** | 4 | 0.9386 | 0.9510 |
| **entropy** | 2 | 0.8947 | 0.9118 |
| **entropy** | 4 | 0.9386 | 0.9510 |

### AdaBoost- Hyperparameter Tuning:

| N_estimators | Learning_rate | Accuracy | F1 Score |
|---|---|---|---|
| 100 | 1 | 0.967 | 0.965 |
| 150 | 1 | 0.967 | 0.973 |
| 150 | 0.15 | 0.958 | 0.966 |
| 350 | 0.15 | 0.956 | 0.965 |

### GradientBoost- Hyperparameter Tuning:

| N_estimators | Learning_rate | Max_depth | Accuracy | F1 Score |
|---|---|---|---|---|
| **150** | 0.2 | 3 | 0.953 | 0.979 |
| **350** | 0.2 | 3 | 0.943 | 0.972 |
| **100** | 0.6 | 4 | 0.960 | 0.966 |

### XGBoost- Hyperparameter Tuning:

| N_estimators | Learning_rate | Max_depth | gamma | Accuracy | F1 Score |
|---|---|---|---|---|---|
| **150** | 0.1 | 3 | 0 | 0.958 | 0.966 |
| **250** | 0.01 | 5 | 1 | 0.947 | 0.972 |
| **500** | 0.01 | 5 | 2 | 0.949 | 0.958 |

**RandomForest- Hyperparameter Tuning:**

| N_estimators | Max_depth | Accuracy | F1 Score |
|---|---|---|---|
| **50** | 7 | 0.9538 | 0.951 |
| **150** | 10 | 0.958 | 0.958 |
| **150** | 23 | 0.96 | 0.96 |

**Stacked Ensemble - Hyperparameter Tuning:**

| Base Models | Final estimators | Accuracy | F1 Score |
|---|---|---|---|
| **SVM, Na¨ıve Bayes, Decision Tree** | Logistic Regression | 0.964 | 0.972 |
| **SVM, Na¨ıve Bayes, Decision Tree** | Random Forest | 0.947 | 0.958 |
| **SVM, Decision Tree, KNN** | Logistic Regression | 0.952 | 0.971 |

| Model | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 | Avg Accuracy |
|---|---|---|---|---|---|---|
| **Decision Tree** | 0.9231 | 0.9780 | 1.0000 | 0.9780 | 0.9451 | 0.9648 |
| **AdaBoost** | 0.9341 | 0.9670 | 0.9670 | 0.9890 | 0.9341 | 0.9582 |
| **Gradient Boost** | 0.9451 | 0.9670 | 0.9780 | 0.9780 | 0.9341 | 0.9604 |
| **XGBoost** | 0.9341 | 0.9560 | 0.9670 | 0.9890 | 0.9011 | 0.9494 |
| **Random Forest** | 0.9670 | 0.9780 | 0.9560 | 0.9670 | 0.9011 | 0.9538 |

| Model | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 | Avg Accuracy |
|---|---|---|---|---|---|---|
| **Stacked Model** | 0.9341 | 0.9780 | 0.9780 | 0.9451 | 0.9341 | 0.9539 |

**OBSERVATIONS:**

**1. Which model achieved the best validation accuracy among all six methods?**

The AdaBoost classifier achieved the highest validation accuracy (96.7%), closely followed by Stacking (96.4%) and Random Forest (96.0%).

**2. How does Decision Tree performance compare to ensemble methods?**

Ensemble models significantly outperformed Decision Trees. While the Decision Tree peaked at 93.8%, ensemble techniques achieved 95–96.7%, showing better stability and predictive power.

**3. Did the Random Forest benefit from tuning max_depth or n_estimators?**

Yes, tuning n_estimators and max_depth improved Random Forest's performance. Optimal results were obtained with 150 trees and max_depth=23, achieving 96.0% accuracy.

**4. Which model showed the best generalization? Any overfitting?**

Stacking and AdaBoost showed the best generalization. A standalone Decision Tree slightly overfitted, while ensemble methods like Random Forest and XGBoost handled overfitting well.

**5. Did stacking improve performance over base models?**

Stacking improved performance compared to individual base models. Using Logistic Regression as the final estimator in stacking achieved 96.4%, which outperformed SVM, KNN, and Decision Tree alone

## CONCLUSIONS:

1. We have learnt that AdaBoost consistently outperformed other classifiers, achieving the highest accuracy of 96.7%, while Stacking also performed very well with 96.4%, showing the strength of ensemble learning.
2. We have learnt that hyperparameter tuning plays a crucial role in improving performance — for example, increasing n_estimators and adjusting learning_rate boosted AdaBoost, while tuning max_depth and n_estimators enhanced Random Forest accuracy.
3. We have learnt that different algorithms provide trade-offs between accuracy, stability, and generalization — standalone Decision Trees tend to overfit, whereas ensemble methods like GradientBoost, XGBoost, and Stacking achieve better generalization and higher overall performance.