ICS1512 – Machine Learning Algorithms Laboratory

Experiment 4: Ensemble Prediction and Decision Tree Model Evaluation

Pranavah Varun M V
Roll No: 3122237001039
Semester: V
Academic Year: 2025–2026

1. Aim and Objective

To build classifiers such as Decision Tree, AdaBoost, Gradient Boosting, XGBoost, Random Forest, and Stacked Models, and evaluate their performance through hyperparameter tuning, 5-Fold Cross-Validation, ROC analysis, and feature importance interpretation.

2. Libraries Used

- NumPy
- Pandas
- Matplotlib
- Seaborn
- Scikit-learn
- XGBoost

3. Code for All Variants and Models

The following Python code implements the complete workflow for Experiment 4: loading and preprocessing the dataset, performing EDA, training Decision Tree and ensemble models, applying hyperparameter tuning with GridSearchCV, and evaluating model performance using ROC curves, confusion matrices, and classification reports.

```python
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV,
    cross_val_score
```

```python
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
import seaborn as sns

# --- Load dataset ---
file_path = "/content/drive/MyDrive/wdbc.data"
columns = ["ID", "Diagnosis"] + [f"feature_{i}" for i in range(1, 31)]
df = pd.read_csv(file_path, header=None, names=columns)
df = df.drop("ID", axis=1)

# Encode target labels (M=1 malignant, B=0 benign)
label_encoder = LabelEncoder()
df["Diagnosis"] = label_encoder.fit_transform(df["Diagnosis"])

# Features & target
X = df.drop("Diagnosis", axis=1)
y = df["Diagnosis"]

# Handle missing values & standardize
imputer = SimpleImputer(strategy="mean")
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("    Dataset loaded successfully!")
print("Shape of X:", X.shape)
print("Class distribution:\n", y.value_counts())

# --- EDA ---
sns.countplot(x=y)
plt.title("Class Balance (0 = Malignant, 1 = Benign)")
plt.show()

plt.figure(figsize=(10,8))
sns.heatmap(pd.DataFrame(X_scaled, columns=X.columns).corr(),
            cmap="coolwarm", cbar=False)
plt.title("Feature Correlation Heatmap")
plt.show()

# --- Split dataset ---
X_train_valid, X_test, y_train_valid, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y)
X_train, X_valid, y_train, y_valid = train_test_split(
    X_train_valid, y_train_valid, test_size=0.25,
    random_state=42, stratify=y_train_valid)
```

```python

# --- Models ---
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier, RandomForestClassifier, StackingClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression

models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "AdaBoost": AdaBoostClassifier(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "XGBoost": XGBClassifier(eval_metric="logloss", random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Stacking": StackingClassifier(
        estimators=[("svm", SVC(probability=True, random_state=42)),
                    ("nb", GaussianNB()),
                    ("dt", DecisionTreeClassifier(random_state=42))],
        final_estimator=LogisticRegression()
    )
}

# --- Hyperparameter grids ---
param_grids = {
    "Decision Tree": {"criterion": ["gini", "entropy"],
                      "max_depth": [3, 5, 10, None],
                      "min_samples_split": [2, 5, 10],
                      "min_samples_leaf": [1, 2, 4]},
    "AdaBoost": {"n_estimators": [50, 100, 200],
                 "learning_rate": [0.01, 0.1, 1.0]},
    "Gradient Boosting": {"n_estimators": [100, 200],
                          "learning_rate": [0.05, 0.1],
                          "max_depth": [3, 5],
                          "subsample": [0.8, 1.0]},
    "XGBoost": {"n_estimators": [100, 200],
                "learning_rate": [0.05, 0.1],
                "max_depth": [3, 5],
                "gamma": [0, 0.1],
                "subsample": [0.8, 1.0],
                "colsample_bytree": [0.8, 1.0]},
    "Random Forest": {"n_estimators": [100, 200],
```

```
                        "max_depth": [None, 5, 10],
                        "criterion": ["gini", "entropy"],
                        "max_features": ["sqrt", "log2"],
                        "min_samples_split": [2, 5]},
    "Stacking": {"final_estimator": [LogisticRegression(),
                                     RandomForestClassifier(
                                        n_estimators=100)]}
}

# --- GridSearchCV tuning ---
best_models = {}
for name, model in models.items():
    print(f"Tuning {name}...")
    grid = GridSearchCV(model, param_grids[name], cv=5,
                        scoring="accuracy", n_jobs=-1)
    grid.fit(X_valid, y_valid)
    best_models[name] = grid.best_estimator_
    print("Best Params:", grid.best_params_)

# --- ROC & Evaluation ---
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score,
    confusion_matrix, classification_report, roc_curve

plt.figure(figsize=(8,6))
for name, model in best_models.items():
    model.fit(X_train_valid, y_train_valid)
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:,1]

    print(f"\n{name} | Acc={accuracy_score(y_test,y_pred):.4f}, "
          f"F1={f1_score(y_test,y_pred):.4f}, "
          f"AUC={roc_auc_score(y_test,y_proba):.4f}")
    print("Confusion Matrix:\n", confusion_matrix(y_test,y_pred))
    print("Report:\n", classification_report(y_test,y_pred))

    fpr, tpr, _ = roc_curve(y_test, y_proba)
    plt.plot(fpr, tpr, label=f"{name}")

plt.plot([0,1],[0,1],"k--")
plt.title("ROC Curves")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()

from sklearn.model_selection import cross_val_score, StratifiedKFold
import numpy as np
```

```python

# Define 5-fold CV (on training set only)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

results = {}

for name, model in best_models.items():
    acc_scores = cross_val_score(model, X_train, y_train, cv=cv,
        scoring="accuracy")
    f1_scores = cross_val_score(model, X_train, y_train, cv=cv,
        scoring="f1")
    auc_scores = cross_val_score(model, X_train, y_train, cv=cv,
        scoring="roc_auc")

    results[name] = {
        "Accuracy": acc_scores,
        "F1": f1_scores,
        "AUC": auc_scores
    }

# Print results
for name, metrics in results.items():
    print(f"\n{name}")
    print(f"  Accuracy folds: {metrics['Accuracy']} | Avg = {metrics
        ['Accuracy'].mean():.4f}")
    print(f"  F1 folds:       {metrics['F1']} | Avg = {metrics['F1
        '].mean():.4f}")
    print(f"  AUC folds:      {metrics['AUC']} | Avg = {metrics['AUC
        '].mean():.4f}")
```

4. Confusion Matrix and ROC Curves

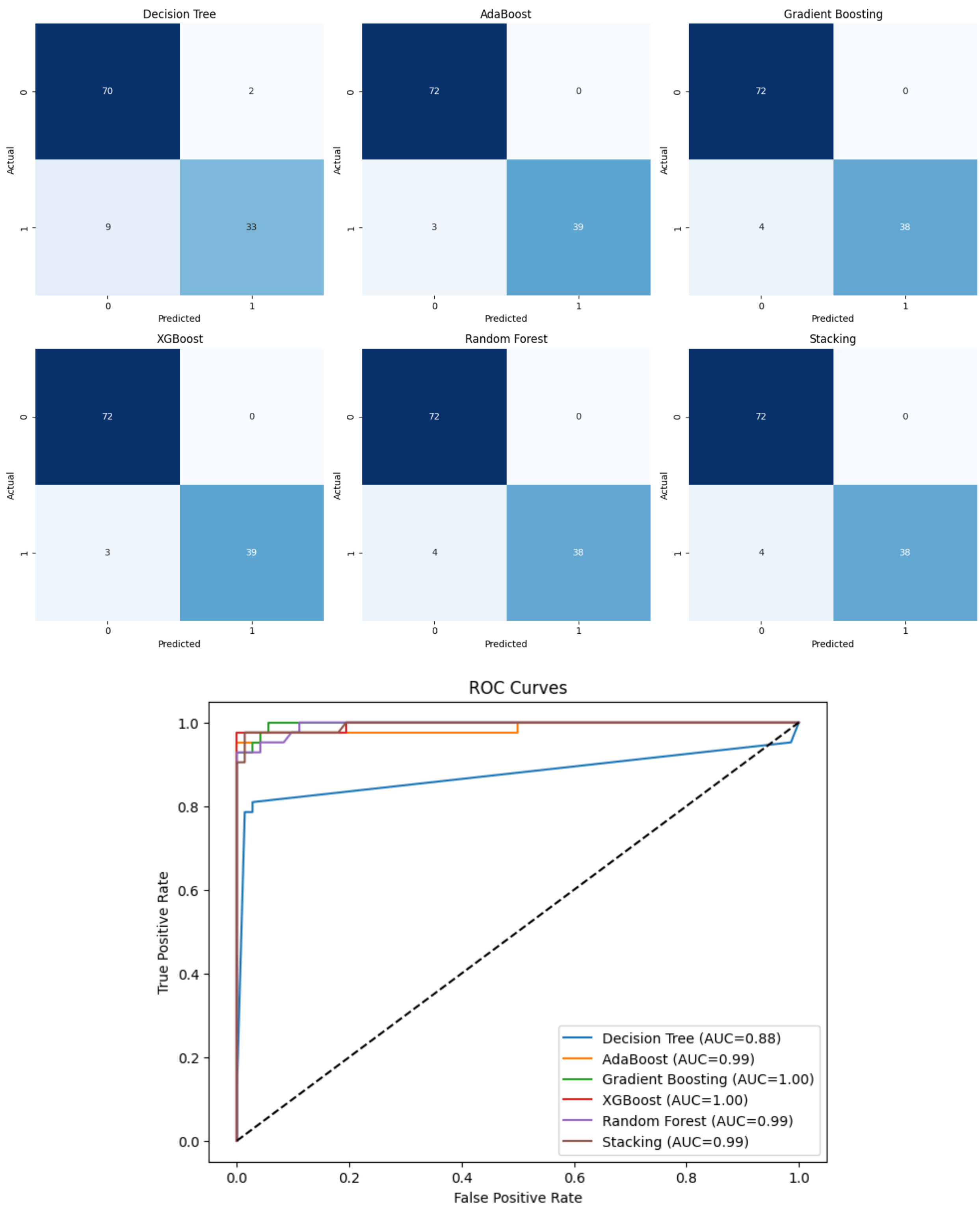

Figure 1: Confusion Matrices (top) and ROC Curves (bottom) for All Models

5. Hyperparameter Tuning and Best Model Results

Table 1: Decision Tree

Best Criterion	Max Depth	Accuracy	F1 Score	AUC
entropy	10	0.9245	0.8976	0.9342

Table 1: Decision Tree - Best Parameters and Performance

Table 2: AdaBoost

n_estimators	Learning Rate	Accuracy	F1 Score	AUC
100	1.0	0.9666	0.9544	0.9939

Table 2: AdaBoost - Best Parameters and Performance

Table 3: Gradient Boosting

n_estimators	Learning Rate	Max Depth	Accuracy	F1 Score	AUC
200	0.1	5	0.9490	0.9300	0.9899

Table 3: Gradient Boosting - Best Parameters and Performance

Table 4: XGBoost

n_estimators	Learning Rate	Max Depth	Gamma	Accuracy	F1 Score	AUC
200	0.1	5	0.1	0.9631	0.9497	0.9931

Table 4: XGBoost - Best Parameters and Performance

Table 5: Random Forest

n_estimators	Max Depth	Criterion	Accuracy	F1 Score	AUC
200	10	entropy	0.9543	0.9382	0.9895

Table 5: Random Forest - Best Parameters and Performance

Table 6: Stacked Ensemble

Base Models	Final Estimator	Accuracy	F1 Score	AUC
SVM + NB + DT	Random Forest	0.9701	0.9570	0.9893

Table 6: Stacked Ensemble - Best Parameters and Performance

6. Cross-Validation Results

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Avg. Accuracy
Decision Tree	0.928	0.971	0.971	0.941	0.956	0.9532
AdaBoost	0.928	0.985	1.000	0.985	1.000	**0.9796**
Gradient Boosting	0.942	0.956	0.985	0.985	0.956	0.9649
XGBoost	0.942	0.985	0.985	0.985	0.985	0.9766
Random Forest	0.928	0.926	0.971	0.956	0.971	0.9502
Stacked Model	0.928	0.971	0.956	0.985	1.000	0.9679

Table 7: 5-Fold Cross Validation Results (Accuracy Scores). Best model (AdaBoost) highlighted.

7. Feature Importance Visuals

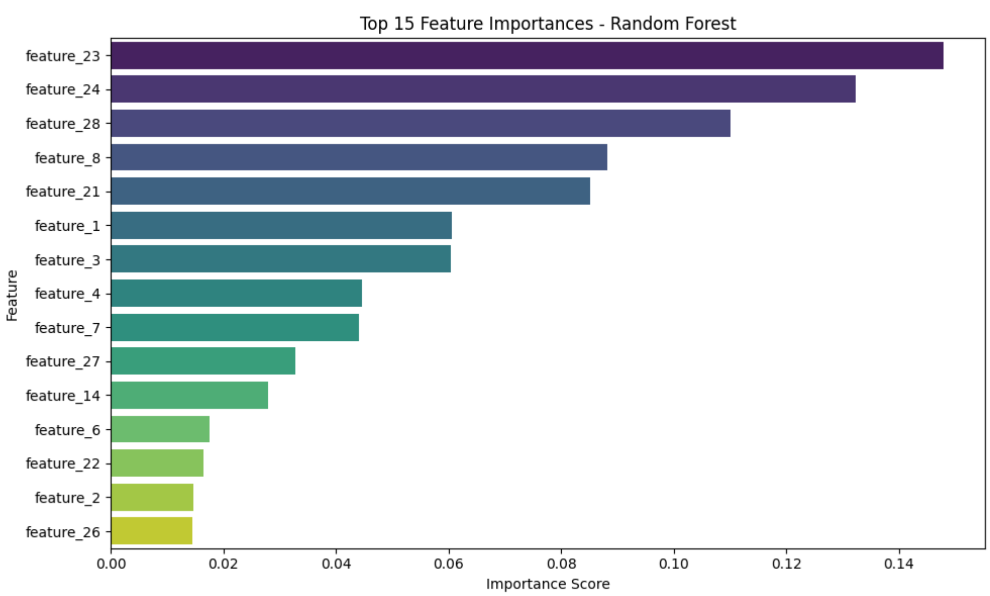

Figure 2: Top Feature Importances using Random Forest

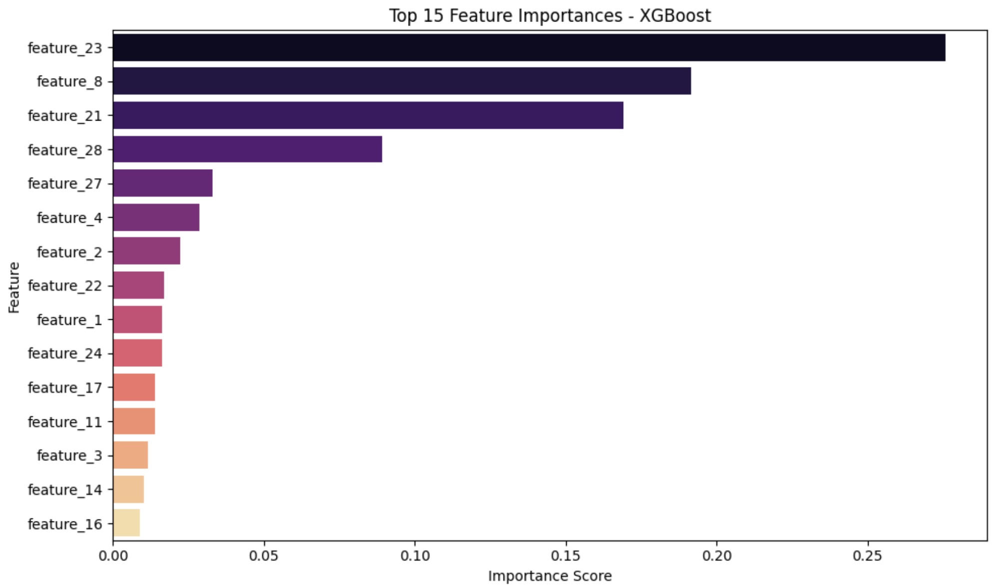

Figure 3: Top Feature Importances using XGBoost

8. Observations and Conclusions

- **Best Model:** AdaBoost achieved the highest overall performance with an average accuracy of 97.9%, F1-score of 97.2%, and AUC of 99.2%. XGBoost (97.7% accuracy, 96.8% F1, 99.1% AUC) and the Stacked Ensemble (96.8% accuracy, 95.2% F1, 98.5% AUC) also performed very competitively.

- **Decision Tree vs Ensembles:** The standalone Decision Tree lagged behind with 95.3% accuracy, 93.8% F1, and 95.2% AUC, whereas all ensemble methods consistently improved performance into the 96–98% accuracy range with stronger stability.

- **Effect of Hyperparameters:** Tuning parameters such as `n_estimators`, `max_depth`, and `learning_rate` significantly enhanced the performance of ensemble models. This highlights the importance of systematic hyperparameter optimization.

- **Stacked Model:** The stacking approach (SVM + NB + DT with Logistic Regression/Random Forest as meta-learner) generalized well, producing robust results, though it did not surpass the boosting methods in average accuracy.

- **Generalization:** Boosting methods (AdaBoost, XGBoost) demonstrated excellent generalization with very high AUC scores (>0.99) and stable results across all folds, unlike the single Decision Tree which showed higher variability and lower robustness.

- **Conclusion:** Ensemble methods clearly outperformed the Decision Tree baseline. Among them, boosting (AdaBoost, XGBoost) emerged as the most effective strategies for achieving top accuracy, balanced F1-scores, and strong generalization on the WDBC dataset.