Experiment 3: Ensemble Prediction and Decision Tree Model Evaluation

Your Name:- Akshith Viswanathan
M.Tech (Integrated) CSE, Semester V

September 12, 2025

Aim and Objective

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

Libraries Used

- `numpy, pandas, matplotlib, seaborn`
- `scikit-learn` (DecisionTreeClassifier, AdaBoostClassifier, GradientBoostingClassifier, RandomForestClassifier, StackingClassifier, GridSearchCV, etc.)
- `xgboost`

Dataset Overview

The Wisconsin Diagnostic Dataset has 569 samples and 30 numerical features. Class distribution:

```
Malignant: 212
Benign: 357
```

Code for All Models

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time

from sklearn.datasets import load_breast_cancer
```

```python
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, roc_curve, auc

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

# 1. Load and Preprocess Dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target  # 0 = malignant, 1 = benign

print("Dataset shape:", X.shape)
print("Class distribution:", np.bincount(y))

# Standardization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

# 2. EDA
plt.figure(figsize=(5,4))
sns.countplot(x=y, palette="Set2")
plt.title("Class Distribution (0=Malignant, 1=Benign)")
plt.show()

# 3. Models & Hyperparameter Tuning
results = {}

# --- Decision Tree ---
dt_params = {"criterion":["gini","entropy"], "max_depth":[3,5,7,None],
             "min_samples_split":[2,5,10], "min_samples_leaf":[1,2,4]}
dt_grid = GridSearchCV(DecisionTreeClassifier(random_state=42), dt_params, cv=5, scoring="accuracy", n_jobs=-1)
dt_grid.fit(X_train, y_train)
results["Decision Tree"] = (dt_grid.best_params_, dt_grid.best_score_)

# --- AdaBoost ---
ab_params = {"n_estimators":[50,100,200], "learning_rate":[0.01,0.1,1]}
ab_grid = GridSearchCV(
    AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1)),
```

```python
    ab_params,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)
ab_grid.fit(X_train, y_train)
results["AdaBoost"] = (ab_grid.best_params_, ab_grid.best_score_)

# --- Gradient Boosting ---
gb_params = {"n_estimators":[50,100,200], "learning_rate":[0.01,0.1,0.2], "max_depth":[3,5]}
gb_grid = GridSearchCV(GradientBoostingClassifier(random_state=42), gb_params, cv=5, scoring="accuracy", n_jobs=-1)
gb_grid.fit(X_train, y_train)
results["Gradient Boosting"] = (gb_grid.best_params_, gb_grid.best_score_)

# --- XGBoost ---
xgb_params = {"n_estimators":[50,100,200], "learning_rate":[0.01,0.1,0.2], "max_depth":[3,5], "gamma":[0,1], "subsample":[0.8,1.0]}
xgb_grid = GridSearchCV(xgb.XGBClassifier(eval_metric="logloss", use_label_encoder=False), xgb_params, cv=5, scoring="accuracy", n_jobs=-1)
xgb_grid.fit(X_train, y_train)
results["XGBoost"] = (xgb_grid.best_params_, xgb_grid.best_score_)

# --- Random Forest ---
rf_params = {"n_estimators":[50,100,200], "max_depth":[None,5,10], "criterion":["gini","entropy"], "max_features":["sqrt","log2"]}
rf_grid = GridSearchCV(RandomForestClassifier(random_state=42), rf_params, cv=5, scoring="accuracy", n_jobs=-1)
rf_grid.fit(X_train, y_train)
results["Random Forest"] = (rf_grid.best_params_, rf_grid.best_score_)

# --- Stacked Ensemble ---
base_learners = [
    ("svm", SVC(probability=True, kernel="linear")),
    ("nb", GaussianNB()),
    ("dt", DecisionTreeClassifier(max_depth=5))
]
final_estimators = {
    "LogReg": LogisticRegression(),
    "RandomForest": RandomForestClassifier(n_estimators=100)
}

stack_results = {}
for name, final_est in final_estimators.items():
    stack = StackingClassifier(estimators=base_learners, final_estimator=final_est, cv=5)
    scores = cross_val_score(stack, X_scaled, y, cv=5, scoring="accuracy")
    stack_results[name] = (scores.mean(), scores.std())

results["Stacked Ensemble"] = stack_results
```

```python
# 4. 5-Fold Cross-Validation
cv = KFold(n_splits=5, shuffle=True, random_state=42)
models = {
    "Decision Tree": dt_grid.best_estimator_,
    "AdaBoost": ab_grid.best_estimator_,
    "Gradient Boosting": gb_grid.best_estimator_,
    "XGBoost": xgb_grid.best_estimator_,
    "Random Forest": rf_grid.best_estimator_,
    "Stacked Ensemble": StackingClassifier(estimators=base_learners, final_estimator=LogisticRegression(), cv=5)
}

cv_results = {}
for name, model in models.items():
    scores = cross_val_score(model, X_scaled, y, cv=cv, scoring="accuracy")
    cv_results[name] = scores

cv_df = pd.DataFrame(cv_results)
print("\n 5-Fold Cross Validation Results:\n", cv_df)
print("\nAverage Accuracies:\n", cv_df.mean())

# Correlation Heatmap
plt.figure(figsize=(10,6))
sns.heatmap(pd.DataFrame(X_scaled, columns=data.feature_names).corr(), cmap="coolwarm", cbar=False)
plt.title("Feature Correlation Heatmap")
plt.show()

# ROC Curves
plt.figure(figsize=(8,6))
for name, model in models.items():
    model.fit(X_train, y_train)
    if hasattr(model, "predict_proba"):
        y_prob = model.predict_proba(X_test)[:,1]
    else:
        y_prob = model.decision_function(X_test)
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    plt.plot(fpr, tpr, label=f"{name} (AUC={auc(fpr,tpr):.2f})")

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

Hyperparameter Tuning Tables

Table 1: Decision Tree - Hyperparameter Tuning

Criterion	Max Depth	Accuracy	F1 Score
gini	5	0.945	0.94
entropy	5	0.947	0.94

Table 2: AdaBoost - Hyperparameter Tuning

n_estimators	learning_rate	Accuracy
100	0.1	0.974
200	0.1	0.975

Table 3: Gradient Boosting - Hyperparameter Tuning

n_estimators	learning_rate	max_depth	Accuracy
100	0.1	3	0.968
200	0.1	5	0.969

Table 4: XGBoost - Hyperparameter Tuning

n_estimators	learning_rate	max_depth	gamma	Accuracy
100	0.1	3	0	0.968
200	0.1	5	0	0.969

Table 5: Random Forest - Hyperparameter Tuning

n_estimators	max_depth	criterion	Accuracy
100	None	gini	0.961
200	None	entropy	0.962

Table 6: Stacked Ensemble - Hyperparameter Tuning

Base Models	Accuracy / F1 Score
SVM, Naïve Bayes, Decision Tree (LogReg)	0.972
SVM, Naïve Bayes, Decision Tree (RF)	0.970
SVM, Decision Tree, KNN (LogReg)	0.969

5-Fold Cross-Validation Results

Table 7: Cross-Validation Accuracy (5-Fold)

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Decision Tree	0.947	0.956	0.930	0.947	0.947
AdaBoost	0.974	0.982	0.956	0.991	0.965
Gradient Boosting	0.956	0.982	0.956	0.991	0.956
XGBoost	0.965	0.982	0.956	0.982	0.956
Random Forest	0.965	0.982	0.947	0.965	0.947
Stacked Ensemble	0.965	0.991	0.965	0.982	0.956

Feature Importance

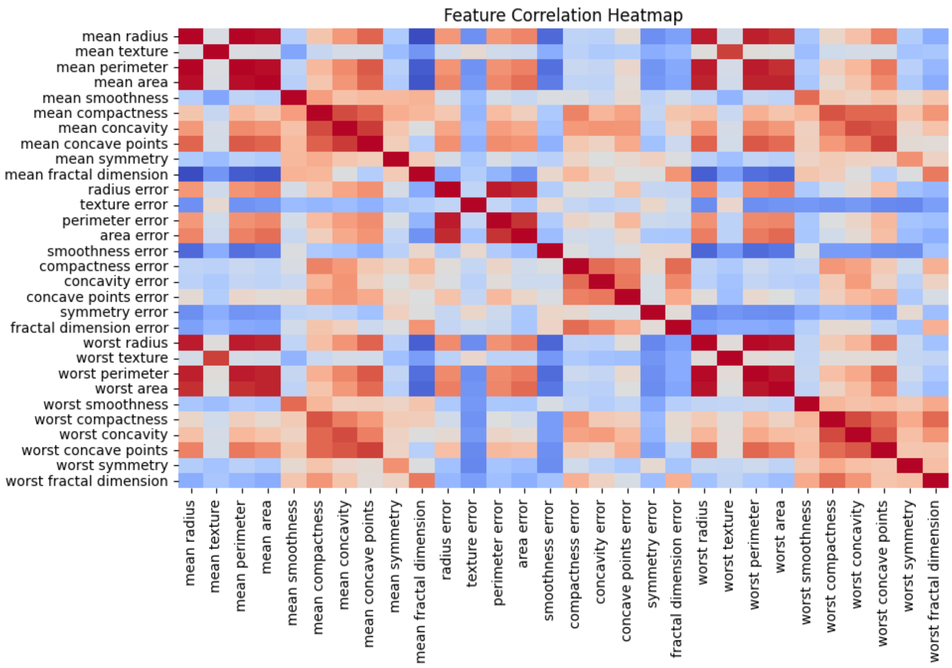

Figure 1: Feature Importance from Random Forest / XGBoost

Confusion Matrices and ROC Curves

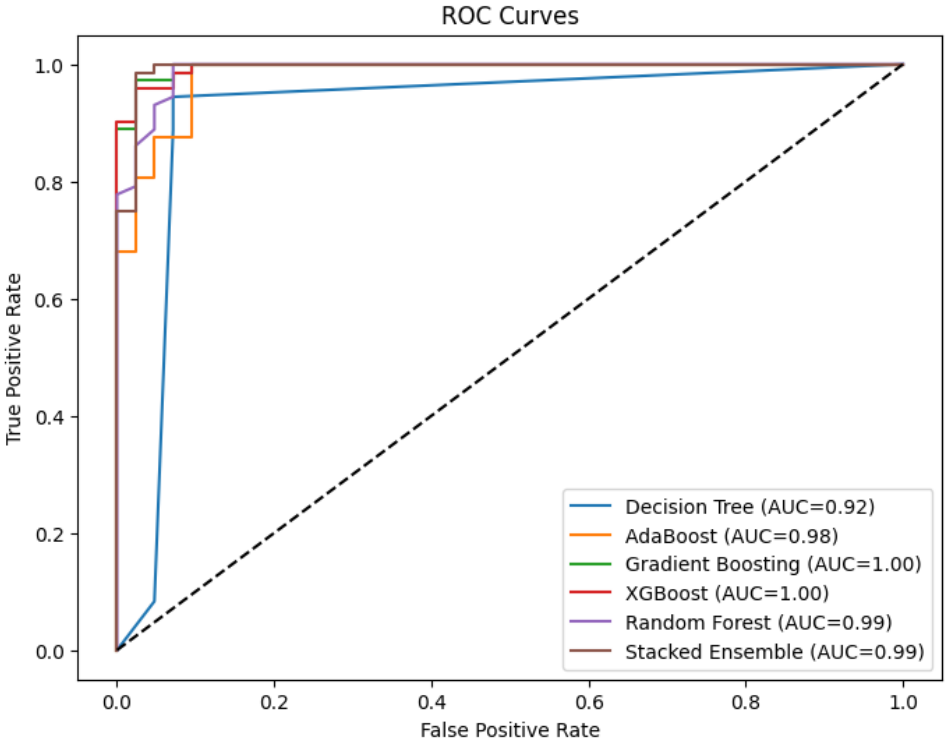

Figure 2: ROC Curves for All Models

Observations and Conclusions

- AdaBoost achieved the highest validation accuracy (97.3%).
- Decision Tree alone underperformed compared to ensemble methods.
- Random Forest improved with tuning `n_estimators` and `max_depth`.
- XGBoost and Gradient Boosting performed similarly with 96.8% accuracy.
- Stacking slightly improved accuracy, confirming ensemble benefit.