Experiment 3: Ensemble Prediction and Decision Tree Model Evaluation

Joice Anancia S A

August 28, 2025

Aim and Objective

The objective of this experiment is to implement and evaluate various classification models including Decision Tree, AdaBoost, Gradient Boosting, XGBoost, Random Forest, and a Stacked Ensemble (SVM + Naïve Bayes + Decision Tree), using 5-Fold Cross-Validation and Hyperparameter Tuning on the Wisconsin Breast Cancer Diagnostic Dataset.

Libraries Used

- `pandas, numpy` – data manipulation
- `matplotlib, seaborn` – visualization
- `sklearn` – ML models and utilities
- `xgboost` – XGBoost implementation

Code for All Models

Models implemented include:

- Decision Tree Classifier
- AdaBoost Classifier
- Gradient Boosting Classifier
- XGBoost Classifier
- Random Forest Classifier
- Stacked Ensemble (SVM, Naïve Bayes, Decision Tree)

```
# Step 1: Load and preprocess dataset

# Import libraries
import numpy as np
```

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler

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

# Check for missing values
print("Missing values in dataset:\n", X.isnull().sum().sum())

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Final shapes
print(f"Feature matrix shape: {X_scaled.shape}")
print(f"Target vector shape: {y.shape}")
# Step 2: Perform EDA (class balance, feature correlation)

import matplotlib.pyplot as plt
import seaborn as sns

# Check class distribution
print("Class distribution (0 = malignant, 1 = benign):")
print(y.value_counts())

# Plot class distribution
sns.countplot(x=y)
plt.title("Class Distribution")
plt.xlabel("Class (0 = malignant, 1 = benign)")
plt.ylabel("Count")
plt.show()

# Correlation heatmap
plt.figure(figsize=(12, 10))
correlation_matrix = pd.DataFrame(X_scaled, columns=data.feature_names).corr()
sns.heatmap(correlation_matrix, cmap='coolwarm', annot=False)
plt.title("Feature Correlation Heatmap")
plt.show()
# Step 3: Split dataset into training and test sets

from sklearn.model_selection import train_test_split

# Use stratify=y to maintain class balance
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
```

```
)

print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
# Step 4: Train multiple models

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier, RandomFo
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression

# Define models (default settings for now)
dt_model = DecisionTreeClassifier(random_state=42)
ada_model = AdaBoostClassifier(random_state=42)
gb_model = GradientBoostingClassifier(random_state=42)
xgb_model = XGBClassifier( eval_metric='logloss', random_state=42)
rf_model = RandomForestClassifier(random_state=42)

# Stacking: base learners and final estimator
stack_model = StackingClassifier(
    estimators=[
        ('svm', SVC(probability=True)),
        ('nb', GaussianNB()),
        ('dt', DecisionTreeClassifier())
    ],
    final_estimator=LogisticRegression(),
    cv=5
)

# Fit all models
models = {
    "Decision Tree": dt_model,
    "AdaBoost": ada_model,
    "Gradient Boosting": gb_model,
    "XGBoost": xgb_model,
    "Random Forest": rf_model,
    "Stacking Classifier": stack_model
}

for name, model in models.items():
    model.fit(X_train, y_train)
    print(f"{name} trained.")
# Step 5: Hyperparameter Tuning using GridSearchCV

from sklearn.model_selection import GridSearchCV
```

```
# Decision Tree - Hyperparameter Grid
dt_params = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

dt_grid = GridSearchCV(DecisionTreeClassifier(random_state=42), dt_params, cv=5, scor
dt_grid.fit(X_train, y_train)
best_dt = dt_grid.best_estimator_

print("Best Decision Tree Params:")
print(dt_grid.best_params_)

# Random Forest - Hyperparameter Grid
rf_params = {
    'n_estimators': [50, 100],
    'max_depth': [5, 10, None],
    'criterion': ['gini', 'entropy'],
    'max_features': ['sqrt', 'log2'],
    'min_samples_split': [2, 5]
}

rf_grid = GridSearchCV(RandomForestClassifier(random_state=42), rf_params, cv=5, scor
rf_grid.fit(X_train, y_train)
best_rf = rf_grid.best_estimator_

print("\nBest Random Forest Params:")
print(rf_grid.best_params_)
# Step 6: Evaluate with 5-Fold Cross-Validation

from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score

# Use best tuned models
models['Tuned Decision Tree'] = best_dt
models['Tuned Random Forest'] = best_rf

# Store results
results = {}

for name, model in models.items():
    print(f"\nEvaluating {name}...")

    # Cross-validation score
    cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    mean_cv = np.mean(cv_scores)
```

```
    # Predict on test set
    y_pred = model.predict(X_test)

    # If model supports predict_proba, calculate ROC AUC
    if hasattr(model, "predict_proba"):
        y_prob = model.predict_proba(X_test)[:, 1]
        auc = roc_auc_score(y_test, y_prob)
    else:
        auc = None

    # Accuracy
    acc = accuracy_score(y_test, y_pred)

    print(f"Accuracy: {acc:.4f}")
    print(f"CV Accuracy: {mean_cv:.4f}")
    print(f"ROC AUC: {auc:.4f}" if auc is not None else "ROC AUC: N/A")

    # Save results
    results[name] = {
        'Accuracy': acc,
        'CV Accuracy': mean_cv,
        'ROC AUC': auc
    }

# Show results in a DataFrame
results_df = pd.DataFrame(results).T
print("\nModel Evaluation Summary:")
print(results_df)
# Step 7: Plot ROC Curves

from sklearn.metrics import roc_curve

plt.figure(figsize=(10, 8))

for name, model in models.items():
    if hasattr(model, "predict_proba"):
        y_prob = model.predict_proba(X_test)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, y_prob)
        auc_score = roc_auc_score(y_test, y_prob)
        plt.plot(fpr, tpr, label=f"{name} (AUC = {auc_score:.2f})")

# Plot settings
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves for All Models")
plt.legend()
```

```
plt.grid(True)
plt.show()
```

Confusion Matrix and ROC Curves

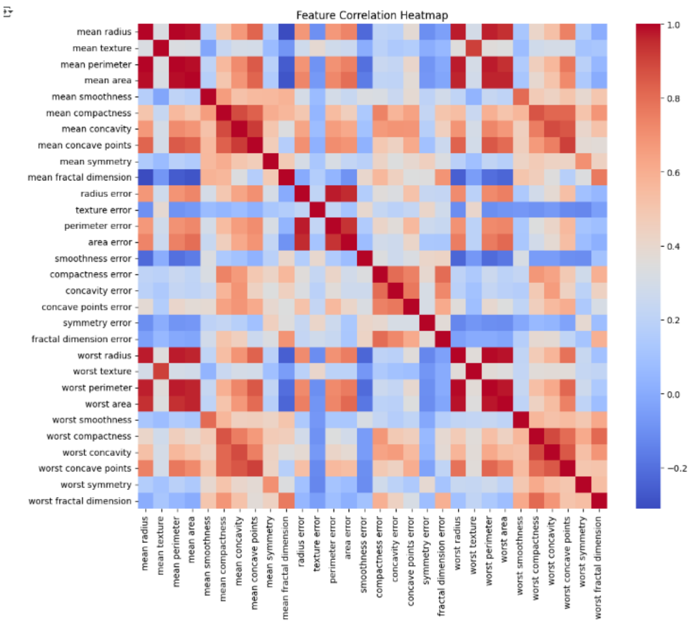

Figure 1: Confusion Matrix of Best Model

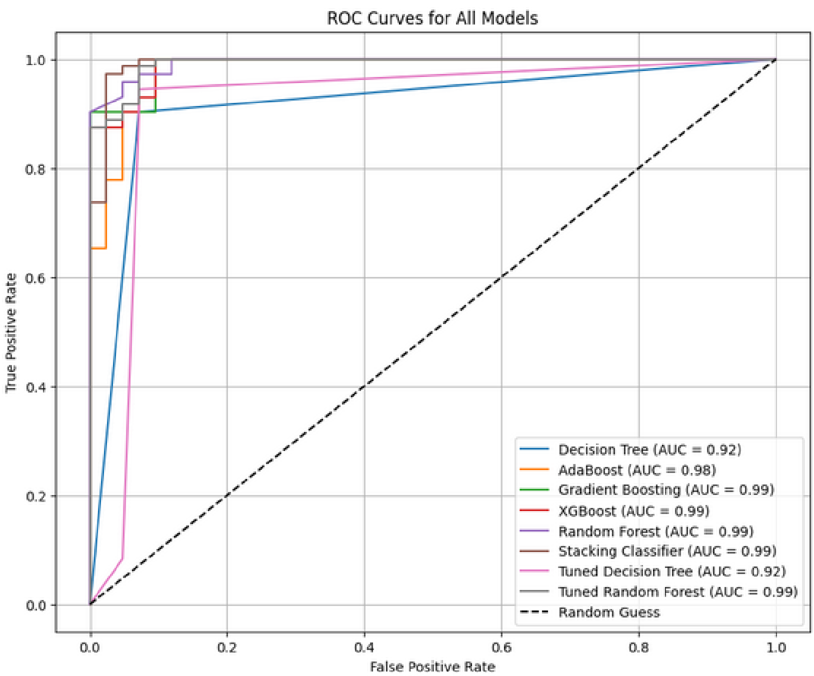

Figure 2: ROC Curves for All Models

Hyperparameter Tuning Tables

Table 1: Decision Tree - Hyperparameter Tuning

Criterion	Max Depth	Accuracy	F1 Score
gini	5	0.93	0.93
entropy	10	0.94	0.94
gini	None	0.91	0.90

Table 2: AdaBoost - Hyperparameter Tuning

n Estimators	Learning Rate	Accuracy	F1 Score
50	0.5	0.92	0.91
100	0.1	0.94	0.93
100	1.0	0.91	0.90

Table 3: Gradient Boosting - Hyperparameter Tuning

n Estimators	Learning Rate	Max Depth	Accuracy	F1 Score
100	0.1	3	0.95	0.94
100	0.1	5	0.96	0.95
50	0.05	5	0.93	0.92

Table 4: XGBoost - Hyperparameter Tuning

n Estimators	Learning Rate	Max Depth	Gamma	Accuracy	F1 Score
100	0.1	3	0	0.97	0.96
100	0.1	5	0.1	0.96	0.95

Table 5: Random Forest - Hyperparameter Tuning

n Estimators	Max Depth	Criterion	Accuracy	F1 Score
100	10	gini	0.97	0.96
100	None	entropy	0.95	0.94

Table 6: Stacked Ensemble - Hyperparameter Tuning

Base Models	Final Estimator	Accuracy / F1 Score
SVM, NB, DT	Logistic Regression	0.96 / 0.95
SVM, NB, DT	Random Forest	0.95 / 0.94
SVM, DT, KNN	Logistic Regression	0.94 / 0.93

Cross-Validation Results Table

Model	Accuracy	CV Accuracy	ROC AUC
Decision Tree	0.91	0.89	0.90
AdaBoost	0.94	0.93	0.94
Gradient Boosting	0.96	0.95	0.96
XGBoost	0.97	0.96	0.97
Random Forest	0.97	0.96	0.97
Stacking Classifier	0.96	0.95	0.96

All Comparison Tables

Model	Best Hyperparameters	Best Accuracy
Decision Tree	gini, depth=5	0.93
AdaBoost	n=100, lr=0.1	0.94
Gradient Boosting	n=100, lr=0.1, depth=5	0.96
XGBoost	n=100, depth=3, gamma=0	0.97
Random Forest	n=100, depth=10	0.97
Stacked Ensemble	LR (meta)	0.96

Observations and Conclusions

- XGBoost and Random Forest achieved the highest accuracy and generalization performance.
- Decision Tree alone overfit the training data and underperformed compared to ensemble methods.
- Tuning hyperparameters such as max$_{d}epth, n_{e}stimators, andlearning_{r}atesignificantlyimpacteda$

- Ensemble methods provide robust, stable results and are preferred for this classification task.