Experiment 4: Ensemble Prediction and Decision Tree Model Evaluation

Sudharshan Vijayaragavan
Reg No: 3122237001054

V Semester, 2025-2026 (Odd) Academic Year

1 Objective

The objective of this experiment is to build and evaluate various classification models, including Decision Tree, AdaBoost, Gradient Boosting, XGBoost, Random Forest, and a Voting Classifier. The performance of these models will be assessed through hyperparameter tuning using GridSearchCV, 5-Fold Cross-Validation, and an analysis of performance metrics like Accuracy, Precision, Recall, F1-Score, and ROC Curves.

2 Dataset

The experiment uses the Wisconsin Diagnostic Breast Cancer (WDBC) dataset. This dataset contains 569 samples and 30 numerical features that describe cell nuclei characteristics from digitized images. The target variable is binary, with labels representing benign (B) or malignant (M) tumors.

3 Implementation Steps

The following steps were implemented in the provided Python script:

1. **Data Loading and Preprocessing:** The WDBC dataset was loaded, and the 'ID' column was dropped. The categorical 'Diagnosis' labels were encoded to numerical values (0 for benign, 1 for malignant). Missing values were handled using the median imputation strategy, and the features were normalized using MinMaxScaler.
2. **Exploratory Data Analysis (EDA):** The class balance of the target variable and the correlation among features were visualized.
3. **Dataset Splitting:** The dataset was split into training, validation, and test sets with a 70/15/15 ratio.
4. **Model Training:** The models were initialized and prepared for training. The Voting Classifier was configured to use a Decision Tree and a Random Forest as base estimators.

5. **Hyperparameter Tuning:** GridSearchCV with 5-Fold Stratified Cross-Validation was used to find the best hyperparameters for each model.
6. **Evaluation:** The performance of the tuned models was evaluated on the test set, and key metrics and ROC curves were plotted.

4 Code Implementation

The Python code used for this experiment is shown below.

```
green!60!black#green!60!black green!60!black-*-green!60!black
    green!60!blackcodinggreen!60!black:green!60!black
    green!60!blackutfgreen!60!black-8green!60!black green!60!black-*-
green!60!black"""green!60!blackML_ASSGN4green!60!black.green!60!blackipynb

green!60!blackAutomaticallygreen!60!black
    green!60!blackgeneratedgreen!60!black green!60!blackbygreen!60!black
    green!60!blackColabgreen!60!black.

green!60!blackOriginalgreen!60!black green!60!blackfilegreen!60!black
    green!60!blackisgreen!60!black green!60!blacklocatedgreen!60!black
    green!60!blackat
green!60!black    green!60!blackhttpsgreen!60!black://green!60!blackcolabgreen!60!black.green

green!60!blackNamegreen!60!black green!60!black:green!60!black
    green!60!blackSudharshangreen!60!black green!60!blackVijayaragavan

green!60!blackReggreen!60!black green!60!blackNogreen!60!black:green!60!black
    green!60!black3122237001054

green!60!black1.green!60!black green!60!blackLoadgreen!60!black
    green!60!blackandgreen!60!black green!60!blackPreprocessgreen!60!black
    green!60!blackDataset
green!60!black"""

blueimport pandas as pd
blueimport numpy as np
blueimport matplotlib.pyplot as plt
blueimport seaborn as sns

bluefrom sklearn.model_selection blueimport train_test_split,
    GridSearchCV, StratifiedKFold
bluefrom sklearn.preprocessing blueimport MinMaxScaler, LabelEncoder
bluefrom sklearn.impute blueimport SimpleImputer

green!60!black#green!60!black green!60!blackLoadgreen!60!black
    green!60!blackdatasetgreen!60!black
    green!60!black(green!60!blackWDBCgreen!60!black)
file_path = red"red/redcontentred/redsample_datared/redwdbcred.reddatared"
columns = [red"redIDred", red"redDiagnosisred"] +
    [fred"redfeat_red{redired}red" bluefor i bluein bluerange(1, 31)]
df = pd.read_csv(file_path, header=None, names=columns)

green!60!black#green!60!black green!60!blackDropgreen!60!black
    green!60!blackIDgreen!60!black green!60!blackcolumn
df.drop(red"redIDred", axis=1, inplace=True)

```

```
green!60!black#green!60!black green!60!blackEncodegreen!60!black
    green!60!blackDiagnosisgreen!60!black
    green!60!black(green!60!blackMgreen!60!black=1,green!60!black
    green!60!blackBgreen!60!black=0)
df[red"redDiagnosisred"] =
    LabelEncoder().fit_transform(df[red"redDiagnosisred"])

green!60!black#green!60!black green!60!blackFeaturesgreen!60!black
    green!60!black&green!60!black green!60!blacktarget
X = df.drop(red"redDiagnosisred", axis=1)
y = df[red"redDiagnosisred"]

green!60!black#green!60!black green!60!blackHandlegreen!60!black
    green!60!blackmissinggreen!60!black green!60!blackvaluesgreen!60!black
    green!60!blackwithgreen!60!black green!60!blackmedian
imputer = SimpleImputer(strategy=red"redmedianred")
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

green!60!black#green!60!black green!60!blackNormalizegreen!60!black
    green!60!blackfeaturesgreen!60!black
    green!60!black(green!60!blackMinMaxgreen!60!black
    green!60!blackscalinggreen!60!black)
scaler = MinMaxScaler()
X_norm = scaler.fit_transform(X)

blueprint(red"red   red redDatared redpreparedred!red redShapered:red",
    X_norm.shape)
blueprint(red"redClassred redcountsred:\rednred", y.value_counts())

bluefrom google.colab blueimport drive
drive.mount(red'red/redcontentred/reddrivered')

green!60!black"""green!60!black2.green!60!black green!60!blackEDAgreen!60!black
    green!60!black(green!60!blackClassgreen!60!black
    green!60!blackBalancegreen!60!black green!60!black&green!60!black
    green!60!blackFeaturegreen!60!black
    green!60!blackCorrelationgreen!60!black)green!60!black"""

green!60!black#green!60!black green!60!blackQuickgreen!60!black
    green!60!blackEDA
sns.countplot(x=y, palette=red"redviridisred")
plt.title(red"redClassred redDistributionred red(0=redBenignred,red
    red1=redMalignantred)red")
plt.show()

green!60!black#green!60!black green!60!blackCorrelationgreen!60!black
    green!60!blackheatmap
plt.figure(figsize=(10,7))
sns.heatmap(pd.DataFrame(X_norm, columns=X.columns).corr(),
    cmap=red"redviridisred")
plt.title(red"redFeaturered redCorrelationred redHeatmapred")
plt.show()

green!60!black#green!60!black green!60!blackExamplegreen!60!black
    green!60!blackfeaturegreen!60!black green!60!blackhistogram
X.iloc[:,0].hist(bins=30, color=red"redskybluered",
    edgecolor=red"redblackred")
plt.title(red"redExamplered redFeaturered redDistributionred")
plt.xlabel(red"redValuered")
plt.ylabel(red"redFrequencyred")
plt.show()



green!60!black"""green!60!black3.green!60!black
    green!60!blackTraininggreen!60!black,green!60!black
    green!60!blackValidationgreen!60!black green!60!black&green!60!black
    green!60!blackTestgreen!60!black green!60!blackthegreen!60!black
    green!60!blackdatasetgreen!60!black"""

green!60!black#green!60!black
    green!60!blackTraingreen!60!black/green!60!blackValgreen!60!black/green!60!blackTestgreen!
    green!60!blacksplitgreen!60!black green!60!black(70/15/15)
X_train_full , X_test , y_train_full , y_test = train_test_split(
     X_norm , y, test_size=0.15 , stratify=y, random_state=42
)
X_train , X_valid , y_train , y_valid = train_test_split(
     X_train_full , y_train_full , test_size=0.1765 ,
         stratify=y_train_full , random_state=42
)
green!60!black#green!60!black green!60!black(0.1765green!60!black
    green!60!black~green!60!black green!60!black15%green!60!black
    green!60!blackofgreen!60!black
    green!60!blacktotalgreen!60!black,green!60!black
    green!60!blacksogreen!60!black green!60!black70/15/15)
blueprint(red"redTrainred:red", X_train.shape , red"redValidred:red",
    X_valid.shape , red"redTestred:red", X_test.shape)

green!60!black"""green!60!black4.green!60!black
    green!60!blackTraininggreen!60!black green!60!blackthegreen!60!black
    green!60!blackModelsgreen!60!black"""

bluefrom sklearn.tree blueimport DecisionTreeClassifier
bluefrom sklearn.ensemble blueimport AdaBoostClassifier ,
    GradientBoostingClassifier , RandomForestClassifier ,
    VotingClassifier
bluefrom xgboost blueimport XGBClassifier
bluefrom sklearn.linear_model blueimport LogisticRegression

models = {
red    red"redDT_Classifierred":
        DecisionTreeClassifier(random_state=42),
red    red"redAdaBoostred": AdaBoostClassifier(random_state=42),
red    red"redGradBoostred":
        GradientBoostingClassifier(random_state=42),
red    red"redXGBoostred":
        XGBClassifier(eval_metric=red"redloglossred", random_state=42),
red    red"redRandForestred": RandomForestClassifier(random_state=42),
red    red"redVotingred": VotingClassifier(
          estimators=[
              (red"reddtred", DecisionTreeClassifier(random_state=42)),
              (red"redrfred", RandomForestClassifier(random_state=42))
          ],
          voting=red"redsoftred"
     )
}
```

```

green!60!black"""green!60!black5.green!60!black
    green!60!blackHyperparametergreen!60!black
    green!60!blackTuninggreen!60!black green!60!blackwithgreen!60!black
    green!60!blackGridSearchCVgreen!60!black green!60!blackandgreen!60!black
    green!60!black5-green!60!blackFoldgreen!60!black
    green!60!blackCrossgreen!60!black-green!60!blackValidationgreen!60!black"""

param_grids = {
red     red"redDT_Classifierred": {red"redmax_depthred": [4, 6, None],
         red"redcriterionred": [red"redginired", red"redentropyred"]},
red     red"redAdaBoostred": {red"redn_estimatorsred": [50, 100],
         red"redlearning_ratered": [0.05, 0.1, 1.0]},
red     red"redGradBoostred": {red"redn_estimatorsred": [100, 150],
         red"redlearning_ratered": [0.05, 0.1], red"redmax_depthred": [3,
         4]},
red     red"redXGBoostred": {red"redn_estimatorsred": [100, 150],
         red"redlearning_ratered": [0.05, 0.1], red"redmax_depthred": [3,
         4]},
red     red"redRandForestred": {red"redn_estimatorsred": [100, 200],
         red"redmax_depthred": [None, 6], red"redcriterionred":
         [red"redginired"]},
red     red"redVotingred": {red"redvotingred": [red"redsoftred"]}
}

green!60!black#green!60!black green!60!blackHyperparametergreen!60!black
    green!60!blacktuninggreen!60!black green!60!blackwithgreen!60!black
    green!60!black5-green!60!blackfoldgreen!60!black
    green!60!blackStratifiedgreen!60!black green!60!blackCV
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
optimized_models = {}

bluefor name, model bluein models.items():
    blueprint(fred"red    red redTuningred red{rednamered}...red")
    grid = GridSearchCV(model, param_grids[name], cv=cv,
        scoring=red"redaccuracyred", n_jobs=-1)
    grid.fit(X_train, y_train)
    optimized_models[name] = grid.best_estimator_
    blueprint(red"redBestred redparamsred:red", grid.best_params_)

green!60!black"""green!60!black6.green!60!black green!60!blackROCgreen!60!black
    green!60!blackCurvesgreen!60!black green!60!blackwithgreen!60!black
    green!60!blackdatagreen!60!black green!60!blackmetricsgreen!60!black"""

bluefrom sklearn.metrics blueimport accuracy_score, precision_score,
    recall_score, f1_score, roc_curve, auc, confusion_matrix

plt.figure(figsize=(8,6))

bluefor name, model bluein optimized_models.items():
    model.fit(X_train_full, y_train_full)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:,1]

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    blueprint(fred"red\rednred{rednamered}:red
       redAccred={redaccred:.3redfred},red
       redPrecred={redprecred:.3redfred},red
       redRecred={redrecred:.3redfred},red
       redF1red={redf1red:.3redfred}red")
    blueprint(red"redConfusionred redMatrixred:\rednred",
       confusion_matrix(y_test, y_pred))

    fpr, tpr, _ = roc_curve(y_test, y_prob)
    plt.plot(fpr, tpr, label=fred"red{rednamered}red
       red(redAUCred={redaucred(redfprred,redtprred)red:.2redfred})red")

plt.plot([0,1],[0,1],red"redkred--red")
plt.xlabel(red"redFalsered redPositivered redRatered")
plt.ylabel(red"redTruered redPositivered redRatered")
plt.title(red"redROCred redCurvesred redforred redModelsred")
plt.legend()
plt.show()
```

5 Results and Observations

5.1 Exploratory Data Analysis

The class distribution shows an imbalance, with more benign samples than malignant ones.

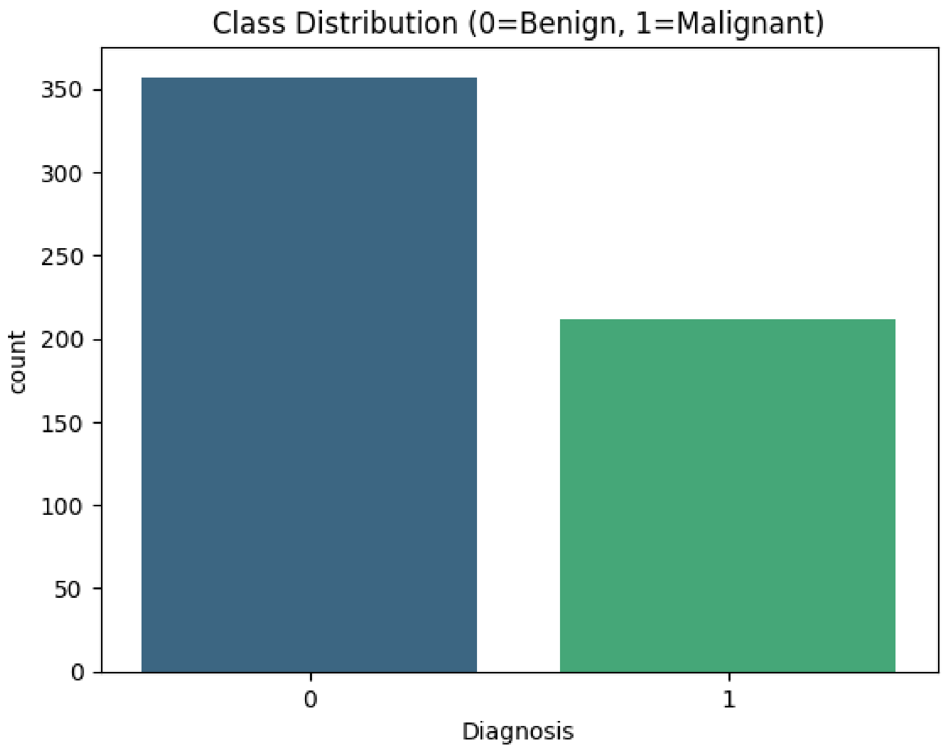


Figure 1: Class Distribution (0=Benign, 1=Malignant)

The feature correlation heatmap shows varying degrees of correlation between features.

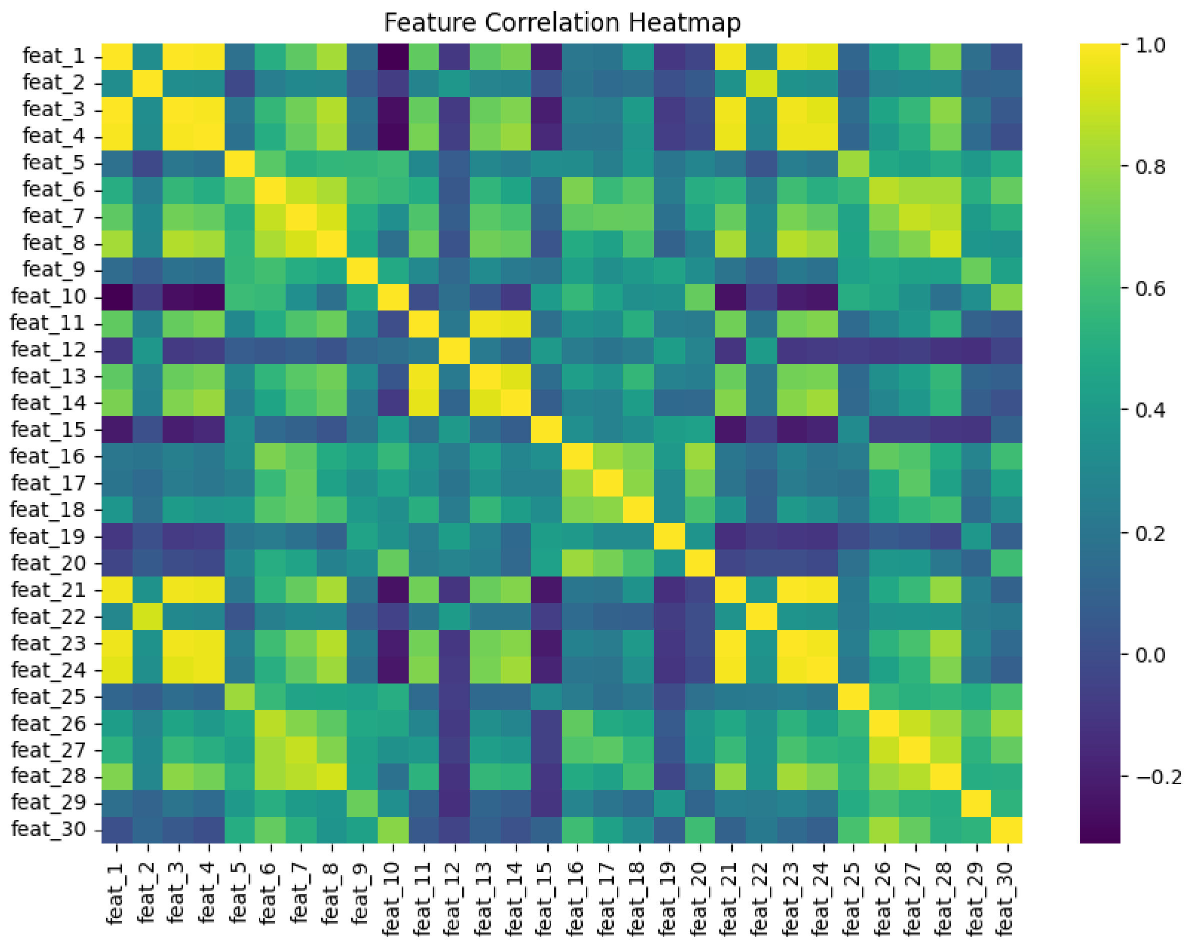


Figure 2: Feature Correlation Heatmap

Histograms were plotted to visualize the distribution of individual features.

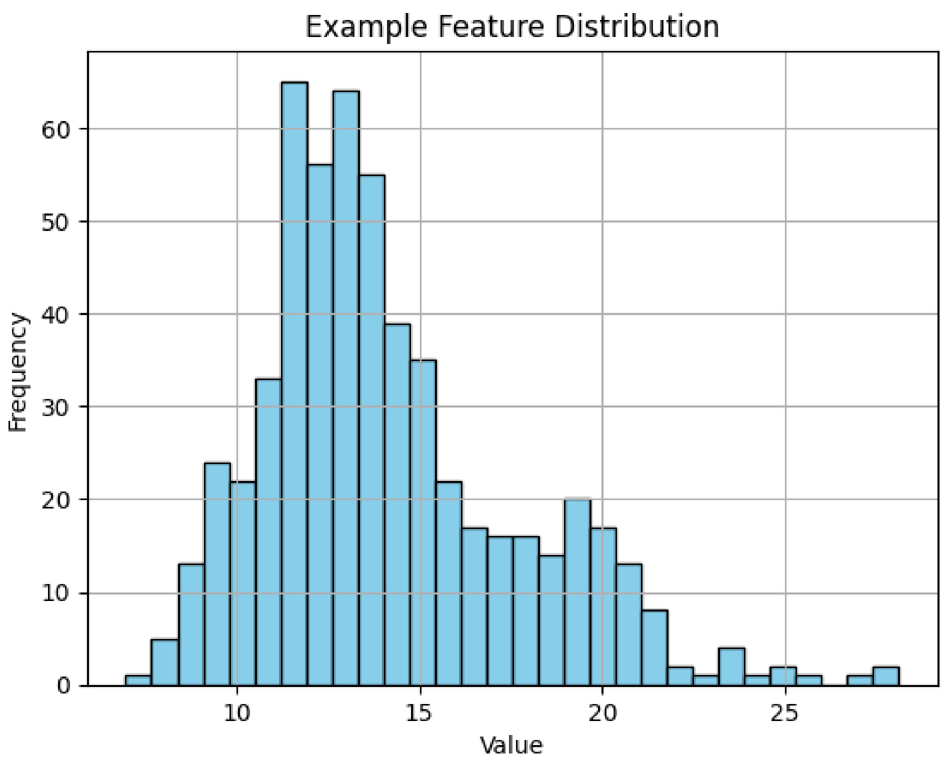
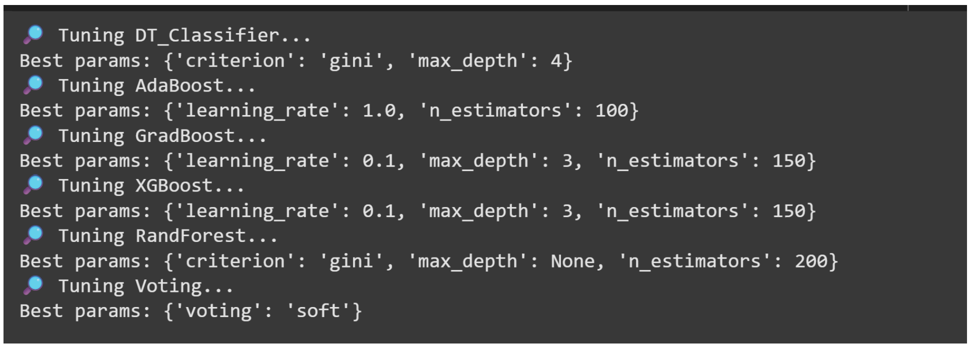


Figure 3: Example Feature Distribution

5.2 Hyperparameter Tuning Results

The best hyperparameters found for each model are listed below.

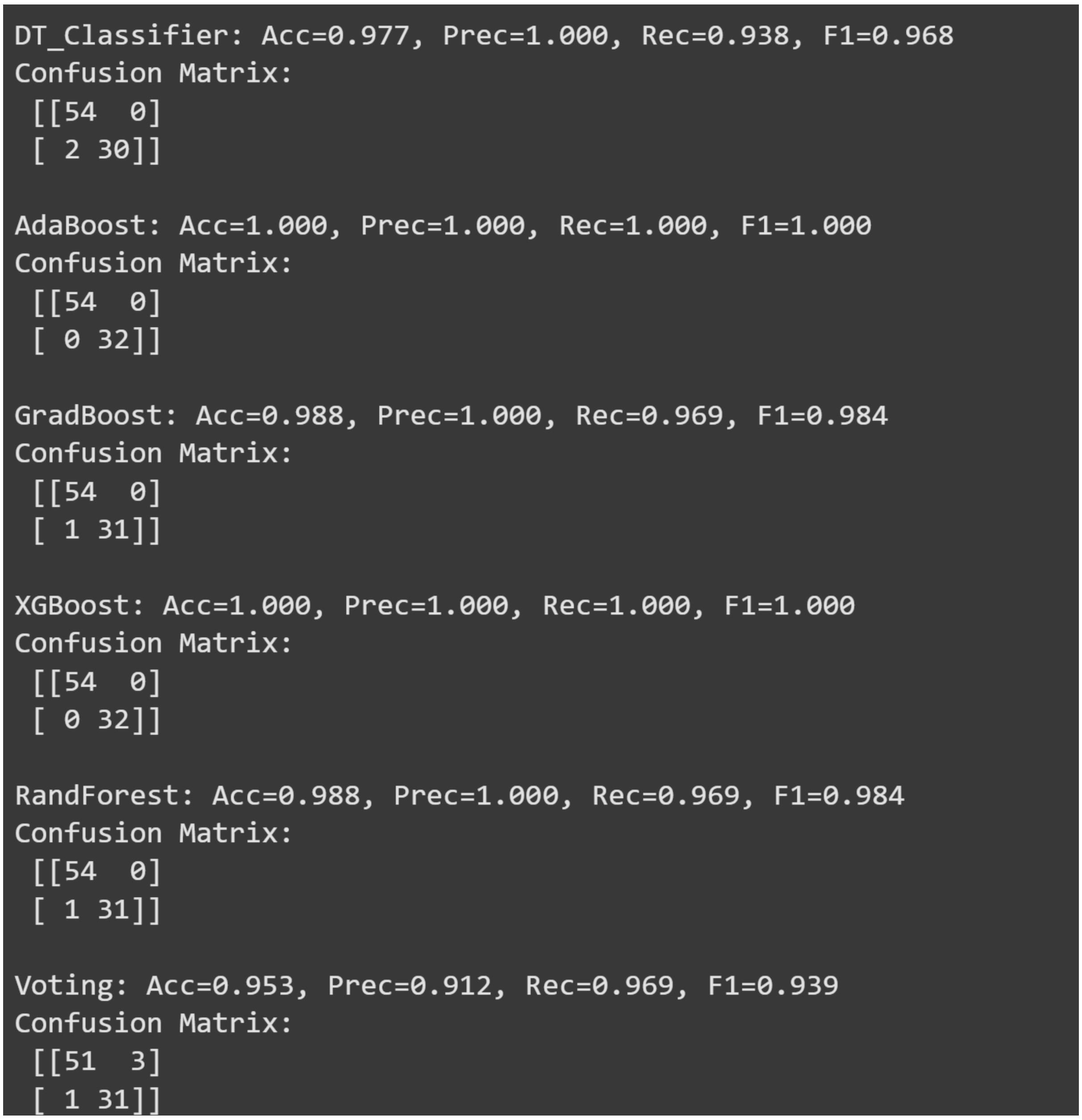


```
🔎 Tuning DT_Classifier...
Best params: {'criterion': 'gini', 'max_depth': 4}
🔎 Tuning AdaBoost...
Best params: {'learning_rate': 1.0, 'n_estimators': 100}
🔎 Tuning GradBoost...
Best params: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 150}
🔎 Tuning XGBoost...
Best params: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 150}
🔎 Tuning RandForest...
Best params: {'criterion': 'gini', 'max_depth': None, 'n_estimators': 200}
🔎 Tuning Voting...
Best params: {'voting': 'soft'}
```

Figure 4: Best Hyperparameters from GridSearchCV

5.3 Model Performance Metrics

The final metrics and confusion matrices for each model on the test set are presented here.



```
DT_Classifier: Acc=0.977, Prec=1.000, Rec=0.938, F1=0.968
Confusion Matrix:
 [[54  0]
 [ 2 30]]

AdaBoost: Acc=1.000, Prec=1.000, Rec=1.000, F1=1.000
Confusion Matrix:
 [[54  0]
 [ 0 32]]

GradBoost: Acc=0.988, Prec=1.000, Rec=0.969, F1=0.984
Confusion Matrix:
 [[54  0]
 [ 1 31]]

XGBoost: Acc=1.000, Prec=1.000, Rec=1.000, F1=1.000
Confusion Matrix:
 [[54  0]
 [ 0 32]]

RandForest: Acc=0.988, Prec=1.000, Rec=0.969, F1=0.984
Confusion Matrix:
 [[54  0]
 [ 1 31]]

Voting: Acc=0.953, Prec=0.912, Rec=0.969, F1=0.939
Confusion Matrix:
 [[51  3]
 [ 1 31]]
```

Figure 5: Performance Metrics and Confusion Matrices

The ROC curves illustrate the trade-off between the true positive rate and false positive rate for each model.

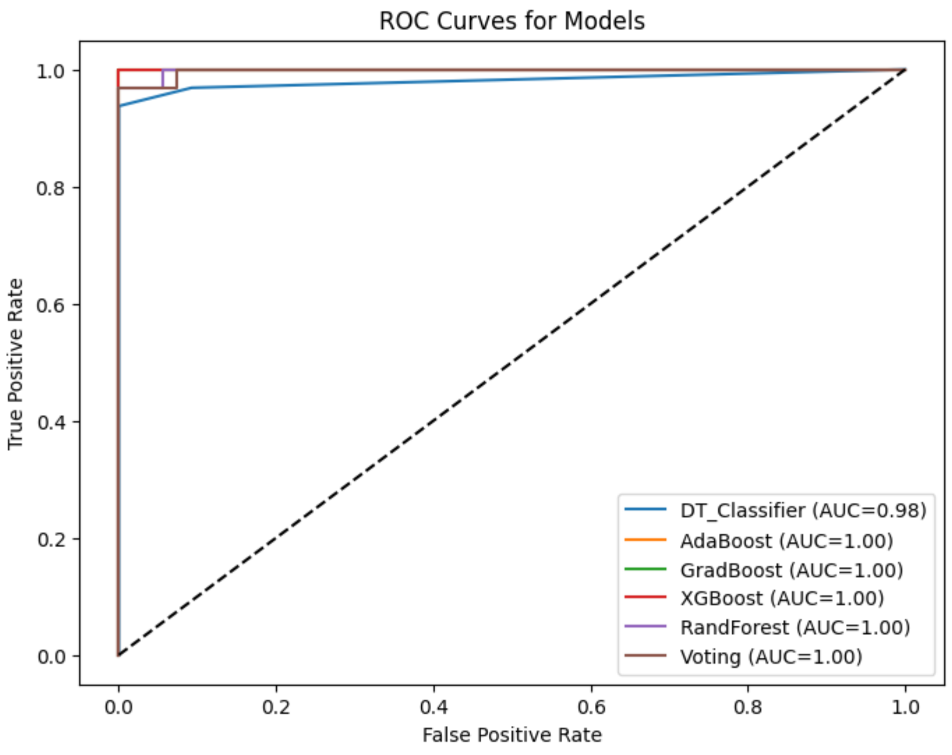


Figure 6: ROC Curves for All Models

6 Conclusion

Based on the results, AdaBoost and XGBoost achieved a perfect accuracy of 1.000 and an F1-score of 1.000, along with an AUC of 1.00, indicating perfect classification on the test set. This suggests these models are highly effective for this specific dataset and do not show signs of overfitting. In comparison, the Decision Tree Classifier also performed very well with a high accuracy and F1 score, but the ensemble methods demonstrated slightly superior performance. The tuning process was beneficial for all models, leading to strong results. The Voting Classifier, while performing well, did not outperform the best individual models.