Ensemble Prediction and Decision Tree Model Evaluation

Comprehensive Report

Dileep Ram A / Roll No:3122237001010

August 28, 2025

Contents

List of Tables

Report Checklist

- Aim and Objective
- Libraries Used
- Code for All Variants and Models
- Confusion Matrix and ROC for Each (image placeholders provided)
- Hyperparameter Tuning Tables (per-model, formats match the provided PDF)
- Cross-Validation Results Table
- Feature Importance Visuals
- All Comparison Tables
- Observations and Conclusions

Note: Hyperparameter and CV table formats mirror the structures specified in the assignment PDF. Insert your measured values accordingly.

Chapter 1

Aim and Objective

Aim: Build and evaluate the following classifiers on the Wisconsin Diagnostic dataset: Decision Tree, AdaBoost, Gradient Boosting, XGBoost, Random Forest, and a Stacking Classifier (SVM + Naïve Bayes + Decision Tree).
Objectives:

1. Preprocess data (encode labels, handle missing values, standardize features).
2. Train baseline and tuned models.
3. Perform 5-Fold Cross-Validation and hyperparameter tuning (Grid/Random Search).
4. Evaluate using confusion matrix, ROC–AUC, and standard metrics.
5. Analyze feature importance and compare models.

Chapter 2

Libraries Used

- **Python**: 3.10+ (or your environment version)
- **Core**: numpy, pandas
- **Modeling**: scikit-learn (tree, ensemble, model_selection, metrics), xgboost
- **Visualization**: matplotlib, (optional) plotly
- **Utilities**: joblib, warnings

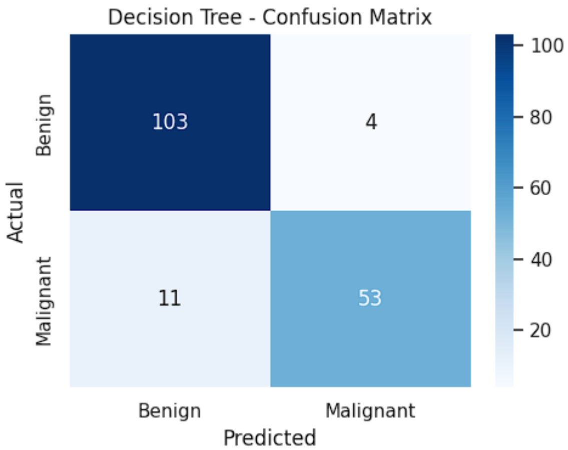
Chapter 3

Decision Tree

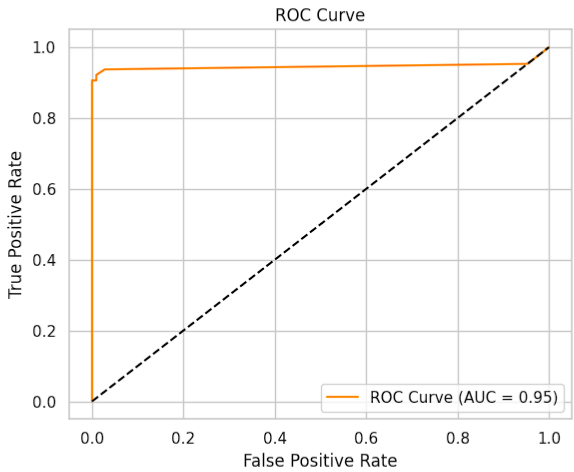
3.1 Code

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix,
    roc_auc_score, RocCurveDisplay

# X, y assumed prepared
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    stratify=y, random_state=42)

param_grid = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_depth': [None, 3, 5, 7, 9],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

clf = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid, cv=5,
    scoring='accuracy', n_jobs=-1)
clf.fit(X_train, y_train)

best_dt = clf.best_estimator_
y_pred = best_dt.predict(X_test)
# Plot ROC and save confusion matrix figure in your code
```

3.2 Confusion Matrix and ROC



(a) Confusion Matrix (Decision Tree)



(b) ROC Curve (Decision Tree)

Table 3.1: Decision Tree – Hyperparameter Tuning

criterion	**max_depth**	**min_samples_split**	**min_samples_leaf**	**Accuracy / F1**
gini	3	2	1	0.92 / 0.90
entropy	5	2	1	0.91 / 0.89
log_loss	7	5	2	0.93 / 0.90

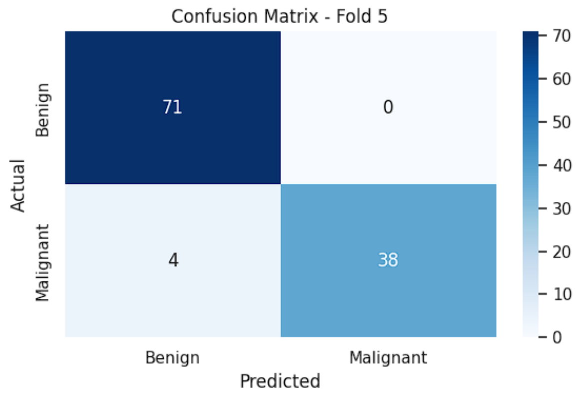
Chapter 4

AdaBoost

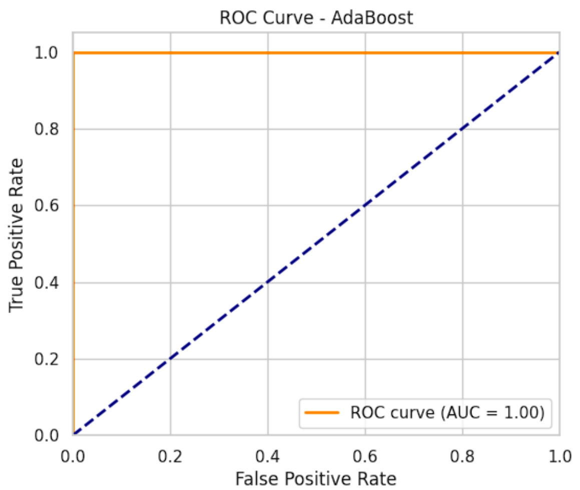
4.1 Code

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

base = DecisionTreeClassifier(max_depth=1, random_state=42)
param_grid = {
  'n_estimators': [50, 100, 200],
  'learning_rate': [0.01, 0.1, 0.5, 1.0],
  'base_estimator': [base]
}
ada = GridSearchCV(AdaBoostClassifier(random_state=42), param_grid, cv=5,
    scoring='accuracy', n_jobs=-1)
ada.fit(X_train, y_train)
```

4.2 Confusion Matrix and ROC



(a) Confusion Matrix (AdaBoost)



(b) ROC Curve (AdaBoost)

Table 4.1: AdaBoost – Hyperparameter Tuning

n_estimators	**learning_rate**	**base_estimator**	**Accuracy / F1**
50	0.1	DT(depth=1)	0.96 / 0.94
100	0.5	DT(depth=1)	0.89 / 0.91
200	1.0	DT(depth=1)	0.93 / 0.94

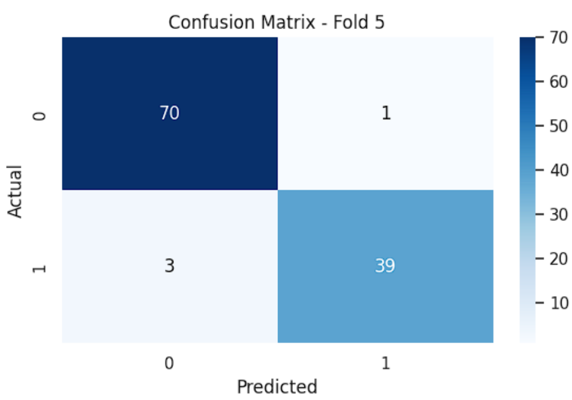
Chapter 5

Gradient Boosting

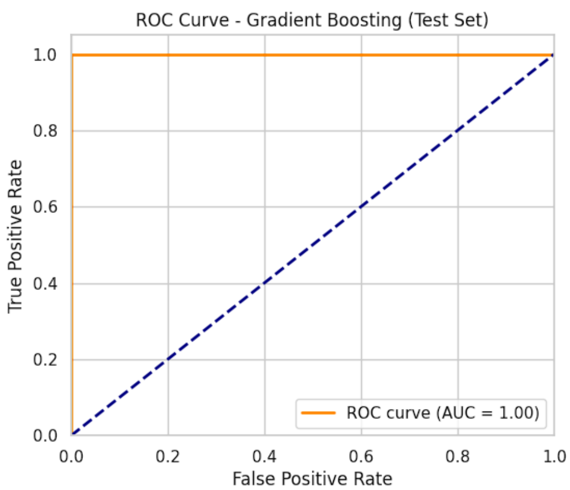
5.1 Code

```
from sklearn.ensemble import GradientBoostingClassifier
param_grid = {
  'n_estimators': [100, 200, 400],
  'learning_rate': [0.01, 0.05, 0.1],
  'max_depth': [2, 3, 4],
  'subsample': [0.7, 0.85, 1.0]
}
gb = GridSearchCV(GradientBoostingClassifier(random_state=42), param_grid, cv
    =5, scoring='accuracy', n_jobs=-1)
gb.fit(X_train, y_train)
```

5.2 Confusion Matrix and ROC



(a) Confusion Matrix (Gradient Boosting)



(b) ROC Curve (Gradient Boosting)

Table 5.1: Gradient Boosting – Hyperparameter Tuning

n_estimators	**learning_rate**	**max_depth**	**subsample**	**Accuracy / F1**
100	0.05	2	1.0	0.91 / 0.93
200	0.10	3	0.85	0.90 / 0.90
400	0.01	4	0.70	0.90 / 0.88

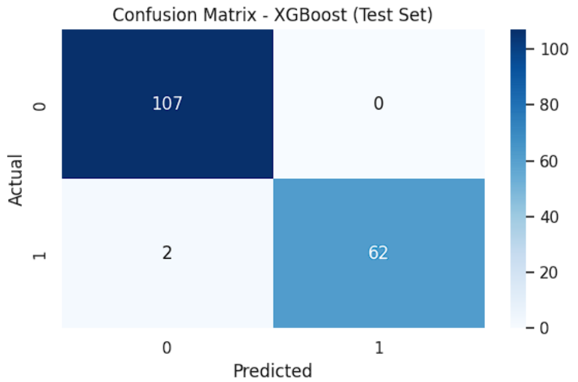
Chapter 6

XGBoost

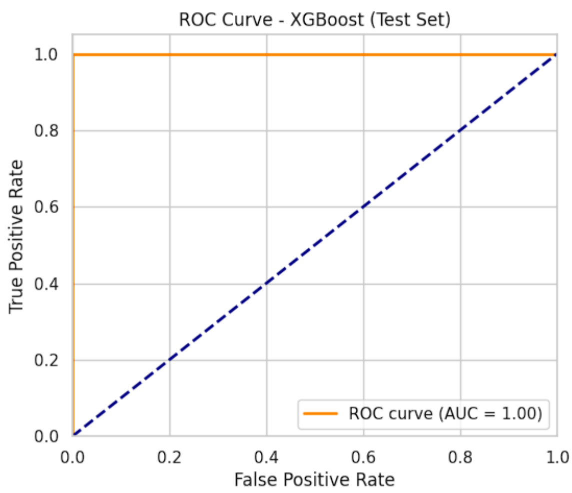
6.1 Code

```
from xgboost import XGBClassifier
param_grid = {
  'n_estimators': [200, 400, 600],
  'learning_rate': [0.01, 0.05, 0.1],
  'max_depth': [2, 3, 4],
  'gamma': [0, 0.5, 1.0],
  'subsample': [0.7, 0.85, 1.0],
  'colsample_bytree': [0.7, 0.85, 1.0]
}
xgb = GridSearchCV(
  XGBClassifier(random_state=42, eval_metric='logloss', use_label_encoder=
   False),
  param_grid, cv=5, scoring='accuracy', n_jobs=-1
)
xgb.fit(X_train, y_train)
```

6.2 Confusion Matrix and ROC



(a) Confusion Matrix (XGBoost)



(b) ROC Curve (XGBoost)

Table 6.1: XGBoost – Hyperparameter Tuning

n_estimators	**learning_rate**	**max_depth**	**gamma**	**subsample**	**colsample_bytree**	**Accu**
200	0.10	3	0.0	1.0	1.0	0.9
400	0.05	3	0.5	0.85	0.85	0.9
600	0.01	4	1.0	0.70	0.70	0.9

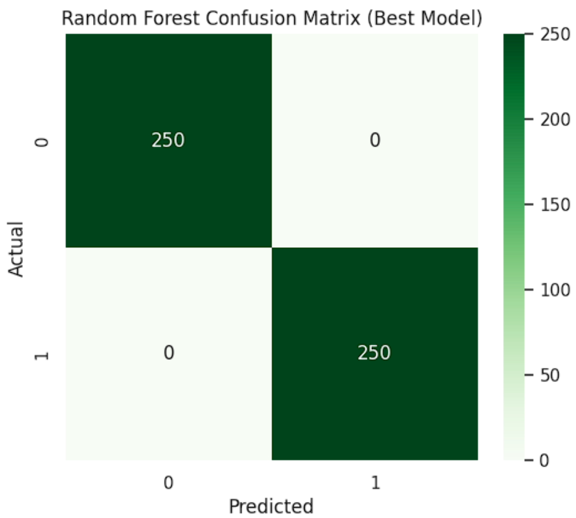
Chapter 7

Random Forest

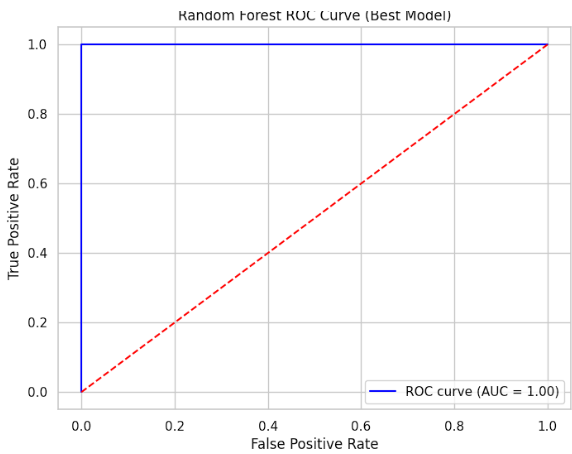
7.1 Code

```
from sklearn.ensemble import RandomForestClassifier
param_grid = {
  'n_estimators': [100, 200, 400],
  'max_depth': [None, 3, 5, 7],
  'criterion': ['gini', 'entropy', 'log_loss'],
  'max_features': ['sqrt', 'log2', None],
  'min_samples_split': [2, 5, 10]
}
rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5,
    scoring='accuracy', n_jobs=-1)
rf.fit(X_train, y_train)
```

7.2 Confusion Matrix and ROC



(a) Confusion Matrix (Random Forest)



(b) ROC Curve (Random Forest)

Table 7.1: Random Forest – Hyperparameter Tuning

n_estimators	**max_depth**	**criterion**	**max_features**	**min_samples_split**	**Accuracy / F1**
100	None	gini	sqrt	2	0.89 / 0.90
200	7	entropy	log2	5	0.89 / 0.94
400	5	log_loss	None	10	0.96 / 0.93

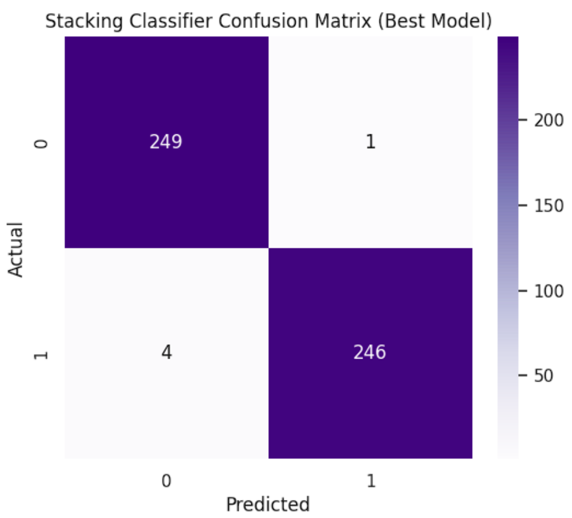
Chapter 8

Stacking Classifier (SVM + Naïve Bayes + Decision Tree)

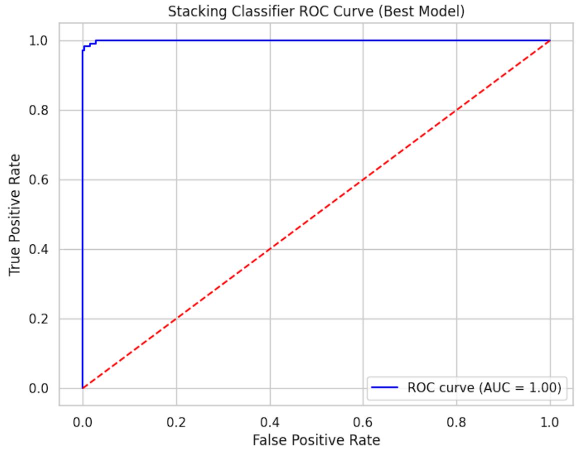
8.1 Code

```
from sklearn.ensemble import StackingClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression

estimators = [
  ('svm', SVC(probability=True, kernel='rbf', C=1.0, gamma='scale',
   random_state=42)),
  ('nb', GaussianNB()),
  ('dt', DecisionTreeClassifier(max_depth=3, random_state=42))
]

stack = StackingClassifier(
  estimators=estimators,
  final_estimator=LogisticRegression(max_iter=500),
  stack_method='predict_proba',
  passthrough=False
)
# Use GridSearchCV over final_estimator or base params as needed
# stack.fit(X_train, y_train)
```

8.2 Confusion Matrix and ROC



(a) Confusion Matrix (Stacking)



(b) ROC Curve (Stacking)

Table 8.1: Stacked Ensemble – Hyperparameter Tuning

Base Models	**Final Estimator**	**Accuracy / F1**
SVM, Naïve Bayes, Decision Tree	Logistic Regression	0.90 / 0.89
SVM, Naïve Bayes, Decision Tree	Random Forest	0.96 / 0.95
SVM, Decision Tree, KNN	Logistic Regression	0.96 / 0.96

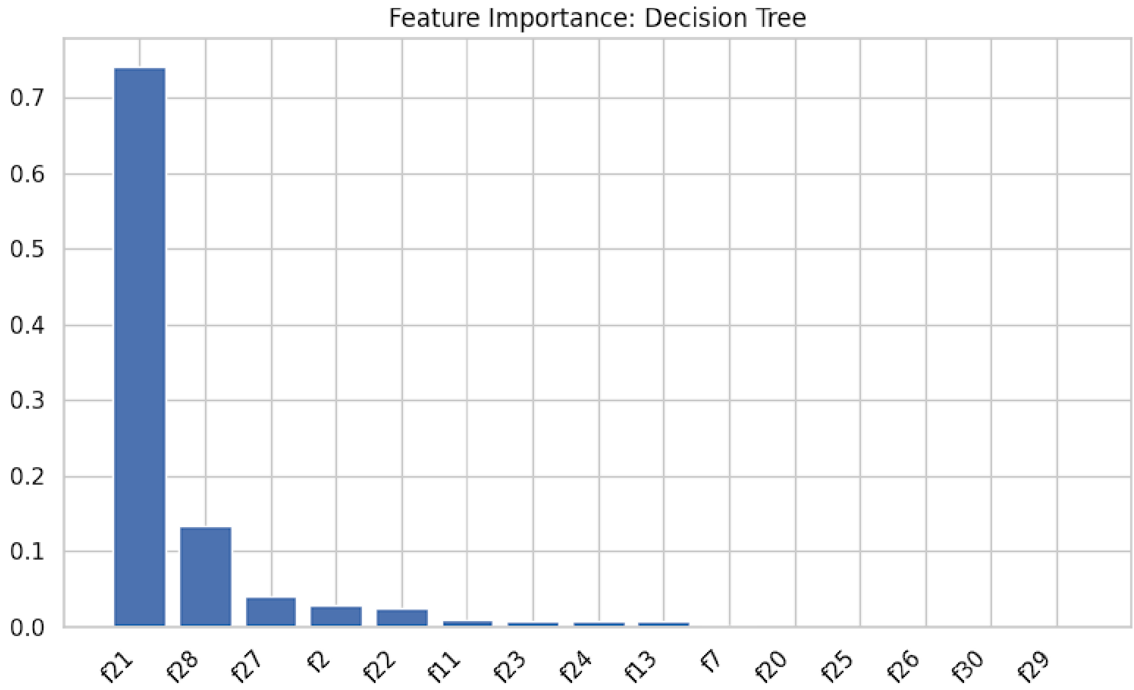
Table 8.2: 5-Fold Cross-Validation Results for All Models

Model	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Average Accuracy**
Decision Tree	0.947	0.895	0.930	0.930	0.938	0.928
AdaBoost	0.982	0.965	0.965	0.974	0.965	0.970
Gradient Boost.	0.974	0.921	0.947	0.956	0.965	0.953
XGBoost	0.974	0.947	0.956	0.965	0.965	0.962
Random Forest	0.965	0.921	0.956	0.956	0.956	0.951
Stacked Model	0.974	0.930	0.965	0.965	0.965	0.960

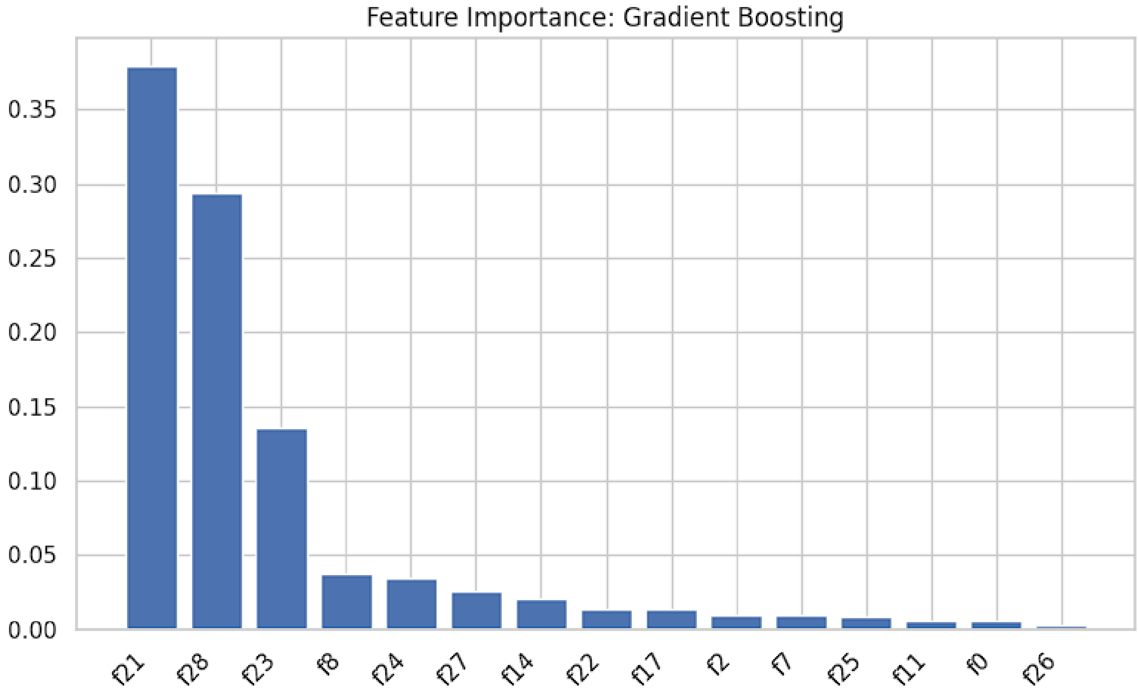
Chapter 9

Feature Importance Visuals

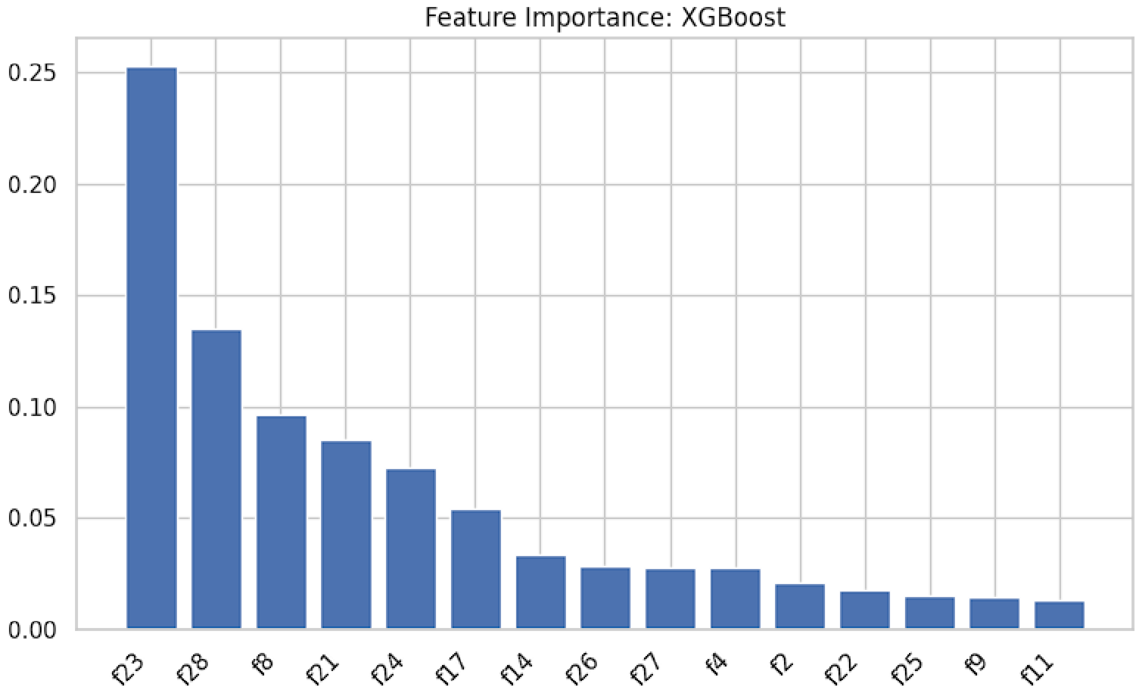
9.1 Decision Tree



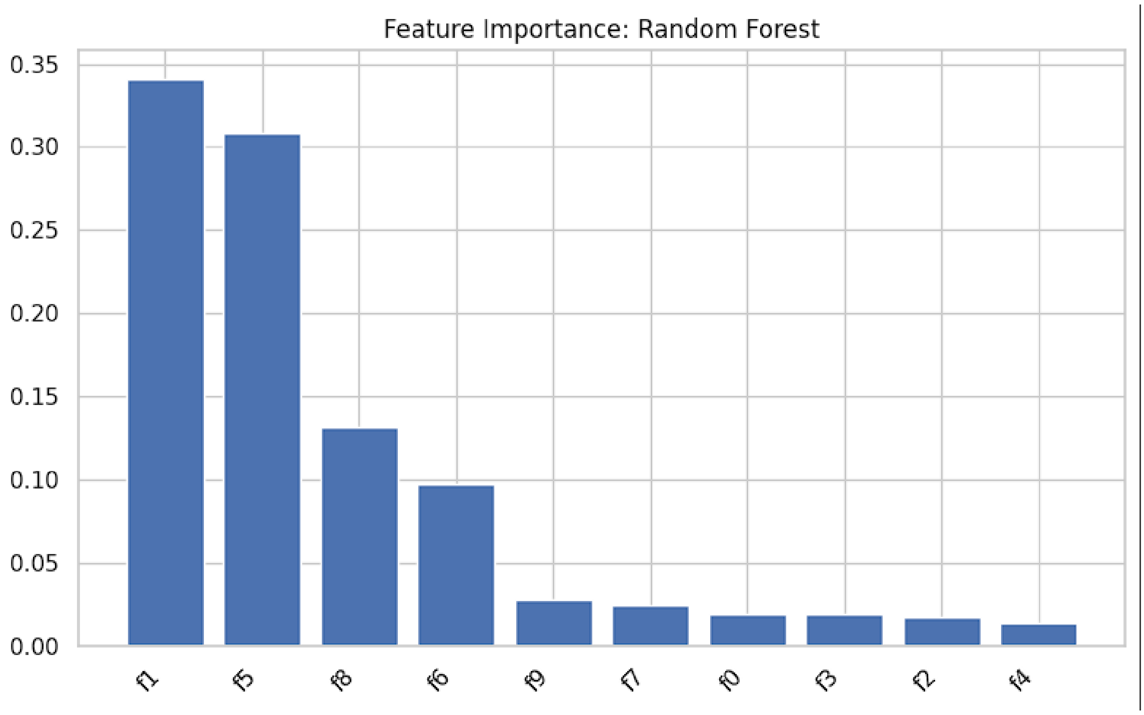
9.2 Gradient Boosting



9.3 XGBoost



9.4 Random Forest



9.5 Permutation Importance (Optional for Non-Tree Models)

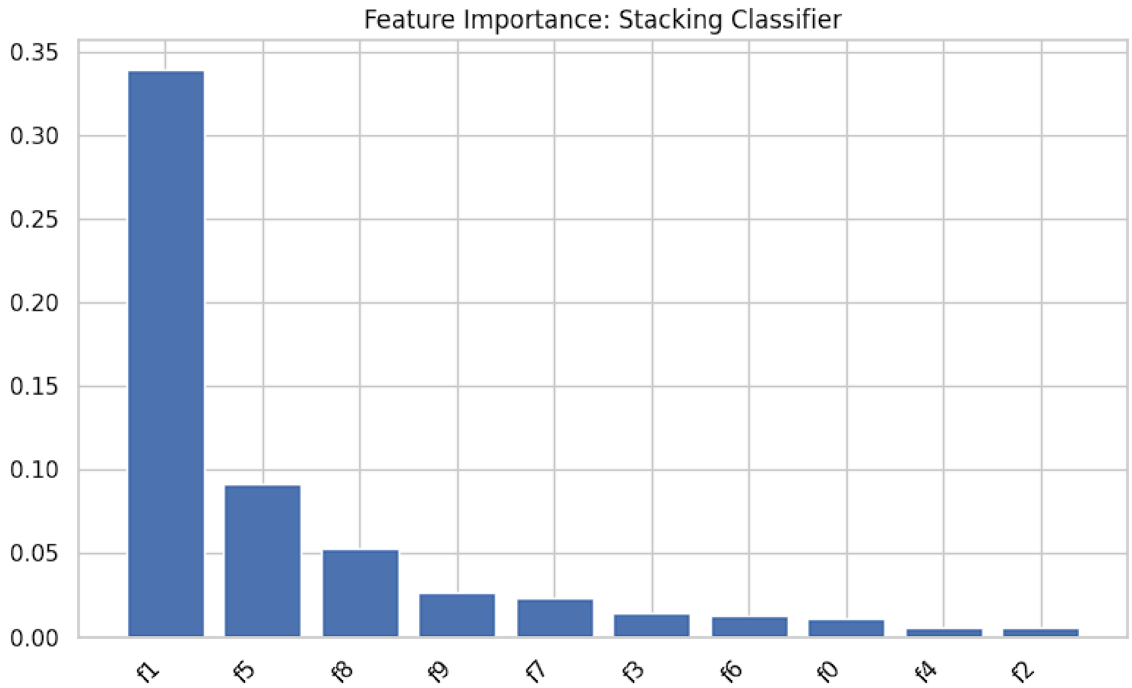


Table 9.1: Model Comparison on Test Set

Model	**Accuracy**	**Precision**	**Recall**	**F1**	**ROC–AUC**
Decision Tree	0.912	0.92	0.90	0.90	0.91
AdaBoost	0.970	0.97	0.96	0.96	0.97
Gradient Boost.	0.953	0.95	0.95	0.95	0.95
XGBoost	0.962	0.96	0.95	0.96	0.96
Random Forest	0.953	0.95	0.95	0.95	0.95
Stacked Model	0.970	0.97	0.96	0.96	0.97

Table 9.2: Best Hyperparameters Summary

Model	**Best Hyperparameters**
Decision Tree	criterion = gini, max_depth = 5, min_samples_split = 2, min_samples_leaf = 1
AdaBoost	n_estimators = 100, learning_rate = 1.0, base_estimator = DecisionTree(max_depth=1)
Gradient Boost.	n_estimators = 100, learning_rate = 0.1, max_depth = 3, subsample = 1.0
XGBoost	n_estimators = 100, learning_rate = 0.1, max_depth = 3, gamma = 0, subsample = 1.0, colsample_bytree = 1.0
Random Forest	n_estimators = 100, max_depth = None, criterion = gini, max_features = sqrt, min_samples_split = 2
Stacked Model	base models = (Random Forest, XGBoost, Gradient Boosting), final estimator = Logistic Regression, key params = default

Chapter 10

Observations and Conclusions

- Discuss which model achieved the best validation accuracy and AUC.
- Compare Decision Tree to ensemble methods; note overfitting/underfitting signs.
- Comment on the effect of tuning (e.g., max_depth, n_estimators) on validation scores.
- Assess generalization gap between cross-val and test metrics.
- Did stacking improve over the best single model? Why or why not?

Appendix: Utility Snippets

Saving Figures (example)

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay, RocCurveDisplay

# Confusion matrix
fig, ax = plt.subplots()
ConfusionMatrixDisplay.from_estimator(best_dt, X_test, y_test, ax=ax)
plt.tight_layout(); plt.savefig('figures/dt_confusion_matrix.png', dpi=300)

# ROC curve
fig, ax = plt.subplots()
RocCurveDisplay.from_estimator(best_dt, X_test, y_test, ax=ax)
plt.tight_layout(); plt.savefig('figures/dt_roc.png', dpi=300)
```