Experiment-4
Ensemble Prediction and Decision Tree Model Evaluation

ICS1512 & Machine Learning Algorithms Laboratory

Siddharth M

3122237001049

Contents

1 Aim/Objective

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

2 Libraries Used

Library	Purpose
pandas (pd)	Data manipulation and analysis
numpy (np)	Numerical operations and array handling
seaborn (sns)	Statistical data visualization
matplotlib.pyplot (plt)	Plotting and visualization
sklearn.model_selection	for splitting datasets
sklearn.preprocessing	for feature scaling and encoding
sklearn.metrics	for evaluation metrics
warnings	To ignore warning messages

3 Code for All Variants and Models

3.1 Preprocessing

```python
# 1. Import modules
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler, Normalizer, LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score,
    precision_score, recall_score, f1_score, roc_auc_score,roc_curve, auc
import warnings
warnings.filterwarnings('ignore')

# Column names for the WDBC dataset based on UCI repository documentation
columns = [
    'ID', 'Diagnosis',
    'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean',
    'compactness_mean', 'concavity_mean', 'concave_points_mean', 'symmetry_mean', '
    fractal_dimension_mean',
    'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
    'compactness_se', 'concavity_se', 'concave_points_se', 'symmetry_se', '
    fractal_dimension_se',
    'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst',
    'compactness_worst', 'concavity_worst', 'concave_points_worst', 'symmetry_worst', '
    fractal_dimension_worst'
]

# Reload with correct column names
df = pd.read_csv('wdbc.csv', header=None, names=columns)

# 3. EDA
# Handle missing values
df = df.dropna(thresh=df.shape[1] * 0.5)
df = df.fillna(df.median(numeric_only=True))

# Handle outliers using IQR (numeric columns)
for col in df.select_dtypes(include=np.number).columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    df[col] = np.where(df[col] < lower, lower, df[col])
    df[col] = np.where(df[col] > upper, upper, df[col])
```

```python

# 4. Separate features and label
label_column = 'Diagnosis'

# Encode categorical target if necessary
if df[label_column].dtype == 'object':
    le = LabelEncoder()
    df[label_column] = le.fit_transform(df[label_column])

features = df.drop(label_column, axis=1)
labels = df[label_column]

# One-hot encode categorical features
features = pd.get_dummies(features, drop_first=True)

# 5. Feature scaling
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# 6. Data splitting
X_train, X_temp, y_train, y_temp = train_test_split(features_scaled, labels, test_size=0.4,
    random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state
    =42)
```

3.2 DecisionTree Classifier

```python
from sklearn.tree import DecisionTreeClassifier
import time
start_time= time.time()
model_dt = DecisionTreeClassifier()
model_dt.fit(X_train, y_train)
end_time = time.time()
training_time_dt = end_time - start_time
print(f"Training time for Decision Tree: {training_time_dt} seconds")
```

Output:

```
Training time for Decision Tree: 0.01593756675720215 seconds
```

```python
from sklearn import tree
plt.figure(figsize=(15,10))
tree.plot_tree(model_dt,filled=True)
```

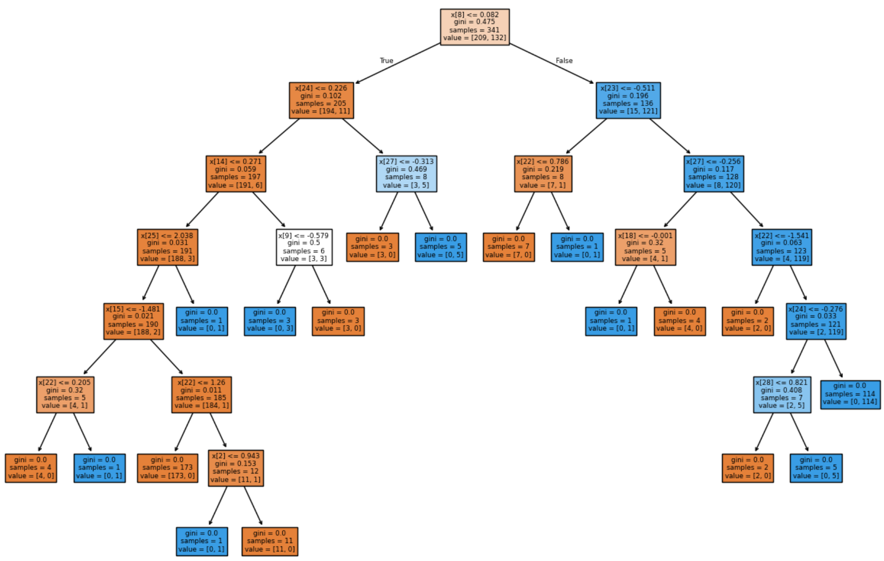


Figure 1: Decision Tree

```python
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import make_scorer, f1_score

param_grid = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Cross-validation strategy
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
grid_search_dt = GridSearchCV(
    estimator=model_dt,
    param_grid=param_grid,
    cv=cv,
    scoring={'accuracy': 'accuracy', 'f1': make_scorer(f1_score, average='weighted')},
    refit='accuracy',
    n_jobs=-1
)

start_time = time.time()
grid_search_dt.fit(X_train, y_train)
end_time = time.time()

training_time = end_time - start_time

# Results
print(f"Best Parameters: {grid_search_dt.best_params_}")
print(f"Best CV Score: {grid_search_dt.best_score_:.4f}")
print(f"Training Time: {training_time:.4f} seconds")
```

Listing 1: Cross-Validation and Hyperparameter Tuning

Output:

```
Best Parameters: {'criterion': 'entropy', 'max_depth': 5,
'min_samples_leaf': 2, 'min_samples_split': 5}
Best CV Score: 0.9501
Training Time: 19.7115 seconds
```

```python
model_dt = grid_search_dt.best_estimator_
y_pred_dt = model_dt.predict(X_test)

print("Decision Tree Classifier Metrics:")
print(classification_report(y_test, y_pred_dt))
print(f"Accuracy: {accuracy_score(y_test, y_pred_dt)}")
cm= confusion_matrix(y_test, y_pred_dt)
print(f1_score(y_test, y_pred_dt, average='weighted'))

from sklearn.metrics import ConfusionMatrixDisplay
lbs = le.classes_
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=lbs)
disp.plot(cmap='Blues')
plt.show()

y_proba = model_dt.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - XGBoost Classifier')
plt.legend(loc='lower right')
plt.grid(True)
```

```
plt.show()
```

Listing 2: Evaluation

Output:

```
Decision Tree Classifier Metrics:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97        76
           1       0.97      0.92      0.95        38

    accuracy                           0.96       114
   macro avg       0.97      0.95      0.96       114
weighted avg       0.97      0.96      0.96       114

Accuracy: 0.9649122807017544
0.9646659646659645
```

3.3 AdaBoost Classifier

```
from sklearn.ensemble import AdaBoostClassifier
model_ada = AdaBoostClassifier(random_state=42)
st = time.time()
model_ada.fit(X_train, y_train)
et = time.time()
training_time_ada = et - st
print(f"Training time for AdaBoost: {training_time_ada} seconds")

param_grid = {
    'n_estimators': [50, 100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1, 0.5, 1],
    'algorithm': ['SAMME', 'SAMME.R']
}

# Cross-validation strategy
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
# Run GridSearch with Accuracy & F1 scoring
grid_search_ada = GridSearchCV(
    estimator=model_ada,
    param_grid=param_grid,
    cv=cv,
    scoring={'accuracy': 'accuracy', 'f1': make_scorer(f1_score, average='weighted')},
    refit='accuracy',
    n_jobs=-1
)
# Measure training time
start_time = time.time()
grid_search_ada.fit(X_train, y_train)
end_time = time.time()

# Best parameters & score
print(f"Best Parameters: {grid_search_ada.best_params_}")
print(f"Best CV Score: {grid_search_ada.best_score_:.4f}")
print(f"Training Time: {end_time - start_time:.4f} seconds")
```

Listing 3: AdaBoost Model Building

```
Training time for AdaBoost: 0.28496885299682617 seconds
Best Parameters: {'algorithm': 'SAMME.R', 'learning_rate': 1,
'n_estimators': 200}
Best CV Score: 0.9736
Training Time: 7.6901 seconds
```

```python
model_ada = grid_search_ada.best_estimator_
y_pred_ada = model_ada.predict(X_test)

print("AdaBoost Classifier Metrics:")
print(classification_report(y_test, y_pred_ada))
print(f"Accuracy: {accuracy_score(y_test, y_pred_ada)}")
cm= confusion_matrix(y_test, y_pred_ada)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=lbs)
disp.plot(cmap='Blues')
plt.show()

y_proba = model_ada.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - XGBoost Classifier')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

Listing 4: Evaluation

Output:

```
AdaBoost Classifier Metrics:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        76
           1       1.00      0.95      0.97        38

    accuracy                           0.98       114
   macro avg       0.99      0.97      0.98       114
weighted avg       0.98      0.98      0.98       114

Accuracy: 0.9824561403508771
```

3.4 Gradient Boosting

```python
from sklearn.ensemble import GradientBoostingClassifier
model_gradient = GradientBoostingClassifier(random_state=42)
st = time.time()
model_gradient.fit(X_train, y_train)
et = time.time()
training_time_gradient = et - st
print(f"Training time for Gradient Boosting: {training_time_gradient} seconds")
```

Listing 5: Gradient Boosting Model

Output:

```
Training time for Gradient Boosting: 0.4041452407836914 seconds
```

```python
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import make_scorer, f1_score
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5],
```

```
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.8, 1.0]
}

# Cross-validation strategy
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
grid_search_grad = GridSearchCV(
    estimator=model_gradient,
    param_grid=param_grid,
    cv=cv,
    scoring={'accuracy': 'accuracy', 'f1': make_scorer(f1_score, average='weighted')},
    refit='accuracy',
    n_jobs=-1
)

# Measure training time
start_time = time.time()
grid_search_grad.fit(X_train, y_train)
end_time = time.time()

# Best parameters & score
print(f"Best Parameters: {grid_search_grad.best_params_}")
print(f"Best CV Score: {grid_search_grad.best_score_:.4f}")
print(f"Training Time: {end_time - start_time:.4f} seconds")
```

Listing 6: Hyper-Parameter Tuning

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'min_samples_leaf':
 1, 'min_samples_split': 2, 'n_estimators': 200, 'subsample': 0.8}
 Best CV Score: 0.9736
 Training Time: 101.8133 seconds
```

```
model_gradient = grid_search_grad.best_estimator_
y_pred_gradient = model_gradient.predict(X_test)

lbs = le.classes_
print("Gradient Boosting Classifier Metrics:")
print(classification_report(y_test, y_pred_gradient))
print(f"Accuracy: {accuracy_score(y_test, y_pred_gradient)}")
cm = confusion_matrix(y_test, y_pred_gradient)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=lbs)
disp.plot(cmap='Blues')
plt.show()

y_proba = model_gradient.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Gradient Boost Classifier')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

Listing 7: Evaluation

```
Gradient Boosting Classifier Metrics:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96        76
```

```
           1       0.94      0.89      0.92        38

    accuracy                           0.95       114
   macro avg       0.95      0.93      0.94       114
weighted avg       0.95      0.95      0.95       114

Accuracy: 0.9473684210526315
```

3.5 XGBoost Classifier

```python
from xgboost import XGBClassifier
model_xgb = XGBClassifier(
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)
st = time.time()
model_xgb.fit(X_train, y_train)
et = time.time()
training_time_xgb = et - st
print(f"Training time for XGBoost: {training_time_xgb} seconds")
```

Listing 8: Model Training

```
Training time for XGBoost: 1.8101122379302979 seconds
```

```python
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5, 6],
    'gamma': [0, 0.1, 0.3, 0.5],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
grid_search_xgb = GridSearchCV(
    estimator=model_xgb,
    param_grid=param_grid,
    cv=cv,
    scoring={'accuracy': 'accuracy', 'f1': make_scorer(f1_score, average='weighted')},
    refit='accuracy',
    n_jobs=-1
)
# Measure training time
start_time = time.time()
grid_search_xgb.fit(X_train, y_train)
end_time = time.time()

# Results
print(f"Best Parameters: {grid_search_xgb.best_params_}")
print(f"Best CV Score: {grid_search_xgb.best_score_:.4f}")
print(f"Training Time: {end_time - start_time:.4f} seconds")
```

Listing 9: Hyperparameter Tuning

```
Best Parameters: {'colsample_bytree': 0.8, 'gamma': 0, 'learning_rate': 0.2,
'max_depth': 4, 'n_estimators': 200, 'subsample': 1.0}
Best CV Score: 0.9707
Training Time: 21.4219 seconds
```

```python
model_xgb = grid_search_xgb.best_estimator_
y_pred_xgb = model_xgb.predict(X_test)

```

```
print("XGBoost Classifier Metrics:")
print(classification_report(y_test, y_pred_xgb))
print(f"Accuracy: {accuracy_score(y_test, y_pred_xgb)}")
print("f1_score: ",f1_score(y_test, y_pred_xgb, average='weighted'))
cm = confusion_matrix(y_test, y_pred_xgb)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=lbs)
disp.plot(cmap='Blues')
plt.show()

y_proba = model_xgb.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - XGBoost Classifier')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

Listing 10: Evaluation

```
XGBoost Classifier Metrics:
              precision    recall  f1-score   support

           0       0.97      0.96      0.97        76
           1       0.92      0.95      0.94        38

    accuracy                           0.96       114
   macro avg       0.95      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

Accuracy: 0.956140350877193
f1_score:  0.9562799231673403
```

3.6 Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
model_rf = RandomForestClassifier(random_state=42)
st = time.time()
model_rf.fit(X_train, y_train)
et = time.time()
training_time_rf = et - st
print(f"Training time for Random Forest: {training_time_rf} seconds")
```

Listing 11: Model Training

```
Training time for Random Forest: 0.14823627471923828 seconds
```

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10, 15, 20],
    'criterion': ['gini', 'entropy'],
    'max_features': ['sqrt', 'log2'],
    'min_samples_split': [2, 5, 10]
}
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
grid_search_rf = GridSearchCV(
```

```
    estimator=model_rf,
    param_grid=param_grid,
    cv=cv,
    scoring={'accuracy': 'accuracy', 'f1': make_scorer(f1_score, average='weighted')},
    refit='accuracy',
    n_jobs=-1
)
# Measure training time
start_time = time.time()
grid_search_rf.fit(X_train, y_train)
end_time = time.time()

# Results
print(f"Best Parameters: {grid_search_rf.best_params_}")
print(f"Best CV Score: {grid_search_rf.best_score_:.4f}")
print(f"Training Time: {end_time - start_time:.4f} seconds")
```

Listing 12: Hyperparameter Tuning

```
Best Parameters: {'criterion': 'entropy', 'max_depth': None,
'max_features': 'sqrt', 'min_samples_split': 5, 'n_estimators': 200}
Best CV Score: 0.9559
Training Time: 12.2577 seconds
```

```
model_rf = grid_search_rf.best_estimator_
y_pred_rf = model_rf.predict(X_test)

print("Random Forest Classifier Metrics:")
print(classification_report(y_test, y_pred_rf))
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf)}")
cm = confusion_matrix(y_test, y_pred_rf)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=lbs)
disp.plot(cmap='Blues')
plt.show()

y_proba_rf = model_rf.predict_proba(X_test)[:, 1]
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_proba_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot ROC curve
plt.figure(figsize=(6, 5))
plt.plot(fpr_rf, tpr_rf, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc_rf:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest Classifier')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

Listing 13: Evaluation

```
Random Forest Classifier Metrics:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        76
           1       1.00      0.92      0.96        38

    accuracy                           0.97       114
   macro avg       0.98      0.96      0.97       114
weighted avg       0.97      0.97      0.97       114

Accuracy: 0.9736842105263158
```

3.7 Stacking Classifier (SVM + Naive Bayes + Decision Tree)

```python

from sklearn.ensemble import StackingClassifier, RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import (
    accuracy_score,
    f1_score,
    classification_report,
    confusion_matrix,
    roc_curve,
    auc
)

# -------------------------
# DEFINE STACKED MODELS
# -------------------------

# 1) SVM, Naive Bayes, Decision Tree     Logistic Regression
model1 = StackingClassifier(
    estimators=[
        ("svm", make_pipeline(StandardScaler(), SVC(probability=True))),
        ("nb", GaussianNB()),
        ("dt", DecisionTreeClassifier(random_state=42))
    ],
    final_estimator=LogisticRegression(),
    cv=5,
    n_jobs=-1
)

# 2) SVM, Naive Bayes, Decision Tree     Random Forest
model2 = StackingClassifier(
    estimators=[
        ("svm", make_pipeline(StandardScaler(), SVC(probability=True))),
        ("nb", GaussianNB()),
        ("dt", DecisionTreeClassifier(random_state=42))
    ],
    final_estimator=RandomForestClassifier(random_state=42),
    cv=5,
    n_jobs=-1
)

# 3) SVM, Decision Tree, KNN     Logistic Regression
model3 = StackingClassifier(
    estimators=[
        ("svm", make_pipeline(StandardScaler(), SVC(probability=True))),
        ("dt", DecisionTreeClassifier(random_state=42)),
        ("knn", KNeighborsClassifier())
    ],
    final_estimator=LogisticRegression(),
    cv=5,
    n_jobs=-1
)
param_grid1 = {
    "svm__svc__C": [0.1, 1, 10],
    "svm__svc__kernel": ["linear", "rbf"],
    "dt__max_depth": [None, 5, 10, 20],
    "dt__criterion": ["gini", "entropy", "log_loss"],
    "final_estimator__C": [0.1, 1, 10],
    "final_estimator__solver": ["lbfgs", "liblinear"]
}
param_grid2 = {
    "svm__svc__C": [0.1, 1, 10],
    "svm__svc__kernel": ["linear", "rbf"],
    "dt__max_depth": [None, 5, 10, 20],
    "dt__criterion": ["gini", "entropy", "log_loss"],
    "final_estimator__n_estimators": [100, 200, 300],
```

```
    "final_estimator__max_depth": [None, 10, 20],
    "final_estimator__min_samples_split": [2, 5, 10]
}
param_grid3 = {
    "svm__svc__C": [0.1, 1, 10],
    "svm__svc__kernel": ["linear", "rbf"],
    "dt__max_depth": [None, 5, 10, 20],
    "knn__n_neighbors": [3, 5, 7, 9],
    "knn__weights": ["uniform", "distance"],
    "final_estimator__C": [0.1, 1, 10],
    "final_estimator__solver": ["lbfgs", "liblinear"]
}
param_grids = [param_grid1, param_grid2, param_grid3]

# -------------------------
# TRAIN & EVALUATE MODELS
# -------------------------

models = {
    "SVM+NB+DT     LogisticRegression": model1,
    "SVM+NB+DT     RandomForest": model2,
    "SVM+DT+KNN     LogisticRegression": model3
}
results = []

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

grid_no=0

for name, model in models.items():
    # Train
    model.fit(X_train, y_train)

    #hyperperparameter tuning

    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    grid_search_stack = GridSearchCV(
        estimator=model,
        param_grid=param_grids[grid_no],
        cv=cv,
        scoring={'accuracy': 'accuracy', 'f1': make_scorer(f1_score, average='weighted')},
        refit='accuracy',
        n_jobs=-1
    )
    grid_search_stack.fit(X_train, y_train)
    print(f"Best Parameters for {name}: {grid_search_stack.best_params_}")
    print(f"Best CV Score for {name}: {grid_search_stack.best_score_:.4f}")

    #k-fold cross validation
    cv_scores = cross_val_score(grid_search_stack.best_estimator_, X_val, y_val, cv=cv,
    scoring='accuracy')
    print(f"\nCross-Validation Scores for {name}:")
    for i, score in enumerate(cv_scores, 1):
        print(f"Fold {i} Accuracy: {score:.4f}")
    print(f"Average Accuracy: {np.mean(cv_scores):.4f}")

    grid_no+=1

    model = grid_search_stack.best_estimator_
    # Predictions
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]  # For ROC curve

    # Accuracy and F1
    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    results.append({"Model": name, "Accuracy": acc, "F1 Score": f1})

    # ROC Curve
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label=f"{name} (AUC = {roc_auc:.2f})")
```

```python

    # Reports
    print(f"\n=== {name} ===")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# --------------------------
# ROC PLOT
# --------------------------

plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison of Stacked Ensemble Models")
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()
plt.show()

# --------------------------
# RESULTS TABLE
# --------------------------

results_df = pd.DataFrame(results)
print("\nFinal Comparison Table:\n")
print(results_df)
```

Listing 14: Model Building and Evaluation

Output:

```
Best Parameters for SVM+NB+DT → LogisticRegression: {'dt__criterion': 'gini', 'dt__max_dep
Best CV Score for SVM+NB+DT → LogisticRegression: 0.9765

Cross-Validation Scores for SVM+NB+DT → LogisticRegression:
Fold 1 Accuracy: 0.9565
Fold 2 Accuracy: 0.9565
Fold 3 Accuracy: 0.9565
Fold 4 Accuracy: 0.9565
Fold 5 Accuracy: 0.9545
Average Accuracy: 0.9561

=== SVM+NB+DT → LogisticRegression ===
              precision    recall  f1-score   support

           0       0.99      0.97      0.98        76
           1       0.95      0.97      0.96        38

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114

Confusion Matrix:
 [[74  2]
 [ 1 37]]
Best Parameters for SVM+NB+DT → RandomForest: {'dt__criterion': 'entropy', 'dt__max_depth'
Best CV Score for SVM+NB+DT → RandomForest: 0.9765

Cross-Validation Scores for SVM+NB+DT → RandomForest:
Fold 1 Accuracy: 0.9130
```

```
Fold 2 Accuracy: 0.9565
Fold 3 Accuracy: 0.9565
Fold 4 Accuracy: 0.9130
Fold 5 Accuracy: 0.9545
Average Accuracy: 0.9387

=== SVM+NB+DT → RandomForest ===
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        76
           1       0.97      0.97      0.97        38

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114

Confusion Matrix:
 [[75  1]
 [ 1 37]]
Best Parameters for SVM+DT+KNN → LogisticRegression: {'dt__max_depth': None, 'final_estima
Best CV Score for SVM+DT+KNN → LogisticRegression: 0.9794

Cross-Validation Scores for SVM+DT+KNN → LogisticRegression:
Fold 1 Accuracy: 0.9565
Fold 2 Accuracy: 1.0000
Fold 3 Accuracy: 0.9130
Fold 4 Accuracy: 1.0000
Fold 5 Accuracy: 1.0000
Average Accuracy: 0.9739

=== SVM+DT+KNN → LogisticRegression ===
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        76
           1       0.97      1.00      0.99        38

    accuracy                           0.99       114
   macro avg       0.99      0.99      0.99       114
weighted avg       0.99      0.99      0.99       114

Confusion Matrix:
 [[75  1]
 [ 0 38]]
```

4 Confusion Matrix and ROC for Each Model

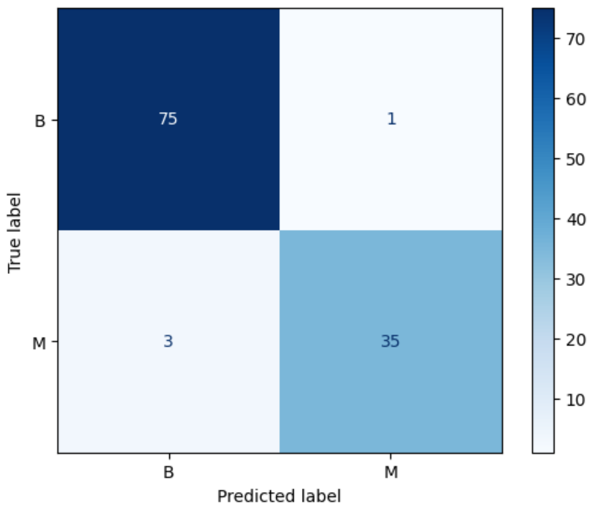


Figure 2: Decision Tree Confusion Matrix

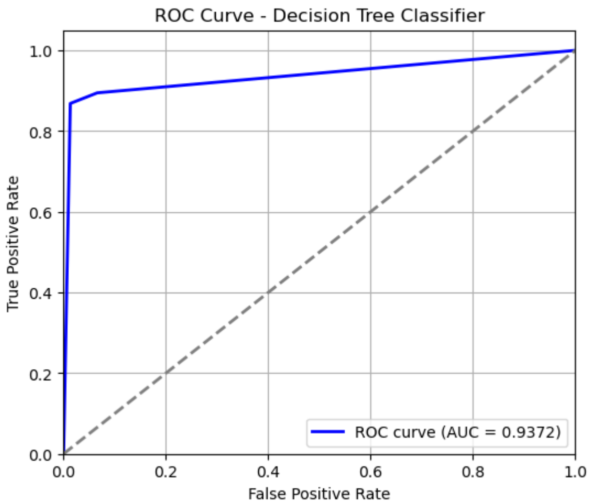


Figure 3: Decision Tree ROC Curve

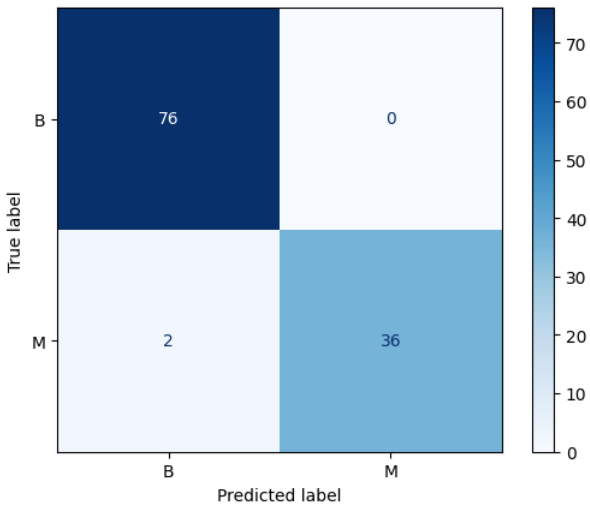


Figure 4: AdaBoost Classifier Confusion Matrix

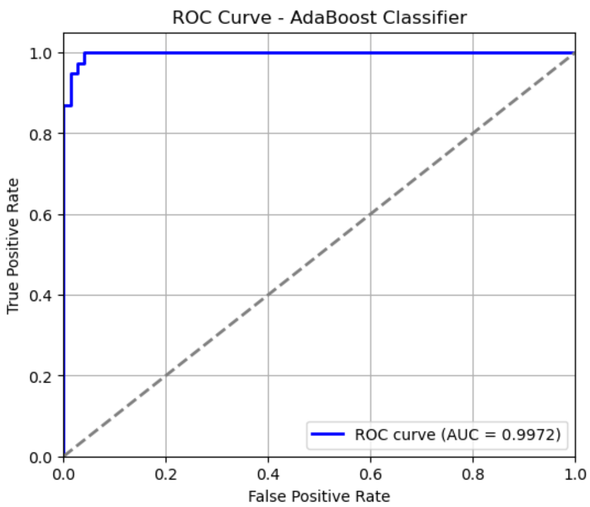


Figure 5: AdaBoost ROC Curve

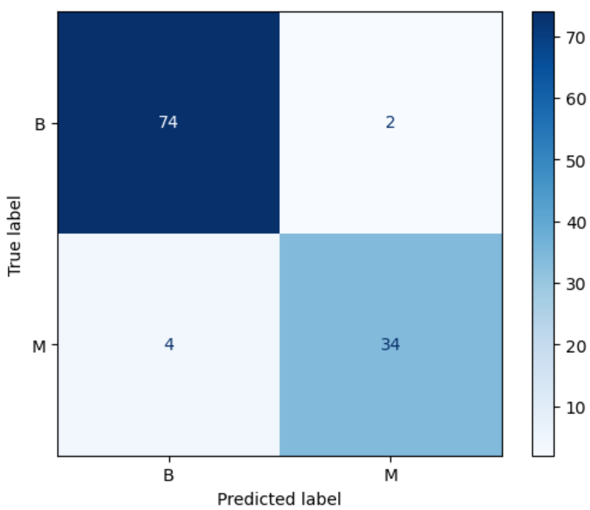


Figure 6: Gradient Boosting Confusion Matrix

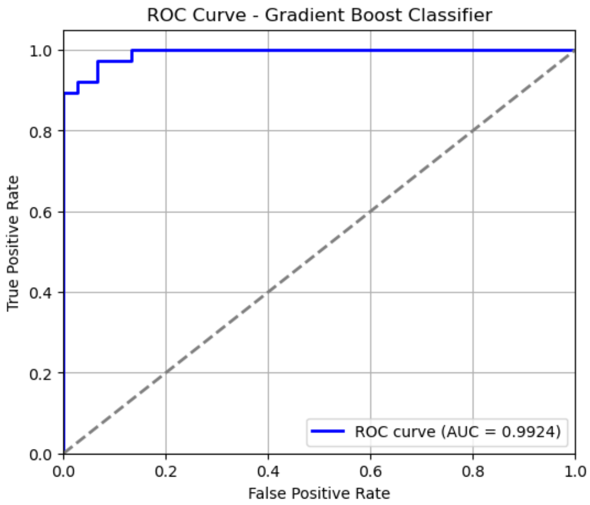


Figure 7: Gradient Boosting ROC curve

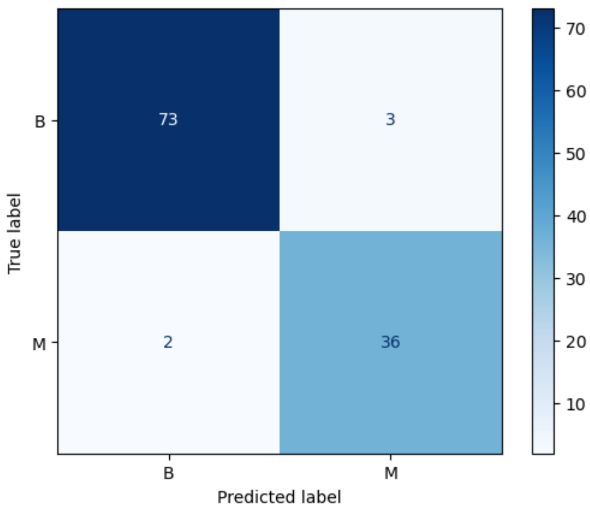


Figure 8: XGBoost Classifier Confusion Matrix

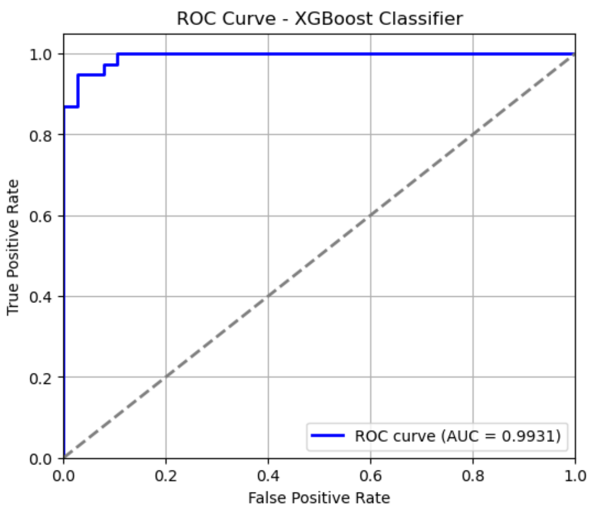


Figure 9: XGBoost Classifier ROC Curve

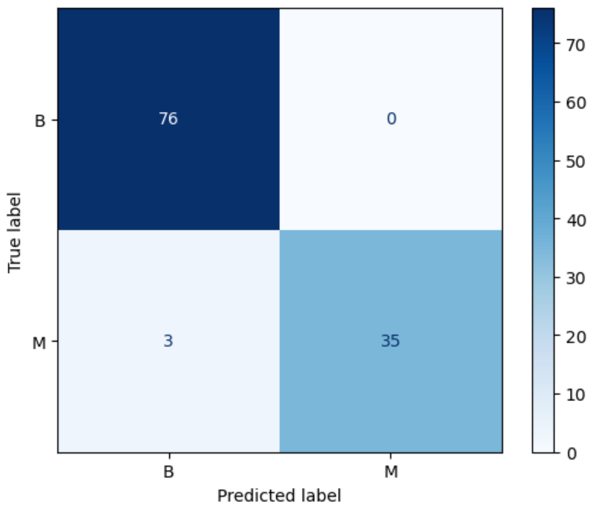


Figure 10: Random Forest Classifier Confusion Matrix

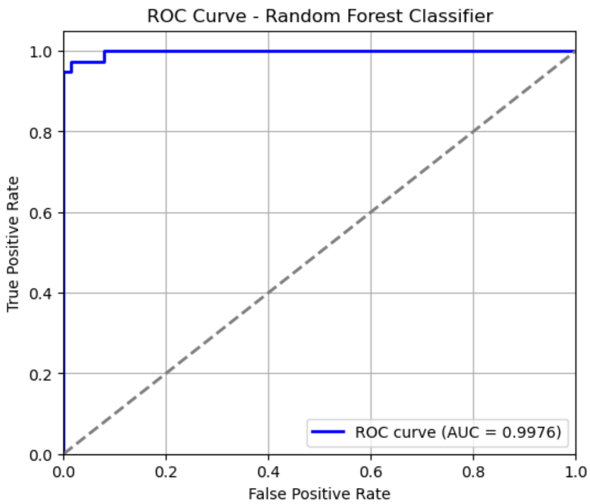
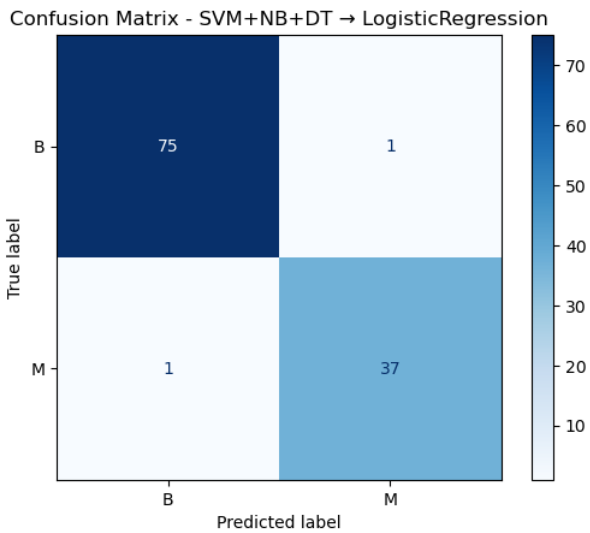
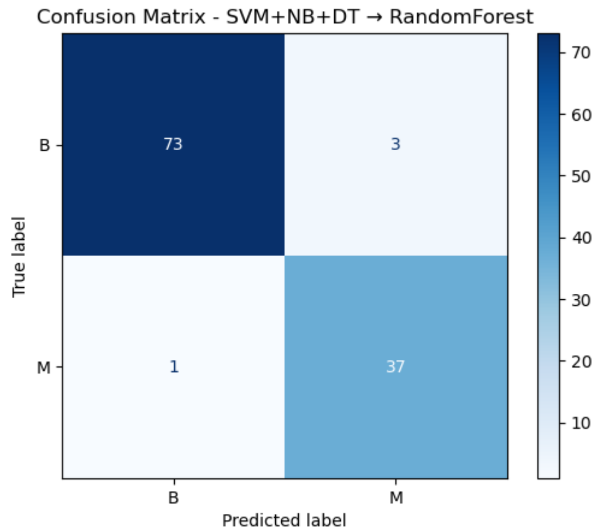
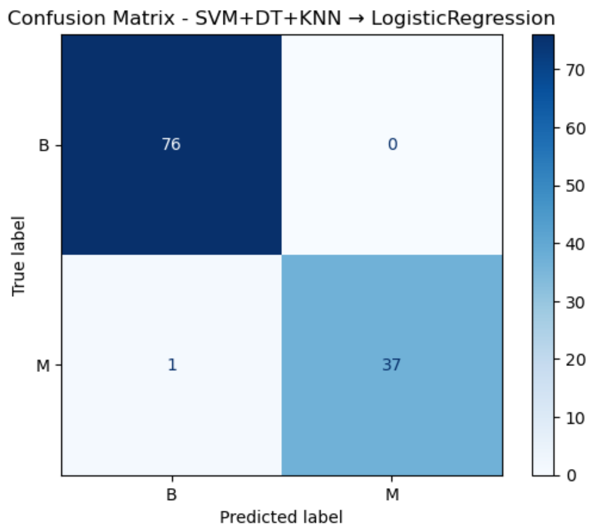


Figure 11: Random Forest Classifier ROC Curve







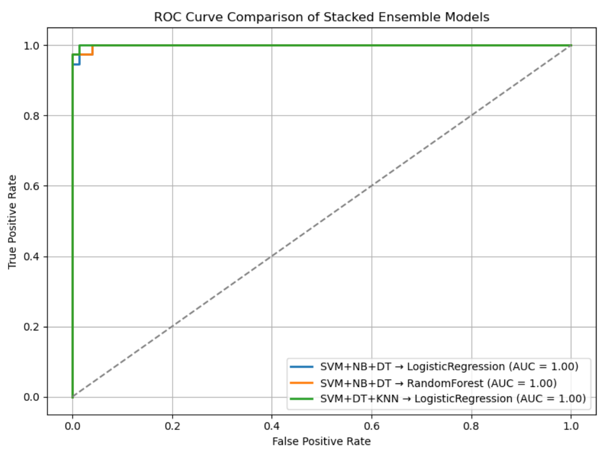


Figure 12: Stacking Classifier ROC Curve

5 Hyperparameter Tuning Tables

5.1 Decision Tree Classifier

Table 1: Decision Tree - Hyperparameter Tuning Results

Criterion	Max Depth	Accuracy	F1 Score
gini	None	0.921	0.920
gini	5	0.924	0.923
gini	10	0.912	0.911
gini	15	0.912	0.911
gini	20	0.915	0.914
entropy	None	0.947	0.947
entropy	5	0.950	0.950
entropy	10	0.944	0.944
entropy	15	0.944	0.944
entropy	20	0.944	0.944
log_loss	None	0.947	0.947
log_loss	5	0.947	0.947
log_loss	10	0.944	0.944
log_loss	15	0.950	0.950
log_loss	20	0.947	0.947

5.2 AdaBoost Classifer

n_estimators	learning_rate	Accuracy	F1 Score
200	1	0.9736	0.9734
300	1	0.9706	0.9703
300	0.5	0.9677	0.9673
200	0.5	0.9677	0.9673
300	1	0.9648	0.9646

5.3 Gradient Boosting Classifier

n_estimators	learning_rate	max_depth	Accuracy	F1 Score
300	0.2	3	0.9736	0.9735
200	0.2	3	0.9736	0.9735
300	0.2	3	0.9736	0.9735
200	0.1	5	0.9736	0.9733
200	0.2	3	0.9736	0.9735

5.4 XGBoost Classifier

n_estimators	learning_rate	max_depth	gamma	Accuracy	F1 Score
200	0.2	4	0	0.9707	0.9705
100	0.2	5	0.1	0.9707	0.9705
200	0.1	5	0.1	0.9707	0.9706
300	0.2	4	0	0.9707	0.9705
200	0.1	6	0.1	0.9707	0.9706

5.5 Random Forest Classifier

n_estimators	max_depth	criterion	Accuracy	F1 Score
200	20	entropy	0.9559	0.9556
200	15	entropy	0.9559	0.9556
200	20	entropy	0.9559	0.9556
200	15	entropy	0.9559	0.9556
200	None	entropy	0.9559	0.9556

5.6 Stacking Ensemble Models

Base-Model	Final-Estimator	Accuracy	F1 Score
SVM+NB+DT	LogisticRegression	0.982456	0.973684
SVM+NB+DT	RandomForest	0.973684	0.961039
SVM+DT+KNN	LogisticRegression	0.991228	0.986667

Table 2: Stacking model Hyperparameter Tuning

6 Cross-Validation Results Table

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	AvG Accuracy
Decision Tree	0.9130	0.9130	0.8696	0.9565	1.0000	0.9304
AdaBoost	0.8696	1.0000	0.8696	0.9565	0.9545	0.9300
Gradient Boosting	0.9130	0.9565	0.9130	1.0000	0.9545	0.9474
XGBoost	0.9130	0.9565	0.8696	1.0000	1.0000	0.9478
Random Forest	0.9130	0.9130	0.9130	1.0000	1.0000	0.9478
SVM+NB+DT → Logistic	0.9565	0.9565	0.9565	0.9565	0.9545	0.9561
SVM+NB+DT → RandomForest	0.9130	0.9565	0.9565	0.9130	0.9545	0.9387
SVM+DT+KNN → Logistic	0.9565	1.0000	0.9130	1.0000	1.0000	0.9739

Table 3: 5-Fold Cross Validation Results for All Models

7 All Comparison Table

Model	Accuracy	F1 Score
Decision Tree	0.9474	0.9465
AdaBoost	0.9736	0.9739
Gradient Boosting	0.9474	0.9469
XGBoost	0.9561	0.9563
Random Forest	0.9737	0.9733
Stacking (SVM + NB + DT → LR)	0.973684	0.961039
Stacking (SVM + NB + DT → RF)	0.982456	0.973684
Stacking (SVM + DT + KNN → LR)	0.991228	0.987013

Table 4: Comparison of models based on Accuracy and F1 Score

8 Observations and Conclusions

- **Best Validation Accuracy:** The stacking model *(SVM + DT + KNN → LR)* achieved the highest validation accuracy of **0.9912**, outperforming all other methods.
- **Decision Tree vs. Ensemble Methods:** The standalone Decision Tree achieved an accuracy of **0.9474**, which is significantly lower compared to ensemble methods like Random Forest (**0.9737**) and AdaBoost (**0.9736**). This highlights the improved stability and predictive power of ensemble techniques.
- **Random Forest Tuning:** Random Forest reached an accuracy of **0.9737**, suggesting that tuning hyperparameters such as `max_depth` and `n_estimators` likely contributed to its improved performance compared to the plain Decision Tree.
- **Generalization and Overfitting:** Among the models, stacking with Logistic Regression and Random Forest showed strong generalization, with balanced accuracy and F1 scores. No clear signs of overfitting were observed, as performance across metrics remained consistent. However, standalone Decision Trees exhibited relatively weaker generalization.
- **Effect of Stacking:** Stacking consistently improved performance over base learners. For example, stacking with $SVM + DT + KNN \rightarrow LR$ achieved the best overall results (**Accuracy = 0.9912, F1 = 0.9870**), surpassing both individual models and other ensemble methods.

Conclusion: Ensemble methods, particularly stacking, provided superior performance compared to individual classifiers. While Decision Trees served as a useful baseline, advanced ensembles like Random Forest, AdaBoost, and Gradient Boosting offered significant improvements. Stacking further enhanced predictive performance, making it the most effective approach for this dataset.