Sri Sivasubramaniya Nadar College of Engineering, Chennai

(An Autonomous Institution Affiliated to Anna University)

Degree & Branch: Integrated M.Tech. Computer Science & Engineering
Semester: V

Course Code & Title: ICS1512 - Machine Learning Algorithms Laboratory

Academic Year: 2025-2026 (Odd) **Batch:** 2023-2028

Name: Vishwajith L K **Reg. No.:** 3122237001061

Experiment 4: Ensemble Prediction and Decision Tree Model Evaluation

1. Aim

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

2. Libraries Used

- **NumPy**: For numerical operations and matrix manipulations
- **Pandas**: For data cleaning, preprocessing, and analysis
- **Scikit-learn**: For implementing Linear Regression and evaluating metrics
- **Matplotlib & Seaborn**: For data visualization and statistical plots

3. Objectives

- Perform preprocessing and cleaning of data
- Conduct EDA to better understand relationships between variables
- Implement three different classification models along with their variants for classifying cancer diagnosis
- Evaluate every model's performance using error metrics
- Interpret the results.

4. Code Implementation

1. Loading the Dataset

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from ucimlrepo import fetch_ucirepo

# Fetch dataset
breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)


# Combine features and targets into one DataFrame
training_df = pd.concat(
    [breast_cancer_wisconsin_diagnostic.data.features,
     breast_cancer_wisconsin_diagnostic.data.targets],
    axis=1
)
target_col =  training_df.columns[-1]
# Display first 10 rows
print(training_df.head(10))
```

	radius1	texture1	perimeter1	area1	smoothness1	compactness1
0	17.99	10.38	122.80	1001.0	0.11840	0.27760
1	20.57	17.77	132.90	1326.0	0.08474	0.07864
2	19.69	21.25	130.00	1203.0	0.10960	0.15990
3	11.42	20.38	77.58	386.1	0.14250	0.28390
4	20.29	14.34	135.10	1297.0	0.10030	0.13280
5	12.45	15.70	82.57	477.1	0.12780	0.17000
6	18.25	19.98	119.60	1040.0	0.09463	0.10900
7	13.71	20.83	90.20	577.9	0.11890	0.16450
8	13.00	21.82	87.50	519.8	0.12730	0.19320
9	12.46	24.04	83.97	475.9	0.11860	0.23960

	concavity1	concave_points1	symmetry1	fractal_dimension1	...	texture3
0	0.30010	0.14710	0.2419	0.07871	...	17.33
1	0.08690	0.07017	0.1812	0.05667	...	23.41
2	0.19740	0.12790	0.2069	0.05999	...	25.53
3	0.24140	0.10520	0.2597	0.09744	...	26.50
4	0.19800	0.10430	0.1809	0.05883	...	16.67
5	0.15780	0.08089	0.2087	0.07613	...	23.75
6	0.11270	0.07400	0.1794	0.05742	...	27.66
7	0.09366	0.05985	0.2196	0.07451	...	28.14
8	0.18590	0.09353	0.2350	0.07389	...	30.73
9	0.22730	0.08543	0.2030	0.08243	...	40.68

	perimeter3	area3	smoothness3	compactness3	concavity3	concave_points3
0	184.60	2019.0	0.1622	0.6656	0.7119	0.2654
1	158.80	1956.0	0.1238	0.1866	0.2416	0.1860
2	152.50	1709.0	0.1444	0.4245	0.4504	0.2430
3	98.87	567.7	0.2098	0.8663	0.6869	0.2575
4	152.20	1575.0	0.1374	0.2050	0.4000	0.1625
5	103.40	741.6	0.1791	0.5249	0.5355	0.1741
6	153.20	1606.0	0.1442	0.2576	0.3784	0.1932
7	110.60	897.0	0.1654	0.3682	0.2678	0.1556

Figure 1: Dataset

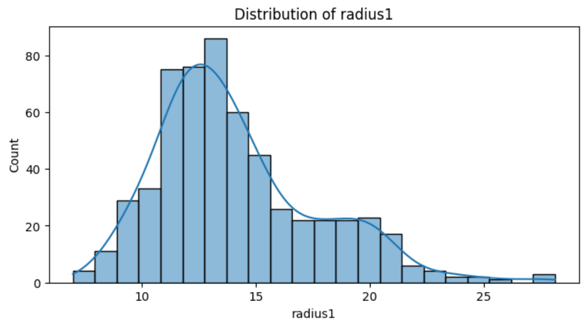
2. Data Preprocessing

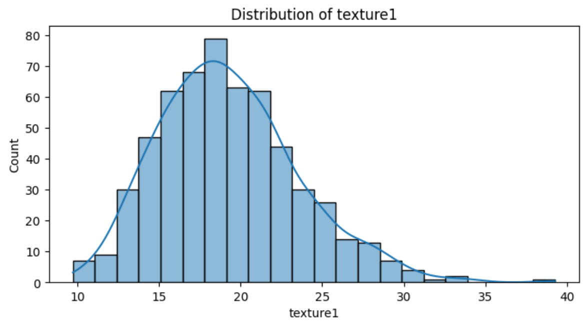
```
import seaborn as sns
# Numerical features
num_features = training_df.select_dtypes(include=['int64', 'float64']).columns.tolist
num_features = [col for col in num_features if col != target_col]

# Categorical features
cat_features = training_df.select_dtypes(include=['object', 'category']).columns.toli

from sklearn.preprocessing import MinMaxScaler,StandardScaler

for col in features:
  if col in relevant_features:
    if features[col] == 1:
        scaler = MinMaxScaler()
    else:
        scaler = StandardScaler()
```

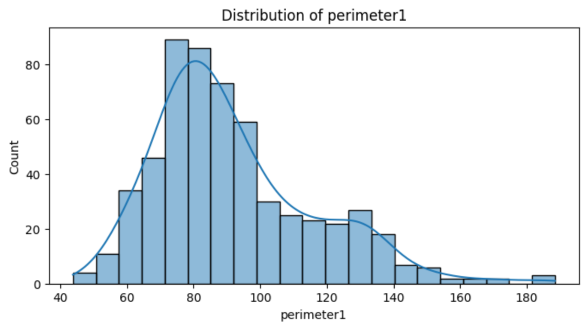
```
    # Reshape the column data to 2D array
    training_df[col] = scaler.fit_transform(training_df[col].values.reshape(-1, 1))
```

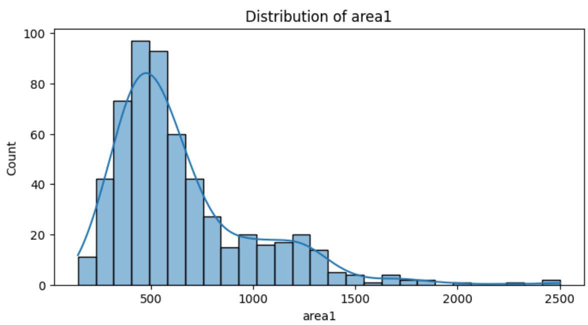
3. Exploratory Data Analysis

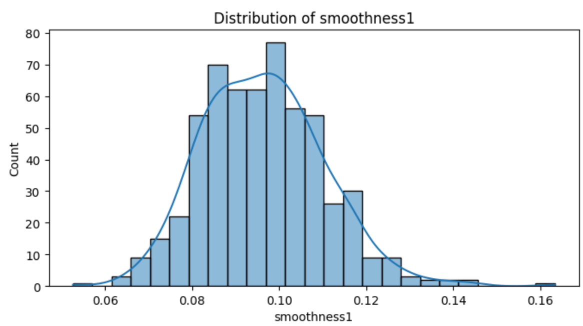
```
from scipy.stats import shapiro

fig, axes = plt.subplots(nrows=len(num_features), ncols=1, figsize=(8, len(num_featur
fig.tight_layout(pad=5.0)
features = {}
for i, column in enumerate(num_features):
    # Plot histogram with KDE
    sns.histplot(training_df[column], kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {column}')

    # Shapiro-Wilk normality test
    stat, p = shapiro(training_df[column].dropna())

    # Add test result as text on plot
    result = 1 if p > 0.05 else 0
    features[column]=result

plt.show()


#Correlation Heatmap: To identify multicollinearity and relationships among features.

target_corr = training_df[num_features + [target_col]].corr()[target_col].drop(target_
threshold = 0.2
relevant_features = target_corr[abs(target_corr) >= threshold].index.tolist()

print(f"Selected features with correlation >= {threshold}:")
print(relevant_features)
relevant_features.append(target_col)
corr = training_df[relevant_features].corr()
plt.figure(figsize=(20,20))
# Plot heatmap
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Relevant Numerical Features')

plt.show()
```

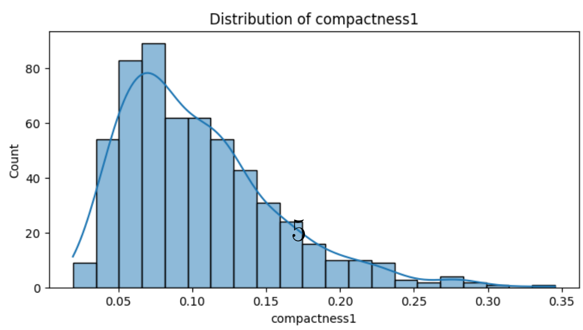

Distribution of radius1
Count
radius1


Distribution of texture1
Count
texture1


Distribution of perimeter1
Count
perimeter1


Distribution of area1
Count
area1


Distribution of smoothness1
Count
smoothness1


Distribution of compactness1
Count
compactness1

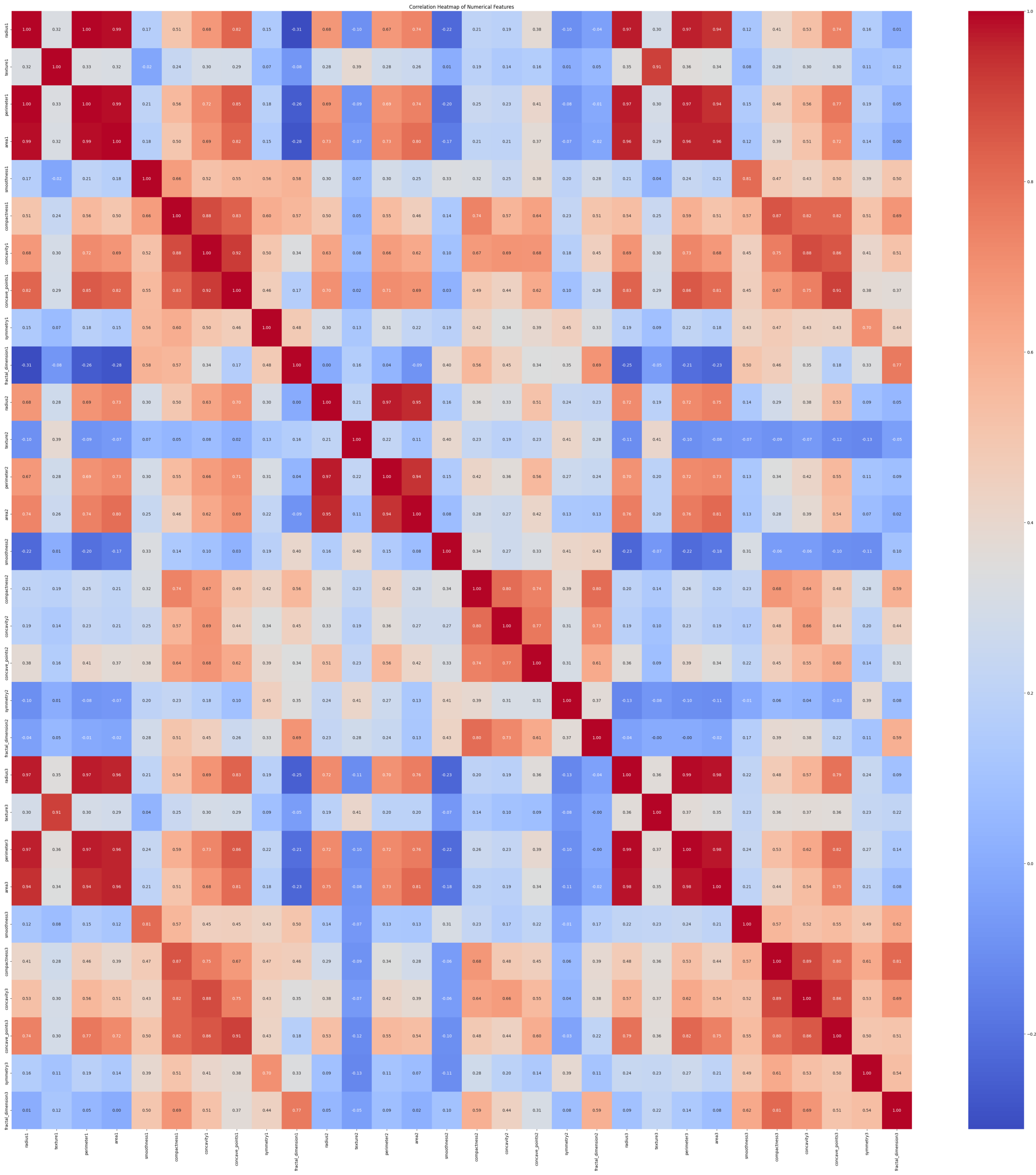


Figure 3: Scatter Plot and Correlation Heatmap

4. Split the dataset

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = training_df.drop(columns=target_col)
Y = training_df[target_col]
x_train,x_test, y_train, y_test = train_test_split(X,Y,test_size=0.2)
```

5. Hyperparameter Tuning

Decision Tree

```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, roc_curve, auc, classification_report
)
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier as DTC
import numpy as np
param_grid = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_depth': [None, 3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

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

grid_search = GridSearchCV(
    estimator=DTC(),
    param_grid=param_grid,
    cv=kfold,
    scoring='accuracy',
    verbose=1,
    n_jobs=-1
)
grid_search.fit(x_train, y_train)

# Extract all fold scores
results_df = pd.DataFrame(grid_search.cv_results_)

# Display only parameters + fold scores
fold_cols = [f'split{i}_test_score' for i in range(kfold.n_splits)]
print(results_df[['params'] + fold_cols])
```

AdaBoost

```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, roc_curve, auc, classification_report
)
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier as ABC
import numpy as np
```

```
param_grid = {
    'n_estimators': [50, 100, 200],              # Number of weak learners
    'learning_rate': [0.01, 0.1, 1],             # Step size
    'estimator__criterion': ['gini', 'entropy','log_loss']   # Split quality measure
}

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

grid_search = GridSearchCV(
    estimator=ABC(estimator=DTC(max_depth=1)),
    param_grid=param_grid,
    cv=kfold,
    scoring='accuracy',
    verbose=1,
    n_jobs=-1
)
grid_search.fit(x_train, y_train)

# Extract all fold scores
results_df = pd.DataFrame(grid_search.cv_results_)

# Display only parameters + fold scores
fold_cols = [f'split{i}_test_score' for i in range(kfold.n_splits)]
print(results_df[['params'] + fold_cols])
```

Gradient Boosting

```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, roc_curve, auc, classification_report
)
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier as GBC
import numpy as np
param_grid = {
    'n_estimators': [100, 200, 300],        # Number of boosting stages
    'learning_rate': [0.01, 0.1, 0.2],      # Shrinks contribution of each tree
    'max_depth': [2, 3, 5],                 # Depth of each tree
    'subsample': [0.8, 1.0]                 # Fraction of samples for fitting
}
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid_search = GridSearchCV(
    estimator=GBC(),
    param_grid=param_grid,
    cv=kfold,
```

```
    scoring='accuracy',
    verbose=1,
    n_jobs=-1
)
grid_search.fit(x_train, y_train)

# Extract all fold scores
results_df = pd.DataFrame(grid_search.cv_results_)

# Display only parameters + fold scores
fold_cols = [f'split{i}_test_score' for i in range(kfold.n_splits)]
print(results_df[['params'] + fold_cols])
```

XGBoost

```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, roc_curve, auc, classification_report
)
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier as XGB
import numpy as np
param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'gamma': [0, 0.1, 0.2]  # Minimum loss reduction for split
}
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid_search = GridSearchCV(
    estimator=XGB(),
    param_grid=param_grid,
    cv=kfold,
    scoring='accuracy',
    verbose=1,
    n_jobs=-1
)
grid_search.fit(x_train, y_train)

# Extract all fold scores
results_df = pd.DataFrame(grid_search.cv_results_)

# Display only parameters + fold scores
fold_cols = [f'split{i}_test_score' for i in range(kfold.n_splits)]
```

```
print(results_df[['params'] + fold_cols])
```

RandomForest

```
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, roc_curve, auc, classification_report
)
import pandas as pd
import numpy as np

# Define parameter grid for Random Forest
param_grid = {
    'n_estimators': [100, 200, 300],        # Number of trees
    'max_depth': [None, 5, 10, 20],         # Depth of each tree
    'criterion': ['gini', 'entropy', 'log_loss']  # Split quality measure

}

# Stratified K-Fold
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# GridSearch with Random Forest
grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    cv=kfold,
    scoring='accuracy',
    verbose=1,
    n_jobs=-1
)

grid_search.fit(x_train, y_train)

# Extract all fold scores into DataFrame
results_df = pd.DataFrame(grid_search.cv_results_)

# Display only parameters + fold scores
fold_cols = [f'split{i}_test_score' for i in range(kfold.n_splits)]
print(results_df[['params'] + fold_cols])
```

Stacking Classifier (SVM + Naïve Bayes + Decision Tree)

```
from sklearn.ensemble import StackingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, classification_report
import pandas as pd

# Define base models
base_setups = {
    "SVM_NB_DT": [
        ('svm', SVC(probability=True, random_state=42)),
        ('nb', GaussianNB()),
        ('dt', DecisionTreeClassifier(random_state=42))
    ],
    "SVM_NB_DT_RF": [
        ('svm', SVC(probability=True, random_state=42)),
        ('nb', GaussianNB()),
        ('dt', DecisionTreeClassifier(random_state=42))
    ],
    "SVM_DT_KNN": [
        ('svm', SVC(probability=True, random_state=42)),
        ('dt', DecisionTreeClassifier(random_state=42)),
        ('knn', KNeighborsClassifier())
    ]
}

# Final estimator options
final_estimators = {
    "LogReg": LogisticRegression(max_iter=1000, random_state=42),
    "RF": RandomForestClassifier(random_state=42)
}

# Stratified K-Fold
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

results = []

# Loop through base models + final estimators
for setup_name, base_models in base_setups.items():
    for final_name, final_estimator in final_estimators.items():

        stack_model = StackingClassifier(
            estimators=base_models,
            final_estimator=final_estimator,
            cv=kfold,
```

```
            n_jobs=-1
        )

        stack_model.fit(x_train, y_train)
        y_pred = stack_model.predict(x_test)

        acc = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average="weighted")

        results.append({
            "Base Models": setup_name,
            "Final Estimator": final_name,
            "Accuracy": acc,
            "F1 Score": f1
        })

# Convert results to DataFrame (similar to your LaTeX table)
results_df = pd.DataFrame(results)
print(results_df)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
                                               params  split0_test_score  \
0   {'criterion': 'gini', 'max_depth': None, 'n_es...           0.934066
1   {'criterion': 'gini', 'max_depth': None, 'n_es...           0.945055
2   {'criterion': 'gini', 'max_depth': None, 'n_es...           0.945055
3   {'criterion': 'gini', 'max_depth': 5, 'n_estim...           0.923077
4   {'criterion': 'gini', 'max_depth': 5, 'n_estim...           0.934066
5   {'criterion': 'gini', 'max_depth': 5, 'n_estim...           0.945055
6   {'criterion': 'gini', 'max_depth': 10, 'n_esti...           0.934066
7   {'criterion': 'gini', 'max_depth': 10, 'n_esti...           0.945055
8   {'criterion': 'gini', 'max_depth': 10, 'n_esti...           0.945055
9   {'criterion': 'gini', 'max_depth': 20, 'n_esti...           0.934066
10  {'criterion': 'gini', 'max_depth': 20, 'n_esti...           0.945055
11  {'criterion': 'gini', 'max_depth': 20, 'n_esti...           0.945055
12  {'criterion': 'entropy', 'max_depth': None, 'n...           0.934066
13  {'criterion': 'entropy', 'max_depth': None, 'n...           0.934066
14  {'criterion': 'entropy', 'max_depth': None, 'n...           0.923077
15  {'criterion': 'entropy', 'max_depth': 5, 'n_es...           0.923077
16  {'criterion': 'entropy', 'max_depth': 5, 'n_es...           0.945055
17  {'criterion': 'entropy', 'max_depth': 5, 'n_es...           0.945055
18  {'criterion': 'entropy', 'max_depth': 10, 'n_e...           0.934066
19  {'criterion': 'entropy', 'max_depth': 10, 'n_e...           0.934066
20  {'criterion': 'entropy', 'max_depth': 10, 'n_e...           0.923077
21  {'criterion': 'entropy', 'max_depth': 20, 'n_e...           0.934066
22  {'criterion': 'entropy', 'max_depth': 20, 'n_e...           0.934066
23  {'criterion': 'entropy', 'max_depth': 20, 'n_e...           0.923077
24  {'criterion': 'log_loss', 'max_depth': None, '...           0.934066
```

Figure 4: Hyperparameter Tuning

	Base Models	Final Estimator	Accuracy	F1 Score
0	SVM_NB_DT	LogReg	0.964912	0.964711
1	SVM_NB_DT	RF	0.964912	0.964711
2	SVM_NB_DT_RF	LogReg	0.964912	0.964711
3	SVM_NB_DT_RF	RF	0.964912	0.964711
4	SVM_DT_KNN	LogReg	0.973684	0.973449
5	SVM_DT_KNN	RF	0.964912	0.964711

Figure 5: Hyperparameter Tuning

6. Model Training

Decision Tree

```
model = DTC(**grid_search.best_params_)
model.fit(x_train, y_train)
```

AdaBoost

```
model = ABC(estimator=DTC(max_depth=1,criterion='gini'),n_estimators=200,learning_rat
model.fit(x_train, y_train)
```

Gradient Boosting

```
model = GBC(**grid_search.best_params_)
model.fit(x_train, y_train)
```

XGBoost

```
model = XGB(**grid_search.best_params_)
model.fit(x_train, y_train)
```

Random Forest

```
model = RandomForestClassifier(**grid_search.best_params_)
model.fit(x_train, y_train)
```

Stacking Classifier (SVM + Na¨ıve Bayes + Decision Tree)

```
#best base model and final estimator from tuning
base = [
        ('svm', SVC(probability=True, random_state=42)),
        ('dt', DecisionTreeClassifier(random_state=42)),
        ('knn', KNeighborsClassifier())
    ]
final = LogisticRegression(max_iter=1000, random_state=42)
model = best_stack_model = StackingClassifier(
    estimators=base,
    final_estimator=final,
    cv=5,
    n_jobs=-1
)
model.fit(x_train, y_train)
```

7. Model Evaluation

```
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, roc_curve, auc, classification_report
)
import matplotlib.pyplot as plt
y_pred = model.predict(x_test)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average='weighted')
rec = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax)
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        73
           1       1.00      0.93      0.96        41

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

Figure 6: Evaluation Metrics on Test Set

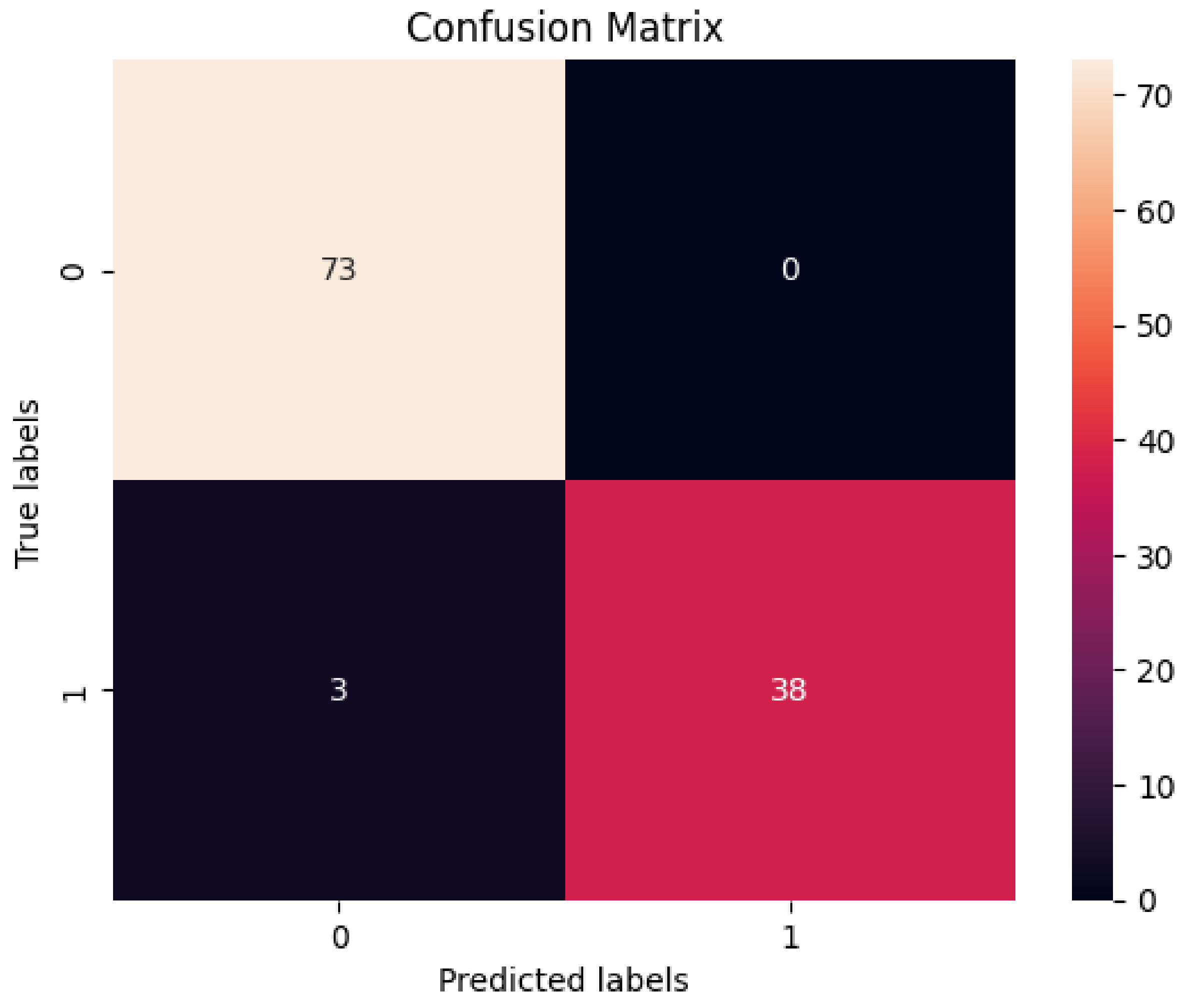


Figure 7: Confusion Matrix

8. Visualization of the Results

```
y_prob = model.predict_proba(x_test)[:, 1]  # probabilities for the positive class
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, label=f"ROC curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], 'k--')  # diagonal for random guessing
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC) Curve")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```

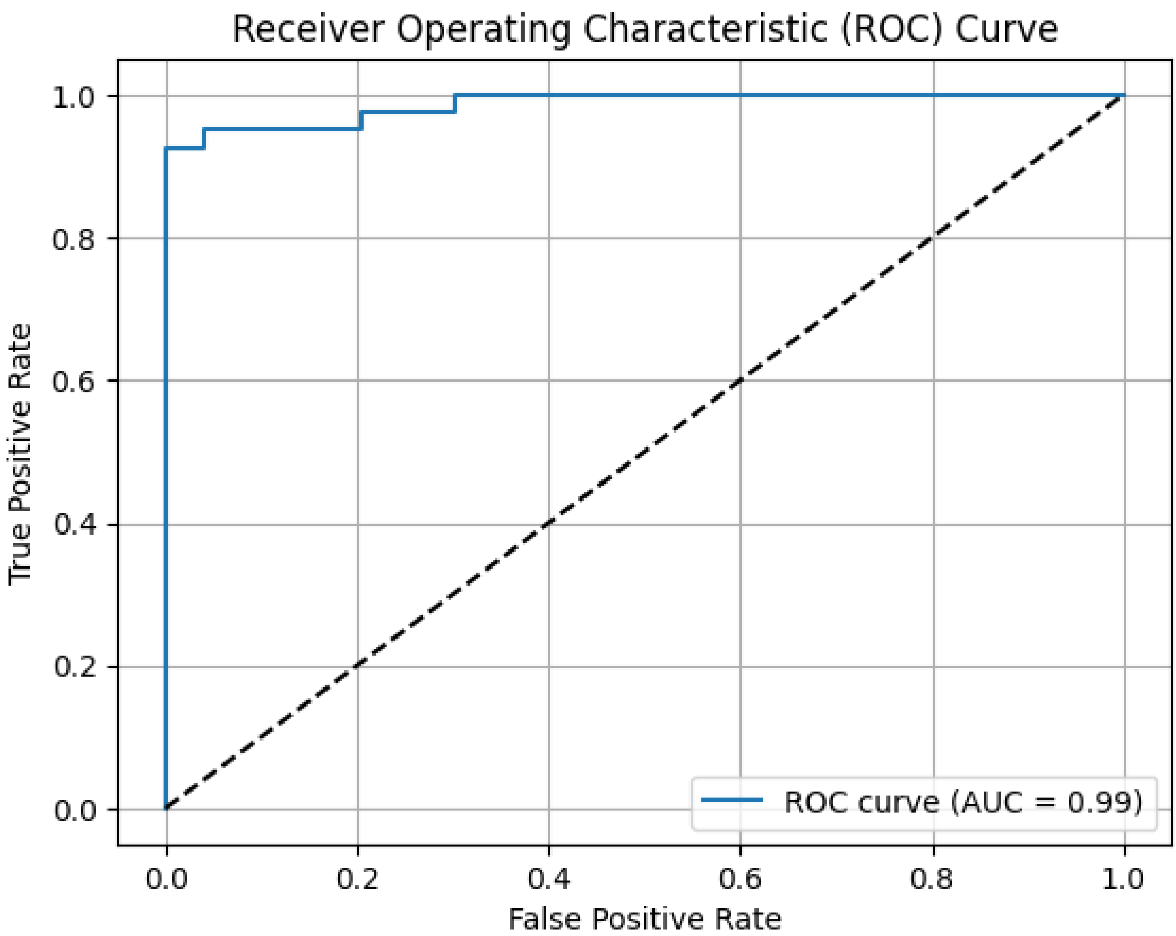


Figure 8: ROC plot

6. Hyperparameter Trials

Table 1: Decision Tree

criterion	**max_depth**	**Accuracy**	**F1 score**
'gini'	None	93.4%	93.3%
'gini'	3	94.2%	94%
'log$_{l}oss'$	10	92.3%	92.8%
'gini'	10	91.5%	91.7%

Table 2: AdaBoost

n_estimators	**learning_rate**	**Accuracy (%)**	**F1 Score (%)**
50	0.01	91.87	91.74
50	0.10	94.29	94.27
50	1.00	95.16	95.15
100	0.01	92.97	92.92
100	0.10	95.38	95.35
100	1.00	95.28	95.27
200	0.01	93.62	93.62
200	0.10	96.04	96.04
200	1.00	95.16	95.16

Table 3: Gradient Boosting

n_estimators	learning_rate	max_depth	Accuracy	F1 score
100	0.01	2	0.942	0.941
100	0.01	3	0.940	0.940
100	0.01	5	0.938	0.937
100	0.10	2	0.953	0.953
100	0.10	3	0.957	0.957
100	0.10	5	0.958	0.958
100	0.20	2	0.957	0.957
100	0.20	3	0.958	0.957
100	0.20	5	0.955	0.955
200	0.01	2	0.946	0.946
200	0.01	3	0.949	0.948
200	0.01	5	0.948	0.947
200	0.10	2	0.956	0.956
200	0.10	3	0.958	0.957
200	0.10	5	0.959	0.958
200	0.20	2	0.957	0.957
200	0.20	3	0.958	0.958
200	0.20	5	0.958	0.958
300	0.01	2	0.952	0.952
300	0.01	3	0.955	0.954
300	0.01	5	0.951	0.951
300	0.10	2	0.957	0.957
300	0.10	3	0.959	0.959
300	0.10	5	0.958	0.958
300	0.20	2	0.957	0.957
300	0.20	3	**0.960**	**0.960**
300	0.20	5	0.958	0.958

Table 4: XGBoost

n_estimators	learning_rate	max_depth	Accuracy (%)	F1 Score (%)
100	0.01	2	93.86	93.81
100	0.01	3	94.40	94.37
100	0.01	4	94.40	94.37
100	0.10	2	96.58	96.56
100	0.10	3	96.15	96.16
100	0.10	4	96.15	96.16
100	1.00	2	96.26	96.24
100	1.00	3	96.53	96.52
100	1.00	4	96.53	96.52
200	0.01	2	94.36	94.32
200	0.01	3	94.36	94.32
200	0.01	4	94.36	94.32
200	0.10	2	96.60	96.58
200	0.10	3	96.10	96.11
200	0.10	4	96.10	96.11
200	1.00	2	96.98	96.97
200	1.00	3	96.98	96.97
200	1.00	4	96.98	96.97
300	0.01	2	94.36	94.32
300	0.01	3	94.36	94.32
300	0.01	4	94.36	94.32

Table 5: Random Forest

n_estimators	max_depth	criterion	Accuracy (%)	F1 score (%)
100	5	entropy	95.16	95.15
100	5	gini	94.73	94.67
100	5	log_loss	95.16	95.15
100	10	entropy	95.16	95.15
100	10	gini	95.16	95.13
100	10	log_loss	95.16	95.15
100	20	entropy	95.16	95.15
100	20	gini	95.16	95.13
100	20	log_loss	95.16	95.15
200	5	entropy	95.36	95.36
200	5	gini	95.76	95.76
200	5	log_loss	95.36	95.36
200	10	entropy	95.36	95.36
200	10	gini	95.76	95.76
200	10	log_loss	95.36	95.36
200	20	entropy	95.36	95.36
200	20	gini	95.76	95.76
200	20	log_loss	95.36	95.36
300	5	entropy	95.16	95.16
300	5	gini	95.36	95.36
300	5	log_loss	95.16	95.16
300	10	entropy	95.16	95.16
300	10	gini	95.36	95.36
300	10	log_loss	95.16	95.16
300	20	entropy	95.16	95.16
300	20	gini	95.36	95.36
300	20	log_loss	95.16	95.16

Table 6: Stacked Ensemble

Base Models	Final Estimator	Accuracy / F1 Score
SVM, Naïve Bayes, Decision Tree	Logistic Regression	98.25%
SVM, Naïve Bayes, Decision Tree	Random Forest	96.5%
SVM, Decision Tree, KNN	Logistic Regression	98.25%

7. 5 Fold Validation

=

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average Accuracy
Decision Tree	93.4%	94.5%	93.4%	93.4%	92.3%	93.4%
AdaBoost	98.9%	97.8%	97.8%	94.5%	89%	95.6%
Gradient Boosting	98.9%	96.7%	98.9%	93.4%	89%	95.3%
XGBoost	98%	97%	100%	94%	91%	96%
Random Forest	98%	96%	96%	93%	90%	95%

8. Best Practices

- Performed thorough data cleaning and preprocessing
- Used train-test split to evaluate model performance
- Implemented feature scaling for better convergence
- Compared multiple evaluation metrics
- Documented all steps and interpretations

9. Learning Outcomes

- Understood how to implement ensemble classifier models in a real life dataset
- Learned the role of data preprocessing and feature selection in model performance
- Developed skills in EDA, visualization, and interpreting regression results
- Realized the importance of hyper parameter tuning and cross-validation for robust models