Performance Analysis of Machine Learning Algorithms for Hepatitis Classification

Department of IT, Univ. Badji Mokhtar Annaba
Islam AIOUNI Master 1 IATI
Author: Islam AIOUNI

March 3, 2025

Abstract

Hepatitis is a liver inflammation that may lead to serious complications and even death if not diagnosed early. In this project, I applied several machine learning algorithms — including Random Forest, Support Vector Machine (SVM), k-Nearest Neighbors (KNN), Logistic Regression, Decision Tree, Neural Network, and an unsupervised K-Means clustering approach — on a hepatitis dataset from the UCI Machine Learning Repository. I evaluated the models using metrics such as accuracy, confusion matrix, precision, recall, F1-score, and log loss, and visualized their performance using combined Precision-Recall and ROC curves (with AUC values). Overall, I found that supervised models, particularly Decision Tree and Logistic Regression, performed best. This document also includes personal reflections and future learning plans.

1 Introduction

Hepatitis refers to the inflammation of the liver, most commonly caused by viral infections (A, B, C, D, and E). Chronic hepatitis, especially types B and C, can lead to liver cirrhosis or cancer. Early and accurate diagnosis is crucial for effective treatment and improved patient outcomes.

Machine learning methods offer promising approaches for medical diagnosis by extracting patterns from clinical datasets. In this project, I analyze a hepatitis dataset consisting of 155 instances and 20 attributes. The target variable, **CLASS** (Die or Live), is used without feature selection to fully evaluate the predictive power of all available attributes. My objective is to compare various classifiers and assess whether an unsupervised method like K-Means can provide additional insight.

2 Dataset Description

The dataset, obtained from the UCI Machine Learning Repository, contains the following attributes:

Table 1: Attributes of the Hepatitis Dataset

Attribute	**Description / Possible Values**
CLASS	Outcome: Die (0) / Live (1)
AGE	Age: 10, 20, 30, 40, 50, 60, 70, 80
SEX	Gender: male, female
STEROID	Steroid treatment: no, yes
ANTIVIRALS	Antiviral treatment: no, yes
FATIGUE	Fatigue: no, yes
MALAISE	Malaise: no, yes
ANOREXIA	Anorexia: no, yes
LIVER_BIG	Enlarged liver: no, yes
LIVER_FIRM	Firm liver: no, yes
SPLEEN_PALABLE	Palpable spleen: no, yes
SPIDERS	Spider angiomas: no, yes
ASCITES	Ascites: no, yes
VARICES	Varices: no, yes
BILIRUBIN	Bilirubin levels: 0.39, 0.80, 1.20, 2.00, 3.00, 4.00
ALK_PHOSPHATE	Alkaline phosphatase: 33, 80, 120, 160, 200, 250
SGOT	SGOT enzyme: 13, 100, 200, 300, 400, 500
ALBUMIN	Albumin levels: 2.1, 3.0, 3.8, 4.5, 5.0, 6.0
PROTIME	Prothrombin time: 10, 20, 30, 40, 50, 60, 70, 80, 90
HISTOLOGY	Histology result: no, yes

3 Methodology

3.1 Data Preprocessing

I handled missing values by replacing "?" with `NaN` and imputing with the most frequent value. Categorical features were encoded using Label Encoding, and all features were standardized (zero mean and unit variance) to ensure that no single feature dominated the learning process.

3.2 Algorithms Overview

I implemented the following models:

- **Random Forest:** An ensemble method that builds multiple decision trees and aggregates predictions via majority voting.
- **SVM:** Finds an optimal hyperplane to separate classes using a linear kernel with probability estimates.
- **KNN:** Classifies a new instance based on the majority vote of the k nearest neighbors.
- **Logistic Regression:** Uses a logistic function to model the probability of a binary outcome.
- **Decision Tree:** Constructs a tree-like model based on splitting features.
- **Neural Network:** Implements a multi-layer perceptron (MLP) to capture non-linear relationships.
- **K-Means Clustering:** An unsupervised method that partitions data into k clusters; its inertia is reported instead of classification accuracy.

3.3 Model Training, Evaluation, and Validation

I split the dataset into 80% training and 20% testing sets. Each model was trained and evaluated using:

- **Accuracy:** Accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$
- **Precision:** Precision $= \frac{TP}{TP+FP}$
- **Recall (Sensitivity):** Recall $= \frac{TP}{TP+FN}$
- **F1-Score:** F1 $= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision}+\text{Recall}}$
- **ROC and Precision-Recall Curves:** These curves evaluate performance over various thresholds.

Metrics are reported for each class (Class 0 for Die and Class 1 for Live). I compared training and test accuracies to check for overfitting or underfitting.

3.4 Confusion Matrix Explanation

The confusion matrix is defined as:

True Negatives (TN)	False Positives (FP)
False Negatives (FN)	True Positives (TP)

From these values, evaluation metrics are computed:

- **Accuracy:** $\frac{TP+TN}{TP+TN+FP+FN}$
- **Precision:** $\frac{TP}{TP+FP}$
- **Recall:** $\frac{TP}{TP+FN}$
- **F1-Score:** $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision}+\text{Recall}}$

4 Raw Data Analysis

Before modeling, I performed exploratory data analysis (EDA) to better understand the dataset. I generated histograms for key features such as **AGE**, **BILIRUBIN**, and **ALBUMIN**, and computed a correlation matrix to examine the relationships between features.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df = pd.read_csv("hepatitis_new.csv")
df.columns = df.columns.str.upper()

# Display basic dataset info
print(df.info())
print(df.describe())

# Check for missing values
print(df.isnull().sum())

# Histograms for key features
plt.figure(figsize=(15, 5))

```

```
plt.subplot(1, 3, 1)
plt.hist(df['AGE'].dropna(), bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Age')

plt.subplot(1, 3, 2)
plt.hist(df['BILIRUBIN'].dropna(), bins=20, color='lightgreen', edgecolor='
    black')
plt.title('Distribution of Bilirubin')

plt.subplot(1, 3, 3)
plt.hist(df['ALBUMIN'].dropna(), bins=20, color='salmon', edgecolor='black')
plt.title('Distribution of Albumin')

plt.tight_layout()
plt.savefig("histogram.pdf")  # Save the combined histograms
plt.show()

# Correlation matrix
plt.figure(figsize=(12, 10))
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix of Features')
plt.savefig("correlation_matrix.pdf")  % Save the correlation matrix image
plt.show()

# Explore the 'CLASS' variable (target)
print(df['CLASS'].value_counts())
```

Listing 1: Python Code for Raw Data Analysis

The generated images, `histogram.pdf` and $\texttt{correlation}_matrix.pdf, areincludedbelow$:

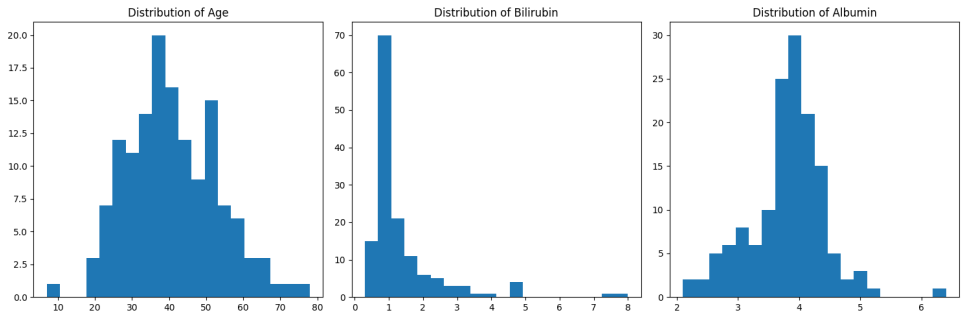

Figure 1: Histogram of Key Features (AGE, BILIRUBIN, ALBUMIN)

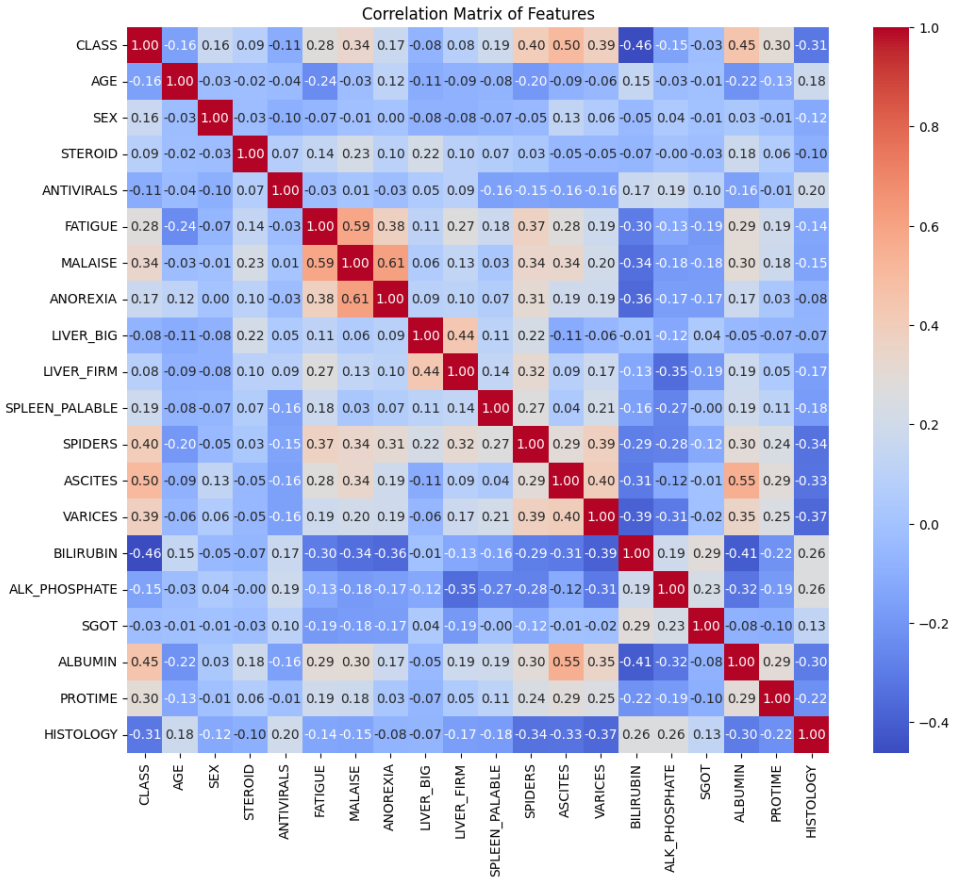

Figure 2: Correlation Matrix of Features

5 Experimental Results

The test set comprised 29 instances. Tables 2 and 3 summarize the performance metrics and confusion matrices, respectively.

5.1 Performance Metrics Table

Table 2: Performance Metrics of Models

Model	Accuracy (%)	Precision (0/1)	Recall (0/1)	F1-Score (0/1)
Random Forest	82.76	0.33 / 0.96	0.67 / 0.85	0.44 / 0.90
SVM	82.76	0.33 / 0.96	0.67 / 0.85	0.44 / 0.90
KNN	79.31	0.29 / 0.95	0.67 / 0.81	0.40 / 0.88
Logistic Regression	86.21	0.40 / 0.96	0.67 / 0.88	0.50 / 0.92
Decision Tree	89.66	0.50 / 0.96	0.67 / 0.92	0.57 / 0.94
Neural Network	82.76	0.33 / 0.96	0.67 / 0.85	0.44 / 0.90
K-Means	—	—	—	—

Note: Metrics for each supervised model are reported as (Class 0 / Class 1), where Class 0 means Die and Class 1 means Live.

5.2 Confusion Matrices Table

Table 3: Confusion Matrices of Models

Model	Confusion Matrix
Random Forest	$\begin{bmatrix} 2 & 1 \\ 4 & 22 \end{bmatrix}$
SVM	$\begin{bmatrix} 2 & 1 \\ 4 & 22 \end{bmatrix}$
KNN	$\begin{bmatrix} 2 & 1 \\ 5 & 21 \end{bmatrix}$
Logistic Regression	$\begin{bmatrix} 2 & 1 \\ 3 & 23 \end{bmatrix}$
Decision Tree	$\begin{bmatrix} 2 & 1 \\ 2 & 24 \end{bmatrix}$
Neural Network	$\begin{bmatrix} 2 & 1 \\ 4 & 22 \end{bmatrix}$
K-Means	$\begin{bmatrix} 0 & 3 \\ 4 & 22 \end{bmatrix}$

5.3 Visualizations

Combined ROC and Precision-Recall curves were generated for models that support probability estimates. These curves provide an aggregated view of model performance across various thresholds.

ROC Curve: The ROC curve plots the True Positive Rate (Recall) against the False Positive Rate (FPR). Its AUC is computed as:

$$\text{AUC} = \int_0^1 \text{TPR}(FPR)\, dFPR$$

A higher AUC indicates better discrimination.

Precision-Recall Curve: This curve plots Precision against Recall, where:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}$$

These curves are particularly useful for imbalanced datasets.

Figures 3 and 4 display the combined ROC and Precision-Recall curves for all applicable models.

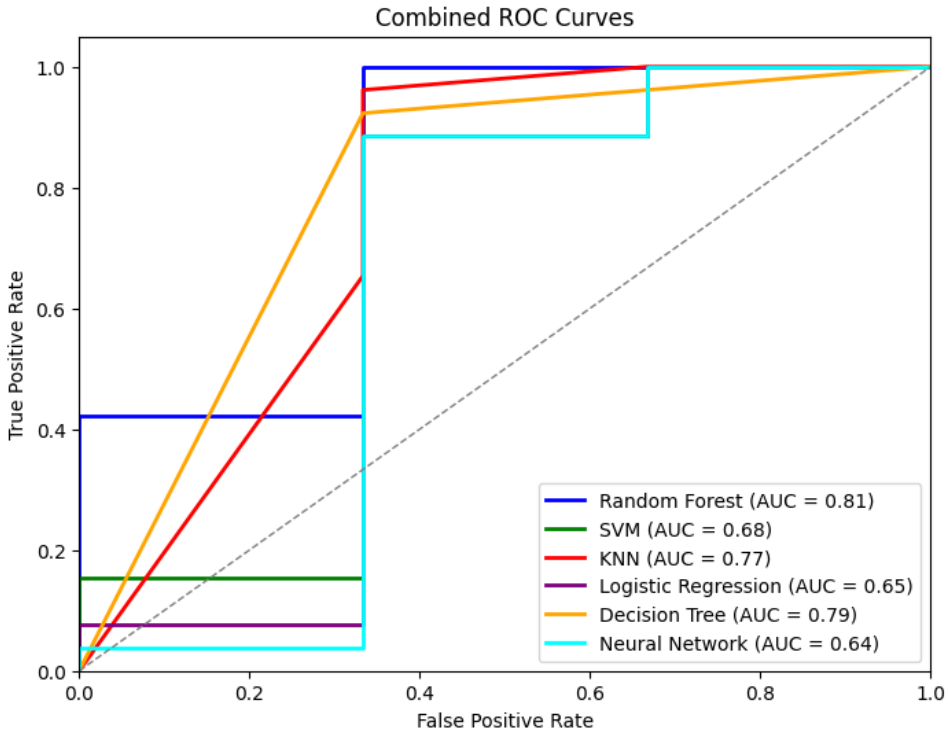

Figure 3: Combined ROC Curves for All Supervised Models

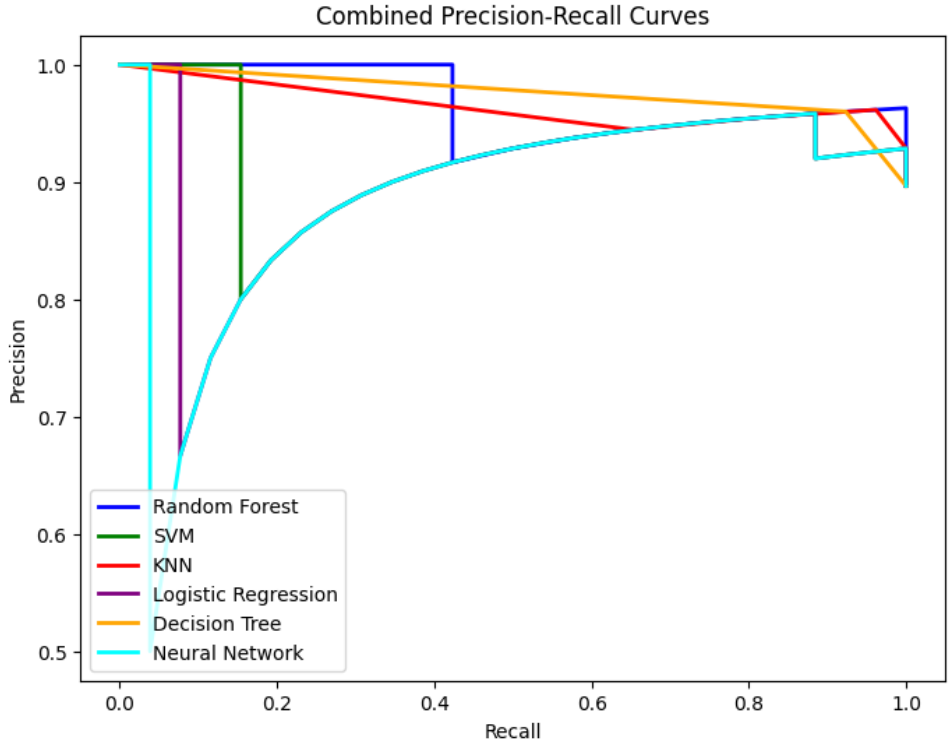

Figure 4: Combined Precision-Recall Curves for All Supervised Models

6 Discussion

`The experimental results indicate:`

- **Decision Tree** achieved the highest accuracy (89.66%), suggesting that its hierarchical structure effectively captures decision boundaries.
- **Logistic Regression** performed strongly with 86.21% accuracy and balanced precision and recall.
- Both **Random Forest** and **SVM** reached 82.76% accuracy, demonstrating competitive performance from ensemble and margin-based methods.
- **KNN** achieved slightly lower accuracy (79.31%), likely due to its sensitivity to the choice of k and feature scaling.
- **Neural Network** obtained 82.76% accuracy, with performance similar to other models.
- **K-Means** does not support probability estimates for ROC/PR curves and provides an inertia score rather than classification accuracy.

The combined ROC and Precision-Recall curves provide further insight into each model's ability to distinguish between classes across various thresholds.

7 Personal Reflection

I faced several challenges during this project, especially in tuning the SVM and Neural Network models. Understanding evaluation metrics such as ROC AUC and Precision-Recal curves was initially difficult, but consulting resources like Analytics Vidhya and freeCodeCamp helped me overcome these hurdles. I was surprised by how well the Decision Tree performed, and overall, I learned a great deal about practical model evaluation and data preprocessing.

8 Future Learning

For future projects, I plan to explore:

- Advanced ensemble methods and deep learning techniques.
- More sophisticated feature selection and dimensionality reduction.
- Ethical implications and interpretability in machine learning for healthcare.
- Integration of real-time diagnostic systems.

9 Conclusion

This study compared multiple machine learning algorithms for hepatitis classification using a dataset from the UCI repository. Supervised models, particularly Decision Tree and Logistic Regression, outperformed others in terms of accuracy and balanced classification metrics. Although K-Means clustering provided insights into the data structure, its unsupervised nature limits its direct applicability for classification tasks. Overall, I gained valuable lessons in data preprocessing, model evaluation, and the challenges of tuning complex models, and I look forward to further exploring advanced techniques.

10 Python Code

Below is the complete Python code used for data preprocessing, model training, evaluation, and visualization.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import logging
import pickle

from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold, cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer  # For handling missing values
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier  # Added Decision Tree
from sklearn.cluster import KMeans
from sklearn.neural_network import MLPClassifier   # Neural Network classifier
from sklearn.metrics import classification_report, confusion_matrix, precision_recall_curve, roc_curve, auc, log_loss

# ============================================================================
# Dataset Description:
#   CSV Header: class, age, sex, steroid, antivirals, fatigue, malaise, anorexia,
#               liver_big, liver_firm, spleen_palable, spiders, ascites, varices,
#               bilirubin, alk_phosphate, sgot, albumin, protime, histology
#
#   Target column: class (e.g., one value represents DIE and the other LIVE)
#
#   No feature selection is applied; all features are used.
# ============================================================================

# Load the dataset (ensure "hepatitis_new.csv" is in your working directory)
df = pd.read_csv("hepatitis_new.csv")

# Convert all column names to uppercase for consistency
df.columns = df.columns.str.upper()

# Print available columns to verify
print("Columns in dataset:", df.columns.tolist())

# Define the target column name (should be 'CLASS' after conversion)
target_col = 'CLASS'
if target_col not in df.columns:
    raise KeyError(f"Target column '{target_col}' not found in dataset.")

# Display first few rows and dataset info
print("\nDataset Head:")
print(df.head())
print("\nDataset Info:")
print(df.info())

# Display distribution of the SEX column (if available)
if 'SEX' in df.columns:
    print("\nDistribution of SEX column:")
    print(df['SEX'].value_counts())
```

```
else:
    print("\nThe 'SEX' column was not found in the dataset.")

# ================================================================================
# Data Preprocessing
# ================================================================================

df.replace('?', np.nan, inplace=True)
imputer = SimpleImputer(strategy='most_frequent')
df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

for col in df.columns:
    try:
        df[col] = pd.to_numeric(df[col])
    except Exception:
        pass

encoder = LabelEncoder()
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = encoder.fit_transform(df[col])

print("\nTarget Class Encoding (e.g., DIE, LIVE):")
le_class = LabelEncoder()
df[target_col] = le_class.fit_transform(df[target_col])
print(dict(zip(le_class.classes_, le_class.transform(le_class.classes_))))

X = df.drop(columns=[target_col])
y = df[target_col]

trainX, testX, trainY, testY = train_test_split(X, y, test_size=0.2, stratify=y
    , random_state=42)

# ================================================================================
# Standardize Features (Zero Mean and Unit Variance)
# ================================================================================
scaler = StandardScaler()
trainX = scaler.fit_transform(trainX)
testX = scaler.transform(testX)

# ================================================================================
# Define Models
# ================================================================================
models = {
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "SVM": SVC(kernel='linear', probability=True, random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "Logistic Regression": LogisticRegression(random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Neural Network": MLPClassifier(hidden_layer_sizes=(100,), random_state=42)
    ,
    "K-Means": KMeans(n_clusters=2, random_state=42)  # Unsupervised clustering
     baseline
}

# ================================================================================
# Define Plotting Functions
# ================================================================================
def plot_conf_matrix(cm, model_name):
    plt.figure(figsize=(4,3))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.title(f"Confusion Matrix - {model_name}")
    plt.xlabel("Predicted Label")
```

```
    plt.ylabel("True Label")
    plt.show()

def plot_precision_recall(y_true, y_scores, model_name):
    precision, recall, _ = precision_recall_curve(y_true, y_scores)
    plt.figure(figsize=(4,3))
    plt.plot(recall, precision, marker='.')
    plt.title(f"Precision-Recall Curve - {model_name}")
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.show()

def plot_roc(y_true, y_scores, model_name):
    fpr, tpr, _ = roc_curve(y_true, y_scores)
    roc_auc = auc(fpr, tpr)
    plt.figure(figsize=(4,3))
    plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
    plt.plot([0, 1], [0, 1], '--', color='gray')
    plt.title(f"ROC Curve - {model_name}")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend(loc="lower right")
    plt.show()

# Combined plots for ROC and Precision-Recall curves for all models that
    support probability outputs.
colors = ['blue', 'green', 'red', 'purple', 'orange', 'cyan']

def plot_combined_roc(models, testX, testY):
    plt.figure(figsize=(8,6))
    i = 0
    for name, model in models.items():
        if hasattr(model, "predict_proba"):
            y_scores = model.predict_proba(testX)[:, 1]
        elif hasattr(model, "decision_function"):
            y_scores = model.decision_function(testX)
        else:
            continue
        fpr, tpr, _ = roc_curve(testY, y_scores)
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, color=colors[i % len(colors)], lw=2,
                 label=f"{name} (AUC = {roc_auc:.2f})")
        i += 1
    plt.plot([0, 1], [0, 1], 'k--', lw=1, color='gray')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Combined ROC Curves")
    plt.legend(loc="lower right")
    plt.show()

def plot_combined_pr(models, testX, testY):
    plt.figure(figsize=(8,6))
    i = 0
    for name, model in models.items():
        if hasattr(model, "predict_proba"):
            y_scores = model.predict_proba(testX)[:, 1]
        elif hasattr(model, "decision_function"):
            y_scores = model.decision_function(testX)
        else:
            continue
        precision, recall, _ = precision_recall_curve(testY, y_scores)
```

```
        plt.plot(recall, precision, color=colors[i % len(colors)], lw=2,
                 label=f"{name}")
        i += 1
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title("Combined Precision-Recall Curves")
    plt.legend(loc="lower left")
    plt.show()

# ================================================================================
# Model Training, Evaluation, and Validation
# ================================================================================

results = []

for name, model in models.items():
    print(f"\n{'='*30}\nModel: {name}\n{'='*30}")
    model.fit(trainX, trainY)
    predictions = model.predict(testX)

    # Validation: Compare training and test accuracy
    train_acc = model.score(trainX, trainY)
    test_acc = model.score(testX, testY)
    print(f"Training Accuracy: {train_acc:.4f}, Test Accuracy: {test_acc:.4f}")
    if train_acc - test_acc > 0.1:
        print("Warning: Model may be overfitting.")
    elif test_acc < 0.6:
        print("Warning: Model may be underfitting.")

    accuracy = test_acc
    cm = confusion_matrix(testY, predictions)
    report = classification_report(testY, predictions)

    print(f"Test Accuracy: {accuracy:.4f}")
    print("Confusion Matrix:")
    print(cm)
    print("Classification Report:")
    print(report)

    results.append([name, round(accuracy * 100, 2), cm])

    plot_conf_matrix(cm, name)

    if hasattr(model, "predict_proba"):
        y_scores = model.predict_proba(testX)[:, 1]
        plot_roc(testY, y_scores, name)
        plot_precision_recall(testY, y_scores, name)
    elif hasattr(model, "decision_function"):
        y_scores = model.decision_function(testX)
        plot_roc(testY, y_scores, name)
        plot_precision_recall(testY, y_scores, name)
    else:
        print(f"{name} does not support probability estimates for ROC/PR curves.")

results_df = pd.DataFrame(results, columns=['Model', 'Accuracy (%)', 'Confusion Matrix'])
print("\nSummary of Model Performance:")
print(results_df)

# Plot combined ROC and Precision-Recall curves for all models that support probability outputs.
plot_combined_roc(models, testX, testY)
```

```
plot_combined_pr(models, testX, testY)
```

Listing 2: Enhanced Hepatitis Classification Code

References

References

[1] Analytics Vidhya, "Machine Learning," Available at: https://www.analyticsvidhya.com/blog/category/machine-learning/?ref=category.

[2] Evaluation of Classification Models in Machine Learning, SciSpace, Available at: https://scispace.com/pdf/evaluation-of-classification-models-in-machine-learning-1u2pog86m5.pdf.

[3] Analytics Vidhya, "Metrics to Evaluate Your Classification Model to Take the Right Decisions," Available at: https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions

[4] Analytics Vidhya, "Pinnacle Plus," Available at: https://www.analyticsvidhya.com/pinnacleplus/?ref=blognavbar.

[5] Kombib, "21 Steps to Design a Machine Learning System from Scratch," Available at: https://medium.com/@kombib/21-steps-to-design-a-machine-learning-system-from-scratch-02c083cb785d.

[6] freeCodeCamp, "Machine Learning Course," Available on YouTube at: https://www.youtube.com/c/Freecodecamp.

[7] GeeksforGeeks, "Machine Learning Algorithms," Available at: https://www.geeksforgeeks.org/machine-learning/.

[8] AssemblyAI, "Machine Learning from Scratch Playlist," Available on YouTube at: https://www.youtube.com/c/AssemblyAI.

11 Personal Reflection

I faced several challenges during this project, especially in tuning the SVM and Neural Network models. Understanding evaluation metrics such as ROC AUC and Precision-Recal curves was initially difficult, but consulting resources like Analytics Vidhya and freeCodeCamp helped me overcome these hurdles. I was surprised by how well the Decision Tree performed, and overall, I learned a great deal about practical model evaluation and data preprocessing.

12 Future Learning

For future projects, I plan to explore:

- Advanced ensemble methods and deep learning techniques.
- More sophisticated feature selection and dimensionality reduction.
- Ethical implications and interpretability in machine learning for healthcare.
- Integration of real-time diagnostic systems.