# Heart Disease Prediction Using Machine Learning

**Abstract**

This notebook explores the application of various machine learning models to predict heart disease from clinical parameters. We employ models such as Logistic Regression, Random Forest, and Gradient Boosting to analyze a dataset containing patient information. The notebook evaluates model performances and identifies the best-performing model based on accuracy metrics. The final selected model can predict heart disease presence with an accuracy exceeding 85%.

**Introduction**

Heart disease is one of the leading causes of death globally. Early prediction and diagnosis can significantly increase the effectiveness of treatment. This notebook uses data science and machine learning techniques to predict the likelihood of heart disease based on physiological and medical test data from patients.

**About the Dataset**

The dataset originates from the UCI Machine Learning Repository and includes 918 records of patients. Each record contains 12 attributes such as age, sex, cholesterol levels, and ECG results, which are used to predict the presence of heart disease (binary outcome).

**Attributes:**

Age: Patient's age in years

Sex: Male (M) or Female (F)

ChestPainType: Type of chest pain (e.g., ATA, NAP)

RestingBP: Resting blood pressure

Cholesterol: Serum cholesterol in mg/dl

**Methodology**

The methodology section explains the use of a preprocessing pipeline for scaling and encoding data, followed by splitting the data into training and test sets. We evaluate five different models: Logistic Regression, Decision Tree, Random Forest, SVM, and Gradient Boosting. Model performance is assessed using accuracy as the primary metric.

**Code**

# Import necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

import joblib

# Load the dataset

data\_path = r'C:\Users\Linata04\Desktop\Final Project\Dataset\heart.csv'  # Use a raw string

heart\_data = pd.read\_csv(data\_path)

# Define categorical and numerical features

categorical\_features = heart\_data.select\_dtypes(include=['object']).columns.tolist()

numeric\_features = [col for col in heart\_data.columns if col not in categorical\_features + ['HeartDisease']]

# Data preprocessing

preprocessor = ColumnTransformer(

    transformers=[

        ('num', StandardScaler(), numeric\_features),

        ('cat', OneHotEncoder(), categorical\_features)

    ])

# Define models to compare

models = {

    "Logistic Regression": LogisticRegression(random\_state=0),

    "Decision Tree": DecisionTreeClassifier(random\_state=0),

    "Random Forest": RandomForestClassifier(random\_state=0),

    "SVM": SVC(random\_state=0),

    "Gradient Boosting": GradientBoostingClassifier(random\_state=0)

}

# Split data into features and target variable

X = heart\_data.drop('HeartDisease', axis=1)

y = heart\_data['HeartDisease']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Initialize dictionary to hold trained pipelines for saving

trained\_pipelines = {}

# Train each model and evaluate

accuracies = {}

for name, model in models.items():

    pipeline = Pipeline([

        ('preprocessor', preprocessor),

        ('classifier', model)

    ])

    pipeline.fit(X\_train, y\_train)

    y\_pred = pipeline.predict(X\_test)

    accuracy = accuracy\_score(y\_test, y\_pred)

    accuracies[name] = accuracy

    trained\_pipelines[name] = pipeline  # Store the trained pipeline

# Sort models based on accuracy and save the top 3

top\_models = sorted(accuracies, key=accuracies.get, reverse=True)[:3]

for model\_name in top\_models:

    joblib.dump(trained\_pipelines[model\_name], f'{model\_name.replace(" ", "\_")}\_model.pkl')

# Print model accuracies and top models saved

print("Model Accuracies:")

for model\_name, acc in accuracies.items():

    print(f"{model\_name}: {acc:.3f}")

print("\nTop 3 Models Saved:")

for model\_name in top\_models:

    print(f"- {model\_name}")

**Import Libraries**

The code begins by importing necessary libraries:

Pandas for data manipulation.

Sklearn.model\_selection for splitting the dataset into training and test sets.

Sklearn.preprocessing for scaling numeric features and encoding categorical features.

Sklearn.compose for applying different transformations to different types of features simultaneously.

Sklearn.pipeline for chaining preprocessors and model training steps into a single callable object.

Sklearn's LogisticRegression, DecisionTreeClassifier, RandomForestClassifier, GradientBoostingClassifier, and SVC for different predictive modeling approaches.

Sklearn.metrics for evaluating model performance (accuracy).

Joblib for saving the trained models.

**Load Dataset**

data\_path: Specifies the file path to the dataset.

heart\_data: Loads the dataset from the specified path into a pandas DataFrame.

Identify Feature Types

categorical\_features: Lists all column names that contain categorical data.

numeric\_features: Lists all column names that contain numerical data excluding the target variable 'HeartDisease'.

**Data Preprocessing**

ColumnTransformer: This component combines two transformations: scaling for numerical features and one-hot encoding for categorical features.

StandardScaler: Normalizes numerical features so they contribute equally to model training.

OneHotEncoder: Converts categorical variables into a format that can be provided to ML algorithms to do a better job in prediction.

**Define and Compare Models**

models: A dictionary of different machine learning algorithms initialized with their default parameters.

Pipeline: Creates a pipeline that first applies the preprocessor and then fits the model. This ensures all steps in the process are applied correctly when making predictions.

**Split the Data**

X and y: Separate the features from the target variable.

train\_test\_split: Splits the data into training and testing sets.

Train Models and Evaluate Accuracy

The code loops through each model, trains it using the training set, and evaluates it on the test set using accuracy as the metric.

Each trained model is stored in a dictionary called trained\_pipelines.

**Save Top 3 Models**

top\_models: Identifies the three models with the highest accuracy.

joblib.dump: Saves these models to the disk for later use.

**Output**

Prints the accuracies of all models and the names of the top three models saved.

**Usage**

This script is useful for comparing the effectiveness of different machine learning models on a specific dataset and can be adapted for use with other datasets or models. Saving the top models allows for easy deployment in a production environment or further evaluation on new data.

import streamlit as st

import pandas as pd

import joblib

# Function to load a saved model

def load\_model(model\_name):

    # Adjust the path if your models are saved in a different directory

    return joblib.load(f'{model\_name.replace(" ", "\_")}\_model.pkl')

# Title of the application

st.title('Heart Disease Prediction Application')

# Description

st.write("This application predicts the likelihood of a heart disease based on input parameters. \

         It uses the top 3 performing models: Gradient Boosting, Random Forest, and SVM.")

# Model names list (make sure these names match the names used in your training script)

model\_names = ["Gradient Boosting", "Random Forest", "SVM"]

# Load models (you might need to adjust this if your environment differs)

models = {name: load\_model(name) for name in model\_names}

# User input fields

st.sidebar.header('User Input Parameters')

def user\_input\_features():

    age = st.sidebar.slider('Age', 18, 100, 50)

    sex = st.sidebar.selectbox('Sex', ('M', 'F'))

    chest\_pain\_type = st.sidebar.selectbox('Chest Pain Type', ('ATA', 'NAP', 'ASY', 'TA'))

    resting\_bp = st.sidebar.slider('Resting BP', 90, 200, 120)

    cholesterol = st.sidebar.slider('Cholesterol', 100, 600, 200)

    fasting\_bs = st.sidebar.selectbox('Fasting Blood Sugar > 120 mg/dl', (0, 1))

    resting\_ecg = st.sidebar.selectbox('Resting ECG', ('Normal', 'ST', 'LVH'))

    max\_hr = st.sidebar.slider('Maximum Heart Rate', 60, 220, 150)

    exercise\_angina = st.sidebar.selectbox('Exercise Induced Angina', ('Y', 'N'))

    oldpeak = st.sidebar.slider('Oldpeak', 0.0, 6.0, 2.0)

    st\_slope = st.sidebar.selectbox('ST Slope', ('Up', 'Flat', 'Down'))

    # Create a DataFrame of the input features

    data = {

        'Age': [age],

        'Sex': [sex],

        'ChestPainType': [chest\_pain\_type],

        'RestingBP': [resting\_bp],

        'Cholesterol': [cholesterol],

        'FastingBS': [fasting\_bs],

        'RestingECG': [resting\_ecg],

        'MaxHR': [max\_hr],

        'ExerciseAngina': [exercise\_angina],

        'Oldpeak': [oldpeak],

        'ST\_Slope': [st\_slope]

    }

    return pd.DataFrame(data)

input\_df = user\_input\_features()

# Display the user input features

st.subheader('User Input parameters for prediction')

st.write(input\_df)

# Button to perform prediction

if st.button('Predict Heart Disease'):

    st.subheader('Prediction results')

    for name, model in models.items():

        prediction = model.predict(input\_df)

        st.write(f"{name}: {'Heart Disease' if prediction[0] == 1 else 'Normal'}")

**What This Script Does:**

**Import Libraries**

Streamlit (st): Used to create the web app interface.

Pandas (pd): Handles data manipulation.

Joblib: Loads trained machine learning models from disk.

**Function to Load Models**

load\_model(model\_name): This function takes a model name as input and loads the corresponding model from disk using Joblib. The model name is formatted to match the filename (spaces are replaced with underscores).

**Streamlit Application Setup**

Title and Description: Sets up the title of the web application and a brief description explaining its purpose.

**Model Management**

model\_names: A list of the model names. These should match the names used when the models were trained and saved.

models: A dictionary comprehension is used to load all the models by their names into memory for use in the application.

**User Input Fields**

Sidebar for Input: Uses Streamlit's sidebar feature to gather user inputs for various health parameters.

user\_input\_features(): A function that collects input from the user through sliders and select boxes. Each input corresponds to a feature used by the machine learning models. This function returns a pandas DataFrame containing the user's inputs.

**Display User Inputs**

The input features provided by the user are displayed under a subheader called 'User Input parameters for prediction'.

**Prediction Button and Results**

Predict Button: When clicked, this button triggers the prediction process using the loaded models.

Display Predictions: For each model, the prediction is displayed, indicating whether the model predicts 'Heart Disease' or 'Normal' based on the user's input.

**How It Works in Practice**

User Interaction: The user adjusts parameters using sliders and select boxes on the sidebar. These parameters include age, sex, type of chest pain, and others relevant to predicting heart disease.

Data Handling: Once the parameters are set, the user\_input\_features() function formats them as a DataFrame which the models can use for making predictions.

Model Predictions: Upon clicking the "Predict Heart Disease" button, the application uses each loaded model to predict the outcome based on the user inputs. Each model's prediction is then displayed, informing the user of the potential health outcome.

Patients with heart disease

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Patients with without heart disease : Normal

A screenshot of a computer

Description automatically generated

**Conclusion**

This notebook presented a comprehensive approach to predicting heart disease using various machine learning models on a clinically relevant dataset. The analysis journey began with data preprocessing to transform raw data into a format suitable for modeling. We implemented a range of models including Logistic Regression, Decision Tree, Random Forest, SVM, and Gradient Boosting, each evaluated based on their accuracy to predict the presence of heart disease.

**Future Work:**

Further research could explore more sophisticated feature engineering techniques to unearth more complex relationships within the data.

Implementing a grid search or random search for hyperparameter optimization could refine model performances further.

Expanding the dataset or using different subsets could help validate the models more robustly and ensure their generalizability to other populations.

# AI Generate Music

I had a great time producing these AI-generated songs using a variety of tools to bring my ideas to life. Initially, I used the Suno application to specify the type of song I wanted, such as a rock climber theme for rock music. I detailed the background music vibe I was aiming for as well.

Next, I used ChatGPT to read the lyrics and obtain corresponding images. For animation, I turned to Pika and Haiper, which helped me create 3-second video clips by animating the images. To compile everything, Capcut was instrumental in synchronizing the AI-generated audio with the motion images to produce a cohesive video album.

Throughout this process, I explored various themes and musical genres, creating a diverse collection of song albums that visually displayed the lyrics in sync with the tune.

A screenshot of a book

Description automatically generated