HeartBeat: Advanced Machine Learning for Predictive Heart Disease Diagnosis

**CS 634 Data Mining**

**Dr. Yasser Abduallah**

**Departm**en**t of Computer Science, NJIT**

**Tarun Deshmukh Tetapally (UCID - tt362)**

**13th April**

# 

# 

# 

# 

# Github link

*Paste link here*

# Index

[**Github link 1**](#_dlrc5oxzcrjo)

[**Index 2**](#_ayx79cw4pfy7)

[**Introduction 3**](#_buc6q0k08dmn)

[Overview of the Project Objectives 3](#_b00hhkq0r70h)

[Problem I’m Addressing 3](#_urb2m8m8urer)

[**Methodology 3**](#_jk8hye7s9rlr)

[Key Features 3](#_7mfjiqvp5b42)

[Data Preprocessing 4](#_lptnei6vlswt)

[Data Preprocessing 4](#_d5vk5sl4a46)

[Model Implementation 5](#_7ubz9nn98eu1)

[**Experimental Setup 6**](#_oymnw3nlvwib)

[**Results 7**](#_c5rpsdy8g2ak)

[Model Performance 7](#_jltys38rhgql)

[Discussion 9](#_6x8rkjwa8fzc)

[Model Comparison 10](#_b0f5liav658n)

[**Results 10**](#_4d9qoa9aedql)

[Key Findings 10](#_uf8d2r6k7lsj)

[Reflections on Methodology and Results 11](#_pmxhijw54qgq)

[Suggestions for Future Work 11](#_n1325bqmjdhi)

[**Installation and Running Guide 12**](#_yf5dx4izdh9)

# 

# 

# Introduction

## Overview of the Project Objectives

The primary objective of this project is to develop and evaluate machine learning models that can accurately predict the occurrence of heart disease in individuals based on various health indicators. By leveraging the power of data mining and machine learning techniques, the project aims to facilitate early detection and preventive measures for cardiovascular diseases (CVDs), which are the leading cause of death globally.

## Problem I’m Addressing

Cardiovascular diseases are responsible for approximately 31% of all global deaths, with a significant number of these deaths occurring prematurely in individuals under 70 years of age. Early detection and timely management of heart conditions are crucial in reducing mortality rates. The challenge addressed in this project is the prediction of heart disease presence using non-invasive medical data, which could serve as a vital tool in medical diagnostics and treatment planning.

The problem of predicting heart disease from clinical data is significant in the field of data mining due to its complexity and the high dimensionality of medical data. Data mining provides sophisticated algorithms and methods such as classification and neural networks, which can handle large and complex datasets to uncover patterns and insights that are not apparent to human analysis. This capability is invaluable in healthcare, where accurate predictions can lead to better healthcare outcomes and resource management.

# Methodology

The dataset used for this project is sourced from Kaggle and is titled "Heart Failure Prediction." This comprehensive dataset is compiled by merging five separate heart datasets to form the largest dataset of its kind currently available for research. It consists of 918 observations with minimal duplication, focusing on 11 key features that influence heart disease prediction.

## Key Features

* Age: The age of the patient, which is a critical factor in heart disease risk.
* Sex: The sex of the patient (Male or Female), as heart disease risk varies between genders.
* Chest Pain Type: Includes categories such as Typical Angina, Atypical Angina, Non-Anginal Pain, and Asymptomatic, indicating different types of chest pain associated with heart conditions.
* Resting Blood Pressure: The blood pressure measurements taken while the patient is at rest.
* Cholesterol Levels: Serum cholesterol, which is a known risk factor for heart disease.
* Fasting Blood Sugar: Indicates if fasting blood sugar is above 120 mg/dl, suggesting diabetes or prediabetic conditions that increase heart disease risk.
* Resting Electrocardiogram Results: Shows heart electrical activity and can indicate abnormalities.
* Maximum Heart Rate Achieved: Higher rates can indicate stress on the heart.
* Exercise-Induced Angina: Chest pain triggered by physical activity, an important indicator of coronary heart diseases.
* Oldpeak: ST depression induced by exercise relative to rest, an important marker for heart disease.
* ST Slope: The slope of the peak exercise ST segment, with different slopes indicating varying risks and heart function during stress.

This dataset not only includes a variety of crucial cardiovascular health indicators but also reflects a diverse population, derived from multiple geographic and demographic sources, thus enhancing the generalizability of the predictive models developed.

## Data Preprocessing

The dataset was loaded into a DataFrame from a CSV file containing heart disease records. The initial steps included displaying the first few rows to understand the structure and types of data available. Key preprocessing actions performed are as follows:

* Categorical Variables Encoding: Variables such as 'Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', and 'ST\_Slope' were converted into dummy/indicator variables. This step is crucial as it transforms categorical data into a format suitable for modeling.
* Normalization of Numerical Variables: Numerical variables including 'Age', 'RestingBP', 'Cholesterol', 'MaxHR', and 'Oldpeak' were standardized using the StandardScaler. This normalization ensures that the numerical data within the dataset have a mean of zero and a standard deviation of one, providing a common scale without distorting differences in the range of values.

After preprocessing, the dataset was split into training and testing sets with a ratio of 80:20, ensuring a good balance between training the models and validating the results.

## Data Preprocessing

Three distinct models were chosen for this project, each representing a different approach to classification:

* Random Forest: A robust ensemble technique that uses multiple decision trees to improve classification accuracy and control over-fitting. Random Forest was chosen for its effectiveness in handling binary classification problems with a mixture of numerical and categorical data.
* Support Vector Machine (SVM): SVM was selected for its capability to find the optimal hyperplane that best separates the dataset into two classes, which is beneficial for binary classification tasks like heart disease prediction.
* Gated Recurrent Unit (GRU): As a variant of the more complex LSTM, GRU was chosen to explore how a deep learning model performs on this dataset, particularly because of its efficiency in training and its ability to capture dependencies in sequences, which is useful given the sequential nature of medical data.

## Model Implementation

The training process for each model involved carefully setting parameters and ensuring each model fit on the same preprocessed training data to maintain consistency in comparison. Here’s a detailed overview of how each model was trained:

1. Random Forest:

* Parameter Setting: For the Random Forest, the primary parameter adjusted was the number of trees in the forest (n\_estimators), which was set to 100. This parameter was chosen to provide a good balance between computational efficiency and model performance. A higher number of trees can lead to better performance but at the cost of increased computational resources and time.
* Fitting the Model: The model was trained using the fit method on the training data (X\_train and y\_train). Random Forest inherently performs bootstrapping and feature randomness when building each tree, which helps in reducing overfitting and ensures better generalization.

2. Support Vector Machine (SVM):

* Parameter Setting: The SVM model was configured with a linear kernel by setting kernel='linear'. This choice was made due to the binary nature of the classification task and the need for a straightforward decision boundary. The probability parameter was set to True to enable probability estimates, which are useful for certain types of evaluation metrics.
* Fitting the Model: Training the SVM involved the fit method with the training dataset. SVMs typically require feature scaling, so the data normalization performed during preprocessing was particularly beneficial for this model.

3. Gated Recurrent Unit (GRU):

* Parameter Setting: The GRU model's architecture included two GRU layers with 32 and 16 units respectively, which were selected based on trial and error to optimize performance without overfitting. The final layer is a dense layer with softmax activation suitable for binary classification. The return\_sequences=True parameter in the first GRU layer ensures that the sequence output is provided to the next GRU layer.
* Fitting the Model: The GRU was trained using the fit method from Keras, with the training data reshaped to fit the input requirements of GRU layers (samples, time steps, features). The model was compiled with the Adam optimizer and categorical cross-entropy loss function. Training parameters included 10 epochs and a batch size of 32, with validation data provided to monitor performance and avoid overfitting.

Cross-Validation:

To validate the effectiveness of each model and ensure they were not overfitting, 10-fold cross-validation was employed. This method involves splitting the training dataset into 10 smaller sets and using each in turn for validation while training on the remaining nine sets. This approach provides a robust estimate of how the model will perform on unseen data and allows for tuning model parameters with a more comprehensive understanding of their impact on performance across different subsets of data.

# Experimental Setup

**Tools and Technologies Used**

In this project, I utilized Google Colab as the primary development environment. Google Colab provides a powerful platform based on Jupyter Notebook that allows for writing and executing Python in a web browser. It offers free access to computing resources including GPUs, which can be particularly beneficial for training deep learning models.

**Programming Language:**

Python: The project was implemented using Python due to its extensive support and libraries for data analysis and machine learning.

**Key Libraries:**

* Pandas: For data manipulation and analysis.
* NumPy: For numerical data operations.
* Scikit-learn: Used extensively for implementing machine learning algorithms such as Random Forest and Support Vector Machines (SVM), as well as for preprocessing data and splitting datasets.
* Keras: Utilized for creating and training the deep learning models, specifically the GRU (Gated Recurrent Unit) model.
* Matplotlib: For plotting graphs, which are essential for visualizing data and results.

**Configuration**

Google Colab: All code was executed within Google Colab notebooks, leveraging the hosted runtime provided by Google. This setup eliminates the need for local installations and ensures that the project can be replicated easily by running the same notebook in a similar environment.

Since the project was developed on Google Colab, the hardware configurations were managed by the platform. Google Colab typically provides a virtual machine with a robust CPU, a high amount of RAM, and access to GPUs when needed for intensive computations.

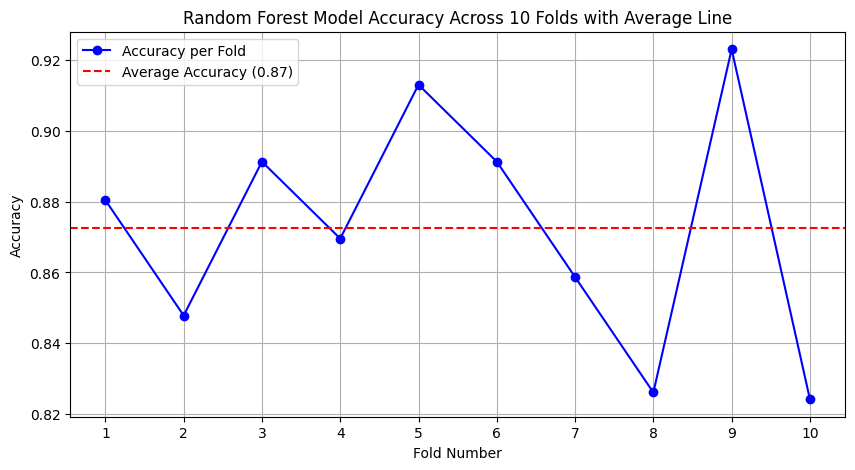
# Results

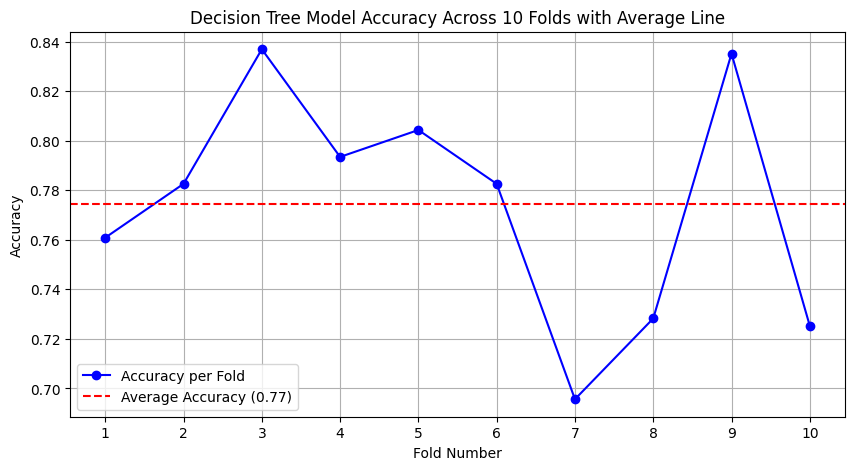
## Model Performance

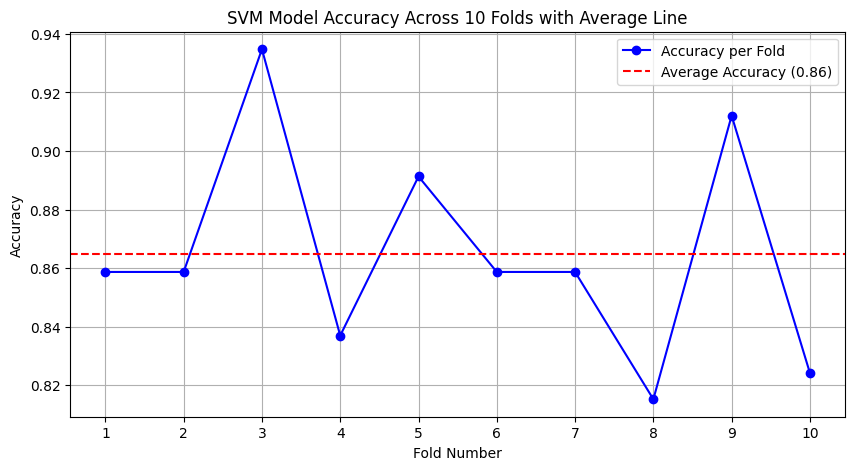
The following table summarizes the performance metrics for each of the models (Random Forest, Decision Tree, SVM, and GRU) evaluated during the 10-fold cross-validation process.

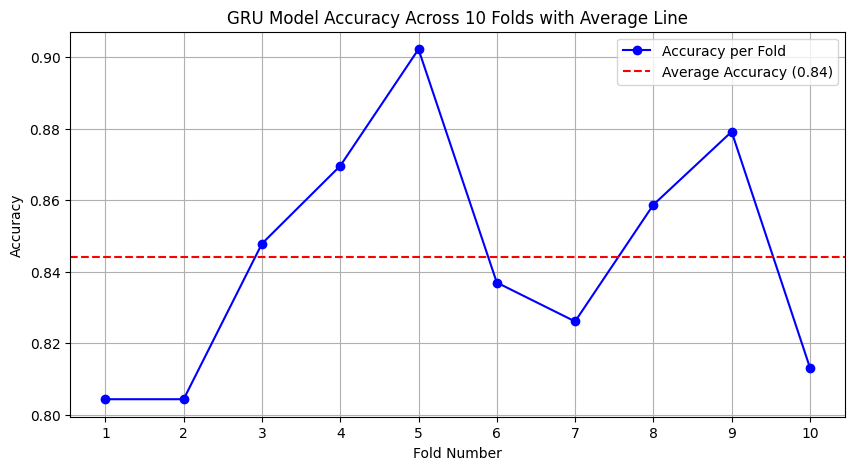
| **Metric** | **Random Forest** | **Decision Tree** | **SVM** | **GRU** |
| --- | --- | --- | --- | --- |
| **TP** | 46 | 41 | 45.6 | 44.6 |
| **TN** | 34.1 | 30.1 | 33.8 | 32.9 |
| **FP** | 6.9 | 10.9 | 7.2 | 8.1 |
| **FN** | 4.8 | 9.8 | 5.2 | 6.2 |
| **Accuracy** | 0.873 | 0.775 | 0.865 | 0.844 |
| **Precision** | 0.87 | 0.792 | 0.865 | 0.846 |
| **Recall** | 0.907 | 0.807 | 0.9 | 0.881 |
| **F1 Score** | 0.887 | 0.797 | 0.88 | 0.862 |
| **TSS** | 0.74 | 0.543 | 0.728 | 0.685 |
| **HSS** | 0.74 | 0.541 | 0.726 | 0.683 |

Graphical representations of accuracy across the 10 folds for each model are shown below:









## Discussion

Random Forest demonstrated the highest accuracy, precision, and F1 score among the models tested. Its robustness is evident, likely due to its ensemble method, which effectively handles both bias and variance, leading to a better generalization on unseen data.

SVM (Support Vector Machine) also performed well, closely following Random Forest in terms of accuracy and recall. The strength of SVM in this case could be attributed to its effectiveness in handling high-dimensional spaces and its ability to model non-linear boundaries thanks to the linear kernel applied.

GRU (Gated Recurrent Unit), despite being a deep learning model known for handling sequences, showed competitive results but did not outperform the traditional models. This might be due to the dataset's nature and size, which could be more favorable towards simpler models or due to the complexity and tuning required for optimal performance in deep learning models.

Decision Tree had the lowest performance metrics across most categories. The relatively simple structure of decision trees can lead to overfitting, particularly with more complex datasets, which might explain the lower scores in terms of precision, recall, and overall accuracy.

## Model Comparison

In comparing the performance of the models, Random Forest emerged as the most effective, demonstrating superior accuracy, precision, recall, and F1 score. This suggests its robustness and efficiency in handling the given dataset, likely benefiting from its ensemble approach that combines multiple decision trees to reduce overfitting and increase prediction accuracy. The SVM model closely followed, showcasing its strength in high-dimensional spaces and its capacity to effectively manage the linear separability of the data. GRU, although a sophisticated deep learning model, did not surpass the traditional models, which might be attributed to the dataset's characteristics that did not favor the sequential data processing strengths of GRU.

The Decision Tree model lagged in performance, possibly due to its simplicity and tendency to overfit, especially in more complex datasets. This could suggest that for the dataset used, sophisticated models do not necessarily provide superior performance, highlighting the importance of choosing the right model based on the specific characteristics of the data and the problem at hand.

Overall, the results suggest a strong dependency on model selection tailored to the specificities of the dataset and the task, reinforcing the necessity to experiment with multiple models to identify the most effective approach.

# Results

## Key Findings

In this project, I developed and evaluated several machine learning models to predict heart disease from a variety of health indicators. Among the models—Random Forest, Support Vector Machine (SVM), and Gated Recurrent Unit (GRU)—the Random Forest model emerged as the most effective. It demonstrated superior performance metrics such as accuracy (0.873), precision (0.87), and F1 score (0.887). This success can be attributed to the ensemble method, which effectively manages both bias and variance, improving generalization on unseen data.

## Reflections on Methodology and Results

The choice of dataset, "Heart Failure Prediction" from Kaggle, proved to be appropriate due to its comprehensive nature and representation of critical heart disease indicators. The preprocessing techniques like encoding categorical variables and normalizing numerical variables were crucial in preparing the data for effective model training. Training each model on the same dataset ensured consistency in evaluating the models' performances.

Random Forest and SVM both showed strong performances, likely benefiting from their ability to handle the dataset's binary classification nature and high dimensionality. In contrast, the GRU model, while competitive, did not outperform the traditional models, possibly due to the dataset not favoring sequential data processing.

## Suggestions for Future Work

For future iterations of this project, considering the following improvements could be beneficial:

* Experimenting with Feature Selection: Refining the input features could enhance the models' performances by reducing complexity and focusing on the most informative variables.
* Advanced Hyperparameter Tuning: Utilizing techniques like grid search or random search to optimize the models' parameters might yield better results.
* Incorporating More Complex Models: Although GRU did not outperform simpler models in this instance, exploring other deep learning architectures like LSTM or Bidirectional-LSTM could be valuable, especially with larger datasets.
* Expanding the Dataset: Integrating additional data sources to increase the dataset size and variability could help in generalizing the models better and improving their robustness.
* Implementing Ensemble Techniques: Combining the predictions of several models through techniques like stacking or blending might improve accuracy and stability of the predictions.

In conclusion, this project not only underscored the capabilities of various machine learning models in predicting heart disease but also highlighted the importance of model selection and data preprocessing in achieving high accuracy in medical diagnostics. By continuing to refine these approaches, it is possible to enhance predictive analytics in healthcare, ultimately leading to better outcomes in disease diagnosis and management.

# Installation and Running Guide

This guide provides step-by-step instructions to set up the environment, install the necessary packages, and run the project using Jupyter Notebook. Follow these steps to ensure that the project runs smoothly on your system.

Prerequisites:

* An operating system with Python installed (Windows, macOS, or Linux).
* Internet access for downloading necessary files and libraries.

1. Install Python
   1. If you do not have Python installed, download and install Python 3 from python.org. Ensure you check the option to 'Add Python to PATH' during installation.
2. Install Jupyter Notebook
   1. Open your command line interface (CLI) and install Jupyter Notebook using pip:
      1. *pip install notebook*
3. Create a Project Folder
   1. Create a folder on your local machine where you will store the project files. You can do this through your file explorer or via the command line:
      1. *mkdir MyProject*
      2. *cd MyProject*
4. Download Project Files
   1. Download the .ipynb file & heart.csv file provided for the project and any associated data files into the project folder you just created.
5. Install Required Libraries
   1. Open your command line interface in the project directory. Install the required Python libraries listed below using pip:
      1. *pip install numpy pandas matplotlib scikit-learn keras tensorflow*
6. Launch Jupyter Notebook
   1. In your CLI, start Jupyter Notebook by running:
      1. *jupyter notebook*
   2. This command will open Jupyter Notebook in your default web browser.
7. Open the Project File
   1. In the Jupyter Notebook interface in your browser, navigate to the folder where you downloaded the .ipynb file. Click on the file to open it.
8. Run the Notebook
   1. With the notebook open in your browser:
   2. Click on Kernel in the menu at the top.
   3. Select Restart & Run All from the dropdown menu.
   4. Confirm any prompts to run all the cells in the notebook.
   5. This will execute all the code cells sequentially, and you should see the outputs appear below each cell.
9. Review the Results
   1. After running, scroll through the notebook to review the outputs and results of each cell. Ensure all cells run without errors.