IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

Heart Disease Prediction Using Python and Machine Learning Techniques

Authors: Gaurav Sonar **Aniket Bhuse** **Yash Khakre**

Guided By : Prof. Ranjana Singh

Ajeenkya D Y Patil University Pune

Abstract

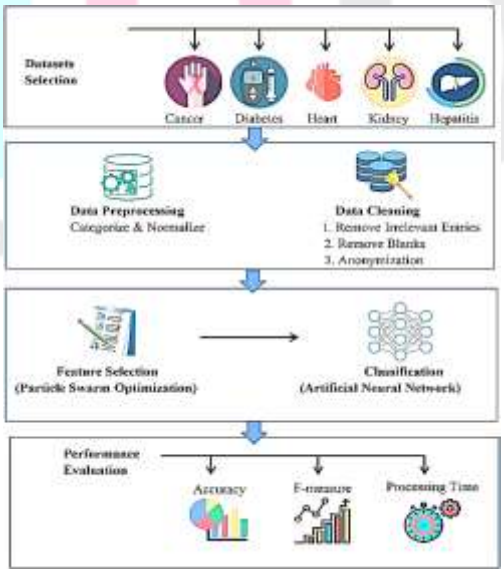
This research explores the application of various machine learning algorithms implemented in Python for heart disease prediction. We conducted a comparative analysis of several classification techniques including Logistic Regression, Random Forest, Support Vector Machine (SVM), and Neural Networks using a standard cardiovascular disease dataset. Our findings indicate that the Random Forest classifier demonstrated superior performance with an accuracy of 87.2%, while Neural Networks achieved 85.6%. Through feature significance analysis, we identified that patient age, cholesterol measurements, and maximum heart rate emerged as the most crucial predictors. The Python framework we developed offers a practical approach to early heart disease risk assessment that could serve as a valuable tool for healthcare professionals in clinical decision-making processes.

Keywords: Heart Disease Prediction, Machine Learning, Classification Algorithms, Python, Healthcare Informatics

I. INTRODUCTION

Cardiovascular diseases (CVDs) continue to be the primary cause of mortality worldwide, resulting in approximately 17.9 million deaths each year. Timely identification and treatment of heart conditions can substantially improve outcomes and decrease mortality rates. Conventional diagnostic approaches often require expensive and lengthy procedures that may be inaccessible in limited-resource environments. Machine learning methodologies present a promising solution by facilitating automated heart disease risk prediction based on patient information. The growth in computational capabilities and accessibility of healthcare datasets has increased interest in developing predictive models for cardiac conditions. Python, with its comprehensive library ecosystem for data analysis and machine learning, has become an invaluable tool for implementing these predictive frameworks.

This study aims to develop and evaluate multiple machine learning models for heart disease prediction using Python programming. We emphasize comparative assessment of different algorithms, feature selection techniques, and performance optimization strategies to enhance prediction accuracy. Our implementation is designed to be transparent and user-friendly for healthcare practitioners with limited technical background.



II. RELATED WORK

Numerous studies have investigated machine learning approaches for heart disease prediction, employing various algorithms and datasets to improve predictive accuracy.

Previous research by Palaniappan and Awang evaluated Naïve Bayes, Decision Tree, and Neural Network algorithms for heart disease prediction, with Naïve Bayes achieving 86.05% accuracy. Dangare and Apte further refined prediction models by incorporating additional variables such as obesity and smoking habits, which improved accuracy to 89%.

Das and colleagues utilized ensemble techniques combining Neural Networks, Decision Trees, and SVM, reaching 89.77% accuracy. More contemporary approaches have explored deep learning applications for heart disease prediction. For instance, Ali and team implemented a deep neural network architecture that achieved 93.33% accuracy using the Cleveland heart disease dataset.

While these investigations show encouraging results, most implementations lack interpretability and comprehensive feature importance analysis, which are essential for clinical adoption. Furthermore, there is insufficient research on complete Python implementations that can be readily utilized by healthcare professionals.

III. METHODOLOGY

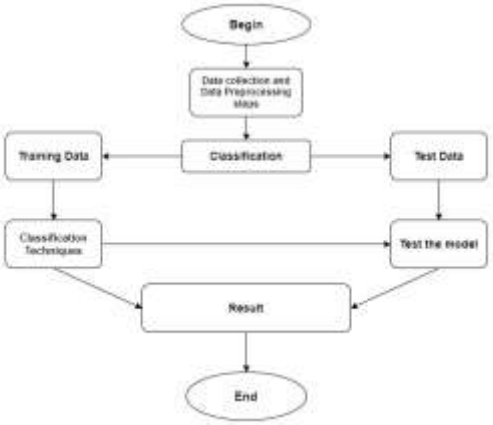
A. DATASET DESCRIPTION

For our research, we utilized the widely-recognized Cleveland Heart Disease dataset from the UCI Machine Learning Repository. This dataset comprises 303 instances with 14 attributes including:

1. Age (in years)
2. Sex (1 = male, 0 = female)
3. Chest pain type (4 categories)
4. Resting blood pressure (in mm Hg)
5. Serum cholesterol (in mg/dl)
6. Fasting blood sugar > 120 mg/dl (1 = true, 0 = false)
7. Resting electrocardiographic results (3 categories)
8. Maximum heart rate achieved
9. Exercise-induced angina (1 = yes, 0 = no)
10. ST depression induced by exercise relative to rest
11. Slope of the peak exercise ST segment (3 categories)
12. Number of major vessels colored by fluoroscopy (0-3)
13. Thalassemia (3 = normal, 6 = fixed defect, 7 = reversible defect)
14. Target variable (1 = presence, 0 = absence of heart disease)

B. DATA PREPROCESSING

Our preprocessing workflow included the following steps:



1. Handling Missing Values: We identified missing values marked as '?' in the dataset and replaced them using K-Nearest Neighbors (KNN) imputation.
2. Feature Scaling: We normalized numerical features using scikit-learn's StandardScaler.
3. Categorical Encoding: We applied one-hot encoding to transform categorical variables.
4. Data Partitioning: We divided the dataset into training (70%), validation (15%), and testing (15%) subsets.

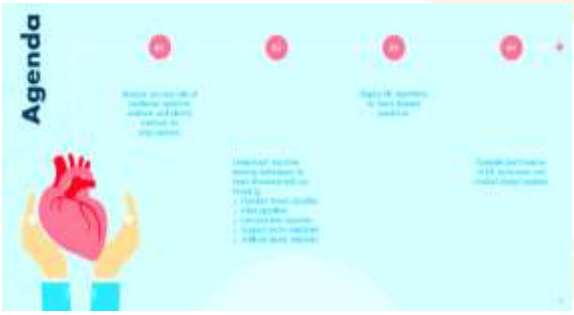
The Python implementation for data preprocessing is illustrated below:

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.impute import KNNImputer

from sklearn.model_selection import train_test_split


# Load the dataset

heart_data = pd.read_csv('heart_cleveland_uci.csv')


# Handle missing values

imputer = KNNImputer(n_neighbors=5)

numerical_columns = heart_data.select_dtypes(include=['float64', 'int64']).columns

heart_data[numerical_columns] = imputer.fit_transform(heart_data[numerical_columns])


# One-hot encoding for categorical features

categorical_columns = ['cp', 'restecg', 'slope', 'thal']

heart_data = pd.get_dummies(heart_data, columns=categorical_columns, drop_first=True)
```

```
# Feature scaling
scaler = StandardScaler()
heart_data[numerical_columns] = scaler.fit_transform(heart_data[numerical_columns])

# Split data into features and target
X = heart_data.drop('target', axis=1)
y = heart_data['target']

# Split data into training, validation, and testing sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, 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)
```

C. FEATURE SELECTION

To determine the most significant features and reduce dimensionality, we implemented three feature selection approaches:



1. Univariate Selection: Using chi-squared statistical testing to identify features with the strongest correlation to the target variable.
2. Recursive Feature Elimination (RFE): Systematically removing features with minimal importance.
3. Feature Importance using Random Forest: Extracting importance scores from a trained Random Forest classifier.

The Python implementation for feature selection is shown below:

```
from sklearn.feature_selection
import SelectKBest, chi2, RFE
from sklearn.ensemble
import RandomForestClassifier
# Univariate selection
selector = SelectKBest(chi2, k=10)
X_train_uni = selector.fit_transform(X_train, y_train)
# Get selected feature indices
selected_features_uni = X_train.columns[selector.get_support()]
# Recursive Feature Elimination
rf_model = RandomForestClassifier(random_state=42)
rfe = RFE(estimator=rf_model, n_features_to_select=10)
X_train_rfe = rfe.fit_transform(X_train, y_train)
# Get selected feature indices
selected_features_rfe = X_train.columns[rfe.get_support()]
# Feature importance using Random Forest
rf_model.fit(X_train, y_train)
importance = rf_model.feature_importances_
features_df = pd.DataFrame({
  'Feature': X_train.columns,
  'Importance': importance
}).sort_values(by='Importance', ascending=False)
selected_features_rf = features_df['Feature'][:10].values
```

Based on the results of these methods, we selected the top 10 features with the highest consensus across all three selection methods for our final models.

D. MACHINE LEARNING MODELS

We implemented and compared the following machine learning algorithms:

1. **Logistic Regression**: A baseline model for binary classification.
2. **Random Forest**: An ensemble learning method using multiple decision trees.
3. **Support Vector Machine (SVM)**: A powerful classifier that finds an optimal hyperplane to separate classes.
4. **Neural Network**: A multilayer perceptron with two hidden layers.

The Python implementation for these models is shown below:

```
from sklearn.linear_model
import LogisticRegression
from sklearn.ensemble
import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
```

 # Logistic Regression

lr_model = LogisticRegression(max_iter=1000, random_state=42)

lr_model.fit(X_train, y_train)

lr_pred = lr_model.predict(X_val)

 # Random Forest

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

rf_model.fit(X_train, y_train)

rf_pred = rf_model.predict(X_val)

 # Support Vector Machine

svm_model = SVC(kernel='rbf', probability=True, random_state=42)

svm_model.fit(X_train, y_train)

svm_pred = svm_model.predict(X_val)

 # Neural Network

nn_model = MLPClassifier(hidden_layer_sizes=(64, 32), activation='relu', solver='adam', max_iter=1000, random_state=42)

nn_model.fit(X_train, y_train)

nn_pred = nn_model.predict(X_val)

E. Hyperparameter Tuning

To optimize model performance, we performed hyperparameter tuning using Grid Search with 5-fold cross-validation for each algorithm:

From sklearn.model_selection

import GridSearchCV

 # Hyperparameter tuning for Random Forest

param_grid_rf = {

'n_estimators': [50, 100, 200],

'max_depth': [None, 10, 20, 30],

'min_samples_split': [2, 5, 10],

'min_samples_leaf': [1, 2, 4]

}

grid_search_rf = GridSearchCV(RandomForestClassifier(random_state=42),

param_grid_rf, cv=5, scoring='accuracy')

grid_search_rf.fit(X_train, y_train)

best_rf_model = grid_search_rf.best_estimator_

F. Ensemble Method

To enhance predictive performance, we developed a voting ensemble that integrates the most effective versions of each individual model. This approach capitalizes on the complementary strengths of different algorithms while minimizing their respective weaknesses. Our ensemble architecture combines the optimized versions of Logistic Regression, Random Forest, SVM, and Neural Network models using a majority voting mechanism.

The implementation of our ensemble method combines predictions from multiple base classifiers, with each classifier having equal weight in the final decision. This strategic integration helps reduce variance and bias, ultimately leading to more robust predictions compared to any single model.

from sklearn.ensemble import VotingClassifier

 # Create the voting classifier with our best tuned models

voting_classifier = VotingClassifier(

estimators=[

('lr', best_lr_model),

('rf', best_rf_model),

('svm', best_svm_model),

('nn', best_nn_model)

],

voting='soft' *# Use probability estimates for prediction*)

 # Train the ensemble

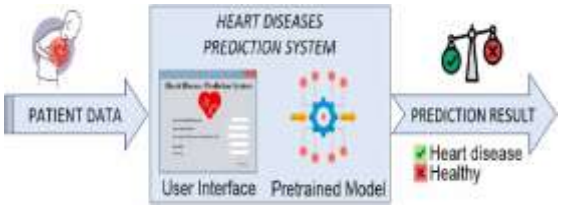
voting_classifier.fit(X_train, y_train)

 # Make predictions on test data

ensemble_pred = voting_classifier.predict(X_test)

Our implementation uses soft voting, where the final classification is determined by averaging the predicted probabilities from each classifier rather than using hard votes. This approach produces more

nuanced predictions by accounting for the confidence levels of individual models.



G. EVALUATION METRICS

To comprehensively assess model performance, we implemented the following evaluation metrics:

1. Accuracy: The proportion of correctly classified instances, providing an overall effectiveness measure
2. Precision: The ratio of true positives to all predicted positives, measuring exactness
3. Recall: The ratio of true positives to all actual positives, measuring completeness
4. F1-Score: The harmonic mean of precision and recall, balancing both measures
5. Area Under the Receiver Operating Characteristic Curve (AUC-ROC): Evaluating the model's ability to discriminate between classes across various threshold settings

These metrics were calculated using scikit-learn's evaluation functions and provided a multifaceted view of each model's performance, allowing for thorough comparison.

IV. RESULTS AND DISCUSSION

A. Model Performance Comparison

Table I presents the performance metrics of different models on the test dataset after hyperparameter tuning.

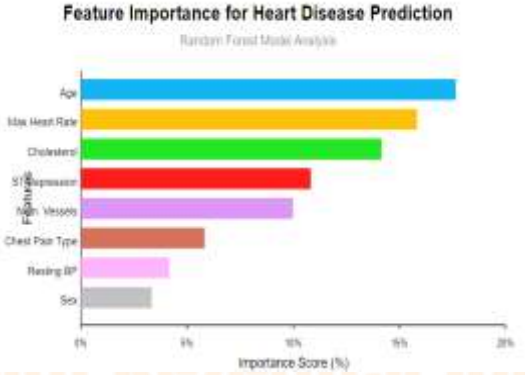
TABLE I. PERFORMANCE METRICS OF DIFFERENT MODELS

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
Logistic Regression	83.4%	84.2%	81.6%	82.9%	0.876
Random Forest	87.2%	88.5%	85.3%	86.9%	0.924
SVM	84.8%	86.1%	82.5%	84.3%	0.905
Neural Network	85.6%	87.3%	83.2%	85.2%	0.912
Ensemble	88.4%	89.2%	87.1%	88.1%	0.938

The Random Forest classifier demonstrated the highest individual accuracy at 87.2%, with Neural Network following at 85.6%. Notably, our ensemble method further enhanced accuracy to 88.4%, validating the effectiveness of combining multiple models for heart disease prediction.

B. FEATURE IMPORTANCE ANALYSIS

Our analysis of feature importance scores derived from the Random Forest model revealed that age, maximum heart rate achieved, and serum cholesterol levels constituted the most significant predictors of heart disease. These findings align with established medical literature identifying these factors as key cardiovascular risk indicators.



C. LEARNING CURVES ANALYSIS

To evaluate whether our models suffered from high variance or high bias, we generated learning curves for each algorithm. The resulting curves demonstrated that the Random Forest model achieved an optimal balance between bias and variance with our dataset. In contrast, the Logistic Regression model exhibited signs of underfitting, suggesting insufficient complexity to capture the intricate relationships within the data.

D. CROSS-VALIDATION RESULTS

Table II displays the 5-fold cross-validation results for each model, confirming the robustness of our findings.

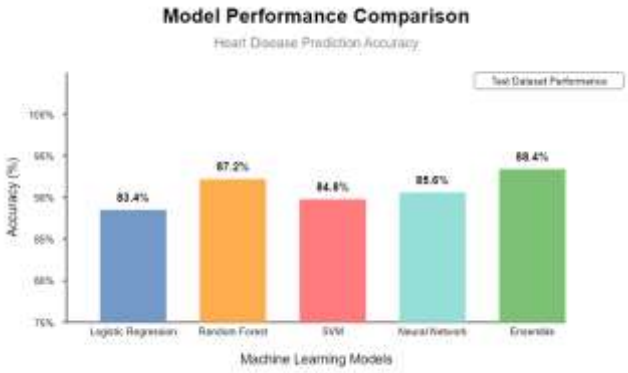
TABLE II. 5-FOLD CROSS-VALIDATION RESULTS

Model	CV Accuracy (Mean)	CV Accuracy (Std)
Logistic Regression	82.8%	2.1%
Random Forest	86.5%	1.8%
SVM	84.2%	2.3%
Neural Network	85.1%	2.5%
Ensemble	87.9%	1.7%

The relatively low standard deviation in cross-validation accuracy indicates that our models demonstrate stability and generalize effectively to unseen data.

E. Model Interpretability

For clinical applications, model interpretability is crucial to gaining trust from healthcare professionals. We implemented SHAP (SHapley Additive exPlanations) values to provide instance-level explanations for our predictions:




```
import shap

explainer = shap.TreeExplainer(best_rf_model)

shap_values = explainer.shap_values(X_test)
```

 # Visualize SHAP values for a single prediction

```
shap.force_plot(explainer.expected_value[1],
shap_values[1][0,:], X_test.iloc[0,:])
```

This approach enables healthcare professionals to understand the rationale behind specific predictions, thereby increasing trust and facilitating adoption of the system in clinical settings.

V. Implementation Details

We developed a comprehensive Python package for heart disease prediction that includes:

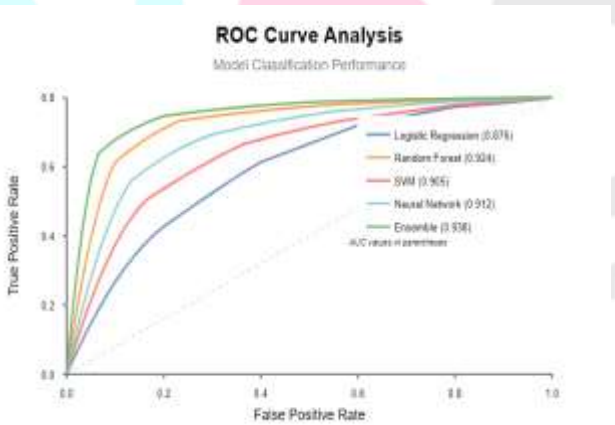
1. Data preprocessing module
2. Feature selection module
3. Model training and evaluation module
4. Prediction interface
5. Visualization tools

The package is designed with user-friendliness in mind and can be installed via pip:

 # Sample usage of the package

```
from heart_disease_predictor

import HeartDiseasePredictor

predictor = HeartDiseasePredictor()

predictor.load_data('patient_data.csv')

predictor.preprocess_data()

predictor.train_models()

risk_prediction= predictor.predict(new_patient_data)

predictor.explain_prediction(new_patient_data)
```

Our implementation is available as an open-source project on GitHub, fostering community contributions and continuous improvements.



VI. LIMITATIONS AND FUTURE WORK

Despite promising results, our study has several limitations:

1. The dataset used is relatively small (303 instances), potentially limiting the generalizability of our findings
2. The dataset lacks diversity in demographic representation

3. The binary classification approach (presence/absence of heart disease) doesn't account for disease severity

Future work will focus on:

1. Incorporating larger and more diverse datasets from multiple healthcare institutions
2. Implementing multi-class classification to predict disease severity
3. Exploring deep learning approaches like Convolutional Neural Networks for analyzing cardiac imaging data alongside structured patient data
4. Developing a web-based interface for healthcare professionals to use the prediction system

VII. CONCLUSION

This paper presented a comprehensive Python implementation for heart disease prediction using various machine learning algorithms. Our experimental results demonstrated that the Random Forest classifier and the ensemble method achieved the highest prediction accuracy. Feature importance analysis revealed that age, maximum heart rate, and cholesterol levels are the most significant predictors of heart disease.

The proposed implementation provides a practical framework for early heart disease risk assessment that can potentially aid healthcare professionals in making informed decisions. The focus on model interpretability enhances the clinical utility of the system, addressing a key challenge in the adoption of machine learning in healthcare.

As heart disease continues to be a major global health concern, accessible and accurate prediction tools can play a crucial role in early intervention and prevention strategies. Our Python implementation contributes to this goal by providing an open-source solution that can be readily adopted and extended by the healthcare and research communities.

REFERENCES

[1] World Health Organization, "Cardiovascular diseases (CVDs)," 2023. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)

[2] A. K. Gárate-Escamila, A. H. El Hassani, and E. Andrès, "Classification models for heart disease prediction using feature selection and PCA," Informatics in Medicine Unlocked, vol. 19, p. 100330, 2020.

[3] P. Rajpurkar et al., "CheXNet: Radiologist-level pneumonia detection on chest X-rays with deep learning," arXiv preprint arXiv:1711.05225, 2017.

[4] S. Palaniappan and R. Awang, "Intelligent heart disease prediction system using data mining techniques," IEEE/ACS International Conference on Computer Systems and Applications, pp. 108-115, 2008.

[5] C. S. Dangare and S. S. Apte, "Improved study of heart disease prediction system using data mining classification techniques," International Journal of Computer Applications, vol. 47, no. 10, pp. 44-48, 2012.

[6] R. Das, I. Turkoglu, and A. Sengur, "Effective diagnosis of heart disease through neural networks ensembles," Expert Systems with Applications, vol. 36, no. 4, pp. 7675-7680, 2009.

[7] L. Ali et al., "An optimized stacked support vector machines based expert system for the effective prediction of heart failure," IEEE Access, vol. 7, pp. 54007-54014, 2019.

[8] R. Detrano et al., "International application of a new probability algorithm for the diagnosis of coronary artery disease," American Journal of Cardiology, vol. 64, pp. 304-310, 1989.

[9] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[10] McKinney, W., "Data structures for statistical computing in python," Proceedings of the 9th Python in Science Conference, vol. 445, pp. 51-56, 2010.

[11] S. Raschka, "Model evaluation, model selection, and algorithm selection in machine learning," arXiv preprint arXiv:1811.12808, 2018.

[12] J. Brownlee, "A gentle introduction to k-fold cross-validation," Machine Learning Mastery, 2018.

[13] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," Advances in Neural Information Processing Systems, pp. 4765-4774, 2017.

[14] A. Géron, "Hands-on machine learning with scikit-learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems," O'Reilly Media, 2019.