

Multiple Disease Detection System Using Machine Learning

Ahaladitha Thamada

BTech, Department of Computer Science and Engineering

Abstract

This study presents a Multiple Disease Prediction System that forecasts diabetes, heart disease, and disease (PD) using machine learning algorithms. The system analyzes complex medical datasets to identify patterns and risk factors linked to these diseases. For heart disease, logistic regression is used, focusing on cardiovascular data and providing a probabilistic assessment of heart health. Random forest and gradient boosting are applied for PD prediction and kidney disease prediction capturing neurological changes, while Support Vector Machines (SVM) predict diabetes using metabolic and genetic markers. The system allows users to identify health issues based on symptoms and vital signs, improving accessibility for early disease detection. By integrating multiple disease predictions into a single interface, the system increases predictive accuracy and power. It compares the performance of algorithms such as Random Forest, Decision Tree, and K-Nearest Neighbor to identify the most accurate model for each condition. The system aims to create a web application for efficient disease forecasting, ultimately saving lives through early detection and diagnosis. This approach could revolutionize chronic disease detection, making healthcare more proactive and efficient.

Keywords: multiple disease prediction, diabetes prediction, heart disease prediction, Kidney Disease, Parkinson's disease (pd) prediction, machine learning algorithms.

1. INTRODUCTION

One of the most important aspects of human life is the ability to predict diseases and one of the most important aspects of treating a disorder is predicting an individual's health early on. Thus, innovation that improves logistics is essential to the healthcare industry. One way to innovate in the medical field in this digital age is to digitalize medical procedures.

Traditional diagnostic methods often involve complex, time consuming, and expensive procedures, which may not be readily available or accessible to all patients, particularly in resource-limited settings. The cost of consultations is prohibitive and the demand on doctors' time is excessive, which are two of the most important issues facing the healthcare business. This problem is mostly highlighted by using patient symptoms as input for disease prediction.

Machine learning-based multiple illness prediction holds the potential to transform healthcare by facilitating more precise and individualised diagnosis, timely interventions, and more efficacious therapies. It involves the use of large datasets for each of the disease consisting of various parameters that will be analysed for the accurate prediction of the disease.

1.1. Problem Definition

This project aims to develop a Multiple Disease Detection System that leverages machine learning

algorithms to accurately predict the likelihood of individuals developing heart disease, diabetes, Parkinson's disease and kidney disease. The system will use easily accessible patient data such as medical history, symptoms, and vital signs, combined with advanced machine learning techniques, to detect potential risks of these diseases early. By providing a tool that can analyze complex medical datasets, identify patterns, and predict disease outcomes, this system aims to enhance the speed, accuracy, and accessibility of disease detection.

The solution will be validated by comparing the performance of several machine learning models (e.g., Random Forest, Decision Tree, K-Nearest Neighbor, SVM, Logistic Regression) to determine the most effective algorithm for each disease. The goal is to create a user-friendly system capable of forecasting the presence of these diseases based on available patient data, offering a cost-effective and non-invasive method for early detection and intervention.

1.2. Existing System

The existing systems focus on utilizing non-invasive methods to measure core health parameters such as BMI, blood pressure, glucose levels, SpO₂, temperature, and pulse rate. These parameters are processed using machine learning algorithms like Decision Trees, Naive Bayes, and K-Nearest Neighbors (KNN) to predict diseases, with an emphasis on low-cost, portable, and real-time diagnostics. The glucose levels, and BMI are analysed for the prediction of diabetes. Logistic regression is used which evaluates cardiovascular data and provides a probabilistic assessment of heart health, offering insights into the likelihood of heart disease development. Parkinson's disease detection systems leverage both traditional clinical approaches and advanced computational techniques to identify the disease in its early stages. Kidney disease detection utilises clinical data by deploying a deep learning model. ML algorithms such as Support Vector Machines (SVM), Random Forests, and XGBoost process features from various datasets to classify individuals as healthy or affected.

1.3. Drawbacks in the Existing Systems

- **Limited Features:** Focusing only on core parameters can overlook critical diagnostic factors, reducing accuracy for complex diseases.
- **Handling Rare Diseases:** Limited data availability for rare diseases hampers the effectiveness of predictions in such cases.
- **Overfitting Issues:** Machine learning models often overfit small or specific datasets, resulting in poor generalization to new data.
- **Lack of Explainability:** Many models, especially advanced ones, act as "black boxes," providing little insight into how predictions are made.
- **Hardware and Integration Challenges:** Dependence on sensors and hardware introduces accuracy and reliability issues, while managing multiple diseases in one framework increases complexity.
- **Cost and Accessibility:** Advanced systems with sophisticated models or imaging data can be costly and require skilled personnel, limiting their use in resource-constrained settings.
- **Ethical and Privacy Concerns:** Risks of data breaches, lack of fairness in predictions, and limited accountability of algorithms pose ethical challenges.

1.4. Proposed Model

The proposed system detects heart diseases, diabetes, Parkinson's disease and kidney disease through a unified machine learning framework. It begins with collecting diverse datasets, including clinical data, patient demographics, and biomarkers specific to each condition. For heart diseases, features like cholesterol levels, blood pressure, and ECG readings are prioritized. Diabetes detection focuses on glucose

levels, BMI, and insulin sensitivity, while Parkinson’s analysis uses motor function data, voice recordings, and neurological biomarkers. Kidney disease detection uses blood cell count, albumin levels, blood pressure, blood urea levels along with various other features.

Next, key attributes are extracted to identify patterns distinguishing healthy individuals from those with these diseases. Machine learning algorithms such as logistic regression, SVM, Gradient Boosting and Random Forest are applied to build predictive models.

The models undergo validation using cross-validation and independent testing to ensure accuracy across populations. Regularization methods minimize overfitting. Once validated, models are deployed in healthcare environments with a user-friendly interface for real-time predictions. The system supports multi-disease assessment, allowing clinicians to evaluate risks for multiple conditions simultaneously.

Continuous updates with new data improve accuracy and generalizability over time. This approach ensures early detection, personalized care, and scalability for diverse healthcare needs.

1.5. Hardware Specifications:

- Processor (Dual-core 2 GHz or higher)
- Memory (8 GB or higher)
- Storage (20 GB or higher)
- OS

1.6. Software Specifications:

- Python 3.7 or higher programming language
- Required Libraries (numpy, pandas, scikit-learn, matplotlib, seaborn, pickle)
- Environment Setup (package manager, IDE)
- Dataset

2. DESIGN METHODOLOGY

2.1. System Architecture

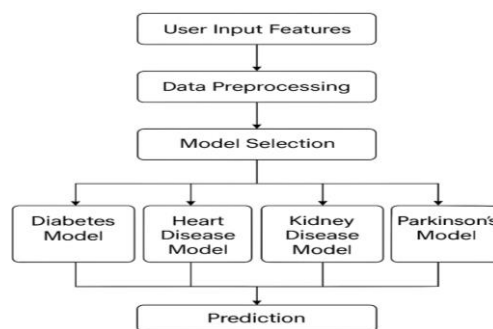


Fig1: System architecture of the Multiple Disease Detection System

As shown in Fig1, the architecture of the multiple-disease detection system involves loading the dataset to train the models and corresponding scalers for each disease type. Input data is collected from the user and preprocessed according to the specific requirements of each model—this includes organizing the features in the required order and applying standardization where necessary to maintain consistency with the model’s training conditions.

Each model is trained on disease-specific datasets using a range of machine learning algorithms such as Random Forest, Logistic Regression, Support Vector Machines (SVM), and K-Nearest Neighbors (KNN).

These trained models are optimized using techniques like 5-fold cross-validation and grid search for hyperparameter tuning. Evaluation metrics such as accuracy are used to identify the best-performing model for each disease. Upon deployment, the system loads the appropriate model based on the user's selection and outputs a prediction—either 'Healthy' or a diagnosis of the respective disease—offering a fast and reliable decision support mechanism. Referenced from [5][6][9][10].

In addition to the prediction, the system incorporates SHAP (SHapley Additive exPlanations) to interpret the model's predictions. SHAP provides a detailed breakdown of how each feature contributes to the prediction, improving the model's transparency and trustworthiness. Using a waterfall plot, the app visually explains which features pushed the prediction toward or away from the diagnosis. This enhances user understanding and supports healthcare professionals in making more informed decisions, by highlighting the most influential health indicators for each individual prediction.

2.2. Model Methodology

1. Data Pre-processing:

- Each disease-specific dataset (e.g., diabetes, heart, kidney, Parkinson's) is loaded from CSV format.
- Exploratory Data Analysis (EDA) is performed to examine feature distributions, identify missing values, and understand data types.
- Irrelevant or highly correlated features are optionally removed, and the target variable (disease outcome) is defined.
- The dataset is then split into input features (X) and target labels (y), followed by an 80-20 train-test split for evaluation.

2. Standardization:

- Numerical input features are standardized using techniques like Z-score normalization also known as standardization to bring them onto a uniform scale.
- Features are standardized to ensure uniform scaling, which improves model performance, especially for distance-based and gradient-sensitive algorithms such as SVM and KNN.

3. Model Training and Evaluation:

- Multiple classifiers are trained for each disease, including Random Forest, Gradient Boosting, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN).
- Each model undergoes hyperparameter optimization with 5-fold cross-validation to find the best configuration.
- Training and testing accuracy for each model is recorded.

4. Prediction:

- A sample user input is collected (e.g., from a Streamlit app), transformed using the same preprocessing pipeline.
- The input is then fed into the best-trained model to generate a prediction, either 'Healthy' or at risk for the specific disease.

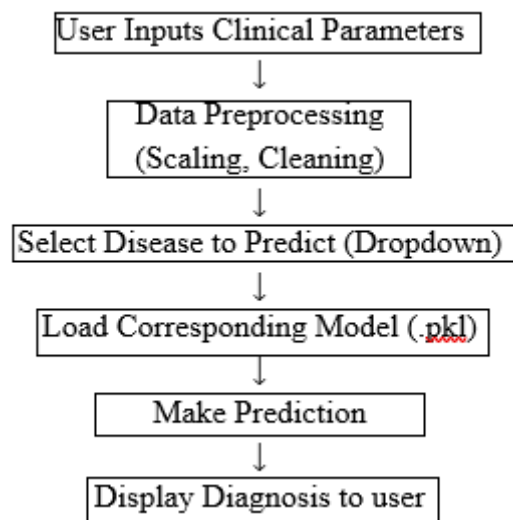
5. Model Persistence:

- The best-performing model for each disease is serialized and saved using joblib for obtaining the model in the “.pkl” format.
- These saved models are later loaded in the deployment phase for fast and consistent predictions without retraining.

6. Model Explainability:

- SHAP (SHapley Additive exPlanations) is integrated to explain individual predictions.
- SHAP visualizations, such as waterfall plots, highlight the influence of each feature on the prediction outcome, improving transparency and trust in the model's decision-making.

2.2. Model Design



Flowchart 1: The design and flow of the Multiple Disease Detection System

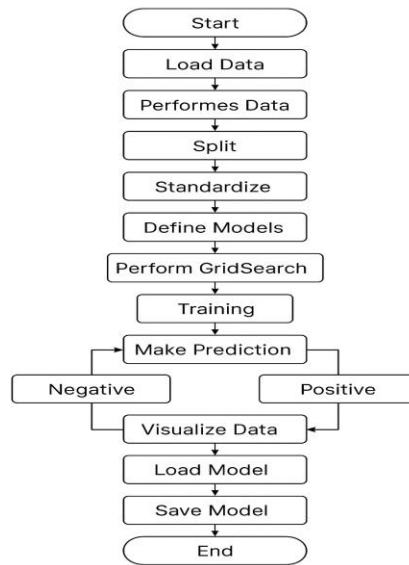
The model design as shown in the Flowchart 1 for the multi-disease prediction system begins with collecting clinical data from the user through an interactive interface. Depending on the disease to be predicted, the user is prompted to enter relevant health parameters.

Once the input is submitted, the system performs data preprocessing to prepare it for model prediction. This preprocessing includes cleaning steps, such as handling missing or invalid values, and feature scaling to bring all input values to a uniform range. Ensuring consistent preprocessing guarantees that the input aligns with the format the model was trained on.

After preprocessing, the user selects the disease they want to predict from a dropdown menu—options include diabetes, heart disease, kidney disease, or Parkinson's disease. Based on the selection, the system dynamically loads the appropriate pre-trained machine learning model stored as a .pkl file.

With the relevant model loaded, the system passes the preprocessed input into the model to generate a prediction. The output is typically a binary classification, where a prediction of 1 indicates a positive diagnosis (e.g., the user is likely at risk), and 0 indicates a negative diagnosis (e.g., the user is not at risk). This result is then displayed to the user in a simple and interpretable format such as "Positive for Diabetes" or "Negative for Heart Disease" along with the SHAP analysis.

3. IMPLEMENTATION



Flowchart 2: Implementation of the Multiple Disease Detection System

Flowchart 2 shows implementation of the disease detection system which involves a combination of data science pipelines and an interactive user interface built using Streamlit.

The process begins by importing necessary libraries and loading pre-trained machine learning models saved as .pkl files for each specific disease: diabetes, heart disease, kidney disease, and Parkinson’s. For each disease module, the system collects user input parameters through sliders and number fields in the web interface, ensuring the data is entered in a structured format.

Behind the scenes, this input is converted into a NumPy array or DataFrame and standardized using a pre-fitted scaler if the model requires it. The data is then split into training and testing sets, typically in an 80-20 ratio, where the training set is used to teach the model and the test set is reserved for later evaluation. The model then predicts the likelihood of disease presence, displaying an easy-to-understand diagnosis. To enhance interpretability, SHAP explainability is integrated, allowing users to see how each feature impacted the prediction visually via a SHAP waterfall plot. The entire implementation ensures seamless model loading, prediction generation, and explainability visualization, all while maintaining efficiency through caching and modular script design. This practical deployment allows non-technical users to interact with complex models through a clean and responsive interface.

3.1. Major Implementation Steps

1. Input Layer:

User-provided features like glucose level, age, blood pressure, serum creatinine, voice measures (for Parkinson’s), etc.

2. Preprocessing:

Standardization/Normalization (using scalers like kidney_scaler.pkl, parkinsons_scaler.pkl) and handling missing values if needed.

3. Saving disease specific models:

- diabetes_model.pkl: Model for predicting diabetes.
- heart_model.pkl: Model for predicting heart disease.
- kidney_model.pkl: Model for predicting chronic kidney disease.

- parkinsons_model.pkl: Model for predicting Parkinson's disease.
4. Prediction engine:
Loading corresponding .pkl models using libraries like joblib or pickle. This is followed by preprocessing input and predicting disease probability.
 5. Output layer:
Disease is predicted in an interpretable format as “Disease detected” or “No disease detected” along with risk analysis for heart disease. The output also contains a graphical representation displaying the impact of each feature towards the disease.

3.2. Model Training and Testing

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

Fig2: Splitting the data for training and testing

Training

The training phase is a critical step where each machine learning model learns to identify patterns and relationships between input features and disease outcomes from historical medical data.

This phase begins after the dataset has been preprocessed—missing values are handled, categorical variables (if any) are encoded, and numerical features are standardized for consistency. As shown in the Fig2, training variables “X_train” and “Y_train” use 80% of data for training which are later used throughout the model.

During training, various algorithms such as Random Forest, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Gradient Boosting are applied. These models are not used with default settings—instead, a process called hyperparameter tuning is carried out using techniques like GridSearchCV with 5-fold cross-validation. This ensures that the model not only fits well to the training data but also generalizes to unseen data. The model iteratively adjusts its internal parameters to minimize prediction errors, effectively "learning" from the input-output mappings in the training set.

Throughout this phase, performance metrics such as accuracy is calculated for each model and configuration. After comparison, the best-performing model (the one with the highest validation accuracy and stable performance) is selected. Finally, this trained model is saved as a .pkl file using joblib so that it can be quickly loaded during the deployment phase without retraining.

Testing

The testing phase is where the performance and reliability of our trained machine learning models is evaluated on data they have never seen before. This step ensures that the models generalize well and are capable of making accurate predictions on real-world inputs.

Once the dataset is preprocessed and split during the training phase, 20% of the data is reserved as the test set. This test set includes samples that were not used during training or validation, making it a true representation of how the model will perform in practical scenarios. Each trained model—whether it's for diabetes, heart disease, kidney disease, or Parkinson's—is evaluated separately using its corresponding test data. As shown in the Fig2, testing variables “X_test” and “Y_test” use 20% of data for test which are later used throughout the model.

Before making predictions on the test set, the features are standardized using the same scaler that was fitted on the training data to ensure consistency. The scaled test data is then fed into the best-performing model, previously selected through GridSearchCV and cross-validation during training.

The model generates predictions for each instance in the test set. These predictions are then compared with the actual (true) labels to compute performance metrics, such as accuracy – the overall percentage of correct predictions.

After the testing phase, the results from different models are compared. The model with the most stable and highest performance on the test set is finalized and saved as a .pkl file for deployment.

3.3. Preprocessing Tool Implementation

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
```

Fig3: Implementation of preprocessing tools for data standardization

Standard Scaler

Standard Scaler is a data preprocessing technique provided by the scikit-learn library that standardizes numerical features by removing the mean and scaling to unit variance. This means that after transformation, each feature will have a mean of 0 and a standard deviation of 1.

Standardization is essential for machine learning algorithms that are sensitive to the scale of input features—such as Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Logistic Regression—because features with larger numeric ranges can disproportionately influence the model. By applying Standard Scaler, the model will be able to treat all features equally during training, improving both convergence speed and accuracy.

In my project, Standard Scaler ensures consistent scaling of clinical parameters like blood pressure, glucose level, and cholesterol before making predictions.

As shown in Fig3, Standard Scaler is implemented using: StandardScaler from sklearn.preprocessing

Label Encoder

Label Encoder is a simple yet powerful utility from scikit-learn used to convert categorical labels into numerical form. It is especially useful for encoding target variables, where class labels such as "Yes" and "No" or "Positive" and "Negative" need to be transformed into a format that machine learning models can understand. For instance, "Yes" might become 1 and "No" becomes 0. This transformation is crucial because most machine learning algorithms operate on numerical data and cannot directly process string-based labels.

In my project, Label Encoder will be used when preparing disease labels (e.g., diabetic vs. non-diabetic) during the model training phase, ensuring the output classes are encoded correctly for classification tasks.

As shown in Fig3, Label Encoder is implemented using: LabelEncoder from sklearn.preprocessing

3.4. Algorithm Implementation

```
## Multiple Disease Prediction System loading ML algorithms
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

Fig4: Various algorithm implementation in the disease detection system

Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm widely used for both classification and regression tasks, though it is particularly effective in solving classification problems in real-world scenarios.

In this approach, each data point is represented as a point in an n-dimensional space, where n corresponds to the number of features, and each feature is mapped to a coordinate axis. The core objective of the SVM algorithm is to identify the optimal hyperplane that separates different classes in the dataset. This hyperplane acts as a decision boundary that maximizes the margin, defined as the distance between the closest data points from each class and the hyperplane itself.

Maximizing the margin helps the model achieve better generalization, reducing the risk of overfitting, which is a common issue in other classification techniques. The classification process is powered by the Support Vector Classifier (SVC), a specific implementation of SVM that excels at handling high-dimensional data.

Furthermore, SVM can use kernel functions to transform non-linearly separable data into a higher-dimensional space, where it becomes easier to find a separating hyperplane. Common kernel types include Linear, Polynomial, and Radial Basis Function (RBF) kernels. Once the hyperplane is defined, the model classifies new data points by determining on which side of the hyperplane they lie. Referenced from [1][3][6].

As shown in Fig4, Support Vector Machine algorithm is implemented using: SVC from sklearn.svm

K Nearest Neighbors

In my disease detection project, K-Nearest Neighbors (KNN) is used as one of the classification algorithms during the training phase for certain disease models. KNN is evaluated alongside other classifiers like SVM, Random Forest, and Gradient Boosting.

The implementation process begins by preprocessing and standardizing the dataset using StandardScaler, which is essential for KNN since it relies on distance calculations between data points. The KNeighborsClassifier from Scikit-learn is then used, and hyperparameters such as the number of neighbors (k), weighting method (uniform or distance), and distance metric are optimized. Referenced from [1][3][4][5][7].

As shown in Fig4, the K Nearest Neighbors algorithm is implemented using: KNeighborsClassifier from sklearn.neighbors

Random Forest

Random Forest is an ensemble learning algorithm that builds multiple decision trees on random subsets of data and features. It uses bagging to reduce overfitting and improve accuracy. During prediction, it aggregates the outputs of all trees using majority voting for classification or averaging for regression, making it robust and effective for complex datasets.

In my disease detection project, Random Forest is used during the training phase as one of the candidate algorithms for predicting conditions such as heart disease, kidney disease, or diabetes.

Random Forest is particularly suited to medical datasets because of its robustness to noise, ability to handle both numerical and categorical data, and lower risk of overfitting compared to individual decision trees. Referenced from [1][3][7][9].

As shown in Fig4, the Random Forest Classifier is implemented using: RandomForestClassifier from sklearn.ensemble.

Logistic Regression

Logistic Regression is a supervised classification algorithm that estimates the probability of a data point belonging to a class using the sigmoid function. It calculates the dot product of features and weights, then maps the result to a range between 0 and 1. A threshold (usually 0.5) determines the final class prediction.

In my project, Logistic Regression is used as one of the baseline classifiers during the model training phase. The trained model learns the optimal weights to separate the classes based on the input features. Logistic Regression is especially effective for binary classification tasks like diabetes prediction due to its simplicity, interpretability, and solid baseline performance. Referenced from [1][3][9].

As shown in Fig4, the Logistic Regression algorithm is implemented using: LogisticRegression from sklearn.linear_model

Gradient Boosting

Gradient Boosting is an ensemble learning technique that builds models in a sequential manner, where each new model tries to correct the errors made by the previous one. It works by minimizing a loss function using a gradient descent approach, and it typically uses decision trees as weak learners.

In my multiple-disease detection system, Gradient Boosting is one of the machine learning algorithms evaluated during the model training phase for diseases like heart disease or kidney disease.

When a user enters clinical parameters, the input is preprocessed, passed into the Gradient Boosting model, and a prediction (positive or negative for disease) is generated. The algorithm's ability to handle non-linear relationships, reduce overfitting, and deliver high accuracy makes it a strong choice for structured medical data used in my project. Referenced from [2][4].

As shown in Fig4, the Gradient Boosting algorithm is implemented using: GradientBoostingClassifier from sklearn.ensemble

3.5. Implementing Libraries

```
import numpy as np
import pandas as pd

from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

Fig5: Importing various libraries to be used in the Multiple Disease Detection System

Numpy

NumPy is a library for numerical computations, particularly used in the project for handling arrays and matrices. It provides efficient tools for mathematical operations.

As in the Fig5, it is implemented as: np from numpy

Pandas

Pandas is used for data manipulation and analysis. It provides powerful data structures like DataFrame and Series for handling and processing structured data.

As in Fig5, it is implemented as: pd from pandas

Splitting Data

Splitting datasets into training and testing subsets helps evaluate a machine learning model's performance by training it on one subset and testing it on another.

Commonly, the dataset is divided into X_train, X_test (features) and Y_train, Y_test (labels) with reference to Fig2.

As in Fig5, splitting of data is done with help of: train_test_split from sklearn.model_selection

Evaluation metrics

Evaluation metrics are crucial for understanding the model's performance. The implemented function calculates the accuracy of a model by comparing its predictions with the true labels. Accuracy is the ratio

of correctly predicted samples to the total number of samples.

As in Fig5, it is implemented using: `accuracy_score` from `sklearn.metrics`

3.6. Implementing the best model using GridSearchCV

```
from sklearn.model_selection import GridSearchCV
```

Fig6: Implementing model selection

GridSearchCV automates the process of hyperparameter tuning by exhaustively searching through a grid of specified hyperparameters for a model.

For each model, a dictionary of possible hyperparameters is defined using 'param_grid'. The grid search trains the model with all possible combinations of hyperparameters, evaluates them using cross-validation and selects the best set of parameters based on the evaluation metric which is the accuracy of the model. After completing the search, the best model and its parameters can be accessed. Thus, it ensures the optimal parameters for each model, improving their predictive performance on the given dataset.

As shown in Fig6, it is implemented in the model using: `GridSearchCV` from `sklearn.model_selection`

3.7. Implementing Disease Prediction Models

Diabetes Prediction Model

The prediction aims to determine a patient's risk of getting diabetes at an early stage using various machine learning algorithms based on the data in the dataset obtained from '<https://github.com/shaadclt/Multiple-Disease-Prediction-System/blob/main/diabetes.csv>' file containing 768 rows and 9 columns. It is trained using SVM, Gradient Boosting and Logistic Regression.

Parkinson's Disease Prediction Model

One of the main components of a multiple disease prediction system is the Parkinson disease prediction module. The patient's outcome is generated based on the data in '<https://github.com/shaadclt/Multiple-Disease-Prediction-System/blob/main/parkinsons.csv>' file containing 195 rows and 24 columns. It is trained using many machine algorithms, including Random Forest, Gradient Boosting, SVM and KNN.

Heart Disease Prediction Model

The patient's outcome is generated based on information about the affected and normal people. It executes machine algorithms, including Random Forest Classifier, Gradient Boosting. A vast and well-structured '<https://github.com/shaadclt/Multiple-Disease-Prediction-System/blob/main/heart.csv>' dataset containing 303 rows and 17 columns allows the model to train efficiently and understand the nuances to predict the type of heart condition as well.

Kidney Disease Prediction Model

The patient's outcome is determined based on clinical information collected from both affected and healthy individuals. The model employs machine learning algorithms such as Random Forest Classifier and Gradient Boosting to accurately assess the risk of chronic kidney disease. By leveraging a well-curated '<https://github.com/shaadclt/Multiple-Disease-Prediction-System/blob/main/kidney.csv>' dataset containing 400 rows and 26 columns, the model will be able to learn subtle patterns in renal function indicators, allowing it to make reliable predictions about the presence or absence of kidney disease.

3.8. Attribute Information

Diabetes Disease

1. Pregnancies
2. Glucose
3. Blood pressure
4. Skin Thickness
5. Insulin
6. BMI
7. Diabetes Pedigree Function
8. Age
9. Outcome (target) – Class variable (0 = No diabetes, 1 = Diabetes)

Parkinson's Disease

1. MDVP: Fo(Hz) – Average vocal fundamental frequency
2. MDVP: Fhi(Hz) – Maximum vocal fundamental frequency
3. MDVP: Flo(Hz) – Minimum vocal fundamental frequency
4. MDVP: Jitter(%) – Variation in pitch
5. MDVP: Jitter(Abs) – Absolute variation in pitch
6. MDVP: RAP – Relative average perturbation
7. MDVP: PPQ – Five-point period perturbation quotient
8. Jitter: DDP – Average absolute difference of differences between cycles
9. MDVP: Shimmer – Variation in amplitude
10. MDVP: Shimmer(dB) – Shimmer in decibels
11. Shimmer: APQ3 – Amplitude perturbation quotient (3-cycle average)
12. Shimmer: APQ5 – Amplitude perturbation quotient (5-cycle average)
13. MDVP: APQ – Amplitude perturbation quotient
14. Shimmer: DDA – Average absolute difference of amplitudes of consecutive periods
15. NHR – Noise-to-harmonics ratio
16. HNR – Harmonics-to-noise ratio
17. RPDE – Recurrence period density entropy (a nonlinear measure)
18. DFA – Detrended fluctuation analysis
19. Spread1 – A nonlinear measure of variation
20. Spread2 – Another nonlinear variation measure
21. D2 – Correlation dimension
22. PPE – Pitch period entropy

Heart Disease

1. Age – Age of the patient (in years)
2. Sex – Gender (1 = male, 0 = female)
3. Chest Pain Type (cp)
 - 0: Typical angina
 - 1: Atypical angina
 - 2: Non-anginal pain
 - 3: Asymptomatic
4. Resting Blood Pressure (trestbps) – In mm Hg

5. Serum Cholesterol (chol) – In mg/dL
6. Fasting Blood Sugar (fbs) – (1 = >120 mg/dL, 0 = ≤120 mg/dL)
7. Resting ECG Results (restecg)
 - 0: Normal
 - 1: ST-T wave abnormality
 - 2: Probable/definite left ventricular hypertrophy
8. Maximum Heart Rate Achieved (thalach)
9. Exercise Induced Angina (exang) – (1 = yes, 0 = no)
10. ST Depression (oldpeak) – Induced by exercise relative to rest
11. Slope of Peak Exercise ST Segment (slope)
 - 0: Upsloping
 - 1: Flat
 - 2: Downsloping
12. Number of Major Vessels Colored by Fluoroscopy (ca) – (0–3)
13. Thalassemia (thal)
 - 0: Normal
 - 1: Fixed defect
 - 2: Reversible defect
14. Target (output) – Diagnosis of heart disease (1 = disease present, 0 = no disease)

Kidney Disease

1. Age – Age of the patient (in years)
2. Blood Pressure (bp) – Measured in mm/Hg
3. Specific Gravity (sg) – Indicator of urine concentration
4. Albumin (al) – Protein level in urine
5. Sugar (su) – Sugar level in urine
6. Red Blood Cells (rbc) – Normal or abnormal
7. Pus Cell (pc) – Normal or abnormal
8. Pus Cell clumps (pcc) – Present or not
9. Bacteria (ba) – Presence of bacteria in urine
10. Blood Glucose Random (bgr) – Random blood glucose level in mg/dL
11. Blood Urea (bu) – Urea concentration in mg/dL
12. Serum Creatinine (sc) – Creatinine level in mg/dL
13. Sodium (sod) – Sodium concentration in mEq/L
14. Potassium (pot) – Potassium concentration in mEq/L
15. Hemoglobin (hemo) – Hemoglobin level in g/dL
16. Packed Cell Volume (pcv) – Packed cell volume (%)
17. White Blood Cell Count (wc) – WBC count (cells per mm³)
18. Red Blood Cell Count (rc) – RBC count (millions/cubic mm)
19. Hypertension (htn) – (1= yes, 0= no)
20. Diabetes Mellitus (dm) – (1= yes, 0= no)
21. Coronary Artery Disease (cad) – (1= yes, 0= no)
22. Appetite (appet) – (1= yes, 0= no)

23. Pedal Edema (pe) – Presence of swelling in feet (1= yes, 0= no)
24. Anemia (ane) – Presence of anemia (1= yes, 0= no)
25. Classification (target) – Chronic Kidney Disease (ckd) or not (notckd)

3.9. UI Implementation

The user interface (UI) of my multiple disease detection system is developed using Streamlit, an open-source, Python-based web application framework tailored for data science and machine learning workflows. Streamlit allows for the rapid creation of interactive dashboards and the deployment of machine learning models with minimal code, making it ideal for real-time applications like this one. Its ease of use and seamless integration with the Python ecosystem enable data scientists to build elegant, functional web apps without the need for extensive frontend development experience.

Once the user inputs relevant clinical data, the system processes it in real time, arranges the inputs into a structured format compatible with the machine learning model, and performs standardization if required using a pre-loaded scaler. The model previously trained and saved as a .pkl file is then loaded using joblib and used to generate predictions. The result is displayed to the user using intuitive feedback like `st.success()` or `st.warning()`, which may indicate messages such as “No Disease Detected” or “Disease Detected”.

To improve transparency and model interpretability, each disease module integrates SHAP (SHapley Additive Explanations) visualizations using Matplotlib. These plots explain the influence of each input feature on the model’s prediction, making it easier for users and healthcare professionals to understand why a particular result was returned. SHAP plots are embedded directly within the Streamlit interface to provide visual feedback in an accessible format.

Streamlit simplifies ML app development by eliminating the need for HTML, CSS, or JavaScript. It allows developers to build full-featured data science apps in hours. Supporting libraries like Pandas, Plotly, and PyTorch, it also offers data caching to streamline computation and boost performance.

Streamlit serves as the backbone of my UI, enabling a clean, responsive, and user-friendly environment where patients or medical professionals can interact with advanced machine learning models. It provides fast, accurate, and explainable predictions by deploying machine learning solutions in healthcare.

4. RESULTS

4.1. Executing in PowerShell

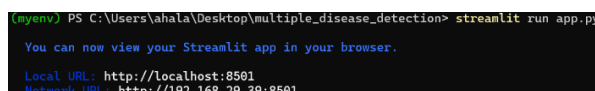


```
(myenv) PS C:\Users\ahala\Desktop\multiple_disease_detection> streamlit run app.py
```

Fig7 : Executing the application deployment command

As shown in Fig7, the “streamlit run app.py” command should be executed in the respective project directory in the PowerShell for Windows.

After successfully executing this command, as shown in Fig8, the PowerShell prompts two URLs, a network URL and a local URL. The local URL will be used to deploy the project in the local environment.



```
(myenv) PS C:\Users\ahala\Desktop\multiple_disease_detection> streamlit run app.py
You can now view your Streamlit app in your browser.
Local URL: http://localhost:8501
Network URL: http://192.168.29.39:8501
```

Fig8: PowerShell prompting project deployment URLs

4.2. Predictive System

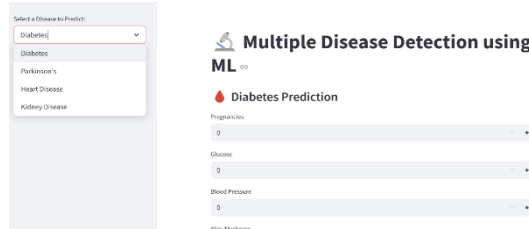


Fig9: UI of the multiple disease detection system displaying the drop down menu and dynamic input fields for the selected disease

The UI page of the multiple disease detection system as shown in Fig9, built with Streamlit, is designed to be clean, minimal, and intuitive perfectly suited for users with or without technical backgrounds. When the app loads, users are greeted with a centered title like “Multiple Disease Detection using ML,” and a sidebar on the left that acts as the main navigation panel.

In the sidebar, users can choose the disease they want to predict Diabetes, Heart Disease, Kidney Disease, or Parkinson's from a simple dropdown menu. Upon selection, the app dynamically loads the corresponding prediction module, without needing to reload or restart the entire application.

On the main page, the selected disease module displays interactive input fields, allowing users to enter medical metrics such as blood pressure, glucose, age, BMI, etc. These inputs are arranged vertically for clarity and simplicity. Below the inputs, a "Predict" button triggers the machine learning model, which then processes the data in real-time.

Once a prediction is made, the result is shown instantly using visual alerts for example:

- No Disease Detected
- Disease Detected

To enhance transparency, each module also includes a SHAP-based explanation, rendered with Matplotlib. These visualizations show how each input contributed to the final decision, using colored bars that make it easy to interpret the model’s reasoning. The UI is responsive and runs entirely in the browser, meaning users don’t need to install anything.

Diabetes Predictive System

As shown in Fig10, the disease “Diabetes” can be selected from the drop down menu. After selection, the user is prompted with a set of input fields to enter the various values useful for the detection of the disease. After providing the values, the user can click on the “Predict” button at the bottom to obtain the diagnosis of the disease as shown in Fig11.

SHAP analysis as shown in Fig12 which is an interactive plot is provided to the user, to provide the detailed understanding of which factors are contributing towards the presence of the disease and which are not.

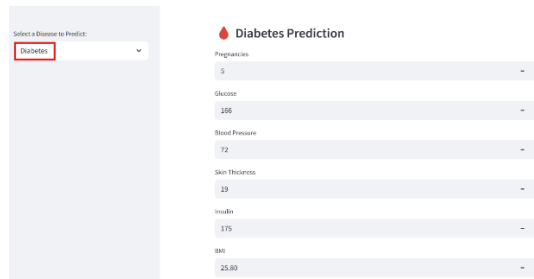


Fig10: Diabetes is selected from the menu and values are inserted on the right



Fig11: Upon clicking the predict button, the diagnosis is obtained

- Features in pink pushed the prediction toward diabetes (1).
- Features in blue pushed the prediction away from diabetes (0).
- The length of each bar shows how much that feature influenced the model's decision.

This helps you understand why the model gave this result.

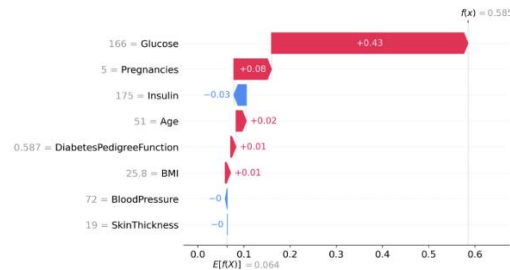


Fig12: SHAP analysis for the predicted Diabetes is displayed

Parkinson's Disease Predictive System

As shown in Fig13, the disease “Parkinson’s” can be selected from the drop down menu. After selection, the user is prompted with a set of input fields to enter the various values useful for the detection of the disease.

After providing the values, the user can click on the “Predict” button at the bottom to obtain the diagnosis of the disease as shown in Fig14.

SHAP analysis as shown in Fig15 which is an interactive plot is provided to the user, to provide the detailed understanding of which factors are contributing towards the presence of the disease and which are not.



Fig13: Parkinson’s Disease is selected from the menu and values are inserted on the right

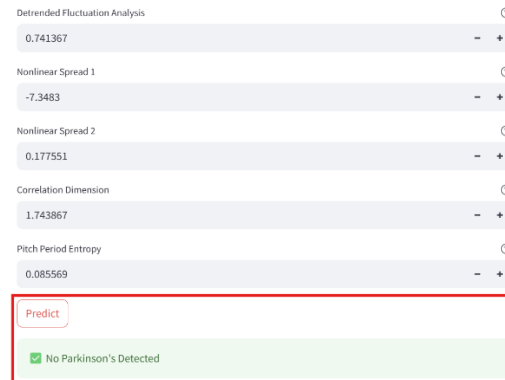


Fig14: Upon clicking the predict button, diagnosis is obtained

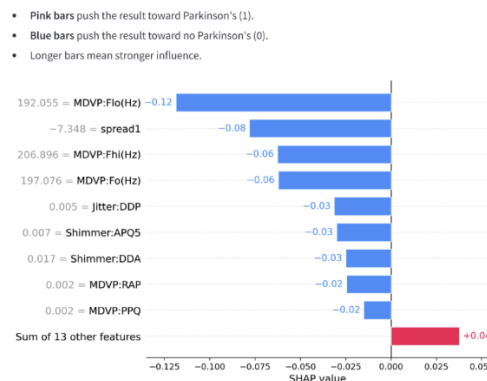


Fig15: SHAP analysis for the predicted disease is displayed

Heart Disease Predictive System

As shown in Fig16, the disease “Heart Disease” can be selected from the drop down menu. After selection, the user is prompted with a set of input fields to enter the various values useful for the detection of the disease.

After providing the values, the user can click on the “Predict” button at the bottom to obtain the diagnosis of the disease along with the stage of risk (moderate risk, healthy, high risk) and the type of heart condition (angina, asymptomatic, no specific chest pain, coronary artery disease) shown in Fig17.

SHAP analysis as shown in Fig18 which is an interactive plot is provided to the user, to provide the detailed understanding of which factors are contributing towards the presence of the disease and which are not.

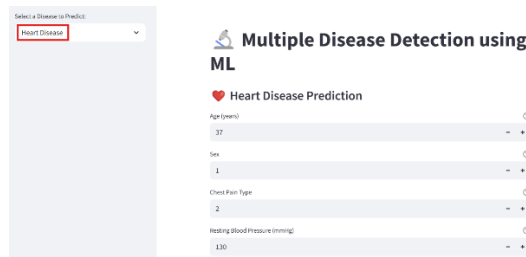


Fig16: Heart Disease is selected from the menu and values are inserted on the right

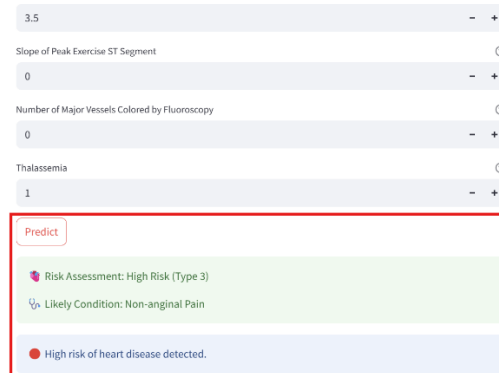


Fig17: Upon clicking the predict button, diagnosis is obtained along with the stage of risk and type of heart condition

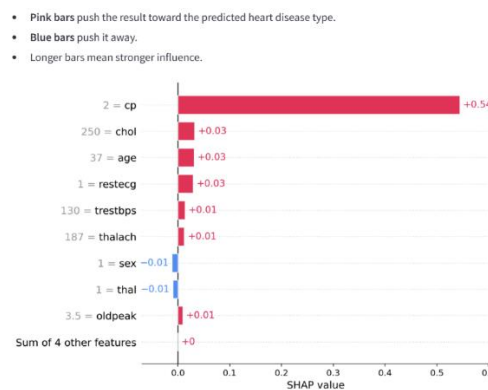


Fig18: SHAP analysis for the predicted disease is displayed

Kidney Disease Predictive System

As shown in Fig19, the disease “Kidney Disease” can be selected from the drop-down menu. After selection, the user is prompted with a set of input fields to enter the various values useful for the detection of the disease.

After providing the values, the user can click on the “Predict” button at the bottom to obtain the diagnosis of the disease as shown in Fig20.

SHAP analysis as shown in Fig21 which is an interactive plot is provided to the user, to provide the detailed understanding of which factors are contributing towards the presence of the disease and which are not.

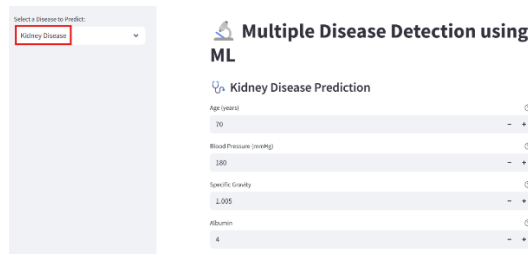


Fig18: Kidney Disease is selected from the menu and values are inserted on the right

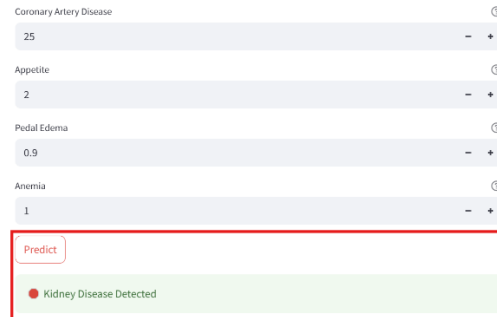


Fig19: Upon clicking the predict button, diagnosis is obtained

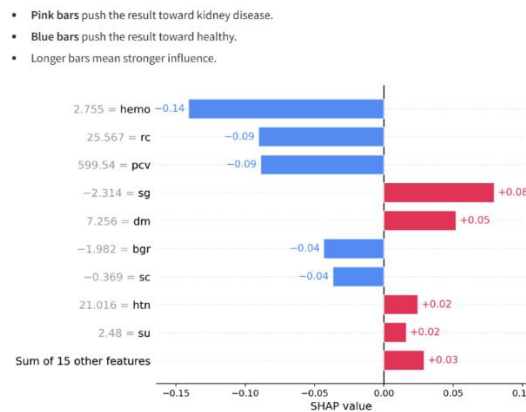


Fig20: SHAP analysis for the predicted disease is displayed

4.3. Accuracy Scores

The accuracy scores of each disease is obtained after training and testing the models with various machine learning algorithms and hyperparameter tuning using GridSearchCV in a Google Colab environment.

Diabetes Disease

As shown in Fig21, the accuracy score of the disease detection model is obtained.

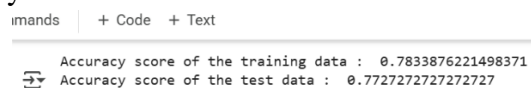


Fig21: Accuracy score of Diabetes model

Parkinson's Disease

As shown in Fig22, the accuracy score of the disease detection model is obtained.

```
Code + Text
Training Random Forest...
Random Forest Training Accuracy: 1.0000
Random Forest Testing Accuracy: 0.9487

Training Gradient Boosting...
Gradient Boosting Training Accuracy: 1.0000
Gradient Boosting Testing Accuracy: 0.8718

Training Support Vector Machine...
Support Vector Machine Training Accuracy: 0.8846
Support Vector Machine Testing Accuracy: 0.8974

Training K-Nearest Neighbors...
K-Nearest Neighbors Training Accuracy: 0.9744
K-Nearest Neighbors Testing Accuracy: 0.9231

Model Performance Summary:
Random Forest: Training Accuracy = 1.0000, Testing Accuracy = 0.9487
Gradient Boosting: Training Accuracy = 1.0000, Testing Accuracy = 0.8718
Support Vector Machine: Training Accuracy = 0.8846, Testing Accuracy = 0.8974
K-Nearest Neighbors: Training Accuracy = 0.9744, Testing Accuracy = 0.9231

Overall Average Training Accuracy: 0.9647
Overall Average Testing Accuracy: 0.9183
```

Fig22: The accuracy score of the Parkinson’s Disease model

Heart Disease

As shown in Fig23, the accuracy score of the disease detection model is obtained.

```
mands | + Code + Text
Training Accuracy: 1.0000
Testing Accuracy: 0.9344
```

Fig23: The accuracy score of the Heart Disease model

Kidney Disease

As shown in Fig24, the accuracy score of the disease detection model is obtained.

```
mands | + Code + Text
Training Accuracy: 0.987
Testing Accuracy: 0.966
```

Fig24: The accuracy score of the Kidney disease model

5. CONCLUSION AND FUTURE SCOPE

5.1. Conclusion

The multiple disease detection system developed in this project demonstrates the effective integration of machine learning with a user-friendly interface to support early and accurate diagnosis of multiple health conditions namely Diabetes, Heart Disease, Kidney Disease, and Parkinson’s Disease. Each model was trained using reliable machine learning algorithms like Logistic Regression, SVM, Random Forest, KNN, and Gradient Boosting, with preprocessing steps such as standardization and hyperparameter tuning via GridSearchCV.

The best models were saved and integrated into the application. SHAP explainability was also implemented, providing users with visual insights into how each feature contributed to the final prediction—crucial for healthcare transparency.

The user interface was built using Streamlit, an open-source framework that allowed for rapid and elegant deployment of the models. The final web application is intuitive and interactive, providing users with the ability to select a disease, input their clinical information, and receive immediate results along with clear visual explanations. The layout is clean and responsive, making it accessible to both healthcare professionals and individuals without technical expertise.

In conclusion, this project successfully bridges the gap between machine learning and practical healthcare applications by delivering a robust, scalable, and explainable disease prediction system. It not only supports early detection but also empowers users with insights into the reasoning behind each prediction.

5.2. Future scope

1. Expansion to More Diseases

Extend the system to detect other diseases such as lung cancer, liver disease, Alzheimer's, breast cancer, and COVID-19.

Integrate multi-label classification to detect co-occurring conditions.

2. Integration with Real-Time Patient Monitoring Systems

Combine with wearable devices or hospital systems to receive live feeds (e.g., heart rate, glucose levels) for real-time predictions and alerts.

3. Electronic Health Record (EHR) Integration

Automatically pull data from patient medical records to reduce manual input and improve data accuracy.

4. Voice & Image-Based Diagnosis

Incorporate models that use medical imaging (X-rays, MRIs) and voice analysis for conditions like lung disorders or neurological diseases.

5. Personalized Treatment Recommendations

Use predictive analytics to suggest customized treatment plans, medication adjustments, or follow-up schedules based on model insights.

REFERENCES

1. Singh, Kumar & Sharma, Ashutosh & Verma, Ashish & Maurya, Ranjeet & Perwej, Dr. Yusuf. (2024). Machine Learning for the Multiple Disease Prediction System. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*. 10. pp. 673-684.
2. Guo, Kehua & Shen, Changchun & Zhou, Xiaokang & Ren, Sheng & Hu, Min & Shen, Minxue & Chen, Xiang & Guo, Haifu. (2022). Traffic Data-Empowered XGBoost-LSTM Framework for Infectious Disease Prediction. *IEEE Transactions on Intelligent Transportation Systems*. PP. 1-12.
3. Deepthi, G Laxmi & Kummera, Srinivas & Pattamsetti, Sai & Kuna, Sneha & Parsi, Niharika & Kodali, Haripriya. (2023). Multiple Disease Prediction System using Machine Learning and Streamlit. pp. 923-931.
4. Vikram, R., and S. Siddarth. "Multiple Disease Predictions using Machine Learning and Deep Learning Algorithms." In *2023 International Conference on System, Computation, Automation and Networking (ICSCAN)*, pp. 1-6. IEEE, 2023.
5. Al Reshan, Mana & Amin, Samina & Zeb, Muhammad & Sulaiman, Adel & Alshahrani, Hani & Shaikh, Asadullah. (2023). A Robust Heart Disease Prediction System Using Hybrid Deep Neural Networks. *IEEE Access*. pp. 1-1.
6. Dasgupta, Dibakar & Mukherjee, Subhojit & Chakraborty, Amit & Majhi, Mou. (2023). Kidney Disease Detection using Machine Learning. *Journal of Mines, Metals and Fuels*. pp. 632-639.
7. A, Vijayalaxmi & S, Sridevi & N, Sridhar & Ambesange, Sateesh. (2020). Multi-Disease Prediction with Artificial Intelligence from Core Health Parameters Measured through Non-invasive Technique. pp.1252-1258.
8. D. Dahiwade, G. Patle and E. Meshram, "Designing Disease Prediction Model Using Machine

- Learning Approach," 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2019, pp. 1211-1215.
9. Mujumdar, A. and Vaidehi, V., 2019. Diabetes prediction using machine learning algorithms. *Procedia Computer Science*, 165, pp.292-299.
 10. Aich, S., Kim, H.C., Hui, K.L., Al-Absi, A.A. and Sain, M., 2019, February. A supervised machine learning approach using different feature selection techniques on voice datasets for prediction of Parkinson's disease. In 2019 21st International Conference on Advanced Communication Technology (ICACT) IEEE pp. 1116-1121.