Artificial Intelligence & machine Learning

1 Introduction

1.1 Project Title

Revolutionizing Liver Care: predicting Liver Cirrhosis Using Advanced Machine Learning

1.2 Team Members

1. Tanam Manoj reddy (Team Leader)
2. Yasam Harsha Vardhan (Team member)
3. Yaragarla Durga Poojitha (Team member)
4. Vuppala Vaishnavi (Team member)

2 Project Overview

2.1 Purpose

The Cirrhosis Prediction App is designed to predict the likelihood of liver cirrhosis in patients based on clinical data using a pre-trained Random Forest machine learning model. The app aims to assist healthcare professionals and individuals with a quick, data-driven risk assessment tool.

2.2 Features

- User-friendly web interface for inputting patient data.
- Prediction of cirrhosis risk based on 18 clinical features (e.g., Age, Bilirubin, Status).
- Display of prediction results with interpretation (Cirrhosis or No Cirrhosis).
- Informative pages for Home, About, and Contact.
- Error handling for invalid inputs or model failures.

3 Architecture

3.1 Frontend

The frontend is built using HTML5 and CSS, with placeholder CSS files (index.css, home.css, about.css, contact.css, result.css, error.css). It includes:

- home.html: Landing page with a call-to-action.
- index.html: Prediction form with JavaScript for age conversion (years to days).
- about.html: Project information.
- contact.html: Contact details.
- result.html: Prediction output.
- error.html: Error display page.

3.2 Backend

The backend is developed using Flask, a lightweight Python web framework. It:

- Handles routing for all pages (/, /about, /contact, /form, /predict).

- Processes form data, applies normalization, and uses a Random Forest model for predictions.
- Implements logging for debugging and error tracking.

3.3 Machine Learning Models

The app uses pre-trained models:

- random_forest_model.pkl: Random Forest classifier.
- normalizer.pkl: Data normalizer.

Input features include 18 clinical parameters (e.g., N_Days, Status, Bilirubin), with categorical features encoded using predefined mappings. Models are loaded using pickle and joblib.

4 Setup Instructions

4.1 Prerequisites

- Python 3.8+
- Flask (pip install flask)
- Pandas (pip install pandas)
- Joblib (pip install joblib)
- Pickle (included in Python standard library)
- Web browser (e.g., Chrome, Firefox)
- Model files: random_forest_model.pkl, normalizer.pkl

4.2 Installation

1. Clone the repository:

```
git clone <repository-url> cd cirrhosis-prediction-app
```

2. Create a virtual environment:

```
python -m venv venv
source venv/bin/activate % On Windows: venv\Scripts\activate
```

3. Install dependencies:

```
pip install flask pandas joblib
```

4. Ensure random_forest_model.pkl and normalizer.pkl are in the project root.
5. Create a static/css directory and add CSS files (e.g., index.css).
6. Create a templates directory and place HTML files (home.html, index.html, about.html, contact.html, result.html, error.html).
7. Set up environment variables (optional):
   - FLASK_ENV=development
   - FLASK_APP=app.py

5 Folder Structure

5.1 Client (Frontend)

```
templates/ home.html
                        % Landing page
  index.html            % Prediction form
  about.html            % Project information
  contact.html          % Contact details
  result.html           % Prediction results
  error.html            % Error display page
static/
  css/
```

```
index.css  % Styles for index.html (placeholder) home.css  % Styles for home.html
(placeholder) about.css          % Styles for about.html (placeholder) contact.css %
Styles for contact.html (placeholder) result.css      % Styles for result.html (placeholder)
error.css  % Styles for error.html (placeholder)
```

5.2 Server (Backend)

```
.
app.py                                  % Main Flask application
random_forest_model.pkl % Pre-trained Random Forest model normalizer.pkl          %          Pre-trained
normalizer
```

6 Running the Application

6.1 Frontend

The frontend is served by Flask's template rendering, so no separate frontend server is needed.

6.2 Backend

Run the Flask server:

```
cd <project-directory> source venv/bin/activate % On Windows: venv\Scripts\activate python
app.py
```

Access the app at http://127.0.0.1:5000.

7 API Documentation

The application exposes the following endpoints:

- **GET /**
  - *Description*: Renders the home page.
  - *Response*: HTML (home.html).
- **GET /about**
  - *Description*: Renders the about page.
  - *Response*: HTML (about.html).

- **GET /contact**
  - *Description*: Renders the contact page.
  - *Response*: HTML (contact.html).
- **GET /form**
  - *Description*: Renders the prediction form.
  - *Response*: HTML (index.html).
- **POST /predict**
  - *Description*: Processes patient data and returns a prediction.
  - *Parameters* (Form Data):
    - N_Days: Number (days since registration).
    - Status: Enum (C, CL, D).
    - Drug: Enum (D-penicillamine, Placebo, Other).
    - Age: Number (age in years, converted to days).
    - Sex: Enum (M, F).
    - Ascites: Enum (N, Y).
    - Hepatomegaly: Enum (N, Y).
    - Spiders: Enum (N, Y).
    - Edema: Enum (N, Y, S).
    - Bilirubin, Cholesterol, Albumin, Copper, Alk_Phos, SGOT, Tryglicerides, Platelets, Prothrombin: Numbers.
  - *Example Request* (Form Data):

```
N_Days=400 Status=C
Drug=D-penicillamine
Age=50
Sex=F
Ascites=N
Hepatomegaly=Y Spiders=N
Edema=N
Bilirubin=1.2
Cholesterol=200 Albumin=3.5
Copper=50
Alk_Phos=1000
SGOT=80
Tryglicerides=120
Platelets=250 Prothrombin=10.5
```

  - *Example Response* (Success): HTML (result.html) with:
    - prediction: 1 (Cirrhosis) or 0 (No Cirrhosis).
    - interpretation: “Cirrhosis” or “No Cirrhosis (Stage 0)”.
  - *Example Response* (Error): HTML (error.html) with:

 error_message: e.g., “Normalizer is not loaded”.

8 Authentication

Current Implementation: The app does not implement authentication or authorization, as it is designed for open access.

Future Consideration: Authentication could be added using Flask-Login or JWT to restrict access to authorized users (e.g., healthcare professionals). User sessions could be managed with Flask’s session handling.

9 User Interface

The UI consists of:

- **Home Page**: Welcomes users with a call-to-action button.
- **Prediction Form**: Inputs for 18 clinical parameters, with dropdowns for categorical features.
- **Result Page**: Displays prediction outcome with a retry option.
- **About Page**: Provides project context.
- **Contact Page**: Lists contact information.
- **Error Page**: Displays error messages with a retry link.

Screenshots: To be added by the developer, as CSS styling is not provided.

10 Testing

Testing Strategy:

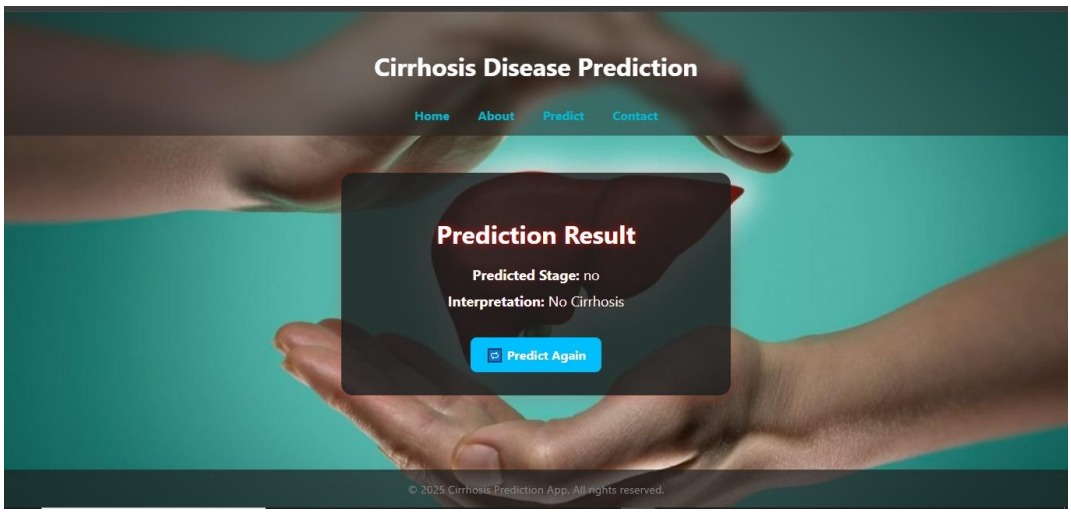
- *Unit Testing*: Test Flask routes and prediction logic using unittest or pytest.
- *Integration Testing*: Verify form submission and prediction pipeline.
- *Model Validation*: Ensure model files produce expected outputs.

Tools:

- pytest for unit tests.
- Manual testing via browser for UI.
- Logging (in app.py) for debugging.

Current Status: No automated tests implemented. Manual testing recommended.

11. Screenshots



12 Known Issues

- Missing CSS files may result in unstyled pages.
- error.html was not initially provided but is referenced.
- Invalid numerical inputs default to 0.0, potentially skewing predictions.
- No client-side form validation beyond HTML5 required attributes.
- No authentication, making the app openly accessible.
- Model loading errors cause prediction failures.

13 Future Enhancements

- Implement client-side form validation using JavaScript.
- Add CSS styling for improved UI/UX.
- Introduce authentication (e.g., Flask-Login or JWT).
- Develop automated tests using pytest.

- Add a MongoDB database for prediction history (transition to MERN if desired).
- Deploy to a cloud platform (e.g., Heroku, AWS).
- Enhance the model with real-time retraining or additional features.