

**The Superior University**

**Project Title**

Vehicle Management System

**Project Details**

1. Course: Artificial Intelligence
2. Instructor: Sir Rasikh Ali
3. Semester: 3rd
4. Section: AI(3B)
5. Submission Date: (10/12/2024)
6. Group Members: Solo

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Roll No** | **Email** | **Contact** |
| Ahmar Hussan Bhatti | SU92-BSAIM-F23-120 | su92-bsaim-f23-120@superior.edu.pk | 0308-4025656 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Abstract**

The **Health Navigator** is a GUI-based application designed to predict potential diseases based on user-input symptoms and provide medication recommendations. Built using Python, it utilizes a Decision Tree Classifier to analyze input symptoms and predict diseases with corresponding suggested medications. The application includes a user-friendly interface developed with Tkinter, a searchable symptom list, and a robust prediction mechanism. This project showcases the integration of machine learning, GUI development, and data processing to solve real-world healthcare problems.

**Table of Contents**

1. Introduction
2. Objectives
3. System Requirements
4. Methodology
5. Implementation
6. Challenges and Solutions
7. Conclusion
8. **Introduction**

**Overview of the project:**

The Health Navigator is a software solution to assist users in identifying potential diseases based on their symptoms and providing medication recommendations. By integrating machine learning with a graphical user interface, the project simplifies healthcare decision-making for non-specialists. **Explanation of the selected topic**:

The Decision Tree Classifier was selected due to its interpretability and suitability for categorical data like symptoms. Tkinter was used for its simplicity and capability to create interactive GUIs, while file handling ensures data persistence and reusability.

1. **Objectives**

* Predict diseases based on symptoms using a trained machine learning model.
* Recommend appropriate medications for diagnosed diseases.
* Develop a GUI for intuitive symptom selection and result display.
* Enable real-time symptom filtering through a search functionality.
* Provide an easily extensible framework for future enhancements.

1. **System Requirements**

Hardware Requirements:

* **Processor**: Intel Core i5 or above
* **Memory**: 4 GB RAM or higher
* **Storage**: At least 500 MB

Software Requirements:

* **Operating System**: Windows 10 /Linux (64 or 32 bit)
* **Programming Language**: Python 3.10 or newer
* **Libraries**
  + **Pandas** for data manipulation
  + **Scikit-learn** for machine learning
  + **Tkinter** for GUI design

1. M**ethodology**

#### Approach:

1. **Data Preparation**:
   * Processed a dataset containing diseases, symptoms, and medications.
   * One-hot encoded symptoms for machine learning compatibility.
2. **Model Training**:
   * Utilized a Decision Tree Classifier for disease prediction.
3. **GUI Design**:
   * Developed a scrollable, searchable interface for symptom selection.
   * Integrated prediction and result display functionality.
4. **Testing**:
   * Conducted extensive testing to ensure accuracy and usability.

#### Workflow

1. Users input symptoms via the GUI.
2. Symptoms are processed and passed to the trained model.
3. The model predicts the disease and retrieves medication recommendations.
4. Results are displayed in a user-friendly format.

**Concepts Used:**

#### ****Decision Tree Classifier****

* **How it Works**:
  + Splits the dataset based on features (symptoms) to make decisions.
  + Reaches a leaf node to predict a disease.
* **Advantages**:
  + Handles categorical data efficiently.
  + Easy to interpret and debug.

#### ****GUI with Tkinter****

* **Features**:
  + Checkboxes for symptom selection.
  + Searchable symptom list.
  + Buttons for prediction, resetting, and exiting.

1. **Implementation:**

**Code Implementation:**

import pandas as pd

import tkinter as tk

from tkinter import messagebox, ttk

from sklearn.tree import DecisionTreeClassifier

def load\_data(file\_path):

    try:

        data = pd.read\_csv(file\_path).dropna()

        symptoms = data['Symptoms'].str.get\_dummies(sep=', ')

        X = symptoms

        y = data['Disease']

        medications = data[['Disease', 'Medicine']].drop\_duplicates().set\_index('Disease').to\_dict()['Medicine']

        return X, y, medications, symptoms.columns.tolist()

    except Exception as e:

        messagebox.showerror("Error", f"Failed to load dataset: {e}")

        return None, None, None, None

def train\_model(X, y):

    model = DecisionTreeClassifier()

    model.fit(X, y)

    return model

def get\_input\_vector(selected\_symptoms, all\_symptoms):

    input\_vector = [1 if symptom in selected\_symptoms else 0 for symptom in all\_symptoms]

    return pd.DataFrame([input\_vector], columns=all\_symptoms)

def diagnose():

    selected\_symptoms = [symptom for symptom, var in symptom\_vars.items() if var.get()]

    if not selected\_symptoms:

        messagebox.showwarning("Input Error", "Please select at least one symptom.")

        return

    input\_data = get\_input\_vector(selected\_symptoms, all\_symptoms)

    predicted\_disease = model.predict(input\_data)[0]

    medication = medications.get(predicted\_disease, "No medication found.")

    result\_text = f"Predicted Disease: {predicted\_disease}\n\nSuggested Medication: {medication}\n\nAdvice: Please consult a doctor for confirmation."

    result\_label.config(text=result\_text)

    new\_prediction\_button.pack(side="left", padx=10, pady=10)

    exit\_button.pack(side="left", padx=10, pady=10)

def new\_prediction():

    for var in symptom\_vars.values():

        var.set(False)

    result\_label.config(text="")

    new\_prediction\_button.pack\_forget()

    exit\_button.pack\_forget()

def exit\_application():

    root.destroy()

def create\_symptom\_box(root):

    symptom\_frame = ttk.Frame(root)

    symptom\_frame.grid(row=1, column=0, padx=20, pady=10, sticky="nsew")

    search\_label = ttk.Label(symptom\_frame, text="Search Symptoms:")

    search\_label.grid(row=0, column=0, sticky="w", padx=10, pady=5)

    search\_entry = ttk.Entry(symptom\_frame, textvariable=search\_var, width=30)

    search\_entry.grid(row=0, column=1, padx=10, pady=5)

    search\_entry.bind("<KeyRelease>", lambda e: update\_symptoms())

    canvas = tk.Canvas(symptom\_frame)

    canvas.grid(row=1, column=0, columnspan=2, sticky="nsew")

    scrollbar = ttk.Scrollbar(symptom\_frame, orient="vertical", command=canvas.yview)

    scrollbar.grid(row=1, column=2, sticky="ns")

    canvas.configure(yscrollcommand=scrollbar.set)

    symptom\_list\_frame = ttk.Frame(canvas)

    canvas.create\_window((0, 0), window=symptom\_list\_frame, anchor="nw")

    for idx, symptom in enumerate(all\_symptoms):

        var = tk.BooleanVar()

        checkbutton = ttk.Checkbutton(symptom\_list\_frame, text=symptom, variable=var, width=25)

        checkbutton.grid(row=idx, column=0, padx=10, pady=5, sticky="w")

        symptom\_vars[symptom] = var

    symptom\_list\_frame.update\_idletasks()

    canvas.config(scrollregion=canvas.bbox("all"))

def update\_symptoms():

    query = search\_var.get().lower()

    for widget in symptom\_widgets.values():

        widget.grid\_forget()

    for idx, symptom in enumerate(all\_symptoms):

        if query in symptom.lower():

            symptom\_widgets[symptom].grid(row=idx, column=0, padx=10, pady=5, sticky="w")

def setup\_gui():

    global root

    root = tk.Tk()

    root.title("Health Navigator")

    root.geometry("700x700")

    root.resizable(False, False)

    global search\_var

    search\_var = tk.StringVar()

    title\_label = ttk.Label(root, text="Health Navigator", font=("Arial", 18, "bold"))

    title\_label.grid(row=0, column=0, columnspan=2, pady=10)

    instruction\_label = ttk.Label(root, text="Select your symptoms from the list below:")

    instruction\_label.grid(row=1, column=0, columnspan=2, pady=10)

    create\_symptom\_box(root)

    predict\_button = ttk.Button(root, text="Predict Disease", command=diagnose)

    predict\_button.grid(row=4, column=0, columnspan=2, pady=20, sticky="ew")

    global result\_label

    result\_label = ttk.Label(root, text="", font=("Arial", 12), wraplength=500, justify="left")

    result\_label.grid(row=5, column=0, columnspan=2, pady=10)

    button\_frame = ttk.Frame(root)

    button\_frame.grid(row=6, column=0, columnspan=2, pady=10)

    global new\_prediction\_button

    new\_prediction\_button = ttk.Button(button\_frame, text="New Prediction", command=new\_prediction)

    new\_prediction\_button.pack\_forget()

    global exit\_button

    exit\_button = ttk.Button(button\_frame, text="Exit", command=exit\_application)

    exit\_button.pack\_forget()

    root.mainloop()

file\_path = r"C:\Users\AHMAR HASSAN\Desktop\AI project\medical data.csv"

X, y, medications, all\_symptoms = load\_data(file\_path)

if X is not None and y is not None:

    model = train\_model(X, y)

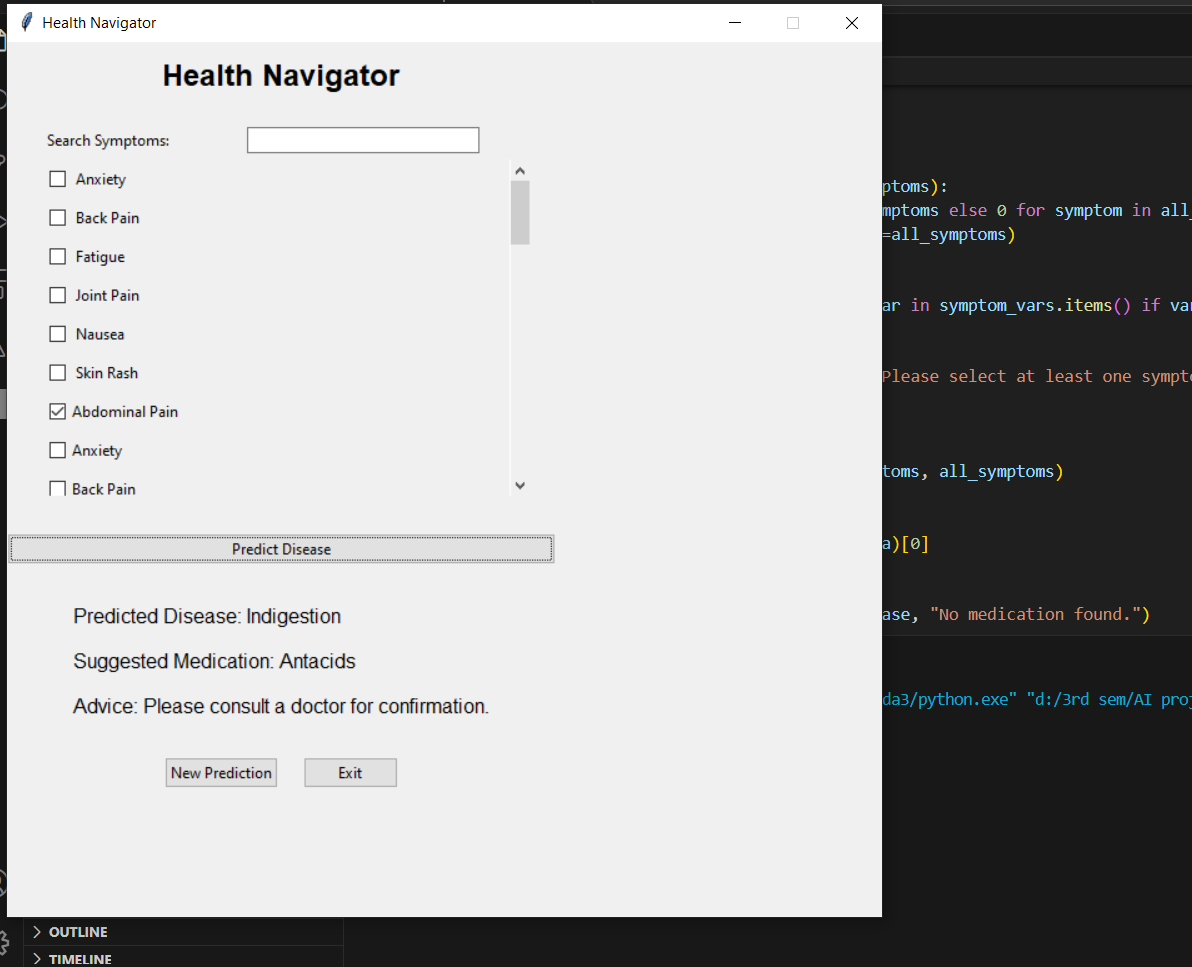
    symptom\_vars = {}

    symptom\_widgets = {}

    setup\_gui()

else:

    print("Error: Dataset could not be loaded.")

**Output: **

#### Core Features:

* **Symptom Selection**: Interactive checkbox-based system.
* **Disease Prediction**: Real-time predictions based on input symptoms.
* **Medication Recommendations**: Provides medication suggestions alongside predictions.
* **Searchable Symptom List**: Filters symptoms dynamically as the user types.
* **GUI**: User-friendly design built with Tkinter.

**Tools or Platform used:**

**Python 3.x , Visual studio code**

1. **Challenges Faced:**
2. **Handling Data**: Cleaning and preparing the dataset.
3. **Designing the GUI**: Making it simple and easy to use.
4. **Model Accuracy**: Ensuring correct predictions.

1. **Conclusion**

The Health Navigator successfully demonstrates the integration of machine learning and GUI development to solve healthcare-related problems. It provides users with a reliable and interactive way to identify potential diseases and medications. The project highlights the application of machine learning and data structures in developing impactful solutions.

**Future Improvements**

1. Add more diseases and symptoms to the dataset.
2. Use more advanced machine learning models.
3. Improve the design for better user experience.