**MULTIPLE DISEASE PREDICTION**

**Internship Project Report**

**Submitted By**

**M.HARSHITHA**

**S. VINESH**

**D. SAI VARSHITH**

**Under the guidance of**

**Prof T. Rama Krishnudu**

**I.INTRODUCTION:**

**1.1 Background**

In today's digital age, the internet has become a vital tool for seeking information and solutions, particularly regarding health concerns. With easier access to the internet than to hospitals or doctors, many people turn to online resources for medical guidance. This reliance on digital solutions has led to the development of web-based applications that can predict diseases based on user-provided symptoms. The Disease Prediction System is designed to assist users by analyzing their symptoms and suggesting the most likely disease, utilizing data collected from various health-related sources.

This system is particularly useful in situations where immediate medical consultation is required but a doctor is unavailable due to prior commitments or other reasons. By entering their symptoms into the application, users can receive prompt and accurate disease predictions, enabling them to seek further treatment from appropriate specialists. The system uses advanced data processing techniques to ensure accurate predictions, providing a valuable tool for urgent medical guidance. Additionally, this application offers free consultation services, making it accessible to a wide range of users. The Disease Prediction System not only addresses the immediate health concerns of individuals but also contributes to the broader goal of improving access to healthcare information and services.

**1.2 Motivation**

In the realm of healthcare, the timely and accurate diagnosis of diseases is crucial for effective treatment and improved patient outcomes. While there have been numerous studies on human disease prediction using various methods, there remains a significant need for accessible and reliable diagnostic tools that can be used by individuals from the comfort of their homes.

Many regions face challenges such as a shortage of skilled medical professionals, long distances to healthcare facilities, and lack of advanced diagnostic equipment. These barriers often lead to delays in diagnosis and treatment, exacerbating health conditions. Traditional diagnostic methods can be time-consuming, subjective, and prone to human error, which further complicates timely intervention. Moreover, the cost of medical consultations and laboratory tests can be prohibitive for many, preventing them from seeking necessary medical care.

The primary contributions of our research are twofold. First, we aim to develop an automated system that can assist individuals in accurately predicting potential diseases based on their symptoms using deep learning techniques. By leveraging deep learning algorithms and extensive health data, our model can provide objective and consistent results, facilitating early disease detection and intervention. Second, our work adds to the growing body of knowledge on deep learning-based disease prediction, promoting advancements in digital health solutions. The motivation behind this project is to enhance the accessibility and accuracy of disease prediction, empowering users to take proactive steps in managing their health and seeking appropriate medical care without financial or logistical barriers.

**II.PROBLEM STATEMENT:**

The objective is to develop a function f(X) that can map a set of input symptoms xi​∈*D* to its corresponding disease label *y*. Here, *D* represents the dataset containing various sets of symptoms and their associated disease labels. The function f(X) will be implemented as a deep learning model designed to accurately identify the disease based on the input symptoms.

This model *f*(*X*) can be expressed with the equation:

*y*pred​=*f*(*x*)

where:

* *y*pred​ denotes the predicted disease label for the input set of symptoms *xi*​
* *f*(*X*) is the trained deep learning model that takes an input set of symptoms 𝑥𝑖*xi*​ and generates the predicted disease label *y*pred​

The primary goal is to train the deep learning model *f*(*X*) using an appropriate algorithm to reduce classification errors and achieve high accuracy in predicting diseases from the provided symptoms.

**III.OBJECTIVE:**

This project aims to develop an advanced predictive model capable of accurately identifying diseases from input symptoms. By training on a diverse range of symptom patterns associated with various diseases, the model will learn to predict the correct disease label for new sets of symptoms. The primary goal is to leverage deep learning techniques to enhance diagnostic accuracy, particularly in distinguishing between overlapping symptoms of different diseases. This capability is crucial for supporting healthcare professionals in early disease detection and intervention, thereby improving patient outcomes and optimizing healthcare resource allocation. Ultimately, the project aims to deliver a robust and scalable solution that contributes to advancing medical diagnostics through the application of deep learning in disease prediction from symptoms.

**IV.SIGNIFICANCE OF STUDY:**

Communicable diseases pose serious threats to public health, exacerbated by the high cost of diagnostics and shortages of skilled medical professionals, especially in rural areas. This study aims to develop a machine learning-based prototype for disease prediction using symptoms, enhancing early diagnosis and reducing reliance on expensive tests. By enabling rapid and accurate assessment, this research supports better disease management and healthcare resource allocation, contributing significantly to advancements in artificial intelligence applications in public health.

**V.DATA COLLECTION**

This study's goal was addressed through a symptom dataset, requiring a large amount of data for training and testing the model. Symptom data can be collected from various medical records, health databases, and patient surveys. In our case, we gathered symptom data from multiple health databases and publicly available datasets, including the "Disease Symptoms Dataset" from Kaggle. Additionally, symptoms could be recorded from patient reports and medical consultations to prepare the dataset.

**VI.PROPOSED METHODOLOGY:**

1. ***Data Loading and preprocessing:***

Disease contains a different number of symptoms while a disease has less number of symptoms the field will be a null value. This dataset contains a total of 18 columns.

B.***Encoding:***

Encode the target labels (diseases) into numerical values using Label Encoder

Split the dataset into features X and Y labels.

1. ***Data Splitting:***

Split the data into training and testing sets.

1. Data Acquisition:
   * Collect data from multiple sources including health organizations, health workers, crowdsourcing, and non-profit organizations.
   * Pre-process data for further analysis.
2. Back-end Data Management System:
   * Implement scalable deep learning algorithms.
   * Develop a computational framework for integrating data from various sources.
   * Post-process and perform statistical analysis on the data.
3. Prediction and Visualization System:
   * Train models with a rich dataset to predict disease spread.
   * Use deep learning methods like CNN for pattern recognition.
   * Develop a user interface for visualizing predictions using web-based tools and mobile apps.
4. Knowledge Dissemination:
   * Create a web portal for sharing knowledge gained from the system.
   * Use graphs and text to communicate information effectively.
   * Ensure the system is responsive and accessible to end-users.

**VII.BLOCK DIAGRAM:**

**A screenshot of a computer

Description automatically generated**

**VIII.CODE:**

**CODE FOR CNN MODEL:**

import numpy as np

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential, save\_model  # Import save\_model

from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout

from tensorflow.keras.utils import to\_categorical

import seaborn as sns

import matplotlib.pyplot as plt

from collections import Counter

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Reading the train.csv by removing the last column since it's an empty column

data = pd.read\_csv('/content/Training (1).csv').dropna(axis=1)

# Checking whether the dataset is balanced or not

disease\_counts = data["prognosis"].value\_counts()

temp\_df = pd.DataFrame({

    "Disease": disease\_counts.index,

    "Counts": disease\_counts.values

})

plt.figure(figsize=(18, 8))

sns.barplot(x="Disease", y="Counts", data=temp\_df)

plt.xticks(rotation=90)

plt.show()

# Encoding the target value into numerical value using LabelEncoder

encoder = LabelEncoder()

data["prognosis"] = encoder.fit\_transform(data["prognosis"])

# Splitting data into features and labels

X = data.iloc[:, :-1]

y = data.iloc[:, -1]

# One-hot encode the labels

y\_categorical = to\_categorical(y)

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_categorical, test\_size=0.2, random\_state=24)

# Reshape input for CNN

X\_train = np.expand\_dims(X\_train.values, axis=2)

X\_test = np.expand\_dims(X\_test.values, axis=2)

print(f"Train: {X\_train.shape}, {y\_train.shape}")

print(f"Test: {X\_test.shape}, {y\_test.shape}")

# Define the CNN model

model = Sequential()

model.add(Conv1D(32, kernel\_size=3, activation='relu', input\_shape=(X\_train.shape[1], 1)))

model.add(MaxPooling1D(pool\_size=2))

model.add(Dropout(0.25))

model.add(Conv1D(64, kernel\_size=3, activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(y\_categorical.shape[1], activation='softmax'))

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, validation\_split=0.2, epochs=50, batch\_size=32)

# Save the trained model to an HDF5 file

model.save('disease\_prediction\_model')

# Plotting the accuracy and loss curves

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Accuracy Curves')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Loss Curves')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

# Load the new test dataset

new\_data = pd.read\_csv('/content/Testing (1).csv').dropna(axis=1)

# Preprocess the new dataset

X\_new = new\_data.iloc[:, :-1]

y\_new = encoder.transform(new\_data.iloc[:, -1])

y\_new\_categorical = to\_categorical(y\_new)

# Reshape input for CNN

X\_new = np.expand\_dims(X\_new.values, axis=2)

# Evaluate the model on the new dataset

new\_loss, new\_accuracy = model.evaluate(X\_new, y\_new\_categorical)

print(f'New dataset accuracy: {new\_accuracy \* 100:.2f}%')

# Make predictions on the new dataset

predictions = model.predict(X\_new)

# Convert predictions to class labels

predicted\_classes = np.argmax(predictions, axis=1)

predicted\_labels = encoder.inverse\_transform(predicted\_classes)

# Print or save the predictions

print(predicted\_labels)

# Confusion matrix

cf\_matrix = confusion\_matrix(y\_new, predicted\_classes)

plt.figure(figsize=(12, 8))

sns.heatmap(cf\_matrix, annot=True)

plt.title("Confusion Matrix for CNN ")

plt.show()

# Function to predict disease based on symptoms

symptom\_index = {symptom: index for index, symptom in enumerate(X.columns)}

def predictDisease(symptoms):

    symptoms = symptoms.split(",")

    # Create input data for the model

    input\_data = [0] \* len(symptom\_index)

    for symptom in symptoms:

        symptom = symptom.strip()

        if symptom in symptom\_index:

            index = symptom\_index[symptom]

            input\_data[index] = 1

        else:

            print(f"Warning: Symptom '{symptom}' not found in dataset")

    # Reshape input data

    input\_data = np.array(input\_data).reshape(1, -1)

    input\_data = np.expand\_dims(input\_data, axis=2)

    # Generate prediction

    prediction = model.predict(input\_data)

    predicted\_class = np.argmax(prediction, axis=1)

    predicted\_label = encoder.inverse\_transform(predicted\_class)[0]

    return predicted\_label

# Test the function

print(predictDisease("fever,dry cough,tiredness"))

**CODE FOR DISEASE PREDICTION SYSTEM:**

import tkinter as tk

from tkinter import \*

import re

# Define the diseases and their symptoms

disease\_data = {

    'Influenza': ['fever', 'cough', 'headache', 'sore throat'],

    'Food Poisoning': ['vomiting', 'weakness', 'fever'],

    'Diarrhea': ['watery stool', 'vomiting', 'abdominal cramps'],

    'Common Cold': ['sneezing', 'runny nose', 'cough'],

    'COVID-19': ['fever', 'dry cough', 'tiredness', 'loss of taste or smell'],

    'Chickenpox': ['itching', 'skin rash', 'fever', 'fatigue'],

    'Measles': ['fever', 'cough', 'runny nose', 'conjunctivitis'],

    'Mumps': ['fever', 'swollen glands (neck)', 'muscle pain', 'loss of appetite'],

    'Tuberculosis': ['cough', 'chest pain', 'weight loss', 'night sweats'],

    'Typhoid': ['high fever', 'stomach pain', 'headache', 'weakness'],

    'Hepatitis A': ['fatigue', 'nausea', 'abdominal pain', 'jaundice'],

    'Malaria': ['fever', 'chills', 'headache', 'nausea', 'vomiting'],

    'Dengue Fever': ['high fever', 'severe headache', 'joint pain', 'pain behind the eyes'],

    'Cholera': ['profuse watery diarrhea', 'vomiting', 'leg cramps', 'dehydration'],

    'Rabies': ['fever', 'headache', 'excessive salivation', 'fear of water'],

    'Impetigo': ['red sores', 'oozing fluid', 'crusty golden-brown sores'],

    'Pneumonia': ['cough', 'fever', 'chills', 'shortness of breath'],

    'HIV/AIDS': ['fever', 'unexplained weight loss', 'extreme tiredness', 'swollen lymph nodes'],

    'Scabies': ['itching', 'skin rash', 'pimple-like irritations'],

    'Chlamydia': ['painful urination', 'lower abdominal pain', 'unusual discharge', 'rectal pain or discharge'],

    'Gonorrhea': ['painful urination', 'discharge from genitals', 'rectal pain', 'sore throat'],

    'Syphilis': ['sores (chancre) on genitals, anus, or mouth', 'rash on palms of hands or soles of feet', 'fever', 'fatigue'],

    'Ringworm': ['itching', 'skin rash', 'redness of skin']

}

# Flatten the disease\_data for the autocomplete list

autocompleteList = sorted(set([symptom for symptoms in disease\_data.values() for symptom in symptoms]))

# Define the autocomplete entry class

class AutocompleteEntry(tk.Entry):

    def \_\_init\_\_(self, autocompleteList, \*args, \*\*kwargs):

        self.listboxLength = 0

        self.parent = args[0]

        # Custom matches function

        if 'matchesFunction' in kwargs:

            self.matchesFunction = kwargs['matchesFunction']

            del kwargs['matchesFunction']

        else:

            def matches(fieldValue, acListEntry):

                pattern = re.compile('.\*' + re.escape(fieldValue) + '.\*', re.IGNORECASE)

                return re.match(pattern, acListEntry)

            self.matchesFunction = matches

        # Custom return function

        if 'returnFunction' in kwargs:

            self.returnFunction = kwargs['returnFunction']

            del kwargs['returnFunction']

        else:

            def selectedValue(value):

                print(value)

            self.returnFunction = selectedValue

        tk.Entry.\_\_init\_\_(self, \*args, \*\*kwargs)

        self.focus()

        self.autocompleteList = autocompleteList

        self.var = self["textvariable"]

        if self.var == '':

            self.var = self["textvariable"] = tk.StringVar()

        self.var.trace('w', self.changed)

        self.bind("<Right>", self.selection)

        self.bind("<Up>", self.moveUp)

        self.bind("<Down>", self.moveDown)

        self.bind("<Return>", self.selection)

        self.bind("<Escape>", self.deleteListbox)

        self.listboxUp = False

    def deleteListbox(self, event=None):

        if self.listboxUp:

            self.listbox.destroy()

            self.listboxUp = False

    def select(self, event=None):

        if self.listboxUp:

            index = self.listbox.curselection()[0]

            value = self.listbox.get(tk.ACTIVE)

            self.listbox.destroy()

            self.listboxUp = False

            self.delete(0, tk.END)

            self.insert(tk.END, value)

            self.returnFunction(value)

    def changed(self, name, index, mode):

        if self.var.get() == '':

            self.deleteListbox()

        else:

            words = self.comparison()

            if words:

                if not self.listboxUp:

                    self.listboxLength = len(words)

                    self.listbox = tk.Listbox(self.parent,

                                              width=self["width"], height=self.listboxLength)

                    self.listbox.bind("<Button-1>", self.selection)

                    self.listbox.bind("<Right>", self.selection)

                    self.listbox.place(

                        x=self.winfo\_x(), y=self.winfo\_y() + self.winfo\_height())

                    self.listboxUp = True

                else:

                    self.listboxLength = len(words)

                    self.listbox.config(height=self.listboxLength)

                self.listbox.delete(0, tk.END)

                for w in words:

                    self.listbox.insert(tk.END, w)

            else:

                self.deleteListbox()

    def selection(self, event):

        if self.listboxUp:

            self.var.set(self.listbox.get(tk.ACTIVE))

            self.listbox.destroy()

            self.listboxUp = False

            self.icursor(tk.END)

    def moveUp(self, event):

        if self.listboxUp:

            if self.listbox.curselection() == ():

                index = '0'

            else:

                index = self.listbox.curselection()[0]

            self.listbox.selection\_clear(first=index)

            index = str(int(index) - 1)

            if int(index) == -1:

                index = str(self.listboxLength - 1)

            self.listbox.see(index)  # Scroll!

            self.listbox.selection\_set(first=index)

            self.listbox.activate(index)

    def moveDown(self, event):

        if self.listboxUp:

            if self.listbox.curselection() == ():

                index = '-1'

            else:

                index = self.listbox.curselection()[0]

            if index != tk.END:

                self.listbox.selection\_clear(first=index)

                if int(index) == self.listboxLength - 1:

                    index = "0"

                else:

                    index = str(int(index) + 1)

                self.listbox.see(index)  # Scroll!

                self.listbox.selection\_set(first=index)

                self.listbox.activate(index)

    def comparison(self):

        return [w for w in self.autocompleteList if self.matchesFunction(self.var.get(), w)]

# Function to predict disease

def prediction():

    symptoms = [symptom\_entry1.get(), symptom\_entry2.get(), symptom\_entry3.get(), symptom\_entry4.get(), symptom\_entry5.get()]

    symptoms = [s for s in symptoms if s]  # Remove empty strings

    max\_match = 0

    predicted\_disease = "Unknown Disease"

    for disease, disease\_symptoms in disease\_data.items():

        match\_count = len(set(symptoms) & set(disease\_symptoms))

        if match\_count > max\_match:

            max\_match = match\_count

            predicted\_disease = disease

    final\_result.delete(0, tk.END)

    final\_result.insert(0, f'You might be suffering from: {predicted\_disease}')

# Set up the main application window

root = tk.Tk()

root.title("Disease Prediction")

frame = LabelFrame(root, padx=10, pady=30, highlightthickness=2)

frame.pack(padx=50, pady=50)

c = Label(frame, text="Disease Prediction System", fg='blue4')

c.grid(row=0, column=0, columnspan=3, pady=(0, 20), padx=(30, 30), sticky="nsew")

c.config(font=("Consolas", 32, 'bold'))

L1 = Label(frame, text='Symptom 1:')

L1.grid(row=1, column=0, sticky=W, pady=(0, 10), padx=(120, 0))

L1.config(font=("Consolas", 20))

symptom\_entry1 = AutocompleteEntry(

    autocompleteList, frame, width=32, fg='orange', bg='black', insertbackground='orange')

symptom\_entry1.grid(row=1, column=1)

symptom\_entry1.config(font=('Consolas', 15, 'bold'))

L2 = Label(frame, text='Symptom 2:')

L2.grid(row=2, column=0, sticky=W, pady=(0, 10), padx=(120, 0))

L2.config(font=("Consolas", 20))

symptom\_entry2 = AutocompleteEntry(

    autocompleteList, frame, width=32, fg='orange', bg='black', insertbackground='orange')

symptom\_entry2.grid(row=2, column=1)

symptom\_entry2.config(font=('Consolas', 15, 'bold'))

L3 = Label(frame, text='Symptom 3:')

L3.grid(row=3, column=0, sticky=W, pady=(0, 10), padx=(120, 0))

L3.config(font=("Consolas", 20))

symptom\_entry3 = AutocompleteEntry(

    autocompleteList, frame, width=32, fg='orange', bg='black', insertbackground='orange')

symptom\_entry3.grid(row=3, column=1)

symptom\_entry3.config(font=('Consolas', 15, 'bold'))

L4 = Label(frame, text='Symptom 4:')

L4.grid(row=4, column=0, sticky=W, pady=(0, 10), padx=(120, 0))

L4.config(font=("Consolas", 20))

symptom\_entry4 = AutocompleteEntry(

    autocompleteList, frame, width=32, fg='orange', bg='black', insertbackground='orange')

symptom\_entry4.grid(row=4, column=1)

symptom\_entry4.config(font=('Consolas', 15, 'bold'))

L5 = Label(frame, text='Symptom 5:')

L5.grid(row=5, column=0, sticky=W, pady=(0, 10), padx=(120, 0))

L5.config(font=("Consolas", 20))

symptom\_entry5 = AutocompleteEntry(

    autocompleteList, frame, width=32, fg='orange', bg='black', insertbackground='orange')

symptom\_entry5.grid(row=5, column=1)

symptom\_entry5.config(font=('Consolas', 15, 'bold'))

# Predict Button

predict\_button = tk.Button(frame, text='Predict', command=prediction, bg='red', fg='white', activebackground='red')

predict\_button.config(font=('Consolas', '18', 'bold'))

predict\_button.grid(row=6, column=1, pady=(10, 50), padx=(0, 180))

# Clear Button

def clear\_entries():

    symptom\_entry1.delete(0, END)

    symptom\_entry2.delete(0, END)

    symptom\_entry3.delete(0, END)

    symptom\_entry4.delete(0, END)

    symptom\_entry5.delete(0, END)

    final\_result.delete(0, END)

clear\_button = tk.Button(frame, text='Clear', bg='red', fg='white', activebackground='red', command=clear\_entries)

clear\_button.config(font=('Consolas', '18', 'bold'))

clear\_button.grid(row=6, column=1, padx=(100, 0), pady=(10, 50), columnspan=2)

# Result Entry

final\_result = Entry(frame, width=50, borderwidth=0, bg='green', fg='white', justify=CENTER, insertbackground='green')

final\_result.grid(row=8, column=0, pady=(0, 20), padx=(60, 0), columnspan=2)

final\_result.config(font=('Consolas', 20, 'bold'))

final\_result.bind("<Key>", lambda e: "break")

tt = Label(frame, text='Note: Use at least 3 symptoms for better results')

tt.grid(row=11, column=0, columnspan=2, padx=(60, 0))

tt.config(font=('', 15, 'bold'))

root.mainloop()

**SYSTEM INTERACE:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**OUTPUT FOR DISEASE PREDICTION SYSTEM:**

A screenshot of a computer

Description automatically generated

**AFTER ENTERING ANY 3 SYMPTOMS THE PREDICTED DISEASE:**

A screenshot of a computer

Description automatically generated

**OUTPUT FOR OTHER MODELS:**

A screenshot of a computer

Description automatically generated

time-consuming for a Patient to know which disease they are suffering from.