



INTERNSHIP REPORT

DISEASE PREDICTION USING MACHINE LEARNING

SUBMITTED BY: JYOTI SHREE S GOTTI

KHYATHI M R

LAVANYA B M

PERIOD : 08/07/2024 – 08/08/2024

DEPARTMENT: BEL Software Technology Centre

UNDER THE GUIDANCE OF:

MURUGAN S

(MANAGER, BSTC)

BHARAT ELECTRONICS LIMITED

DATE OF SUBMISSION: 07/08/2024

ACKNOWLEDGEMENT

It is a great pleasure for us to acknowledge the assistance and support of many individuals who have been responsible for the successful completion of this internship. First, we take this opportunity to express our sincere gratitude to BHARAT ELECTRONICS LIMITED for providing us with the opportunity to undertake this internship.

We would like to express our deepest gratitude to MURUGAN S, our internship guide, for providing us with this valuable opportunity and for their constant guidance, encouragement, and support throughout the duration of our internship. Your insights and expertise have been instrumental in our learning process.

We are also profoundly grateful to Sri. R P Mohan, GM/HR, for granting permission for the internship, Mrs. Durga G K, ED/BSTC, for the opportunity to undertake the internship in the BSTC department, and Mrs. Sujatha Francis, Manager/CLD. We also thank Sri. Ravindra D, Manager/BSTC, for recommending our guide and coordinating the internship. Your support and coordination have been crucial to the success of this internship.

CONTENTS

SERIAL NO	CHAPTER	PAGE NO
1	Introduction	4
2	Objectives	5
3	Description of the company	6 - 8
4	Work Done	9 - 23
5	Skills developed & References	24 - 27
6	Conclusion	28

INTRODUCTION

In recent years, the integration of machine learning technologies to healthcare brought significant advancements in disease prediction and diagnosis. The ability to predict a disease in its early stages can drastically improve patient outcomes and reduce healthcare costs. This report explores the application of machine learning techniques, specifically using a Support vector Machine (SVM) model, for disease prediction.

SVM is a supervised learning algorithm known for its effectiveness in the classification tasks. It operates by finding the hyperplane that best separates different classes in the feature space. In this project, we utilized SVM to classify patient data and predict the likelihood of a specific disease based on various medical parameters.

To make this model accessible and user-friendly, we implemented a web-based interface using Tkinter, a library in python . The Flask API serves as a bridge between the machine learning model and the user interface, allowing users to input their medical data and receive predictions in real-time. The user interface, developed in Python, provides a simple and intuitive platform for users to interact with the system.

This report details the development process of the disease prediction system, including data preprocessing, model training and evaluation, and the integration of the SVM model with the Flask API and Python UI.

OBJECTIVES

1. **To develop a predictive model** : Design and develop a machine learning model that can accurately predict the likelihood of a patient having a specific disease based on their medical history, symptoms, and other relevant factors.
2. **To create a RESTful API** : Develop a RESTful API using Flask that exposes the predictive model's functionality and allows for easy integration with a Python UI.
3. **To design a user-friendly UI**: Design and develop a user-friendly Python UI that interacts with the Flask API to collect user input, display predictions, and provide visualizations of the results.
4. **To integrate the predictive model with the UI** : Integrate the predictive model with the Python UI using the Flask API, allowing users to input data and receive predictions in real-time.
5. **To evaluate the performance of the model** : Assess the performance of the predictive model using metrics such as accuracy, precision, recall, and F1-score, and compare it with existing models or techniques.

DESCRIPTION OF THE COMPANY

Bharat Electronics Limited (BEL) is a leading aerospace and defence company in India, renowned for its significant contributions to the country's defence sector. Established in 1954, BEL operates under the Ministry of Defence, Government of India, and has been pivotal in equipping the Indian armed forces with cutting-edge technology and state-of-the-art equipment.

History and Background

BEL was founded to meet the specialized electronics needs of the Indian defence services. The company's journey began with the manufacture of communication equipment. Over the decades, BEL has expanded its portfolio to include a wide array of products and systems, serving not only the defence sector but also civilian and industrial sectors. The company's headquarters is located in Bangalore, with multiple manufacturing units and regional offices across India.

2. Operations and Divisions

Key Divisions and Locations

BEL operates through a network of nine manufacturing units located in various parts of India, including Bangalore, Ghaziabad, Pune, Machilipatnam, Panchkula, Kotdwara, Navi Mumbai, Chennai, and Hyderabad. Each unit specializes in different product lines and services, ensuring comprehensive coverage of the company's diverse portfolio.

Major Products and Services

BEL's product range is vast, encompassing:

- Radar and Fire Control Systems: BEL produces a variety of radars and fire control systems for land, naval, and airborne applications.
- Communication Systems: These include HF/VHF/UHF radios, encrypted communication devices, and satellite communication equipment.
- Electronic Warfare and Avionics: BEL develops sophisticated electronic warfare systems and avionics for aircraft and helicopters.
- Naval Systems: This division caters to the specific needs of the Indian Navy, providing sonars, torpedoes, and other underwater systems.
- Missiles and Weapon Systems: BEL collaborates on the development and production of missile systems and associated technologies.
- C4I Systems: Command, Control, Communications, Computers, and Intelligence systems are critical for modern warfare, and BEL is a key provider of these integrated solutions.

3. Technological Advancements and Innovations

Research and Development

BEL places significant emphasis on research and development (R&D), investing heavily in cutting-edge technologies. The company has established several R&D centres that focus on developing indigenous solutions tailored to the unique requirements of the Indian defence forces.

Notable Projects and Collaborations

Over the years, BEL has been involved in several high-profile projects, often in collaboration with other leading defence organizations and multinational corporations. Notable projects include the Akash Missile System, Integrated Air Command and Control System (IACCS), and Battlefield Surveillance System (BSS).

4. Achievements and Awards

Milestones

BEL's journey is marked by numerous milestones that underscore its commitment to excellence and innovation. Key milestones include the development of the first indigenous radar, successful implementation of the IACCS, and the production of advanced electronic warfare systems.

Recognitions and Accolades

BEL's contributions have been widely recognized, earning the company several prestigious awards. These include the Defence Technology Absorption Award, SCOPE Meritorious Award for Corporate Governance, and multiple export awards for its contributions to the global defence market.

5. Corporate Social Responsibility (CSR)

Initiatives and Programs

BEL is committed to corporate social responsibility, undertaking various initiatives aimed at improving the quality of life in communities where it operates. The company's CSR programs focus on education, healthcare, rural development, and environmental sustainability. Noteworthy initiatives include setting up educational institutions, organizing health camps, and supporting rural infrastructure projects.

WORK DONE

1. Initial Research and Planning

The primary objective of this project was to develop a machine learning-based system capable of predicting diseases based on symptoms inputted by users in the user interface. This goal was driven by the need for a tool that could assist in early diagnosis and provide a preliminary analysis before consulting healthcare professionals.

To achieve this, we conducted extensive initial research to identify the most relevant diseases and symptoms to include in our study. We also explored existing methodologies and machine learning models commonly used for similar predictive tasks. This research phase was crucial in defining the scope of the project and establishing clear goals.

We began by reviewing research papers, online articles, and existing datasets in the Kaggle to understand the state-of-the-art in disease prediction. This helped us identify potential challenges, such as data quality issues and the need for accurate symptom-disease mappings. With these insights, we formulated a project plan that included data acquisition, model selection, implementation, and evaluation strategies.

2. Studying the machine learning models

Machine learning has revolutionized many fields, including healthcare. Disease prediction is a crucial application of machine learning, as it enables early diagnosis and timely intervention. In this report, we explore the application of machine learning to predict diseases based on symptoms. We evaluated three machine learning models: Naive Bayes, Support Vector Machine

(SVM), and Random Forest. After thorough analysis, we chose SVM for our final model. This report explains our methodology, compares the models, and provides insights into our decision to use SVM.

Machine Learning Models

Naive Bayes

Naive Bayes is a probabilistic classifier based on Bayes' theorem with strong (naive) independence assumptions between features. It is simple, efficient, and works well with high-dimensional data. However, it assumes that all features are independent, which is rarely true in real-world data. This assumption can sometimes limit its accuracy.

Support Vector Machine (SVM)

SVM is a powerful classifier that works by finding the hyperplane that best separates the data into different classes. It is effective in high-dimensional spaces and is versatile, as different kernel functions can be specified for the decision function. SVM tends to perform well with a clear margin of separation and is robust to overfitting, especially in high-dimensional space.

Random Forest

Random Forest is a learning method that operates by constructing multiple decision trees during training and outputting the class that is the mode of the classes of the individual trees. It handles large datasets with higher dimensionality well and reduces overfitting by averaging multiple decision trees. However, it can be computationally intensive and less interpretable compared to other models.

Comparison of Models

To select the best model, we compared Naive Bayes, SVM, and Random Forest based on the following criteria:

Accuracy: The ability of the model to correctly predict diseases based on symptoms.

Speed: The training and prediction speed of the model.

Scalability: The model's ability to handle large datasets.

Interpretability: How easily the model's predictions can be understood.

Results

Criteria	Naive Bayes	SVM	Random Forest
Accuracy	Moderate	High	High
Speed	Fast	Moderate	Slow
Scalability	High	High	Moderate
Interpretability	Moderate Based on these criteria, we chose	High	Low

SVM as our final model due to its high accuracy, scalability, and interpretability. While Random Forest also showed high accuracy, it was less interpretable and slower compared to SVM. Naive Bayes, despite its speed and simplicity, did not perform as well in terms of accuracy.

Implementation of SVM Model

Here is the code for our SVM model, along with explanations of the functions used:

```
class SVMModel:
    def __init__(self):
        self.vectorizer = TfidfVectorizer(stop_words='english')
        self.svm_model = svm.SVC(kernel='linear', C=1.0, random_state=42)
        self.label_encoder = LabelEncoder()
        self.stop_words = ['is', 'are', 'am', 'be', 'been', 'being', 'have', 'has', 'had', 'do',

    def fit(self, X, y):
        X_preprocessed = self.vectorizer.fit_transform(X)
        y_encoded = self.label_encoder.fit_transform(y)
        self.svm_model.fit(X_preprocessed, y_encoded)

    def predict_disease(self, input_symptoms):
        input_vector = self.preprocess_symptoms(input_symptoms)
        prediction = self.svm_model.predict(input_vector)
        return self.get_disease_from_prediction(prediction)

    def get_disease_from_prediction(self, prediction):
        return self.label_encoder.inverse_transform(prediction)[0]
```

```

def save_model(self, filename='svm_model.pkl'):
    """
    Save the trained model and associated objects to a pickle file.
    """
    with open(filename, 'wb') as file:
        pickle.dump({
            'model': self.svm_model,
            'vectorizer': self.vectorizer,
            'label_encoder': self.label_encoder
        }, file)

def run(self):

    df = pd.read_csv('dataset.csv')

    x = df['text']
    y = df['label']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    self.fit(X_train, y_train)
    self.save_model()
    input_symptoms = input("Enter your symptoms: ")
    predicted_disease = self.predict_disease(input_symptoms)

    print("Predicted disease:", predicted_disease)

```

Predict_disease: Predicts the disease based on new input symptoms. It preprocesses the symptoms and uses the trained SVM model to make a prediction.

Get_disease_from_prediction: Converts the numerical prediction back to the corresponding disease name.

Preprocess_symptoms: Preprocesses the input symptoms by tokenizing, removing stop words, and creating a bag-of-words representation.

`train_test_split` is a utility function from the `sklearn.model_selection` module. It splits the dataset into two parts: one for training the model and one for testing it. This split helps in evaluating the model's performance on unseen data, which is essential for assessing its generalization ability.

Example Vector Table:

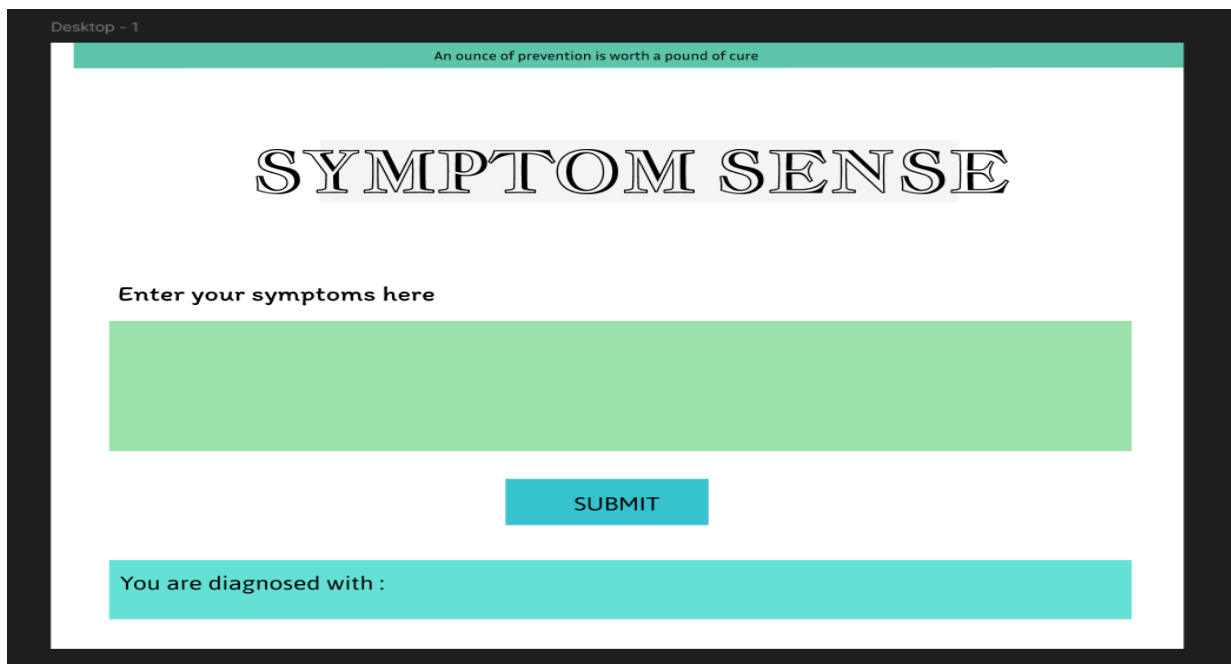
To illustrate the preprocessing step, let's consider an example set of symptoms: nausea, cold, and vomiting.

Symptom	Tokens	After Stop Words Removal	Tfidf Vector
nausea	[nausea]	[nausea]	[0.0, 1.0, 0.0]
cold	[cold]	[cold]	[0.0, 0.0, 1.0]
vomiting	[vomiting]	[vomiting]	[1.0, 0.0, 0.0]

3. Prototype using Figma

Figma is a web-based design tool used primarily for interface design, prototyping, and collaborative work. It allows designers to create vector graphics and user interfaces for websites, apps and other digital products.

The below snapshot shows the design of our interface-



- Figma's real-time collaboration features allow multiple team members to work on the same design simultaneously. Designers, developers, and stakeholders can provide immediate feedback, leading to a more efficient and cohesive design process.
- Designing in Figma allows for rapid prototyping and easy iteration. Changes can be made quickly and visualized instantly, enabling the team to refine the design based on feedback without the need for extensive coding adjustments.
- Creating a detailed design in Figma before coding helps ensure that all team members have a clear understanding of the project requirements and visual expectations. This reduces the risk of miscommunication and ensures that developers have a precise blueprint to follow.

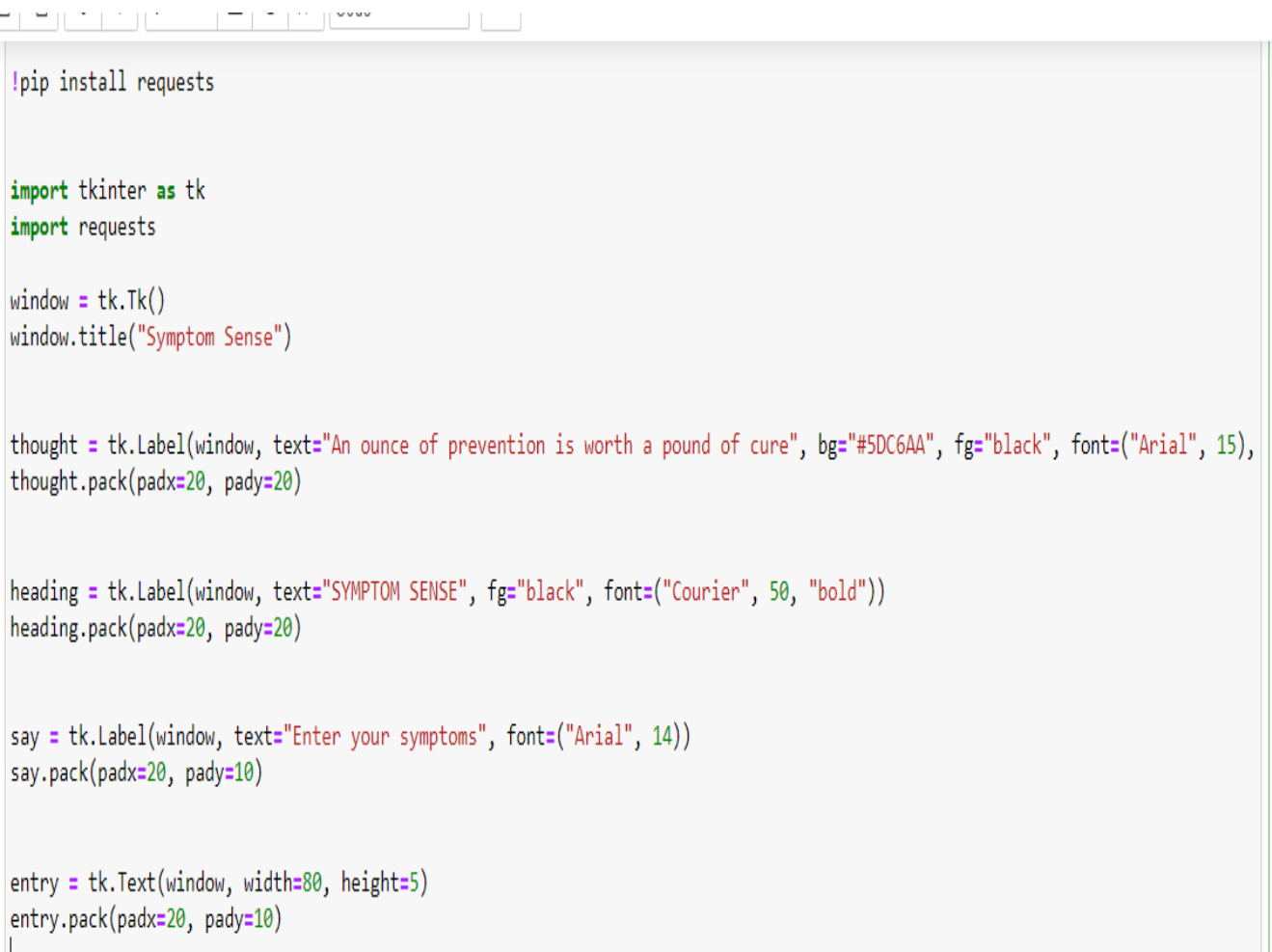
4. CREATING THE USER INTERFACE:

In our study of Python basics, we explored foundational concepts crucial for any beginner programmer. We revised with understanding variables and data types, including integers, floats, strings, and more complex structures like lists and

dictionaries. These elements are the building blocks of any Python program, allowing us to store and manipulate data efficiently.

A key aspect of our learning involved functions, which allow us to encapsulate reusable code blocks. By defining functions, we can simplify complex problems into smaller, manageable tasks, promoting better code organization and readability. We also touched upon the use of modules and libraries, which expand Python's capabilities by providing additional functionalities.

The below code is a snapshot of the user interface:



```
!pip install requests

import tkinter as tk
import requests

window = tk.Tk()
window.title("Symptom Sense")

thought = tk.Label(window, text="An ounce of prevention is worth a pound of cure", bg="#5DC6AA", fg="black", font=("Arial", 15),
thought.pack(padx=20, pady=20)

heading = tk.Label(window, text="SYMPTOM SENSE", fg="black", font=("Courier", 50, "bold"))
heading.pack(padx=20, pady=20)

say = tk.Label(window, text="Enter your symptoms", font=("Arial", 14))
say.pack(padx=20, pady=10)

entry = tk.Text(window, width=80, height=5)
entry.pack(padx=20, pady=10)
```



```

entry.pack(padx=20, pady=10)

def clicked():
    user_input = entry.get("1.0", "end-1c").strip()
    if not user_input:
        result.config(text="Please enter your symptoms.", fg="red")
        return

    symptoms = {'symptoms': user_input}

    try:
        response = requests.post('http://127.0.0.1:5000/predict', data=symptoms)
        response.raise_for_status()

        data = response.json()
        if 'disease' in data:
            result.config(text=f"You are diagnosed with: {data['disease']}", fg="black")
        else:
            result.config(text="Diagnosis not found. Please try again.", fg="red")
    except requests.exceptions.RequestException as e:
        result.config(text=f"Network error: {e}", fg="red")
    except ValueError:
        result.config(text="Invalid response from server.", fg="red")

submit = tk.Button(window, text="SUBMIT", font=("Helvetica", 10), bg="light blue", command=clicked)
submit.pack(padx=20, pady=10)

result = tk.Label(window, text="", bg="yellow", fg="black", width=40, height=10, wraplength=300)
result.pack(padx=40, pady=40)

window.mainloop()

```

This Python code is a simple GUI application called "Symptom Sense," designed to allow users to input their symptoms and receive a potential diagnosis. It uses the Tkinter library to create the graphical user interface and the requests library to send HTTP requests to a server for processing the symptoms. The program's design includes a main window with a thoughtful quote, a heading, input fields, and a submit button, creating an intuitive and user-friendly interface.

The application begins by importing the necessary modules: Tkinter for the GUI and requests for making HTTP requests. The main window is initialized with a title "Symptom Sense," setting the stage for the application. Various widgets are added to the

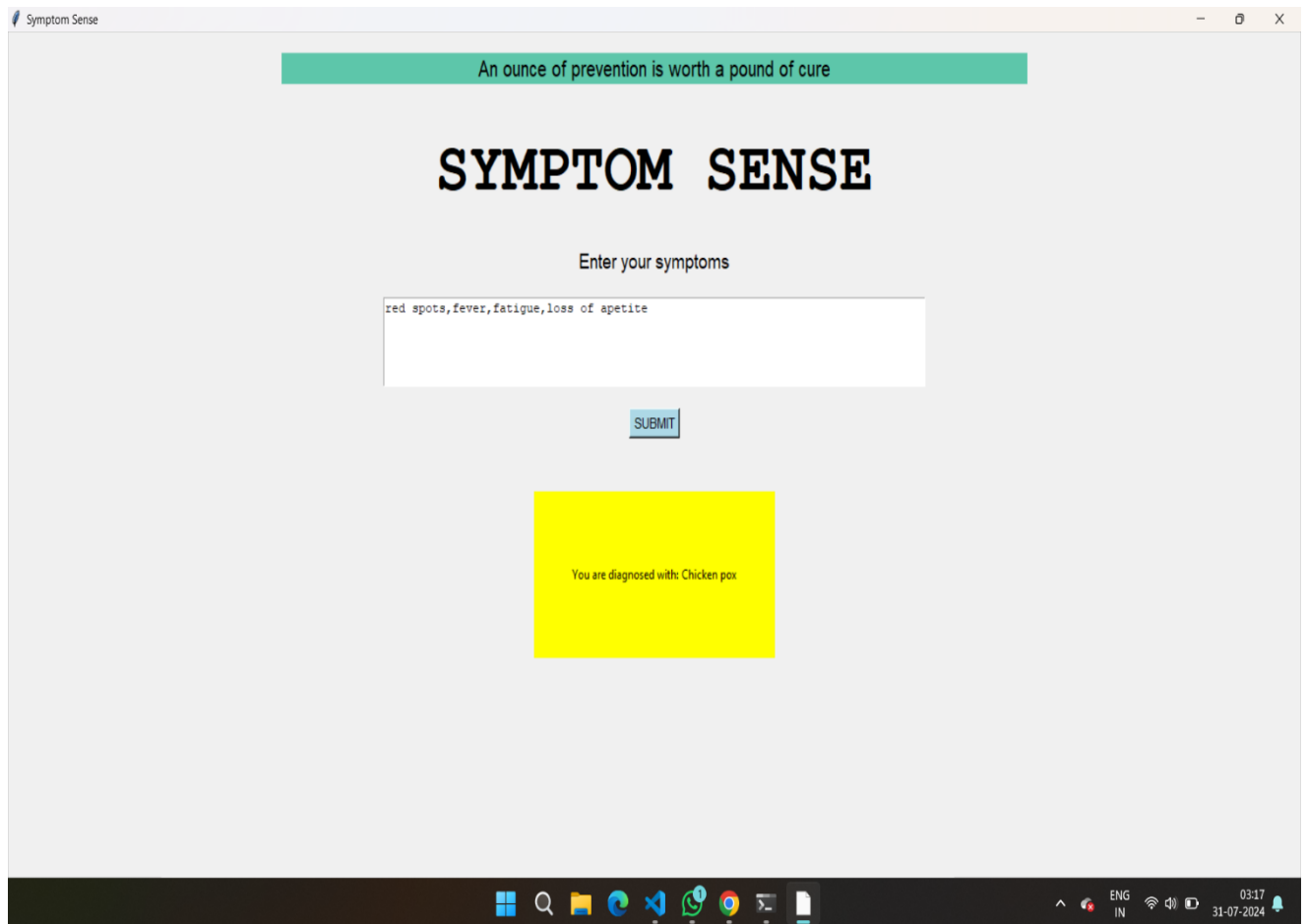
window, including a label displaying a preventive health quote, a bold title label, and another label instructing users to enter their symptoms. These labels help guide the user experience, making the interface clear and easy to navigate.

Users can enter their symptoms into a multi-line text box, and upon clicking the "SUBMIT" button, the `clicked()` function is triggered. This function first retrieves the user's input and checks if it is not empty. If no symptoms are entered, it prompts the user to provide input. The function then prepares the symptoms data for a POST request to a local server endpoint (<http://127.0.0.1:5000/predict>). The server processes this data and returns a response, which the application uses to display the diagnosis or an appropriate error message.

Error handling is a crucial part of this code. The `clicked()` function includes mechanisms to manage potential issues such as network errors or invalid responses from the server. For instance, if the POST request fails due to a network issue, an error message is displayed to the user, helping them understand that the problem lies with connectivity rather than their input. Similarly, if the server returns an unexpected response, the application informs the user that the diagnosis was not found, ensuring a smooth user experience even when things go wrong.

In summary, this code provides a practical example of integrating GUI elements with backend communication in Python. It demonstrates the use of Tkinter for creating a simple yet effective user interface and requests for server communication. By allowing users to input symptoms and receive a diagnosis, the application shows a real-world application of Python programming basics, highlighting the importance of user input handling, server interaction, and error management.

The below snapshot is our user interface(UI)-



5. Making the Rest API:

A REST API (Representational State Transfer Application Programming Interface) is a set of rules and conventions for building and interacting with web services. It relies on stateless, client-server, cacheable communication protocols, typically HTTP.

Firstly, we learned what Flask is, how it works, and where it is used. Then we got to know about HTTP requests and their types, as well as how an API connects our user interface to an SVM model.

Then we started creating the API in Python using Jupyter Notebook. You can see the Flask code below:

```
In [ ]:
from flask import Flask, request, jsonify
from flask_ngrok import run_with_ngrok
from svm_model import SVMModel
import subprocess
import threading

app = Flask(__name__)
run_with_ngrok(app)

svm = SVMModel()
svm.run()

@app.route('/predict', methods=['POST'])
def predict():
    symptoms = request.form.get('symptoms')

    if not symptoms:
        return jsonify({"error": "No symptoms provided"}), 400

    prediction = svm.predict_disease(symptoms)

    return jsonify({"disease": prediction})

def run_app():
    app.run()

def run_jupyter_notebook():
    subprocess.run(["jupyter", "notebook", "frontend/ui.ipynb", "--port=8888", "--no-browser", "--NotebookApp.token=''"])

if __name__ == '__main__':
    flask_thread = threading.Thread(target=run_app)
    flask_thread.daemon = True
    flask_thread.start()

    run_jupyter_notebook()
```

The provided code sets up a web service using Flask to deploy an SVM model for predicting diseases based on user-inputted symptoms. It begins by importing necessary libraries: Flask for the web framework, run_with_ngrok for exposing the local server on the internet, and SVMModel for handling the machine learning model. Additionally, subprocess and threading are

imported to manage external processes and enable concurrent execution, respectively.

An instance of the Flask app is created, and Ngrok is configured to make the app accessible online. The SVMModel instance (svm) is then initialized and trained using the svm.run() method, which prepares the model for making predictions based on the trained data.

The /predict endpoint is defined to handle POST requests. The predict() function within this endpoint retrieves symptoms from the incoming request. It checks if the symptoms are provided, and if not, it returns an error response. When symptoms are provided, the function uses the SVM model to predict the corresponding disease and returns the result in a JSON format.

To run the application, the run_app() function starts the Flask server, while run_jupyter_notebook() launches a Jupyter Notebook, potentially for an interactive development interface. The main block, guarded by if __name__ == "__main__":, ensures that the Flask app runs in a separate thread, allowing the Jupyter Notebook to operate concurrently. This dual setup is especially beneficial in a development environment, as it allows for both the web service and the Jupyter Notebook to be used simultaneously for testing and debugging purposes.

REACT INTERFACE

In addition to using Python Tkinter for the user interface, we also explored React for the frontend while maintaining Flask as the backend API in our disease prediction project. This provided valuable experience in modern web development, enhancing

both the interactivity and user experience of our application. Below is an explanation of the React code used to build the UI.

Purpose: Updates the input data state with the user's input.

```
function Form() {  
  const [inputData, setInputData] = useState({ symptoms: "" });  
  const [result, setResult] = useState("");  
  
  function handleData(e) {  
    setInputData({ ...inputData, [e.target.name]: e.target.value });  
    console.log(inputData);  
  }  
}
```

Parameters: Takes an event object **e** as a parameter.

Functionality:

- Uses the spread operator **...** to copy the existing state.
- Updates the specific property (in this case, symptoms) that the user is modifying based on the name attribute of the input field.
- Logs the current state of inputData to the console for debugging purposes.

```
async function handleSubmit(e) {  
  e.preventDefault();  
  try {  
    const response = await axios.post('/predict', inputData);  
    setResult(response.data.disease);  
  } catch (error) {  
    console.error("There was an error making the request", error);  
    setResult("Error occurred during prediction");  
  }  
}
```

- Purpose: Handles the form submission and sends the input data to the backend server for prediction.
- Parameters: Takes an event object **e** as a parameter.
- Functionality:

- Calls `e.preventDefault()` to prevent the default form submission behavior, which would reload the page.
- Uses `axios.post` to send a POST request to the `/predict` endpoint with `inputData` as the payload.
- Waits for the server's response using `await`.
- If the request is successful, updates the result state with the predicted disease from the server response.
- If an error occurs, logs the error to the console and updates the result state with an error message.

These functions enable the component to manage user input dynamically, send the data to a backend server, and handle the response effectively.

FORM CREATION

```
return (
  <div id="container">
    <h1>SYMPTOM SENSE</h1>
    <form id="symptomForm" onSubmit={handleSubmit}>
      <label htmlFor="symptom">Enter your symptoms:</label>
      <div>
        <input
          type="text"
          name="symptoms"
          value={inputData.symptoms}
          onChange={handleData}
        />
      </div>
      <button type="submit">Predict Disease</button>
    </form>
    <div id="result">{result}</div>
  </div>
);
```

SKILL DEVELOPED

1. Machine Learning:

- Preprocess and Clean Data: Ensuring the data is in the right format for training, handling missing values, and scaling features.
- Feature Selection and Engineering: Identifying and creating relevant features that improve the predictive power of the model.
- Model Training and Evaluation: Training various machine learning models such as Naïve Bayes, support vector machine, and random forests. I also learned to evaluate these models using metrics like accuracy, precision, recall, and F1 score to determine their effectiveness.

2. Programming with Python:

- Pandas and NumPy: For data manipulation and numerical operations.
- Scikit-Learn: For implementing machine learning algorithms and model evaluation.
- matplotlib : For data visualization, helping to understand data distributions and relationships.

3. Backend Development and API Integration

Flask Framework:

- Using Flask to connect the machine learning model with the UI was a critical part of the project. This experience enhanced my understanding of web development and API integration:

- API Development: we learned to create RESTful APIs that allow the frontend to communicate with the machine learning model. This included setting up routes, handling HTTP requests, and sending responses.
- Backend Logic: Implementing the backend logic to process incoming data, run the machine learning model, and return predictions.
- Security and Validation: Ensuring data validation and implementing basic security measures to protect the API.

4. React UI

During the exploration and implementation of the React UI with Flask API for the frontend of our disease prediction project, we developed several key skills in modern web development. These skills are crucial for building dynamic, responsive, and user-friendly web applications. Below is a detailed outline of the skills we acquired:

React and JavaScript

Component-Based Architecture:

- Learned to design and implement reusable components, promoting modularity and maintainability in the application.
- Gained proficiency in managing component state and props to control data flow and interactivity within the application.

5. Soft skills

Problem-Solving:

Working on this project required continuous problem-solving, from debugging code to optimizing model performance. I developed the ability to approach problems methodically, breaking them down into smaller, manageable tasks.

Collaboration and Communication:

The project was a team effort, and collaboration was key. I improved my ability to communicate effectively with team members, share ideas, and provide constructive feedback. Regular meetings and updates helped ensure everyone was aligned and contributed to the project's success.

Time Management:

Balancing multiple aspects of the project, such as model development, UI design, and backend integration, taught me to manage my time efficiently. Prioritizing tasks and meeting deadlines were essential skills I developed during this period.

REFERENCES

Machine Learning

<https://www.geeksforgeeks.org/machine-learning>

<https://youtu.be/Gwlo3gDZCVQ?si=WHBTkDOtdmGA6q-E>

<https://developers.google.com/machine-learning/crash-course>

API

<https://flask.palletsprojects.com/en/3.0.x/>

<https://www.geeksforgeeks.org/python-build-a-rest-api-using-flask/>

React

<https://youtu.be/-mJFZp84TIY?si=qjTeG23maGgUN7Az>

<https://react.dev/learn>

Tkinter

<https://youtu.be/VMP1oQOxfM0?si=Q4rKKjPhpzPbsAFo>

CONCLUSION

The internship project on "Disease Prediction Using Machine Learning using REST API" was a valuable and enriching experience that combined theoretical knowledge with practical application. By developing an end-to-end system for predicting diseases, we not only deepened our understanding of machine learning techniques and their practical implications but also gained hands-on experience in software development, data processing, and user interface design.

Looking forward, we aim to build on this experience by exploring additional machine learning models and techniques, improving the user interface, and considering the scalability of the application. This internship has been a significant stepping stone in our careers, providing a solid foundation for future endeavours in data science, software development, and beyond.

We extend our gratitude to our mentor, who supported us throughout this internship. His guidance and feedback was valuable in helping us achieve our goals and deliver a successful project. We are excited to continue our journey in this field and contribute to meaningful innovations that can positively impact society.