**Automated Disease Prediction Using Machine Learning Ensembles**

**Phase 4: Model Deployment and Interface Development**

**4.1 Overview the Model Deployment and Interface Development [Vasavi C Kulkarni]**

Phase 4 is about launching the already trained models for predicting diseases in a live environment and creating an interactive user interface for the system's end-users. The main goal is to have the models exposed through APIs for real-time forecasts and to have a user-friendly way to interact with the system for both healthcare professionals and individuals. This stage avails cloud platforms for the API and web frameworks for the interface development ensuring scaling, security, and usability.

**4.2 Deploying the Model [Sumeet U Pattan]**

In implementing the then trained ensemble models (Random Forest, Gradient Boosting, and Logistic Regression) the first steps were performed:

**1. Model Export:**

A pickle module is used to deploy the trained machine learning models by saving and exporting them.

**Source Code:**

**Python**

import pickle

# Save Random Forest Model

with open('models/random\_forest.pkl', 'wb') as rf\_file:

pickle.dump(rf\_model, rf\_file)

# Save Gradient Boosting Model

with open('models/gradient\_boosting.pkl', 'wb') as gb\_file:

pickle.dump(gb\_model, gb\_file)

# Save Logistic Regression Model

with open('models/logistic\_regression.pkl', 'wb') as lr\_file:

pickle.dump(lr\_model, lr\_file)

**2. Developing an API for the Models:**

Flask was used to create a RESTful API to the trained models in order to carry out predictions on users' data during the production mode. The API takes in input data as JSON, comprises pre-fitted models, and gives the expected risk level back after processing the data.

**Source Code:**

**Python**

from flask import Flask, request, jsonify

import pickle

import numpy as np

# Load models

with open('models/random\_forest.pkl', 'rb') as rf\_file:

rf\_model = pickle.load(rf\_file)

with open('models/gradient\_boosting.pkl', 'rb') as gb\_file:

gb\_model = pickle.load(gb\_file)

with open('models/logistic\_regression.pkl', 'rb') as lr\_file:

lr\_model = pickle.load(lr\_file)

app = Flask(\_name\_)

# Define API endpoint for predictions

@app.route('/predict', methods=['POST'])

def predict():

data = request.get\_json()

input\_data = np.array(data['input'])

# Generate predictions from each model

rf\_prediction = rf\_model.predict(input\_data)[0]

gb\_prediction = gb\_model.predict(input\_data)[0]

lr\_prediction = lr\_model.predict(input\_data)[0]

# Majority voting

predictions = [rf\_prediction, gb\_prediction, lr\_prediction]

final\_prediction = max(set(predictions), key=predictions.count)

# Map prediction to risk level

risk\_map = {0: "Low risk", 1: "High risk"}

return jsonify({'risk\_level': risk\_map[final\_prediction]})

if \_name\_ == '\_main\_':

app.run(debug=True)

**3. Deploying the API on Cloud:**

The Flask API was containerized and deployed on the cloud platform to make it both scalable and available. The options that could be used for deployment were as follows:

**AWS Lambda:** The API was hosted as a serverless function along with AWS API Gateway for the exposure of the endpoint.

**Google Cloud Functions:** The Flask app was deployed with Google App Engine to get more power and flexibility of the deployment.

**Azure Functions:** The models were deployed as serverless functions with Azure API Management managing access and the way traffic is moved through.

**4.3 Developing the Web Interface [Radha]**

To give the users the possibility to work with the deployed models, a simple web interface was created. The interface gets the input parameters, puts them to the API, and presumes the risk level of the result. Frameworks such as Streamlit and Flask were used to define the web interface.

**Using Streamlit:** Streamlit was decided to be the best tool to establish a light and interactive interface for the healthcare providers and individuals where the former can give their data and the latter can view the predictions.

**Source Code:**

**Python**

import streamlit as st

import requests

import numpy as np

# Set the app title

st.title("Disease Risk Prediction")

# Input fields

name = st.text\_input("Enter Name")

age = st.number\_input("Enter Age", min\_value=1, max\_value=120, value=30)

gender = st.selectbox("Select Gender", options=["Male", "Female"])

bmi = st.number\_input("Enter BMI", min\_value=10.0, max\_value=50.0, value=25.0)

glucose = st.number\_input("Enter Glucose Level", min\_value=50, max\_value=200, value=100)

# Button to predict

if st.button("Predict Risk"):

input\_data = [[age, bmi, glucose]]

response = requests.post("http://127.0.0.1:5000/predict", json={"input": input\_data})

result = response.json()

st.write(f"Prediction: {result['risk\_level']}")

**Using React (Optional):** If the user needs a dynamic and feature-rich interface, React will be used to develop a single-page application. Meanwhile the application sends the input to the Flask API and shows the response in real-time.

**4.4 Cloud Platform Considerations [Shreya Kulkarni]**

When the system was deployed to cloud platforms, a number of factors were taken care of:

**Scalability:** The use of the auto-scaling feature (both on AWS, Google Cloud, and Azure), the system was able to handle an increased amount of traffic and store large volumes of data.

**Security:** API keys and OAuth authentication enabled to make the API access really secure. In this way, only the authorized users could interact with the model.

**Monitoring:** Several tools like AWS CloudWatch and Google Stackdriver were added to the API system to follow its performance in real-time and to identify any issues that may appear.

**Cost Management:** The cloud platforms’ usage-based pricing models were customized by means of the decrease of the API calls that were not necessary and the optimization of the serverless functions.

**4.5 Conclusion of Phase 4 [Vasavi C Kulkarni]**

In Phase 4, the ensemble models for the prediction of disease were deployed to a cloud platform, and then they were exposed by a RESTful API and connected with an interactive web interface. Consequently, this development permitted a faster accessibility to healthcare providers and communities. The success of the project is the combination of the cloud platform, such as the user-friendly web frameworks, which allowed it to move out of the development environment and to be real-world use, hence it gives the end-users the reliable predictions for early disease-risk detection.