EXPERIMENT -6

Aim:

To train a machine learning model for diabetes prediction and integrate it with a user interface using the Flask framework for real-time prediction.

Objective:

- To develop a predictive model using machine learning for health-related applications.
- To understand the process of integrating ML models with web interfaces.
- To build an end-to-end application for disease prediction using Python and Flask.

Theory:

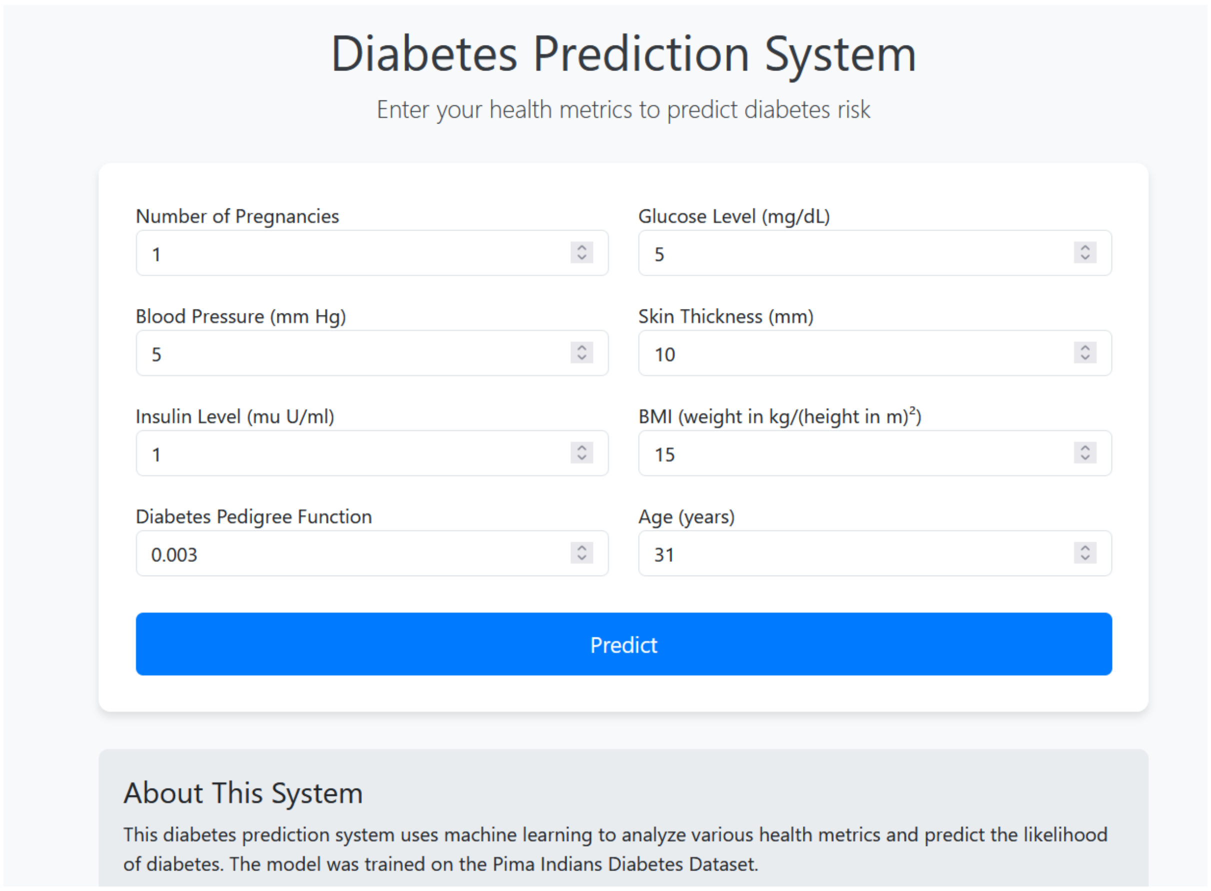
- **Diabetes Prediction:** A binary classification problem where the model predicts whether a person has diabetes based on health-related parameters such as glucose level, BMI, age, blood pressure, etc.
- **ML Algorithm:** Common algorithms used include:
    - Logistic Regression
    - Decision Trees
    - Random Forest
    - Support Vector Machines
- **Dataset:** The PIMA Indians Diabetes Dataset is often used, containing features like glucose level, BMI, age, pregnancies, insulin level, etc.
- **Flask Framework:** A lightweight Python web framework used to:
    - Build RESTful APIs
    - Serve the ML model through a backend
    - Render HTML templates for UI
    - Accept user inputs and return predictions dynamically
- **Workflow:**

1. Load and preprocess the dataset.
2. Train a classification model (e.g., Logistic Regression).
3. Save the trained model using pickle or joblib.
4. Use Flask to create endpoints and forms for user interaction.
5. Display prediction results through the integrated UI.

Code:

```
import numpy as np

import pandas as pd

import pickle

import os

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier


def train_model():

   df = pd.DataFrame(data)

   X = df.drop('Outcome', axis=1)

   y = df['Outcome']

   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

   scaler = StandardScaler()

   X_train_scaled = scaler.fit_transform(X_train)

   X_test_scaled = scaler.transform(X_test)

   clf = RandomForestClassifier(n_estimators=100, random_state=42)

   clf.fit(X_train_scaled, y_train)

   with open('models/diabetes_model.pkl', 'wb') as model_file:

      pickle.dump(clf, model_file)

   with open('models/scaler.pkl', 'wb') as scaler_file:

      pickle.dump(scaler, scaler_file)

   print("Model and scaler saved successfully!"

if __name__ == "__main__":

   if not os.path.exists('models'):

      os.makedirs('models')

   train_model()
```

```
from flask import Flask, jsonify, request

import numpy as np

import pickle

import os

app = Flask(__name__)

# Load the model and scaler

model_path = os.path.join('models', 'diabetes_model.pkl')

scaler_path = os.path.join('models', 'scaler.pkl')

# Check if model files exist, if not train them

if not (os.path.exists(model_path) and os.path.exists(scaler_path)):

    print("Model files not found. Training new model...")

    from train_model import train_model

    model, scaler = train_model()

else:

    print("Loading existing model and scaler...")

    with open(model_path, 'rb') as model_file:

        model = pickle.load(model_file)

    with open(scaler_path, 'rb') as scaler_file:

        scaler = pickle.load(scaler_file)

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

def predict_api():

    """API endpoint for programmatic access"""

    try:

        data = request.get_json()

        features = [

            float(data['pregnancies']),

            float(data['glucose']),

            float(data['bloodpressure']),

            float(data['skinthickness']),

            float(data['insulin']),

            float(data['bmi']),

            float(data['dpf']),

            float(data['age'])

        ]

        features_array = np.array(features).reshape(1, -1)

        features_scaled = scaler.transform(features_array)
    prediction = model.predict(features_scaled)[0]

    prediction_proba = model.predict_proba(features_scaled)[0][1]

    return jsonify({

      'prediction': int(prediction),

      'probability': float(prediction_proba),

      'message': 'Diabetes detected' if prediction == 1 else 'No diabetes detected'

    })

  except Exception as e:

    return jsonify({'error': str(e)})

if __name__ == '__main__':

  app.run(debug=True)
```



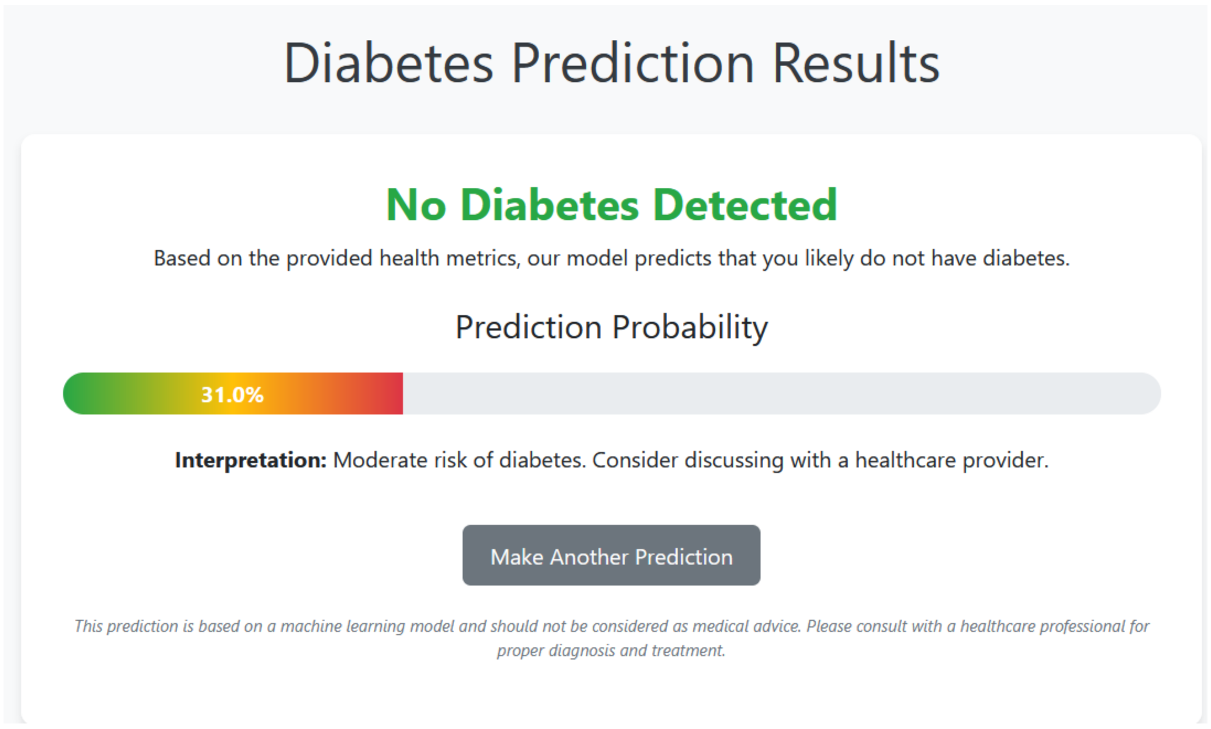
Diabetes Prediction System

Enter your health metrics to predict diabetes risk

Number of Pregnancies: 1	Glucose Level (mg/dL): 5
Blood Pressure (mm Hg): 5	Skin Thickness (mm): 10
Insulin Level (mu U/ml): 1	BMI (weight in kg/(height in m)^2): 15
Diabetes Pedigree Function: 0.003	Age (years): 31

About This System

This diabetes prediction system uses machine learning to analyze various health metrics and predict the likelihood of diabetes. The model was trained on the Pima Indians Diabetes Dataset.



Diabetes Prediction Results

No Diabetes Detected

Based on the provided health metrics, our model predicts that you likely do not have diabetes.

Prediction Probability

Interpretation: Moderate risk of diabetes. Consider discussing with a healthcare provider.

This prediction is based on a machine learning model and should not be considered as medical advice. Please consult with a healthcare professional for proper diagnosis and treatment.

Conclusion:

The experiment successfully demonstrates how a machine learning model for diabetes prediction can be trained and deployed with a user-friendly web interface using Flask. This end-to-end integration enhances students' understanding of practical machine learning applications in healthcare and the importance of UI/UX in real-world systems.