

Experiment-2

(Elements of AI/ML)



Flask App for Diabetes Prediction

Name: Lakshay Dahiya
SAP ID: 500121078
Roll No.: R2142230813
Batch:12

Brief Description:

This Flask-based web application is designed to *predict the likelihood of diabetes in individuals based on their health data*. The app uses a machine learning model, trained on real-world diabetes-related datasets, to make predictions. It allows users to input key health parameters such as glucose levels, BMI, age, and others, and then predicts whether they are at risk for diabetes. The app employs the `scaler` for normalizing the input data and `diabetes_best_model.pkl` for making accurate predictions.

The user-friendly interface built with **HTML** and **CSS** allows users to easily interact with the model and view the results. This application leverages machine learning techniques and Flask to create a simple yet powerful tool for healthcare professionals or anyone interested in assessing diabetes risk based on personal health data.

The backend is structured with **Python**, making use of various libraries such as `joblib` for model serialization, `Flask` for the web framework, and `sklearn` for data preprocessing and model evaluation.

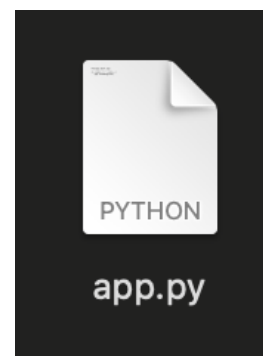
This project is ideal for showcasing the power of machine learning in healthcare applications, specifically for preventive measures related to diabetes.

Working of Flask App

Working of the Flask Diabetes Prediction Web App:

1. Starting the Application:

- To start the Flask web application, you first need to activate the Python virtual environment and install the necessary dependencies.
- Once the dependencies are installed, run the Flask app by executing `python app.py`

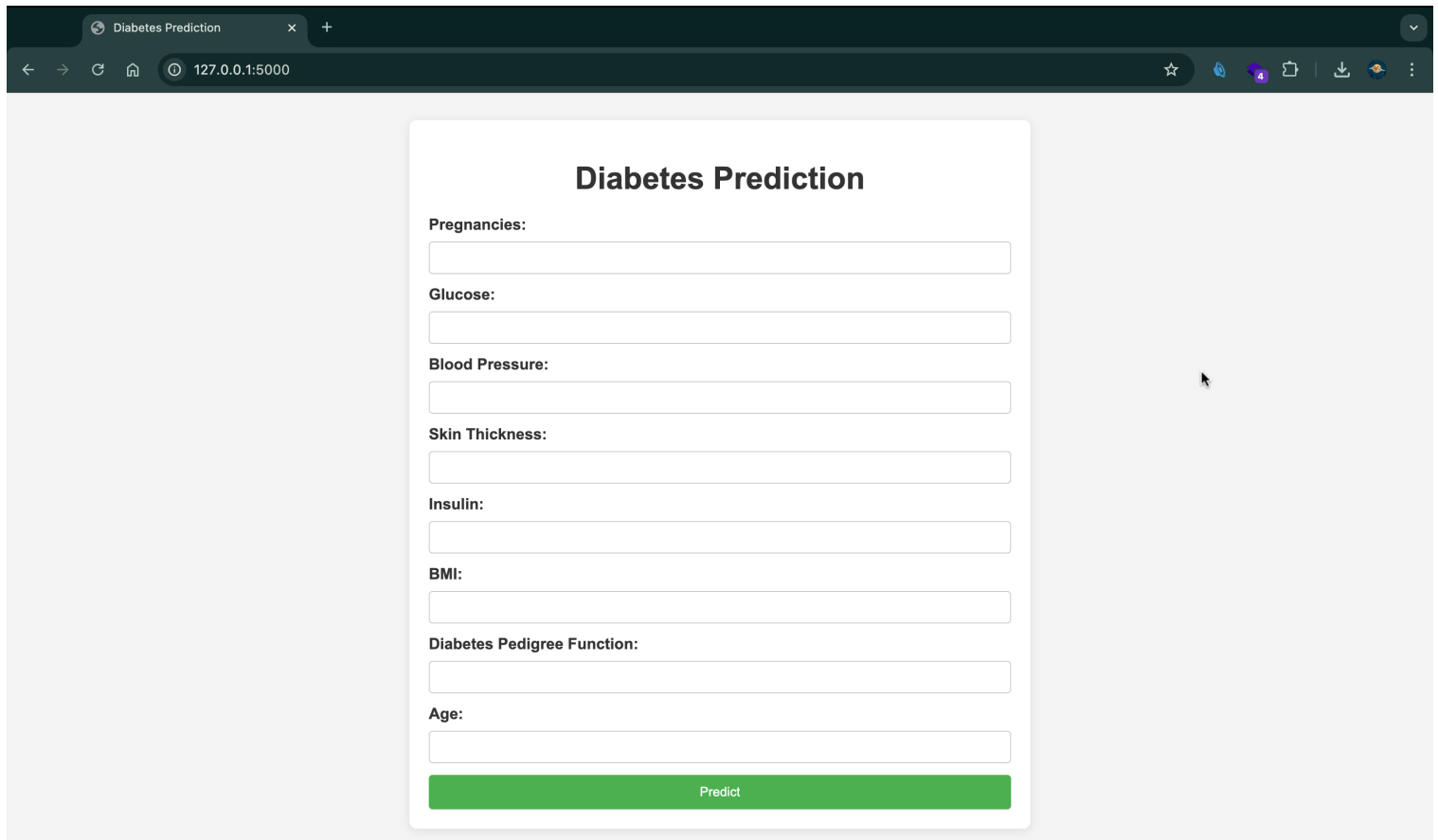


```
lakshaydahiya@Lakshays-MacBook-Air Desktop % cd assignment2
lakshaydahiya@Lakshays-MacBook-Air assignment2 % python3 -m venv venv
(venv) lakshaydahiya@Lakshays-MacBook-Air assignment2 % pip install -r requirements.txt
```

```
(venv) lakshaydahiya@Lakshays-MacBook-Air assignment2 % python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production
WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 910-441-034
```

2. Web Interface (Frontend):

- The user interacts with the app via a simple web interface hosted on **`http://127.0.0.1:5000`**.
- The interface is created using **HTML** (in `index.html`) and styled with **CSS** (in `style.css`).
- The frontend consists of an input form where the user can provide values for the following health parameters:
 - Pregnancies
 - Glucose
 - BloodPressure
 - SkinThickness
 - Insulin
 - BMI (Body Mass Index)
 - DiabetesPedigreeFunction
 - Age



The screenshot shows a web browser window with the title "Diabetes Prediction". The address bar displays "127.0.0.1:5000". The main content area features a white card with the title "Diabetes Prediction" in bold. Below the title, there are eight input fields, each preceded by a label: "Pregnancies:", "Glucose:", "Blood Pressure:", "Skin Thickness:", "Insulin:", "BMI:", "Diabetes Pedigree Function:", and "Age:". At the bottom of the card is a green button labeled "Predict".

Parameter	Input Field
Pregnancies	<input type="text"/>
Glucose	<input type="text"/>
Blood Pressure	<input type="text"/>
Skin Thickness	<input type="text"/>
Insulin	<input type="text"/>
BMI	<input type="text"/>
Diabetes Pedigree Function	<input type="text"/>
Age	<input type="text"/>

3. Data Input & Validation:

- The user enters data for each of the parameters.
- The input values for **BMI**, **Insulin**, and **DiabetesPedigreeFunction** are accepted as decimal values (up to 3 decimal places).
- The input for **Outcome** is also accepted as either 0 (negative) or 1 (positive) (this could be useful if testing a pre-existing dataset).

We take two examples from the dataset diabetes.csv itself to test the validity of the model.

diabetes								
Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0

Now, we test example 1st:

Diabetes Prediction

Pregnancies:

Glucose:

Blood Pressure:

Skin Thickness:

Insulin:

BMI:

Diabetes Pedigree Function:

Age:

Its gives the desired output:

Prediction: Diabetes

Now, we take example 2nd:

Diabetes Prediction

Pregnancies:

Glucose:

Blood Pressure:

Skin Thickness:

Insulin:

BMI:

Diabetes Pedigree Function:

Age:

Predict

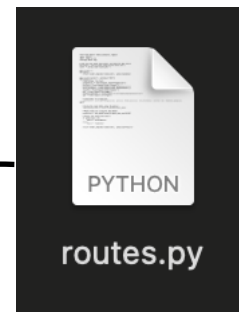
It also gives the desired output:

Predict

Prediction: No Diabetes

4. Sending Data to Backend:

- Once the user submits the form, the data is sent to the backend using a POST request.
- The form data is captured in the `routes.py` file and passed for processing.



```
from flask import render_template, request
import joblib
import numpy as np
from app import app

# Load the best model and scaler once when the app starts
best_model = joblib.load('diabetes_best_model.pkl')
scaler = joblib.load('scaler.pkl')

@app.route('/')
def home():
    return render_template('index.html', prediction=None)

@app.route('/predict', methods=['POST'])
def predict():
    # Get values from form
    pregnancies = float(request.form['Pregnancies'])
    glucose = float(request.form['Glucose'])
    blood_pressure = float(request.form['BloodPressure'])
    skin_thickness = float(request.form['SkinThickness'])
    insulin = float(request.form['Insulin'])
    bmi = float(request.form['BMI'])
    diabetes_pedigree = float(request.form['DiabetesPedigreeFunction'])
    age = float(request.form['Age'])

    # Input data for prediction
    input_data = np.array([[pregnancies, glucose, blood_pressure, skin_thickness, insulin, bmi, diabetes_pedigree, age]])

    # Scale the input data using the scaler
    input_data_scaled = scaler.transform(input_data)

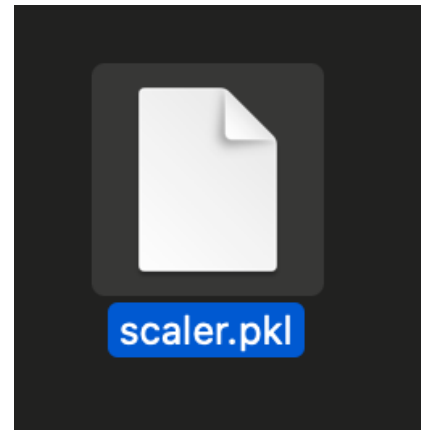
    # Make prediction using the best model
    prediction = best_model.predict(input_data_scaled)[0]

    # Return the prediction result
    if prediction == 0:
        result = "No Diabetes"
    else:
        result = "Diabetes"

    return render_template('index.html', prediction=result)
```

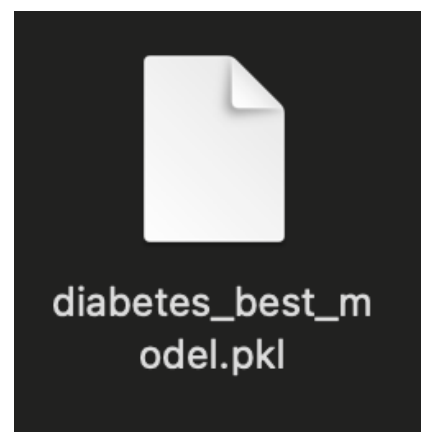
5. Data Preprocessing:

- **Scaling:** The data is normalized using the `scaler.pkl` file, which is a trained `StandardScaler` model. This is crucial because machine learning models generally perform better with normalized data.
- The scaler ensures that all input features are on the same scale, preventing any one feature from disproportionately affecting the prediction.



6. Making Predictions:

- After preprocessing, the processed data is passed to the trained model, which is stored in the `diabetes_best_model.pkl` file.
- This model is a machine learning classifier (e.g., Logistic Regression, Random Forest, or Support Vector Machine) that has been trained on diabetes-related data.
- The model predicts whether the user is at risk of diabetes (1) or not (0) based on the inputs.



7. Output Generation:

- The model's prediction is captured and processed in the backend.
- The result (either 0 or 1, indicating no diabetes risk or high risk) is returned to the frontend and displayed to the user.

8. Displaying Results:

- The prediction is displayed on the same webpage after form submission.
- The result is shown in a user-friendly format, indicating whether the individual is likely to have diabetes or not.
- A message is displayed along with the prediction result to guide the user.

Predict

Prediction: No Diabetes

OR

Predict

Prediction: Diabetes

Where 1 refers to Diabetes and 0 refers to No Diabetes, which basically signify chances of having diabetes in the near future or maybe presently.

9. Error Handling:

- If any of the input fields are left empty or contain invalid data, the app handles the errors gracefully.
- The app may display an error message prompting the user to enter valid information or complete all required fields.

Diabetes Prediction

Pregnancies:

Glucose:



Please fill in this field.

Blood Pressure:

Insulin:

22.3



BMI:



Please enter a valid value. The two nearest valid values are 22 and 23.

2

10. User Interaction & Feedback:

- After receiving the prediction, the user has the option to either try another prediction by resetting the form or exit the app.
- The app is designed to give feedback in a simple and easy-to-understand way to help the user interpret the results effectively.

11. Running the App Locally:

- Once you start the app, it runs locally on **http://127.0.0.1:5000**, where you can access and interact with the prediction model.
- The app is designed to be lightweight and simple, ideal for running on local machines for testing and demonstrations.

12. Technologies Used:

- **Frontend:**
 - HTML for the basic structure of the user interface.
 - CSS for styling and ensuring the app is visually appealing and user-friendly.
- **Backend:**
 - Flask (Python web framework) for building and running the application.
 - `joblib` for loading and using the trained model and scaler.
 - Python libraries like `sklearn` for data preprocessing and machine learning model implementation.

The structure of the project is as follows:

```
assignment2/
|—— app.py
|—— app/
|   |—— __init__.py
|   |—— routes.py
|   |—— templates/
|       |—— index.html
|   |—— static/
|       |—— style.css
|—— models/
|   |—— scaler.pkl
|   |—— diabetes_best_model.pkl
|—— requirements.txt
|—— .gitignore
```

Breakdown of Each File/Folder:

- **app.py:**
 - This is the entry point of the Flask app. It initializes and runs the application. It also configures the app to read from the `app/` folder for routes and templates.
- **app/:**
 - This folder holds the main application logic and related files:
 - **__init__.py:** Initializes the Flask app, making it a Python package. It also sets up configurations.
 - **routes.py:** Contains the route definitions for handling requests. For example, `/predict` where input data is passed to the machine learning model, and `/` to load the home page.
 - **templates/:** Contains HTML templates (views). Here, it includes `index.html`, which is the main webpage that the user interacts with.
 - **static/:** Contains static files like CSS, JavaScript, and images. In this case, `style.css` is used for styling the app's UI.
- **models/:**
 - Contains the trained machine learning models and any necessary preprocessing files (like scalers).
 - **scaler.pkl:** A pretrained scaler (e.g., `MinMaxScaler` or `StandardScaler`) used to preprocess the user input before passing it to the model.
 - **diabetes_best_model.pkl:** The trained machine learning model used for prediction (e.g., logistic regression or random forest model).
- **requirements.txt:**
 - This file lists all the required libraries (such as `Flask`, `scikit-learn`, etc.) that the project depends on. When someone clones the repo, they can install the dependencies with `pip install -r requirements.txt`.

- **.gitignore:**
 - This file tells Git which files or folders should not be tracked. For example, it typically includes the `venv/` folder to avoid uploading virtual environment files to GitHub.