Diabetes Prediction Web Application - Documentation

Author: Satyam Gajjar

This documentation provides an in-depth explanation of a machine learning-powered Diabetes Prediction Web Application created using Streamlit, NumPy, and Pickle. The application loads a pre-trained diabetes classifier model and takes eight health-related parameters as input to predict whether a person likely has diabetes.

1. Project Overview

The ML model, trained on a diabetes dataset, is stored as a .sav file using Pickle and later loaded for real-time predictions. Streamlit provides the UI where users input medical values. Once submitted, values are processed and fed into the model which returns prediction instantly.

2. Technologies Used

- NumPy → For array processing and reshaping. - Pickle → To load and reuse the saved trained model. - Streamlit → To build and display the interactive prediction UI.

3. Code Breakdown

• The model is loaded with pickle.load() from stored .sav file. • Input values are read as text, converted to numpy array and reshaped for prediction. • diabetes_prediction() handles preprocessing and model prediction logic. • Streamlit UI provides fields for user input and displays prediction output.

4. Running The Application

Run the command below inside the project directory to start the Streamlit application: streamlit run app.py The app will launch automatically in the browser (http://localhost:8501) where you can enter values and generate predictions.

End of Documentation

Diabetes Prediction Web Application – Detailed Code Documentation

Author: Satyam Gajjar

This document provides a detailed, line-by-line style explanation of the Python script used to build a Diabetes Prediction Web Application. The app loads a pre-trained machine learning model using Pickle, accepts user input through a Streamlit interface, and predicts whether a person is likely diabetic based on eight health-related parameters.

1. File Header and Metadata

The script begins with a shebang line and encoding declaration generated by the IDE (Spyder):

#!/usr/bin/env python3
 # -*- coding: utf-8 -*-
These indicate that the file should be executed with Python 3 and that the file uses UTF-8 encoding.

The commented block with *Created on* and *@author* is simple metadata for documentation and '' has no effect on code execution.

2. Import Statements

The script imports three key libraries:

import numpy as np
NumPy is used for numerical operations, primarily to convert input data into arrays and reshape it into the format expected by the machine learning model.

import pickle
Pickle is a Python module used for object serialization and deserialization. In this project, it is used to load the pre-trained model saved as a *.sav* file.

import streamlit as st
Streamlit is a Python framework for quickly building web applications. It provides UI elements like text input fields, buttons, and output areas directly from Python code.

3. Loading the Trained Model

The next line loads the trained model from disk:

loaded_model = pickle.load(open("/path/to/trained_model.sav", "rb"))

Here is what happens step-by-step:
• **open(..., "rb")** opens the file in *read-binary* mode.
• **pickle.load(...)** reads the binary file and reconstructs the original trained model object.
• The resulting model object is stored in the variable **loaded_model**, which is then used later to make predictions.

4. The diabetes_prediction() Function

This function encapsulates the logic for preparing input data and getting the prediction from the model:

def diabetes_prediction(input_data):
The function accepts a single argument **input_data**, typically a list or tuple containing eight numerical values " representing the person's health parameters.

Inside the function:

input_data_as_numpy_array = np.asarray(input_data)
• Converts the Python list/tuple into a NumPy array.
• This is necessary because scikit-learn models expect inputs as arrays, not plain Python lists.

input_data_reshaped = input_data_as_numpy_array.reshape(1, -1)
• Reshapes the array into 2D form, with 1 row and as many columns as there are features.
• The model expects data in the shape **(n_samples, n_features)**.
• Since we are predicting for a single person, **n_samples = 1**.

prediction = loaded_model.predict(input_data_reshaped)
• Passes the reshaped array to the model's **predict** method.
• The model returns an array-like output, typically **[0]** or **[1]** for a binary classification task like diabetes detection.

```
if (prediction[0] == 0):
    return "The person is not diabetic"
else:
    return "The person is diabetic"
```

• The function checks the first (and only) prediction value.
• If the value is **0**, the function returns a human-readable message indicating no diabetes.
• If the value is **1**, the function returns a message indicating the person is diabetic.

5. The main() Function – Streamlit User Interface

The **main()** function defines the web application's layout and behavior using Streamlit components.

st.title('Diabetes Prediction Web Application')
• This sets the title of the web application, displayed at the top of the page.

Then, the script collects user input for each of the eight health parameters using **st.text_input()**:

```
Pregnancies = st.text_input('Number of Pregnancies')
Glucose = st.text_input('Glucose level')
BloodPressure = st.text_input('BloodPressure value')
SkinThickness = st.text_input('SkinThickness value')
Insulin = st.text_input('Insulin level')
BMI = st.text_input('BMI value')
DiabetesPedigreeFunction = st.text_input('Diabetes Pedigree Function value')
Age = st.text_input('Age of the person')
```

• Each call creates a text input box on the page.
• The value entered by the user is stored as a string in the corresponding variable.

diagnosis = ''
• Initializes an empty string to hold the prediction message that will later be displayed to the user.

if st.button('Diabetes Test Result'):
• Creates a button labeled "Diabetes Test Result".
• When the user clicks this button, the code inside the *if* block is executed.

Inside the button block:

diagnosis = diabetes_prediction([Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age])
• A list of inputs is passed to the **diabetes_prediction()** function.
• Note: By default, these values are strings. In a more robust version of the code, they should be converted to numeric types (e.g., float) before prediction.
• The function returns either "The person is not diabetic" or "The person is diabetic", which is stored in **diagnosis**.

st.success(diagnosis)
• Displays the **diagnosis** message in a green success box on the Streamlit web page.
• If the button has not been pressed yet, this will show an empty string.

6. Script Entry Point

The final part of the script ensures that **main()** is executed when the script is run directly:

```
if __name__ == '__main__':
    main()
```

• **__name__** is a special Python variable. When the script is executed directly, its value is **'__main__'**.
• This condition prevents **main()** from running if the file is imported as a module in another script.

7. How to Run the Application

To start the Streamlit web application, open a terminal in the project directory and run:

streamlit run app.py

Streamlit will start a local web server and provide a URL (usually http://localhost:8501) where the application can be accessed. " Users can then enter their data, click the button, and immediately see the prediction result.

8. Notes and Possible Improvements

• Convert all text inputs to float before passing them into **diabetes_prediction()** to ensure proper numeric handling.
• Add validation and error handling to catch invalid or missing values.
• Use sliders or number inputs instead of plain text boxes for better UX.
• Deploy the app to Streamlit Cloud or another hosting platform to make it publicly accessible.

Documentation prepared by **Satyam Gajjar**.