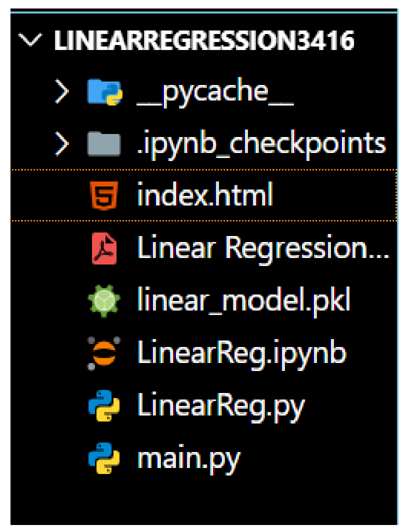
Logistic Regression Model Integrated with Website using FastAPI

Name: Barsha Baibhabi
Roll No: 22053416

1. Overview

This project implements a Logistic Regression model with L1 (Lasso), L2 (Ridge), and ElasticNet (L1& L2) regularization techniques. The model is trained on the Breast Cancer dataset from `sklearn.datasets`. The trained model is integrated into a web application using FastAPI for the backend and an HTML/JavaScript-based frontend for user interaction.

List of Files:



1. **`LogisticReg.ipynb`** - Jupyter Notebook for training and saving models.
2. **`main.py`** - FastAPI backend to handle predictions (Runs on port **8003**).
3. **`index.html`** - Frontend UI with inline CSS & JavaScript for user input and displaying predictions
4. **`model.pkl`** & **`scaler.pkl`** - Saved logistic regression models and data scaler.
5. **LogisticReg.py** -Python script version of the Jupyter Notebook.

---

2. Installation & Setup

Prerequisites

Ensure you have Python installed, along with the following dependencies:

```
pip install fastapi uvicorn scikit-learn pandas numpy pydantic
```

Running the FastAPI Server

Start the FastAPI backend with:

```
uvicorn main:app --host 127.0.0.1 --port 8003 --reload
```

3. Training the Model & Generating Pickle File

The `decision_tree_model.ipynb` notebook does the following:

- Loads the Diabetes dataset from `sklearn.datasets`.
- Splits the data into training and test sets.
- Trains a Decision Tree Regressor.
- Saves the trained model as a pickle file (`decision_tree_model.pkl`).

4. FastAPI Backend (`main.py`)

The backend is implemented using FastAPI to:

- Load the trained model from `decision_tree_model.pkl`.
- Accept feature inputs from the frontend via a POST request.
- Process input features and return the predicted diabetes progression value.

```python
@app.post("/predict")
def predict(input_data: PredictionInput):
    try:
        if input_data.model_type not in models:
            raise HTTPException(status_code=400, detail="Invalid model type")

        model = models[input_data.model_type]
        data = np.array(input_data.data).reshape(1, -1)
        data = scaler.transform(data)
        prediction = model.predict(data)
        return {"prediction": int(prediction[0])}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8003)
```

- Runs on `http://127.0.0.1:8006/`

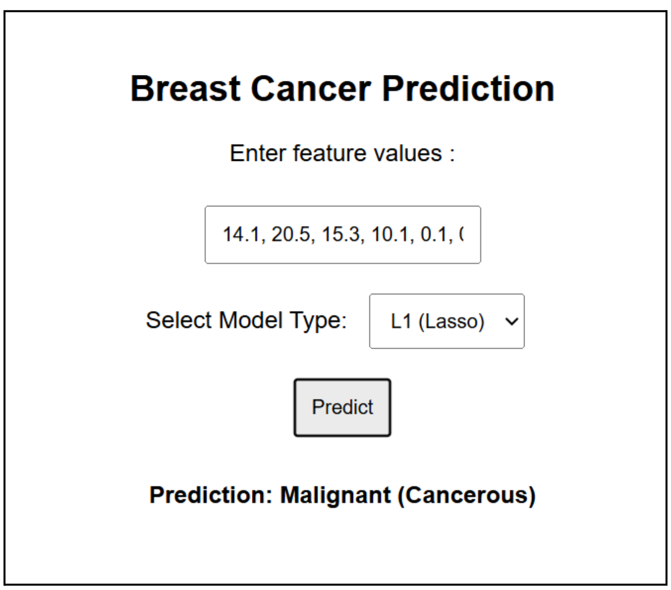
5. Frontend (`index.html`) with Inline CSS & JavaScript

The frontend:

- Provides an input form with 10 fields for feature values.
- Uses JavaScript to send AJAX requests to the FastAPI backend.

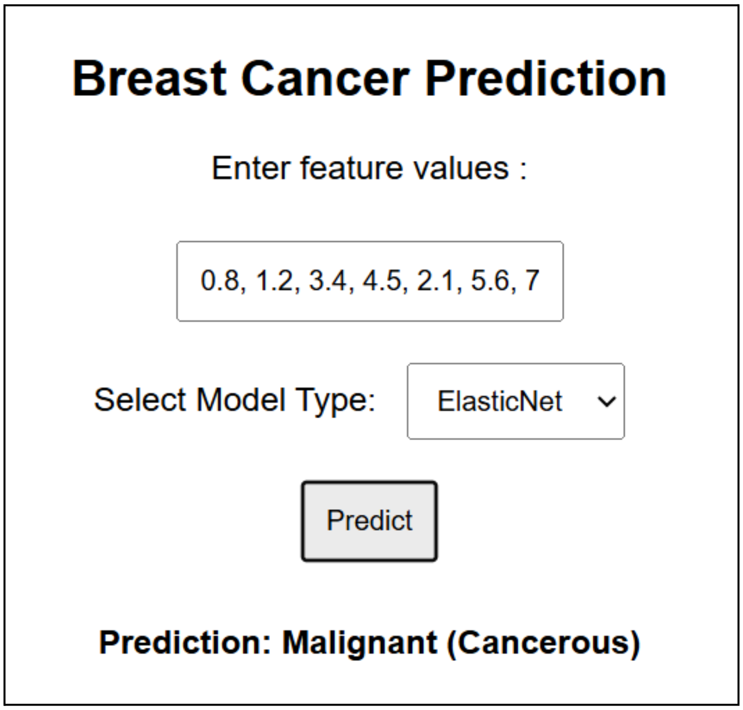
```
function predict() {
    let features = document.getElementById("features").value.split(",").map(Number);
    let modelType = document.getElementById("model_type").value;

    fetch("http://127.0.0.1:8003/predict", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ data: features, model_type: modelType })
    })
    .then(response => response.json())
    .then(data => {
        let predictionText = data.prediction === 1 ? "Malignant (Cancerous)" : "Benign (No Cancer)";
        document.getElementById("result").innerText = "Prediction: " + predictionText;
    })
    .catch(error => {
        document.getElementById("result").innerText = "Error: " + error;
    });
}
```

- Displays the predicted progression value, categorized as "High" or "Low" based on a threshold.



6. Testing the Integration

- Enter 10 numerical feature values in `index.html`.
- Click Predict to send a request to `FastAPI`.
- The API returns the predicted diabetes progression score.



- The result is displayed on the webpage as "High" or "Low".

7. Conclusion

This project successfully integrates a Decision Tree regression model with a FastAPI backend and a simple web frontend. It provides a foundation for further enhancements, such as improved UI, database integration, and additional model tuning.