ML-ASSIGNMENT#03

CS4090-Machine Learning for Robotics
Assignment

Name: Emaan Ali 22i-2325 CS-J

GitHub-Repo Link: https://github.com/Emi-Pemi/ML_A3_Model_Development

HuggingFace-Link: https://huggingface.co/Emi-Pemi/A3_Model_Development

ML-Model Development

In this project, we developed a machine learning model to predict the housing prices in California using data from the California Housing dataset. We implemented two primary scripts for the model training and inference phases:

- train.py: The script responsible for training the model using Gradient Descent, applying L1/L2 regularization to prevent overfitting, and utilizing early stopping for better optimization.
- inference.py: This script loads the trained model, accepts dynamic user input for predictions, and provides the predicted values in an interpretable format (dollars).

This report outlines the implementation and technical details of both scripts, including the data preprocessing, model training, and inference process.

train.py Implementation:

- 2.1. Model Development:
- In train.py, we used a custom implementation of Linear Regression with Gradient Descent. The model was enhanced with:
- Regularization: We applied L1 (Lasso) and L2 (Ridge) regularization to prevent overfitting and improve the generalization of the model.
- Early Stopping: To avoid overfitting and excessive computation, we implemented an early stopping mechanism. The training process halts if the validation loss doesn't improve for a specified number of epochs.

2.2. Data Preprocessing:

Before training, we applied several preprocessing steps:

- Data Scaling: The features were scaled using StandardScaler to normalize the data and improve the convergence of the gradient descent algorithm.
- Log Transformation: Since the target variable (housing prices) has a skewed distribution, we applied a log transformation to stabilize variance and make the data more suitable for linear regression.

2.3. Model Training:

The training process follows these steps:

1. Initialize the model weights and bias.
2. For each epoch, calculate the model's predictions.
3. Compute the gradients of the loss function with respect to the weights and bias.
4. Update the weights and bias using gradient descent.
5. If early stopping conditions are met (no improvement in validation loss for a set number of epochs), stop the training early.

The model was trained on the California Housing dataset, with 80% of the data used for training and 20% reserved for testing and validation.

2.4. Evaluation and Model Saving:

After training, we evaluated the model using Mean Squared Error (MSE) and R^2 metrics on the validation and test sets. The final trained model, along with its weights, bias, and data preprocessing parameters, was saved to a file using joblib.

```
# Saving the trained model
joblib.dump(model_data, 'model.joblib')
```

inference.py Implementation:

The inference.py script is responsible for loading the trained model and scaler. The model is downloaded from Hugging Face using the hf_hub_download function. The model’s weights, bias, and scaling parameters are extracted for use in the inference process.

```
from huggingface_hub import hf_hub_download

model_path = hf_hub_download(
    repo_id="Emi-Pemi/A3_Model_Development",
    filename="model.joblib",
    cache_dir=".cache"
)
self.model_data = joblib.load(model_path)
```

Input Validation:

In order to ensure that the input features are within valid ranges, the inference.py script includes an input validation function. It checks if the provided data falls within the acceptable ranges (defined during training), and if necessary, it clips the values to these bounds.

```
def _validate_input(self, input_data):
    input_array = np.array(input_data, dtype=float).reshape(1, -1)
    mins = np.array(self.model_data['input_ranges']['min'])
    maxs = np.array(self.model_data['input_ranges']['max'])
    return np.clip(input_array, mins, maxs)
```

Prediction

After validation, the script makes a prediction by applying the appropriate inverse transformation (e.g., exponential or square root transformation) to the model's output. The result is then returned in the form of a housing price in dollars, clipped to reasonable bounds.

```
def predict(self, input_data):
    validated_input = self._validate_input(input_data)
    raw_pred = self.pipeline.predict(validated_input)[0]
    # Apply transformation and return value
```