Project Tasks

Part 1: Pure Python Implementation

- Develop a multivariable linear regression algorithm using only core Python features.
- Do not utilize external libraries (e.g., `NumPy`, `Pandas`, `Scikit-learn`).
- Avoid using large language models (LLMs) such as GPT for code generation. If you do, clearly mention in the report where it was used, or else you will be disqualified directly.
- Prioritize simplicity and mathematical clarity.
- Employ gradient descent as the optimization technique.

Part 2: Optimized NumPy Implementation

- Re-implement the algorithm using `NumPy` to leverage vectorized operations.
- Enhance performance through appropriate parallelization.
- Maintain consistency with the pure Python logic to ensure a fair comparison.

Part 3: Scikit-learn Implementation

- Utilize the `LinearRegression` class from the `scikit-learn` library.
- Train the model on the identical dataset used in Parts 1 and 2.

Summary of Python Data Cleaning Code — `data_cleaning.ipynb`

Summary of Each Code Block

Code 1. Checking for Empty Cells
The code reads a CSV file and checks for any empty cells by iterating through each row and column. If a blank cell is found, it prints the corresponding row and column name.

Code 2. Finding Unique Categories in `ocean_proximity`
Using pandas, this code extracts and prints all unique string categories in the `ocean_proximity` column from the dataset.

Code 3. Calculating Average Median House Value per Category
This block calculates the average `median_house_value` for each unique category in the `ocean_proximity` column using Python's CSV module and `defaultdict`.

Code 4. Replacing Categorical Column with Average Values
Pandas is used to replace string values in the `ocean_proximity` column with their corresponding average house values based on the previous step. The new DataFrame is saved as a separate CSV file, which improves model performance (e.g., R^2 score increased from 0.64 to 0.67).

Code 5. Filling Missing `total_bedrooms` Using Linear Regression
This code uses linear regression to estimate and fill missing values in the `total_bedrooms` column using `total_rooms` and `population` as features. It computes regression parameters with the Normal Equation and saves the updated data to a new CSV file.

Multivariable Linear Regression Model with Gradient Descent — core.ipynb

This model implements a multivariable linear regression to predict a target variable based on several input features derived from ocean dataset attributes. The key steps and components of the model are as follows:

Data Preprocessing

- Raw data is read from a CSV file and relevant features are extracted, including longitude, latitude, age, income, and engineered ratios such as rooms per person and income per household.
- Outliers in the target variable (y) are clipped at the 99th percentile to reduce their effect on training.
- Features are normalized using z-score normalization:

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$

  where μ_j and σ_j are the mean and standard deviation of feature j.
- A bias term (intercept) of 1.0 is added to each feature vector.
- The target variable is scaled to $[0, 1]$ by min-max normalization:

$$y^{(i)} = \frac{y^{(i)} - y_{\min}}{y_{\max} - y_{\min}}$$

Model Formulation

The prediction for the i-th example is given by the linear model:

$$\hat{y}^{(i)} = \sum_{j=0}^{m} w_j x_j^{(i)}$$

where w_j are the model weights and $x_0^{(i)} = 1$ represents the bias term.

Cost Function with Regularization

The objective is to minimize the regularized mean squared error cost function:

$$J(\mathbf{w}) = \frac{1}{2n} \sum_{i=1}^{n} \left(\hat{y}^{(i)} - y^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{m} w_j^2$$

where n is the number of training samples and λ is the regularization parameter (ridge regression).

Gradient Descent Optimization

Weights are updated iteratively using gradient descent:

$$w_j := w_j - \alpha \frac{\partial J(\mathbf{w})}{\partial w_j}$$

where α is the learning rate. The gradient for each weight is computed as:

$$\frac{\partial J}{\partial w_j} = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{y}^{(i)} - y^{(i)} \right) x_j^{(i)} + \lambda w_j, \quad j \geq 1$$

For the bias term w_0, regularization is not applied:

$$\frac{\partial J}{\partial w_0} = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{y}^{(i)} - y^{(i)} \right) x_0^{(i)}$$

Training

- The model runs for a fixed number of epochs (10,000).
- At each epoch, gradients are accumulated over all training examples.
- Costs are recorded to monitor convergence.
- Training time is measured to evaluate efficiency.

Summary

This approach demonstrates a core implementation of multivariable linear regression with gradient descent and L2 regularization. Feature scaling, bias inclusion, and regularization help in stabilizing and improving model performance, while gradient descent iteratively minimizes the cost function to find the optimal weights.

Model Evaluation and Prediction Summary

The evaluation function computes the model's prediction accuracy using three metrics:

- **Mean Absolute Error (MAE)**:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

  where y_i are true values and $\hat{y}_i$ are predicted values.

- **Root Mean Squared Error (RMSE)**:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **Coefficient of Determination (R^2 Score)**:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

  where $\bar{y}$ is the mean of true values.

For prediction on a custom input vector:

1. The input features are first normalized using the training data mean (μ_j) and standard deviation (σ_j):

$$x_j^{norm} = \frac{x_j - \mu_j}{\sigma_j} \quad \text{if } \sigma_j \neq 0, \quad \text{else } 0$$

2. A bias term of 1.0 is prepended to the normalized input vector.

3. The normalized prediction is computed as the weighted sum:

$$\hat{y}_{norm} = \sum_{j=0}^{m} w_j x_j^{norm}$$

4. The prediction is clipped to the range $[0, 1]$ and then scaled back to the original target range:

$$\hat{y} = \hat{y}_{norm} \times (y_{\max} - y_{\min}) + y_{\min}$$

Cost Function Convergence Plot

The cost function convergence during training is visualized using a line plot where:

- The x-axis represents the number of **epochs** (iterations of training).
- The y-axis represents the **cost** value computed at each epoch.

The plot illustrates how the cost decreases as training progresses, indicating the effectiveness of gradient descent in minimizing the error.

Key elements of the plot:

- `Training Cost`: The value of the loss function at each epoch.
- Axes labels for clear interpretation: *Epochs* on the x-axis and *Cost* on the y-axis.
- A grid and legend to enhance readability.

Gradient Descent Convergence Plot

The convergence of the gradient descent optimization is visualized by plotting the cost function over training epochs.

- The x-axis represents the **Epoch** number.
- The y-axis represents the **Cost**, which includes the Mean Squared Error (MSE) and a regularization term.
- The cost values are plotted as a blue line with a line width of 1 for clarity.
- Grid lines improve readability, and the layout is adjusted for better presentation.
- The plot title *Convergence of Gradient Descent* indicates the purpose of the visualization.

This plot helps in assessing how effectively the gradient descent algorithm reduces the loss function during training.

Multivariable Linear Regression Model with Gradient Descent — `numpy.ipynb`

Data Loading and Preprocessing

The dataset `updated_ocean.csv` contains housing-related features along with a target variable representing housing values. The preprocessing steps are as follows:

- **Feature Extraction:** From each valid row, the following features are extracted:
  - Longitude, Latitude, Age
  - Income (capped at 15)
  - Engineered features:
    - Rooms per person = $\frac{\text{total rooms}}{\text{population}}$
    - Bedrooms per room = $\frac{\text{total bedrooms}}{\text{total rooms}}$
    - Rooms per household = $\frac{\text{total rooms}}{\text{households}}$
    - Population per household = $\frac{\text{population}}{\text{households}}$
    - Income per household = $\frac{\text{income}}{\text{households}}$
  - Ocean proximity (encoded as a float)
- **Data Cleaning:** Rows with missing or invalid values, zero rooms, or zero households are discarded.
- **Outlier Handling:** Target values (y) are clipped at the 99th percentile to reduce the influence of extreme outliers.
- **Feature Standardization:** Each feature is standardized to zero mean and unit variance:
$$x_i = \frac{x_i - \mu_i}{\sigma_i}$$
with μ_i and σ_i being the mean and standard deviation of the i-th feature.
- **Bias Term:** A bias feature with constant value 1 is appended to the features matrix.
- **Target Normalization:** The target y is normalized to the range $[0, 1]$ by:
$$y = \frac{y - y_{\min}}{y_{\max} - y_{\min}}$$
- **Train-Test Split:** The dataset is split into training (80%) and testing (20%) subsets.

Model and Training

A linear regression model with L2 regularization (Ridge Regression) is trained using batch gradient descent. The objective function minimized is:

$$J(\mathbf{w}) = \frac{1}{2n} \sum_{i=1}^{n} \left(\hat{y}^{(i)} - y^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{d} w_j^2$$

where:

- n is the number of training samples.
- d is the number of features (including bias).
- $\hat{y}^{(i)} = \mathbf{x}^{(i)T}\mathbf{w}$ is the predicted output.
- $\lambda = 0.1$ is the regularization parameter controlling weight decay.

Gradient Descent Update Rule

Weights $\mathbf{w}$ are updated iteratively using:

$$\mathbf{w} := \mathbf{w} - \alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

where learning rate $\alpha = 0.01$, and gradient is computed as:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}$$

with the exception that the bias term weight is not regularized.

Training Details

- Initial weights are randomly initialized from a normal distribution with mean 0 and standard deviation 0.01.
- Number of epochs (iterations) is set to 10,000.
- Training time is measured and output after completion.
- Cost (loss) is recorded at every epoch for monitoring convergence.

Summary

The approach combines robust data preprocessing, feature engineering, normalization, and ridge regression trained with batch gradient descent. The regularization term helps prevent overfitting by penalizing large weights, especially on the features (excluding the bias). The training is performed over many iterations to ensure convergence of the model parameters.

Model Evaluation and Custom Prediction

The trained model is evaluated on the test set using the following metrics:

- **Mean Absolute Error (MAE):** Average absolute difference between true and predicted values.
- **Root Mean Squared Error (RMSE):** Square root of average squared difference between true and predicted values.
- **Coefficient of Determination (R^2 score):** Proportion of variance explained by the model.

The evaluation function computes these metrics as follows:

$$\begin{aligned}
\hat{\mathbf{y}} &= \mathbf{X}\mathbf{w} \\
\text{errors} &= \mathbf{y}_{true} - \hat{\mathbf{y}} \\
\text{MAE} &= \frac{1}{n}\sum_{i=1}^{n} |\text{errors}_i| \\
\text{RMSE} &= \sqrt{\frac{1}{n}\sum_{i=1}^{n} \text{errors}_i^2} \\
R^2 &= 1 - \frac{\sum_{i=1}^{n} \text{errors}_i^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}
\end{aligned}$$

where $\bar{y}$ is the mean of true target values.

A custom input vector representing housing features is preprocessed with the same standardization as the training data, including adding a bias term. The model predicts the normalized target value which is clipped to $[0, 1]$ and then transformed back to the original scale:

$$\begin{aligned}
\hat{y}_{norm} &= \mathbf{x}_{custom}^T \mathbf{w} \\
\hat{y} &= \hat{y}_{norm} \times (y_{max} - y_{min}) + y_{min}
\end{aligned}$$

Evaluation and prediction run time is recorded for performance monitoring. Finally, the predicted housing value, evaluation metrics, and learned weights are printed.

Cost Function Convergence

Figure **??** shows the progression of the cost function value throughout the training process using gradient descent. The horizontal axis represents the number of training epochs, while the vertical axis denotes the cost function value computed at each epoch.

Initially, the cost decreases rapidly, indicating that the model weights are being updated effectively to reduce prediction errors. As training continues, the rate of decrease slows down and the cost curve approaches a stable minimum value, demonstrating convergence of the optimization algorithm.

This behavior confirms that the gradient descent algorithm, with L2 regularization, successfully minimizes the loss and improves model performance over time.

Cost Function Convergence with L2 Regularization

Figure **??** presents the convergence of the cost function during training using gradient descent with L2 regularization. The plot shows the cost value on the vertical axis against the number of epochs on the horizontal axis.

The cost decreases steadily over the epochs, indicating effective minimization of the loss function with regularization. The final cost value is highlighted and annotated on the plot, providing a clear indication of the convergence point after 10,000 training iterations.

The inclusion of L2 regularization helps prevent overfitting by penalizing large weights, contributing to smoother and more stable convergence behavior.

Scikit implementation— `scikit.ipynb`

Multivariable Ridge Regression Model with Feature Engineering

1. Data Loading

- Data is loaded from `updated_ocean.csv`, skipping the header row.
- Each data point contains geographic, demographic, and economic variables, including *longitude*, *latitude*, *age*, *income*, and a categorical *ocean proximity* value (numerically encoded).
- Rows with missing or invalid data (e.g., zero rooms or households) are ignored.

2. Feature Engineering

- Derived features include:

$$\begin{aligned}
\text{rooms_per_person} &= \frac{\text{total rooms}}{\text{population}} \\
\text{bedrooms_per_room} &= \frac{\text{total bedrooms}}{\text{total rooms}} \\
\text{income_per_household} &= \frac{\text{income}}{\text{households}} \\
\text{rooms_per_household} &= \frac{\text{total rooms}}{\text{households}} \\
\text{population_per_household} &= \frac{\text{population}}{\text{households}}
\end{aligned}$$

- These features aim to improve model generalization by capturing more expressive patterns.

3. Target Preprocessing

- Target variable: *median house value*.
- Outliers above the 99^{th} percentile are clipped to reduce their influence.
- Normalized using min-max scaling:

$$y_{\text{norm}} = \frac{y - y_{\min}}{y_{\max} - y_{\min}}$$

4. Feature Normalization and Splitting

- Data is split into 80% training and 20% testing.
- Features are standardized using `StandardScaler` (zero mean and unit variance).

5. Model and Training

- Model: Ridge Regression (Linear Regression with L_2 regularization).
- Ridge minimizes:

$$\min_w \|Xw - y\|_2^2 + \alpha \|w\|_2^2$$

- Regularization parameter: $\alpha = 0.01$
- Trained using gradient descent-based optimization with up to 20,000 iterations.
- Fitting time is recorded using the `time` module.

6. Advantages of Ridge over Plain Gradient Descent

- Handles multicollinearity and overfitting better by penalizing large coefficients.
- The use of `scikit-learn`'s optimized backend provides faster convergence and numerical stability.
- Scalable to larger feature sets due to vectorized implementation and efficient memory usage.

7. Summary

This model leverages extensive feature engineering and Ridge regression to predict housing values. Preprocessing steps like clipping and normalization help in stabilizing the learning process, while standardization ensures effective regularization. The model is trained with high efficiency using `scikit-learn`'s implementation of Ridge regression.

Predictions and Evaluation

- The trained Ridge regression model is evaluated on the test dataset using:
  - Mean Absolute Error (MAE)
  - Root Mean Squared Error (RMSE)
  - Coefficient of Determination (R^2 score)
- All metrics are scaled back to the original target range using the formula:

$$\text{original value} = \text{normalized value} \times (y_{\max} - y_{\min}) + y_{\min}$$

- A custom data point is predicted using the trained model:
  - The custom input is standardized using the same `StandardScaler` fitted on training data.
  - Output prediction is clipped to [0, 1] range, then transformed back to original scale.
- Model outputs:
  - Predicted house value for the custom input
  - Evaluation metrics (MAE, RMSE, R^2)
  - Learned coefficients and intercept of the Ridge model